Performance and Ease of Use in 3D on the Web
Comparing Babylon.js with Three.js

Julia Johansson
This bachelor thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for a bachelor’s degree in Software Engineering. The thesis is equivalent to 10 weeks of full-time studies.

Contact Information
Author:
Julia Johansson
E-mail: jujin18@student.bth.se

University advisor:
Emil Folino
Department of Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Abstract

3D JavaScript frameworks are used for creating interactive 3D in web applications. There are two prominent frameworks: Babylon.js and Three.js. They are often compared to one another where performance and ease of use are mentioned as important factors.

A web developer could face choosing a 3D framework. This thesis aims to aid in that process. It investigates how Babylon.js compares to Three.js when it comes to performance such as memory use and frames per second for the end-user consuming a web app. It also investigates how they compare when it comes to ease of use for the developer in terms of initial learnability.

For the performance part, a web app with a minimalistic one box animation was created with each framework and deployed on Netlify. The web app was accessed with a browser’s developer tools to record memory and frames per second. For the ease of use part, observation sessions were made with nine developers who were tasked with getting a basic 3D model up and running locally with each framework with the official documentation as a starting point, one hour limit respectively.

Results showed that both frameworks performed equally when it came to frames per second, and Babylon.js used 2,2 MB, 46 %, more memory. Only five out of eight or 63 % of the participants managed to implement a 3D model in Babylon.js and five out of nine or 56 % in Three.js. Three.js scored slightly higher on ease of use than Babylon.js, but neither framework scored high on the overall scale. Participants favoured Babylon.js when asked to choose. However, almost all had a frustrating experience with both during the sessions.

Performance is similar except for memory use when it comes to lightweight implementations in these frameworks. Initial learnability could be a gatekeeper. If users do not get past the first hurdle fast with quick feedback that a framework works in the most basic sense, a conceivable risk is they could give up and never use it.

Keywords: 3D JavaScript frameworks, performance, ease of use
Contents

Abstract 3

1. Introduction 6
  1.1 Background 6
  1.2 Focus Area 7
  1.3 Motive and Value 7

2. Research Questions 9

3. Method 10
  3.1 Literature Study 10
  3.2 Empirical Study 10
  3.3 Experiment 11
  3.4 Observation Study and Questionnaires 12
  3.5 Finding Participants 15
  3.6 Validity Threats 15
    Internal validity 15
    External validity 17

4. Literature Study 19
  4.1 Performance in 3D Web Apps 19
  4.2 Important Metrics in 3D Performance 19
  4.3 How To Measure Performance 20
  4.4 Science on Performance and Usability 21
  4.5 Terminology on Ease of Use and Scores 21
  4.6 Learnability and Getting Started 23
  4.7 Influx of New Software and Techniques 24
  4.8 Research Comparing Ease of Use or Learnability in 3D 24

5. Results 26
  5.1 Experiment: Performance 26
  5.2 Observation Study and Questionnaires 28
    Minutes to implement 3D model 29
    Prerequisites 29
    Usability evaluation 29
    Notes on instructions 30
    Notes on struggles 30
    Notes on documentation 30
    Notes on choice 30
  5.3 Overview Performance and Ease of Use 31

6. Analysis and Discussion 32
  6.1 Performance in Babylon.js and Three.js 32
  6.2 Ease of use in Babylon.js and Three.js 33
1. Introduction

On code.talks 2019 David Catuhe, the creator of Babylon.js, has just finished the talk titled *Create stunning 3D web experiences with Babylon.js*. It is now time for questions and the moderator says “So, what’s the advantage of Babylon.js over Three.js?” Catuhe then mentions they are a working team of ten people; they fix bugs in 24 hours, have backwards-compatibility and that Three.js is not an inferior tool, they just have different philosophies. Babylon.js aims more for professionals because they have support and quick bug fixing. The moderator then adds on a personal note that coming from Three.js and trying out Babylon.js the moderator had something up and running in an hour and “Getting started at least for me was way easier.”. Catuhe replies “That is why we have documentation. No, I’m kidding.” [1].

1.1 Background

3D graphics on the web have evolved during the last nine years on top of the Web Graphics Library, WebGL, API, which connects between browsers and computer hardware. WebGL makes it possible to render interactive 2D and 3D graphics on a low level [2], [3], [4]. During this time, several abstraction layer frameworks working through WebGL have emerged. They make it easier and faster to produce interactive 3D graphics. MDNWeb Docs on WebGL mention two prominent frameworks: Three.js and Babylon.js [3].

Developers often compare the two frameworks to one another, and popular opinion varies. Three.js is suggested for small projects because it is light and non-backwards compatible, Babylon.js for bigger longer projects where backwards compatibility is an issue. Ease of use is mentioned as a result of how good the documentation is. Babylon.js is said to have better documentation and Three.js sometimes worse and sometimes equal [5], [6], [7], [8], [9], [10], [11], [12], [13]. A developer who posted on Babylon.js community forum implemented the same code in both frameworks and said there is no difference, except Three.js is faster for simple code [6].

Performance in computing has always been a vital issue. With abstraction and adaptation to humans comes better usability but can be accompanied by an idea of a trade-off in less performance [14], [15]. The pursuit of having the cake and eating it, having both great computation and high usability, is one of the drivers for the evolution and adoption of programming languages [16].

Recently a joint browser vendor effort with representatives from influential browsers took it upon themselves to develop a new graphics web API by the name of WebGPU [15].
Self-reported tests from browser vendors say everything from a 10-15 per cent increase in performance (Google Chrome) to over seven times faster (Apple’s Safari) [15]. It is now a W3C task group [15].

1.2 Focus Area

This thesis focuses on performance and ease of use. Performance is narrowed down to runtime software performance for the end-user consuming a 3D web app. Ease of use is a concept in usability practices [17]. The ease of use part centers around the developer experience. It looks at the process from download to creating a simple ‘Hello, World!’ program or similar for Three.js and Babylon.js.

By looking at ease of use in an initial implementation phase, the thesis also centers around the concept of learnability. Learnability can be defined as how easy it is for users when they first encounter a software application and want to perform specific basic tasks [18]. For this thesis, that is the definition used and the participants perform the basic task of implementing a 3D primitive within the framework (cube or sphere). Learnability influences productivity, the adoption/rejection of software and general success of software [18].

The author decided early on that the frameworks had to adhere to certain requirements to be interesting in a thesis such as this. The frameworks had to be based on JavaScript and actively maintained as in have commits the last three months. They had to be used by professionals, used in both large and small productions and often mentioned in comparisons. Further, they had to be built on top of WebGL, focused on 3D and not be work in progress. They had to be released and have been around for at least a few years. Fifteen candidates came up when looking at 3D frameworks on the web and only two matched all requirements: Babylon.js and Three.js. See Appendix F for that research.

1.3 Motive and Value

JavaScript is a language where frameworks are in the thousands. Many factors can influence when someone decides which framework to use. One study examining factors leading to the adoption of a framework used The Unified Theory of Acceptance and Use of Technology for grouping factors [19]:

The factors are performance expectancy (performance, size), effort expectancy (automatization, learnability, complexity, understandability), social influence (competitor analysis, collegial advice, community size, community responsiveness), facilitating conditions (suitability, updates, modularity, isolation, extensibility), and price value. [19, p. 1]
In the making of this thesis, it has become apparent that 3D web graphics is an evolving field with several stakeholders and many opinions. The approach of new technology seems to fire up the spirits further. It could be argued there is a need for a more precise overview and a connection to facts in the areas of performance and ease of use because those are two notable factors that come up in comparisons, see 1.1 Background. These factors are also harder to look into than say size, collegial advice or community size/responsiveness. Insights drawn from facts on performance and ease of use could be useful for developers planning a project involving 3D and choosing among frameworks. To also take a step back and join two fields often separated by tradition, Software Engineering (focused on performance) and Human-Computer Interaction (usability), could be beneficial. Doing so could add an extra dimension to and perspective on framework choice.

From a business perspective, which tools, such as 3D frameworks, developers choose can have ramifications on many levels. In the first paragraph to this introduction, there is a mention of David Catuhe, creator of Babylon.js, presenting at a conference. What is important to think about is the larger context, framework creators do not present at conferences only because they are nice and love what they do. They exist in a space where they or nearby stakeholders make money directly or indirectly. Microsoft would probably not provide salaries for the team of ten to work with Babylon.js if they did not think it would somehow affect the bottom line positively. Which means they could be invested in developers choosing Babylon.js and that could in turn affect the overall landscape.

Furthermore, both Babylon.js and Three.js are open source and free to download. That does not mean there are no costs or gain benefits to take into account for a developer choosing a 3D framework. A framework’s runtime performance experience for end-users can affect how much developers need to work to compensate for bad response times or slow applications. Work translates to money, slow applications could also translate to a direct cost of losing users, either through e.g. advertising or subscription fees from say a game. When it comes to ease of use, less effort for the developer equals less time which also means less money. For an organisation, it could reduce the costs of training. Time-to-deployment is another time and work-related factor. It becomes a competitive advantage to work with tools that are easy to use. Hence why it is important to investigate this area with performance, ease of use and in particular Babylon.js and Three.js.
2. Research Questions

RQ1: How does Babylon.js compare to Three.js when it comes to performance?
Performance breaks down to these key indicators.

- Memory usage
- Frames per second

These indicators in RQ1 are a combination of what is relevant when measuring 3D performance as per the literature study and what is convenient to track within the scope of this project, see chapter 3. Method for further reading. Measuring memory usage and frames per second can give a picture of how the frameworks perform. By extension, we can then also see if they differ. Performance is significant on several levels. For an end-user consuming a web app, it could mean the difference between a positive, negative and no experience at all. On an organisational level, it concerns costs associated with workarounds to improve performance. On a societal level, performance affects availability and accessibility to web apps, a question of equality. Furthermore, performance connects to resources which are of concern to the planet and its people at large.

RQ2: How does Babylon.js compare to Three.js when it comes to ease of use?
Which framework is easier to get started with for a web developer without previous experience of a said library? Are there any influencing factors? When comparing the ease of use by answering RQ2 as well as the performance in RQ1, one can get quantitative data and compare ease of use to performance. With observation notes, one can get a qualitative angle as well. What participants say in the evaluation can differ from what they do and say during the session. Ease of use is relevant for several reasons. On an individual level, it enables developers to work with less effort. On an organisational level, it reduces the costs of training and time-to-deployment. On a market level, it becomes an advantage to work with easily implemented tools.

RQ3: What are some insights around performance and ease of use that could be useful for a developer when choosing a 3D framework? By just comparing performance in RQ1 and ease of use in RQ2, we could lose valuable conclusions. What if there are no differences? What other factors need to be considered and how significant are they? RQ3 adds a layer of what a developer could do with the information. There can be conclusions that developers can use to apply this thesis in their work.
3. Method

The thesis consists of two parts, a literature study and an empirical study. Methodology for both is as follows.

3.1 Literature Study

The search strategy for finding literature was based on using Summon@BTH, Blekinge Institute of Technology’s gathered databases. The search began with the peer-reviewed filter active, depending on results this was turned off. Turning off this filter was a regular occurrence. Snowballing was used to complement direct search (both forward and backward snowballing). Other criteria for identifying relevant literature were publication date and the number of citations—the closer to present-day, the more relevant because of the evolving field of 3D. When there was no academic literature available, technical specifications, documentation, forum posts and videos from conferences and talks were used to piece together information on browsers, WebGPU, frameworks and context.

Relevant keywords (used in combination or separate):


Learnability in software is also connected to machine learning, but this was outside of scope.

The literature study laid the foundation for answering RQ1, RQ2 and RQ3. It helped in particular with the choice of evaluation methods and scales. An additional benefit was the creation of a context, setting a frame for understanding the field and its relation to technology at large.

To find studies that were relevant on specifically 3D in either performance or usability proved hard and only returned a few.

3.2 Empirical Study

This part is divided into two, an experiment focusing on performance and an observation study with questionnaires focusing on ease of use. One overall influencing factor was the
different release cycles of Babylon.js and Three.js. Babylon.js releases about twice every year, one big update in the middle of this thesis period, Three.js does so once a month. That could have affected download pages, the stability of the software and further. However, it only affected the documentation links. The first observation session started one week after Babylon.js entire documentation rewrite, so it did not affect the consistency of sessions.

3.3 Experiment

The goal of the experiment was to generate a set of data on key performance indicators connected to Babylon.js and Three.js implementations, thereby being able to answer RQ1. Two websites were set up with an animated 3D cube and a skeleton to get the apps up and running without introducing influencing factors [20]. They were placed in separate builds on a deployment platform called Netlify, see Appendix A. Time constraint led to the choice of minimalistic apps.

<table>
<thead>
<tr>
<th></th>
<th>Three.js</th>
<th>Babylon.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework</td>
<td>r123</td>
<td>4.2</td>
</tr>
<tr>
<td>Webpack</td>
<td>5.6</td>
<td>5.6</td>
</tr>
</tbody>
</table>

The build size of each app with the bundled JavaScript code was first 3.4 MB in the Babylon.js app and 5.8 KB in the Three.js app. Babylon.js can be imported with ES modules, this is not the default in guides, but was recommended when digging more in-depth in the documentation and through posting on the official forum.

After just using the Babylon.js core and importing selected modules, the build size came down to 962 KB. That process drew efforts from making the visual side of the apps similar, hence why they look different, see Figure 1 under 5. Results. The Babylon.js cube moves to the left and right, the Three.js rotates around. Because the Three.js implementation was relatively small already and the initial implementation process took longer, the decision was made to skip importing separate modules and just use import * as THREE from ‘three’;

Standard framework-specific materials were used and default cameras from the documentation. The Babylon.js camera can be rotated around the object, the Tree.js is a stationary one. This could have an impact on the result because of Babylon.js having a more complex implementation.
Both frameworks recommend practices for downsizing such as tree-shaking, gzipping and more. That was not implemented.

After the literature study, the choice fell on measuring metrics manually with Firefox Developer Edition due to a combination of factors: 1) Browser automation framework Selenium discourages performance evaluation use and 2) The gains by automating did not outweigh the time and effort required to automate. There were ten test runs for each app.

There was a timing issue when measuring with Firefox that appeared during test runs. The interface does not allow seeing exactly when ten seconds have passed, and this made it hard to get an accurate reading. Thus the measurements were between ten and thirteen seconds. In retrospect, one could have kept track of the time in a complementary way. The risks were mitigated by using averages and timing several runs.

3.4 Observation Study and Questionnaires

The observation study and questionnaires focused on answering RQ2 and RQ3. They were made with nine developers who had at least two years of experience and no experience in either framework. Each participant answered three questionnaires, one short about Three.js, one short about Babylon.js and one medium-length with general questions about themselves as well as which framework they would choose and why. Everything took part within an observation session which together with the other sessions made up the observation study.

When looking for insights on the usability aspect of software, a qualitative study with five users is enough. That will reveal about 75% of the usability problems with the design. To cover all, one must perform at least 15 user tests [21].

This thesis did not intend to find all problems but a big enough portion and where those problems were. The purpose was to see if there were any differences between the two frameworks. Because of this, the above approach was deemed suitable. The choice was made to gather quantifiable data from these sessions as well: task completion time and user-generated score-based answers. Both could then be used for further analysis.

Providing the participants with the links to entry points in official guides and documentation pages, the author then instructed these steps for each framework.

1. Download the framework via the Node package manager.
2. Get a basic object to render in a browser according to instructions provided by each framework.
The entire session script is available in Appendix B. The participants were instructed to think-aloud. After each framework implementation was finished or when an hour had passed, there was a pause, and the participants answered a questionnaire rating the ease of use and to what degree the framework’s capabilities met their requirements, see Appendix C. During the entire session, the author took notes, see Appendix D.

No audio or video recordings were made due to two reasons. First; observation notes, answers to three questionnaires and timed tasks seemed sufficient to provide enough data to answer the research questions. Second, the substantial amount of material this would have created with more than eighteen hours of recordings was too much to handle within the scope of this thesis, transcribing them would have taken a great deal of time. The benefits of being able to go back and hear or see exactly what happened did not outweigh the disadvantages.

However, after the first few sessions, it was apparent that the pace was slow enough that the notes could catch almost all the participants’ actions and what they said. The notes thus became as close to transcripts that are possible without doing an exact transcript from a recording.

After each session, the author scanned the notes for themes and then summarised thoughts on themes and overall impressions. When all the sessions were finished, the notes were translated from Swedish to English, spell-checked and some action descriptions were concentrated to one row instead of e.g. three rows for clarity. These notes can be viewed in full in Appendix D. The themes and summarised thoughts are the ones seen under chapter 5.2 Observation Study and Questionnaires.

It was not apparent beforehand how long the above steps of implementing a 3D model could take. The author performed the implementation and managed to with Babylon.js in 40 minutes and Three.js in first 8 minutes with a basic setup (only the Three.js code in a separate file per the instructions in the documentation) and then 2 hours and 40 minutes for a working setup with Node package manager involved. Based on this and the moderator’s mention of an hour [1], see chapter 1. Introduction, it was deemed feasible to set an hour.

Other factors to account for involved unexpected bugs and external factors interfering with the participants’ processes such as their hardware or other software. The frameworks are platform agnostic. So was the observation study as well to be as close to reality as possible where developers use different platforms. The participants used macOS, Microsoft Windows and different Linux distributions, see Appendix E. No notable interference occurred, except one Linux user (participant H) who had a computer crash but rebooted and returned two minutes later and had no further issues. The two minutes were included in the timed score because it was deemed accurate to what could happen in a developer’s life.
The process during the observation sessions could differ from how developers learn, being observed by someone could interfere. To think-aloud is perhaps not natural but it is still a well-researched area with overall benefits and combined with other techniques it is valuable [22].

To collect some factors that could have an impact on the result, the participants answered an extra post-session questionnaire. This questionnaire asked about their hardware, operating system, which techniques and languages they had used before, if they were familiar with 3D, how many years they had coded, what their education consisted of and their age. They were also asked to choose a framework and motivate why. See Appendix D for the questions and E for answers.

There was a risk that there could be substantial differences in the approach of each framework’s instruction which then could make it hard to compare. When deciding which links to provide to the user, this resolved on its own. The first link provided for each framework was the starting point in the documentation, named some way of “getting started”. The second link was to a page explaining how to use the framework in conjunction with Node package manager, see Appendix B for links.

The order of frameworks tested could influence one another. Changing the order randomly for participants could mitigate this. However, to try one 3D web framework presumably helps to learn another due to the testing effect. The testing effect describes how memory retention increases after repeated tests [23].

Why use Node package manager? Three.js documentation says “For most users, installing from npm is the best choice.” [24]. Babylon.js on the same note; “The NPM package manager is one of the best way to define and organize your project's dependencies.” [25]. They both offer instructions for using Content Delivery Networks, and Babylon.js has a playground. Participants were instructed to install Node.js beforehand if they did not have it installed.

UMUX-LITE was chosen due to its compact nature and correlation with other usability scales, see 4.5 Terminology on Ease of Use and Scores. To have a scale that included an evaluation of an entire system and at the same time consist of only two questions, of which one entirely focused on ease of use, seemed like the perfect match. Finally, questionnaire scales longer than the System Usability Scale were excluded from the literature study due to their perceived irrelevance for this thesis.
3.5 Finding Participants

Participants were searched for in the author’s extended network as well as posted in two Facebook groups for developers and on the Discord chat server for web programming students, teachers and alumni at Blekinge Institute of Technology. That meant a convenience selection, but with an eye on a good sample of different experiences and prerequisites. Should the participants be fully-fledged JavaScript developers? Several potential participants could not partake due to already working with or having tried one of the frameworks. The ongoing COVID-19 pandemic also further narrowed down possible participants due to an increased workload and less inclination to devote time to participating in a bachelor thesis study.

However, at the beginning of the documentation on the Babylon.js website, it says “Beginners to coding in JavaScript should be able to learn a lot from the examples, however we strongly suggest that you gain a working knowledge of JavaScript before starting to develop your own projects.” [26]. That would suggest one could be able to get it at least up and running if one has some knowledge of JavaScript, which eight out of nine developers in this study had.

The final nine participants were developers with different backgrounds, some working with JavaScript every day, some not. Their ages varied from 25 to 44, with both median and average of 35. When it came to their experience, it varied from two to more than 20 years, median eight and an average of 15 years. Some were self-taught to a higher degree. Some had an extensive programming-related university education.

3.6 Validity Threats

This section elaborates on important validity threats and discusses mitigation strategies.

Internal validity

The results from the experiments could be unreliable because it could require more than measuring ten times and use an average to get accurate results. The human factor when pressing start and stop on the measurements could also influence the result. It could also be that measuring frames per second and memory give a skewed image of performance. Perhaps including more test runs, automatic measurements and more factors in a larger study would give a fuller picture. Because this thesis could not be super extensive, these delimitations are to be kept in mind when reading.
A mitigation strategy for external factors when measuring performance was to use the same environment for testing, clearing the cache and not having other programs running. To come closer to the users’ environments one would have to use several computers, preferably different operating systems and browser. It could reveal other results to perform tests at a larger scale with many more tests during different times and with a variety of clients.

When comparing two framework implementations, one strategy is to ensure the apps are as similar as possible, however, this proved slightly hard. The time constraint made it difficult to come all the way. Even though standard materials, default cameras and so on were used, the cubes in the different apps move differently and the apps do not look the same. Still, the respective framework made the standard materials and decided to make those cameras default, so it makes sense to investigate and compare just that. However, the results could reflect these particular cameras and materials and there could be other results e.g. more similar cameras if there are such.

The Babylon.js implementation build size is larger than the Three.js, even with the use of ES modules. Efforts were made to mitigate the differences Babylon.js still comes out larger, as well as use more memory in runtime. It is unclear if those factors of build size and memory use correlate.

When it came to the observation study there were several mitigation strategies to avoid inaccurate results and inconsistencies. Using a script to make the sessions similar, switch the order of the frameworks to make sure they had the same preconditions the same number of times and using questionnaires to get quantifiable data were some. One aspect of the questionnaires was that the participants self-reported information about themselves in text answers. There were no rules on how to calculate years of experience for example. Although, the inputs where overseen and could be discussed with the author, and sometimes were, there was no formal assessment of their experience.

One threat that was hard to anticipate before the study was how UMUX-LITE would be received. Several participants were unsure if they could answer if the respective framework’s capabilities met their requirements and to what extent they were easy of use, the two questions which make up the UMUX-LITE scale. The author trying to mitigate this then stated during sessions that this is from what the participant know as of now and then they answered. Still, this could of course skew the results. A larger sample and an either modified questionnaire or a control group with another scale could have mitigated this further.

Think-aloud is not a natural situation, but it is one way to gain access to what people think. The participants were generally comfortable with doing it, but the thoughts could be filtered. To mitigate that, also writing down observations of what they did and assessing a clear goal with getting the 3D model to show in the browser helped in coming closer to it. The interpretation of the think-alouds could also be faulty in that there was only one human
interpreter who coded and extracted themes. There could be a bias there in some way of course. In hindsight, letting someone else look at the think-aloud transcript notes could have mitigated this.

Finally, participant D broke off the second Babylon.js test due to unforeseen circumstances. Because this participant had a relevant background, had performed the first test and answered all the questionnaires, the results were partly kept with the exemption of minutes to implementation and overall fail/success rate. This meant the minute score for Babylon.js is calculated on eight people and Three.js on nine people and could lead to a worsened comparison. However, since this participant had a relevant background, experience and views it was decided it was valuable to keep. Because despite having all prerequisites, one could fail. If that last fail hade been included it had increased the average implementation time for Babylon.js by 14 minutes. Without it, the average is close to Three.js with only one minute apart and this is something to bear in mind.

**External validity**

How applicable are the results? First, the experiment says nothing about performance in larger implementations and it could be argued that those are very relevant to developers. However, for someone who wants to know what a small implementation performs they might be aided to choose one before the other. This in turn could lend a hand in what to look out for and optimise for if they want to create a larger application, e.g. look out for that Babylon.js does not increase memory use exponentially.

On thing valuable to know is how an app with these frameworks perform in a “normal” environment with interfering programs, plugins and a strained cache in a browser. However, that is not possible to know at this time. The same goes for “normal” application usages, a developer might not use the standard shaders or those particular cameras which could render this study ineffective.

Moving on to the observation study and questionnaires. That part cannot be applied to larger samples of participants, with say more than 15 or more than 100. A larger diversity of participants in terms of age and experience could of course be welcome as well.

The chosen path of entering learnability through the concept of ease of use could affect the application of the results. One would have to know more and compare more on how others have done to be able to see exactly how scientifically applicable this is. There is also no golden rule to say that implementing a 3D model in a framework is the same as initial learnability. Overall, more research is needed for context. The lack of conforming the results from UMUX-LITE with a usability scale comparison formula also makes it less applicable.
The experience of being observed could not be mitigated and it could be a validity threat. That matters as to how applicable the results are, one cannot know for sure if developers really could or could not perform this task in another way without being observed.
4. Literature Study

4.1 Performance in 3D Web Apps

For this study, the focus is the performance experience for the end-user consuming a web app with 3D in it. That is what for example, Babylon.js mean when they talk about improving their performance [27]. Also, that is what is investigated in a research project on 2D graphics framework performance in this area [28].

A user can access a 3D web application with a web browser. The environment for each end-user can vary significantly with regards to hardware and software. To find an exact estimation to account for everyone is therefore hard. When investigating different frameworks’ performance, one can extract reliable data by making sure the environment is identical, and the web apps tested as well [28].

Other than one, there were no studies found which compared 3D JavaScript framework performance. The one study is from 2016 and focused on comparing a commercial proprietary interface system component, GraphWorX64, used in factories and building automation systems with an interface built by researchers with Three.js. Muennoi and Hormdee found that the Three.js version could maintain a frame rate of 60 frames per second. There was also an indication of more consistency with the time per frame than the commercial one [29].

4.2 Important Metrics in 3D Performance

When simulating the three-dimensional space in web applications, it is useful to explore different angles, either move around the viewer or user’s perspective within the scene or move around an object or a scene in front of the user. If not, one could just use 2D representation. Thus, 3D graphics connect closely to animation.

When the human eye sees pictures in a high enough paced concession, it appears animated or alive [30]. Early movies and onwards have used this to their advantage [30]. Today, in computer graphics, 3D animation is still rendered through frames in the browser. Frame rate is the rate with which the frames can resolve in the user’s browser is of importance [31]. A frame rate of 60 Frames per Second (FPS) is considered vital [31]. Frame time or the time each frame has to resolve should be 1000ms/60 ~ 16.7ms [31]. For a smooth experience, the time per frame should be consistent [31][32].
The computation of the simulation of 3D requires a level of complexity. Thus 3D performance is concerned with memory. On a basic level, memory in computing means how much space the code reserves to store information to be able to execute. Web application sizes are of concern to the web community at large. That is due to site speed being a factor to retain users [33] and the increasing internet growth’s impact on sustainability [34].

Hardware-wise, both the central processing unit (CPU) and the graphics processing unit (GPU) are of interest in 3D performance. WebGPU is a work in progress API set to replace WebGL, the underlying API that Babylon.js and Three.js use. The prospect of WebGPU is to move computation from CPU to the GPU and use caching more efficiently [35][15].

4.3 How To Measure Performance

Safari, Edge, Chrome and Firefox all have developer tools which can be accessed within the browser [36]. Here one can extract data on and analyse performance manually such as visually identify gaps in frame rate. They are not standardised and in various states of evolution.

The first developer tools attached to a browser publicly released was Firebug in 2006 with Firefox [37]. Mozilla Firefox’s documentation on the memory profile section has an instruction video at the top which dates back to 2015 when it was first released [38]. There is a frame timing API, but the specification lists it as experimental/not implemented [39]. Chrome announced this year (2020) they are working on a new API to address memory more accurately [40].

Another way to measure performance is to automate measurements using drivers provided by browser vendors. One of the most popular tools for browser automation is Selenium [41]. It works by way of downloading a compatible web driver for the browser, importing this into code and letting the code access a URL and then perform actions like extracting data. However, Selenium lists performance testing with Selenium as a worst practice because of the interference the web driver implementation can cause, browser startup speed and more [42].

Still, Selenium is used for performance testing. Sitespeed.io is a performance tool suite built on top of Selenium, created by Peter Hedenskog, who works on the Wikimedia Foundation performance team [43], [44]. Sitespeed.io does not have easy access to memory evaluation or frames per second [45].

Another application that uses browser drivers is Puppeteer, a Node.js library maintained by the Chrome DevTools team. Puppeteer does have a section on performance testing [46].
Finally, Selenium lists the application called JMeter as suitable for performance testing [41]. JMeter is not a browser and works at the protocol level [47].

In summary, there seems to be no agreed-upon and easily accessible way to run performance tests automatically. To choose one way would mean to lose benefits, and each way requires a tailored approach specific to that situation. For this thesis, the concentrated approach of measuring frames per second and memory via web browser tools is the most beneficial due to time constraints, with the added disadvantages outlined under 3.6 Validity Threats.

### 4.4 Science on Performance and Usability

Software performance and usability are traditionally placed in two different fields, Software Engineering and Human-Computer Interaction. The techniques, methods and scores have originated in different communities and different trains of thoughts. It is possible to understand Human-Computer Interaction through three paradigms [48]. The concept originates from Kuhn’s theory on scientific revolution/evolution in waves [49] and subsequently adapted to the Human-Computer Interaction field to three paradigms by Harrison et al. [50].

The first paradigm centers around solving problems in the interaction between humans and machines. The second paradigm revolves around the interaction as communication and from here comes many scores that try to help us generalise and optimise the communication between humans and computers. The third paradigm takes a broader view: what goes around the system is more interesting than what happens in the interface. A score can only give us so much, and we have to have a phenomenological matrix with several methods and a variety of approaches. Each paradigm lets us focus on different but significant questions [50].

### 4.5 Terminology on Ease of Use and Scores

Ease of use as a concept can have several interpretations. In [51, p. 603] ease of use "represents how comfortable users feel in using the system.", it is here a part in determining the software quality and derived from an international standard published by the International Organization for Standardization.

System Usability Scale or SUS is a ten-question scale created by John Brooke in 1984 and then made freely available. The initial purpose was to have a “quick and dirty” scale to measure the perceived usability. Since then, it has become the most widely used scale. If a participant cannot answer a question, they should mark the middle one in the scale [52].
Another notable scale is the Usability Metric for User Experience created by Kraig Finstad in 2010. UMUX is a concentrated four-question scale that tries to capture the subjective assessment of the perceived usability of an application [53]:

1. [This system’s] capabilities meet my requirements.
2. Using [this system] is a frustrating experience.
3. [This system] is easy to use.
4. I have to spend too much time correcting things with [this system].

Finstad aspired to create a questionnaire that could be used in diagnosing problems with software to make it easier to put efforts into the right aspect of it, something that SUS was inferior at. Therefore the questionnaire is designed to align with the ISO 9241-11’s definition of usability where effectiveness, efficiency, and satisfaction is in focus, fields that have organised methods to work with. Aside from that, UMUX also meant to replace SUS when one needs a more compact scale because of the overall questionnaire length. For example, when measuring other aspects than usability, such as support experience [53].

UMUX-LITE appeared in 2013 and consisted only of the positive notes in the UMUX scale [54]:

1. [This system’s] capabilities meet my requirements.
3. [This system] is easy to use.

There is a tradition to include both positive and negatively worded questions in usability questionnaires. Sauro and Lewis conclude in one study [55] that the advantages of positive questions outweigh the advantages of having both positive and negative ones. To have both positive and negative are supposed to minimise acquiescence and extreme response bias. Though, only positive questions lead to less miscoding by researchers and fewer mistakes from participants as in accidentally agreeing with negative ones.

All of the above questionnaires are answered after using a system and evaluate a system as a whole. They all contain ease of use-specific questions. There are scales only limited to ease of use, one-question post-task scores such as Single Ease Question, SEQ and Subjective Mental Effort Questionnaire, SMEQ [56]. This scale SMEQ, in particular, focus on how difficult a task was, perceived difficulty. Two studies found that task time and perceived difficulty correlate [57]. There is also the ease of use of the scale for the researcher to think of, using a SMEQ score requires vertical scale [57].

One of the major criticisms against usability as a useful scientific construct points out that researchers in Human-Computer Interaction have not dealt with the word in itself to any greater extent despite its general popularity. This criticism says that if one does not have a
stable term, but this umbrella word, then one does not have a way to create empirical
evidence, thus no science [58].

Subsequent reactive research focuses on defending the word and moving forward. One article
focuses on acceptance and chooses to develop the Technology of Acceptance Model (TAM)
and work with Perceived Usefulness and Perceived Ease-of-Use. They compare different
usability scores such as SUS, UMUX and mTAM (modified TAM). They find a correlation
between different scores and UMUX [54]. That is, the scales seem to measure the same
things which would then suggest there is a stable concept behind. What one names this
concept is still debatable.

One interesting note is the integral part of ease of use within usability, especially with a scale
such as UMUX-LITE that boils down to two questions. Here it makes ease of use not only a
part of usability but almost equal to the word, ease of use means usability that is.

4.6 Learnability and Getting Started

While usability is under discussion, there is at least a somewhat consensus that learnability is
a part of usability. The definition of learnability is complex. A meta-study on 88 articles
found several variations: easy to use, easy to learn, first-time performance and more.
Grossman et al. then created a taxonomy over learnability listing two main categories [59, p.
651]:

1. Initial Learnability: Initial performance with the system.
2. Extended Learnability: Change in performance over time.

Note that they mean task performance, not software performance. The first category informed
the author’s decision to talk about initial learnability in the thesis. The previously mentioned
definition from [18] in 1.2 Focus Area of how easy it is for users when they first encounter an
application and want to perform basic tasks were found to be similar to that definition, but
also more specific and aided in defining the basic task of implementing a 3D model that
became part of the observation study.

One of the earliest groundlaying works is a Nielsen article from 1994, a meta-study
comparing subjective preference to objective performance (as in how fast the users could
perform tasks) [60]. Subjective preference was measured with different scores and found a
common denominator; the higher the scored number, the more positive attitude to the system
in question.

However, there are instances where users prefer a system in which they perform worse than
in another comparable one [60]. Wetzlinger et al. conducted a study where tablets were found
preferable to laptops, despite users completing tasks more efficiently in the latter. They concluded there are more factors involved, such as the attractiveness of a tablet and hedonic qualities such as stimulation and identity [56].

Chistyakov et al. mean the scores that take learnability into account only record the participant’s subjective experience, claiming the need for an objective measure. They proceed to do so and provide a method calculating the number of clicks and task completion time. They also suggest this method can be used to supplement other methods but also be used on its own in some instances [61].

4.7 Influx of New Software and Techniques

No browser supports WebGPU by default yet, but some can be enabled to do so in experimental mode [62]. The gains of WebGPU are a new way of utilising the Graphics Processing Unit more effectively, and move computation from the CPU to the GPU, what is mentioned as increased overall performance [15].

For WebGPU to be fully implemented, several criteria need to be fulfilled. The W3C task group must finish the API, and the API has to be stable, browser vendors must ship their browsers with support for this and frameworks such as Babylon.js and Three.js have to build support for this. Babylon.js has support for some WebGPU functionality at the time of writing [63].

The overall quest for improved performance is a story that runs aligned to technological evolution at large. Here some drivers are the need to reduce costs, earn more money and the increasing capacity of hardware components (e.g. Moore’s law and the doubling of transistors on a microchip) [64]. The also increasing energy consumption [34] is made actual by the need to preserve the planet against environmental changes. The physical limit to Moore’s law makes it all the more critical to improve hardware use [64], [65]. Thus, it is possible to say that WebGPU’s intent of using hardware more effectively could not come at a better time.

4.8 Research Comparing Ease of Use or Learnability in 3D

There were no studies found that compared the ease of use or learnability in 3D frameworks. One study mentioned previously where UMUX was used had the objective to compare scales but by extension touched on related issues to this study. It was divided in three surveys where two user groups answered questionnaires on common tools. The first survey was on Powerpoint, the second on Gmail and the third on IBM notes. Each participant in the part-survey answered three questionnaires on the tool, the questionnaires were SUS, UMUX
and mTam and the order was randomised. However, the paper on the study does not list the answers, only the subsequent correlation results between the scales [54].

The tablet and laptop comparison study by Wetzlinger et al. investigated ease of use, efficiency and more [56]. The respondents had to perform basic tasks on the devices such as create a calendar entry. After each task they answered the Single Ease Question: Overall, how difficult or easy did you find this task? The scale went from 1 (very easy) to 7 (very difficult). As a post-system use questionnaire, the researchers used the System Usability Scale. The SUS went from 1 (strongly disagree) to 5 (strongly agree). The overall user group consisted of eight people where half started using the laptop and the other half started with the tablet. The researchers also collected task time.

The results were listed in mean values and the task of creating a calendar entry was 2.00 for the laptop and 1.63 for the tablet, write an email 1.13 for laptop and 3.25 for tablet, browse and fill out forms 1.25 for laptop and 1.63 for tablet. Finally, the task to find and gather information resulted in both 1.25 for both devices. Within SUS there is the question of “I thought the product was easy to use.”. The mean values for answers to this question was 4.00 for the laptop and 4.125 for the tablet which are both high on the scale meaning they are considered quite easy to use. The aforementioned difference between a post-task question versus a post-system use questionnaire is visible here. The SUS scale results differ less than the post-task questions. Overall the researchers conclude there is a difference between subjective ratings and measured characteristics such as time, they also say the sample size of eight participants is unreliable [56].
5. Results

5.1 Experiment: Performance

Two web applications were made with a minimal implementation of the respective framework [20]. They were deployed on Netlify and looked like this see Fig 1, both with a moving cube.

Fig 1. Screenshots of 3D apps, left Babylon.js, right Three.js.

Links to web apps:
https://babylon-light.netlify.app/
https://three-light.netlify.app/

The test runs were made manually in Firefox Browser Developer Edition 84.0b7. There were ten test runs for each app; below are the averages for each app. The recorded files are available in a public repo [20], and the script for the metric recordings can be found in appendix A.

Table 2. Results from performance experiment.

<table>
<thead>
<tr>
<th>Performance</th>
<th>Three.js</th>
<th>Babylon.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average memory in MB</td>
<td>4.74</td>
<td>6.94</td>
</tr>
<tr>
<td>Average min frames per second</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Average max frames per second</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>Average frames per second</td>
<td>57</td>
<td>58</td>
</tr>
</tbody>
</table>
The measurements in Table 2 show that the apps consume different amounts of memory when accessing them with a browser. The difference is 2,2 MB. Babylon.js consumes 46% more memory than Three.js. When it comes to the minimum frame per second, they came up the same on average. That is, what was the average of how few frames they could produce. That means that sometimes they did not produce 60 frames per second. Most of the time though they were able to uphold a maximum of 60 frames per second. The slight difference of average FPS can be accounted to rounding numbers.
5.2 Observation Study and Questionnaires

Nine observation sessions resulted in data based on answers from three questionnaires and notes taken during observations. The usability evaluation is based on the UMUX-LITE scale. It is a Likert type scale which goes from one (1) to seven (7). Where one signifies “strongly disagree” and seven is “strongly agree”. See Appendix C for a visual representation of the questionnaires.

Table 3. Results from the observation study and questionnaires.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D*</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minutes to implement Three.js</td>
<td>FAIL</td>
<td>FAIL</td>
<td>16</td>
<td>FAIL</td>
<td>33</td>
<td>18</td>
<td>FAIL</td>
<td>4</td>
<td>31</td>
</tr>
<tr>
<td>Minutes to implement Babylon.js</td>
<td>33</td>
<td>FAIL</td>
<td>FAIL</td>
<td>N/A</td>
<td>36</td>
<td>16</td>
<td>FAIL</td>
<td>38</td>
<td>50</td>
</tr>
</tbody>
</table>

### Prerequisites

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Years of coding experience</td>
<td>17</td>
<td>2,5</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>20+</td>
<td>8</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>Age</td>
<td>38</td>
<td>30</td>
<td>34</td>
<td>35</td>
<td>32</td>
<td>44</td>
<td>41</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>Familiarity with JavaScript</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Order of implementation</td>
<td>b, t</td>
<td>t, b</td>
<td>b, t</td>
<td>t, b</td>
<td>b, t</td>
<td>t, b</td>
<td>b, t</td>
<td>b, t</td>
<td></td>
</tr>
</tbody>
</table>

### Usability evaluation

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“Three.js’ capabilities meets my requirements.”</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>“Three.js is easy to use.”</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>“Babylon.js’ capabilities meets my requirements.”</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>“Babylon.js is easy to use.”</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Which framework would I choose?</td>
<td>b</td>
<td>b</td>
<td>t</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>t</td>
<td>t</td>
<td>b</td>
</tr>
</tbody>
</table>

* Participant D had a baby who woke up during the second test and the participant decided to give up, which meant the Babylon.js part ended prematurely after 19 minutes. The participant still answered the questionnaires before leaving and so the author decided to include that as part of the data but rule out the minutes to implementation and calculate the overall scores on that factor on eight people instead of nine.

Now follows the author’s observations and reflections compiled from the notes taken during
the participants’ think-alouds in the observation sessions, see Appendix E for transcripts. Below is referencing Table 3.

Minutes to implement 3D model

Five out of eight or 63 % of the participants managed to implement a 3D model in Babylon.js and five out of nine or 56 % in Three.js. Four participants managed to implement within both and two just one of the frameworks. An average implementation time for Three.js based on above is 20,4 minutes. An average for Babylon.js was 21,6 minutes.

Prerequisites

Who managed? Participants A, C, E, F, H and I succeeded in implementing one or both frameworks. Zooming in on the data in appendix E, we find that participants A, C, E, H and I have worked with TypeScript, Webpack or 3D in some way. Participants B, D and G were not a typical JavaScript programmer. However, it is not unrealistic to think they would one day need or want to work with 3D, and then get discouraged by not managing to implement the most basic task in these frameworks. Participant F falls into a separate category with implementation times under 20 minutes for both frameworks without extensive JavaScript knowledge, but with a long overall experience.

Age did not seem to factor in success rate within this sample given that the youngest person (25), as well as the oldest (44), managed to implement a basic model within both frameworks. 67 % of the participants were between 30 and 40 years old and there it varied. The number of years of coding experience seemed to influence slightly though, however, it was not clear because some who had five years of experienced fared well and more experienced, e.g. 17 years, did not. It is important to note that everyone had previous familiarity with JavaScript except participant B. No one was a hobbyist; they were all either professionals or studying to become professionals.

The order of implementation did not seem to influence the outcomes, except in some cases it provided context when a participant reused code or a concept from an earlier implementation. Participant A started with Babylon.js, managed to implement and then did not manage to implement Three.js. Participant H managed to implement Three.js fast and then Babylon.js took significantly more time. In other cases, the second successful framework implementation took a few minutes shorter.

Usability evaluation

A summary of the ease of use can be viewed under 5.3 Overview Performance and Ease of Use. All but one participant were reluctant in grading their experience because they did not feel qualified to answer after trying something for a short period. The scale ran from one to seven. Several participants resorted to answering a four as a kind of neither/nor answer in the middle. The author emphasised they could answer the questions from what they knew right
now. Three people accepted this; the rest did not but answered nevertheless. Some changed their view after completing the second test when, as they said, it was easier to have something to compare.

Notes on instructions
Several participants did not follow both links provided for each framework. Some used the first, and some used the other. There was no apparent pattern here. Not even if the author stressed there are two links did this change. However, there is some evidence that they were aware of two links, one participant referred to the unfollowed link when thinking aloud when they googled and clicked through to find the material “Oh, now I am at where your other link was”. Another participant was sure they had used both when asked afterwards.

Notes on struggles
The participants struggled with a couple of principal areas. One area was paths and where to place files in connection to one another in order to get the software to work. Another was an abundance of text information, where to find what they needed and make sense of what they experienced as a large mass. Additionally, and frequently mentioned was how and if their knowledge was sufficient to implement the frameworks.

Notes on documentation
Generally, they were reluctant to read the documentation, especially piecing together examples from different pages. Almost all participants expressed opinions of wanting to know if it works before learning. They all copied and pasted code, and when they could, they bypassed small code snippets in favour of large masses that looked complete. One participant exclaimed “Frankenstein programming at its best!” when copying and mixing code from both frameworks, from their previous unrelated project and the provided documentation from both frameworks. Some participants quickly went to npmjs.org to find instructions rather than babylonjs.com. Three participants found a guided learning section [66] on Babylon.js in another part of the documentation, which helped them get it to work.

Notes on choice
The study and questionnaire design did not leave room for indecision, so all participants made a choice, although some reluctantly. The participants also answered in writing as to why. Notable reasons for choosing were: better documentation (Three.js and Babylon.js), a nicer looking example model (Babylon.js), that it was easier to use (Three.js), quicker feedback (Babylon.js), heard more about it (Three.js) and better impression of capabilities (Babylon.js). See Appendix E for their answers in full. Notable on the esthetics was also that several participants during the sessions mentioned Three.js documentation as more pleasing to the eye in terms of colour and spacing.
5.3 Overview Performance and Ease of Use

Memory usage differs by 46%; the frame rate does not differ. As seen in Table 4, the choice is not consistent with ease of use or implementation time in minutes. The first question in UMUX-LITE about the framework’s capabilities seemed to confuse the participants more than the second about the framework’s ease of use. Therefore the ease of use is also displayed separately for comparison with the full score. No extra calculation has been made to make these results compare with other usability scales.

Table 4. Overview results from performance and ease of use.

<table>
<thead>
<tr>
<th>Performance</th>
<th>Three.js</th>
<th>Babylon.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average memory in MB</td>
<td>4,74</td>
<td>6,94</td>
</tr>
<tr>
<td>Average frames per second</td>
<td>58</td>
<td>57</td>
</tr>
</tbody>
</table>

Ease of use

<table>
<thead>
<tr>
<th></th>
<th>Three.js</th>
<th>Babylon.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average score easy to use</td>
<td>4,2</td>
<td>3,6</td>
</tr>
<tr>
<td>Average UMUX-LITE</td>
<td>4,3</td>
<td>3,9</td>
</tr>
<tr>
<td>Average implementation time minutes</td>
<td>20,4</td>
<td>21,6</td>
</tr>
<tr>
<td>How many would choose</td>
<td>33 %</td>
<td>67 %</td>
</tr>
<tr>
<td>Implementation success rate</td>
<td>56 %</td>
<td>63 %</td>
</tr>
</tbody>
</table>
6. Analysis and Discussion

6.1 Performance in Babylon.js and Three.js

The results suggest there is no measured difference in performance when it comes to maintaining a frame rate for these frameworks with a minimalistic one cube animation-implementation. However, there are differences in memory use. Babylon.js consumes 46% more memory than Three.js. To know for sure if this difference can be attributed to more code or inefficient use of code in the frameworks, one would have to a) compare larger implementations, b) look into functions and execution of code and c) streamline the apps further, especially when it comes to how much each module uses and pick even more similar ones like cameras. It would also be possible to investigate whether the above could mean that Babylon.js perform better in more complex projects. Either way, 46% is a substantial difference in memory consumption at this level of app complexity.

While using a browser’s built-in developer tools for measurements is convenient, it could also be inaccurate. The plans for new, more accurate tools and the time frame from when they first appeared five years ago suggest there is room for improvement. A browser’s developer tools could also influence the measurements. So could automating tests if run with, e.g. a browser automation suite such as Selenium does. However, the difference between the same measurements taken at intervals could point to changes in performance over time. A study with, e.g. one hundred automated tests could give a more accurate result if the result focuses on spotting differences and if there is an agreed-upon base level.

One possibility is to use a tool like JMeter [47] to measure. Further research is needed to explore the benefits. However, to use a non-web browser to measure the experience of using a browser brings up philosophical questions. Additionally, the perspective of bridging Software Engineering and Human-Computer Interaction fields further lends the thought of using humans to experience the performance as well, subjectively. Why not let users say how the apps perform? In the end, human users will experience the optimised 3D web applications, not browsers.

To also measure CPU and GPU metrics as well as time per frame could provide a more comprehensive picture, especially in light of new technology such as WebGPU. All in all, though, frames per second and memory usage do give us a general picture of the performance, which is no different when it comes to frame rate but some difference when it comes to memory use. Three.js thus outperforms Babylon.js within the bounds of this thesis.
6.2 Ease of use in Babylon.js and Three.js

Three.js is slightly easier to use than Babylon.js according to scores compiled from the participants’ answers. If the numbers had been rounded to integers, they would score the same. However, a four (4) on the UMUX-LITE scale ends up in the middle of Strongly disagree and Strongly agree. The frameworks' ease of use is not alarming, but not good either. To be able to say that one framework is much easier to use than the other in terms of initial learnability, one would have to see a difference. Preferably, one of the numbers should be a six (6) or a seven (7) to say that a framework is easy to use. Numbers aside, almost all participants had a frustrating experience.

The issue of paths, pre-existing knowledge and JavaScript as a whole is also notable. To explore this further, one could devise an instruction page for each framework. This instruction page could list all the steps needed to download and implement a 3D example model, show an overview of how a basic setup is connected with lines drawn to show how the files are connected. It could also explain in short and link to more information about Webpack and as well as ES module functionality. It would have all commands needed to get started, including how to initiate an npm project and a complete working example with a full index.html, main.js with code for a basic object, and an image of what the implementation is supposed to look like when one is successful, plus package.json and a server setup. All of this would be listed first and then link to a step by step breakdown if someone gets into trouble. If participants score the frameworks easier to use after trying these instructions, one could assume it is possible to influence the initial learnability.

6.3 Learnability and Choice of Framework

As evident, not all participants manage to implement a 3D model in one or either framework. That could be a result of the observation study setup. One could argue how natural it is for a developer to have another person watch them implement a framework with a time limit of one hour. While it is hard to do an observation without observing, at least performing tasks under a time constraint should be familiar with developers. Nevertheless, it is possible to implement a basic 3D model in Three.js in four minutes (participant H) and Babylon.js in sixteen minutes (participant F).

Looking at these results where only 63 % managed to implement Babylon.js and 56 % in Three.js, get the most basic model and setup up and running that is, one could wonder how many users overall these frameworks lose during initial learnability. It is one thing to come to a framework with an exact idea of what to do, another just to check it out and see how capable it is. Both scenarios require a quick feedback to let the developer over the first hurdle. The need for wanting to know if it works before investing more is quite clear from
these observations. Again, it lends thoughts to the unrecorded number of people who have tried and given up.

Furthermore, neither the task completion time nor the choice ratio correlates with perceived ease of use. These results suggest there are other factors than ease of use that come into play. The participants themselves cite reasons such as social influence and extensibility, which would be in line with [19]. One factor not mentioned in [19] but mentioned by one participant is esthetics. The actual test model to work with in Babylon.js has a shader and a specific purple background. The model in Three.js is a bright green box on a black background.

Overall learnability can be viewed through the lens of different paradigms. Task completion time lends itself more readily to the first or second paradigm, generalising learnability could help to improve it. Using a scale for the experience is somewhere in between. One particular perspective that the 3rd paradigm could give us is on the reluctance to extensively read the manual before knowing if something works [50]. This is how people do, so how can they be supported in their learning process? No matter the scores and task times, many of the observed developers now do not have a pleasant attitude towards Babylon.js and Three.js. They did not have a joyful and smooth initial experience with either framework.

6.4 The Technology Context and the Developers

Technology does not exist in a vacuum. There is a constant interplay between many factors, as seen in the literature study. Neither the developer nor technology are clearly defined entities with essential properties. However, there are some factors to consider, such as what the developer already knows. Previous knowledge in bundling tools, a good grasp on JavaScript and an understanding of paths could help to learn a 3D framework such as Three.js or Babylon.js.

One idea for the inclined is to try both frameworks, but perhaps not expect great initial learnability from either one separate. Trying both frameworks could increase chances and give context. Comparing frameworks is common, but the competition sometimes clouds how it often works, one uses bits and pieces from different software and puzzles them together. Does it have to be a competition?

From a business perspective, it could be good to budget for training. If an organisation is branching out to use 3D, especially with a workforce who has different experiences, it could benefit to set aside time and money to get on the same page when it comes to e.g. ES modules. Because 3D can be memory consuming as well, it would do well to have a strategy for this and investigate advantages and disadvantages. What does the overall stack look like and how this particular organisation can economise when adding 3D to the mix could be two questions to address.
Finally, improved performance and the introduction of WebGPU could sound like Babylon.js is ahead of the game in the competitive sense. However, both frameworks are open source. What is not to say Babylon.js just writes many implementations and then Three.js copies and adjusts for themselves? For the developer who uses the framework, it might not mean much more than a check if the end-user has a WebGPU compatible browser. Either way, it will probably mean more code, thus bigger files and perhaps more memory use.

6.5 The Research Context

The results from this study are hard to set into context with other research mainly since there has not been a great deal of specific research on either performance with the particular indicators memory and FPS or ease of use on 3D web frameworks. The maintained frame rate in both the Babylon.js and Three.js apps is comparable to the Muennoi and Hormdee study even though they study another type of application with an interface aimed at a factory setting [29]. Memory was not part of that or any similar study found and therefore cannot be compared.

What is possible to say on ease of use is that this thesis’ findings are somewhat similar to Wetzlinger et al. study where the mean value of a post-system question received similar results with both a tablet and a laptop, in the sense that the two frameworks here also scored similarly with a post-system questionnaire. Although, it is hard to compare the 5 points SUS used there with the 7 points UMUX-LITE without using a conversion formula. However, they also found a discrepancy between how long participants took to finish a task compared to how they perceive how easy it was [56]. Furthermore, to compare these results to only two studies leaves more to be desired.

How to measure ease of use is also relevant. The use of the UMUX-LITE scale as a way to evaluate an entire system’s initial learnability was not excellent in this case. A post-task questionnaire with a question such as “To what extent do you agree with… This task was easy to complete?” could have steered around the developers’ unease and provided more focus on the task at hand. However, there could be a factor to the developer being a specific group of professionals who want to know more before they provide advice or opinion on a tool. In [54] they used the same scale for professional users of Gmail and Powerpoint, and they did not conclude anything similar. One factor could be that they had more time and used the tools more. Nevertheless, if all scales correlate to some extent, a post-task one-question scale should provide the same results. But, as seen in [56] they did not, which makes the research somewhat inconclusive. Still, it is important to say that [56] had eight participants and [54] was a study with hundreds of participants.
7. Conclusions and Future Work

7.1 Conclusions

This thesis has looked at performance and ease of use in the web 3D frameworks Babylon.js and Three.js. It has explored how they compare when it comes to performance as well as ease of use. In addition, it has provided insights that could be useful for a developer when choosing a 3D framework.

In answer to how the frameworks compare when it comes to performance, there are no differences between Babylon.js and Three.js when it comes to maintaining the frame rate for a minimalistic one cube animation implementation. Babylon.js uses 2.2 MB, 46% more memory than Three.js, which concludes that Three.js performs better in that regard.

With regards to how the frameworks compare surrounding ease of use, Three.js is slightly easier to use when looking at task completion time and subjective measures of ease of use measured with UMUX-LITE in the initial learnability phase. However, neither framework has overall good ease of use in terms of initial learnability for observed developers.

This thesis also asked about useful insights on performance and ease of use for a developer who faces a 3D framework choice. The observed developers prefer Babylon.js over Three.js to a greater extent, valuing perceived better documentation, a more pleasing looking 3D model and quicker feedback. Ease of use as a factor is part of choosing a 3D framework, but not the only factor. The choice does not correlate with task completion time and perceived ease of use as measured with UMUX-LITE.

Furthermore, when it comes to useful insights, developers who want to increase their odds of implementation success could look into reading up on Node package manager, paths, build tools such as Webpack and on the JavaScript module system, ES modules. They could also search elsewhere than the documentation for readymade working boilerplates which contain basic up to date examples. From a business perspective, an organisation could strategise around memory use and budget for training in previously mentioned areas.

Finally, the most substantial contribution this thesis offers is within initial learnability and ease of use. Documentation and setup procedures matter significantly when developers use a framework for the first time. That finding from this thesis should be valuable for framework developers such as the creators of Babylon.js, Three.js and other frameworks. Initial learnability could be a gatekeeper. If users do not get past the first hurdle fast with quick feedback that the framework works in the most basic sense, a conceivable risk is they could give up and never use it.
7.2 Future Work

Future work could include devising an instruction and use the results from this study as a benchmark to see if there could be a way to increase initial learnability. Another field to look into could be to evaluate performance on complex 3D implementations. A third could be a comprehensive study on scientifically measuring performance for a practical purpose.
References


[58] N. Tractinsky, "The Usability Construct: A Dead End?" *Human-Computer Interaction*, vol. 33, (2), pp. 131-177, 2017;2018;.


Appendix A: Script for Performance Data Collection

1. Restart computer
2. Use Firefox Browser Developer 84.0b7 (64-bits)
3. Make sure all other programs are close (except basic text editor)
4. Clear the cache in the browser at about:preferences#privacy
5. Open an incognito window
6. Access one of the app urls
7. Open Firefox devtools (cmd/ctrl + shift + i)
8. Go to memory tab
9. Press record a snapshot
10. Go to performance tab
11. Press record and finish after 10 seconds
12. Go back to memory tab, copy the overall MB usage to your text editor
13. Save the profile
14. Go to the performance tab, copy the min, max and avg frames per second to your text editor
15. Save the profile
16. Now redo for next text run (skip step one)

Web app locations
https://babylon-light.netlify.app/
https://three-light.netlify.app/

The computer used for the experiment
MacBook Air (13-inch, Early 2015)
macOS Catalina 10.15.2

Processor 1.6 GHz Dual-Core Intel Core i5
Memory 8 GB 1600 MHz DDR3
Graphics Intel HD Graphics 6000 1536 MB
Appendix B: Script for Observation Sessions

1. Hi, how are you? Small talk.
2. I’ll follow a script now and explain a little bit more and how this is going to be
3. I’m studying at Blekinge Institute of Technology
4. This thesis is about performance compared to usability and especially how developers choose and learn new tools
5. So I am comparing how big, heavy, fast these frameworks versus how easy they are to use. But the performance part I’ll investigate separately, not now in this session
6. This will take a maximum of two hours
7. I will not record this
8. You will share your screen with me
9. We will begin by installing the framework [insert framework]
10. You will get links to the instructions
11. When you are done or when an hour has passed you will get to answer a form rating your experience
12. Done means when you have a local installation on your computer and can see a standard model that the framework provides, it can vary like a box or sphere
13. You have to install it with NPM, Node package manager
14. The activity is to make the framework run locally
15. The way we will know is when you can see their model in your browser, locally
16. Then we will do the same thing with the other framework
17. When you do thinks I would like you to think out loud
18. That is: verbalise your actions, intentions and thoughts
19. Are you familiar with this?
20. I have an example here  
   https://s3.amazonaws.com/media.nngroup.com/media/editor/2014/08/18/thinking-aloud-demo.mp4
21. Now you try it (if applicable)
22. Focus on completing the task as you would normally, you can use different ways to solve problems and resource that you would normally use, but use the documentation first-hand
23. I will take some notes
24. This exercise is not about evaluating your skills but focuses on how good the documentation and the software is
25. After both frameworks you will get a form with questions on languages and technologies you have used before etc
26. This is to see if there are factors that can influence the result
27. Do you have any questions?

The links provided to the documentation

Link 1: https://doc.babylonjs.com/start
Link 2: https://doc.babylonjs.com/divingDeeper/developWithBjs/npmSupport
Appendix C: Questionnaire Ease of Use

Which is the framework? *
- Babylon.js
- Three.js

Who are you? Enter your designated letter. *
Ditt svar

[Framework's] capabilities meet my requirements. *

1 2 3 4 5 6 7
Strongly disagree ○ ○ ○ ○ ○ ○ ○ Strongly agree

[Framework] is easy to use. *

1 2 3 4 5 6 7
Strongly disagree ○ ○ ○ ○ ○ ○ ○ Strongly agree
Appendix D: Questionnaire About Participants

What hardware and OS did you use for this session? Primary device where you installed and displayed 3D models. *
Ditt svar

Which techniques and languages have you used before? Please state if you are familiar with 3D in some way. *
Ditt svar

How many years have you coded? *
Ditt svar

What does your education consist of? *
Ditt svar

How old are you? *
Ditt svar

If you were to choose a framework to continue with which would you choose? *
- Three.js
- Babylon.js

Why would you choose this? *
Ditt svar

Your participation is voluntary and anonymous. All responses and data emanating from your participation will be analysed and used for this bachelor thesis, separately and compiled together as a group. *
- I certify that I have read and understood the conditions for my participation in this survey


Appendix E: Notes from Observation Sessions

Participant A

Hardware + OS: MacBook Pro MacOS 11.0.1
Languages + techniques: Not used 3D libraries. Java, Ruby, Python, Perl, Prolog, TypeScript, JavaScript, ObjC, Swift, PHP, npm, webpack, css, html, React …

Education: Master in engineering physics and one year of behavioural sciences

Babylon.js
Only reads link 1, skips link 2 entirely.
“Feels like it’s the first scene I need, hello world.”
“This does not look like something you install.”
Performs a google search and ends up at babylon.js page on npmjs.com.
Follows instructions there, runs npm install babylonjs on the command line.
“I did not get a package.json.”
Creates a helloWorld.js file in a local repository.
“I want to run it.”
Clicks on a documentation link at npmjs.com but ends up at a 404.
“I want something that works, I don’t really understand how to run it, I want something that works first, then I can learn.”
Performs a google search on how to run babylon.js from terminal, ends up in the documentation.
https://doc.babylonjs.com/guidedLearning/createAGame/gettingSetUp, reads this.
“I need a dist folder.”
Realises it’s based on TypeScript and starts making modifications for it, moves around in folders.
“Aha, npm init.”
“This was not what it said on their web page on npmjs.com.”
“I do lose some confidence in them, get less eager at investing a lot of my time in learning things they have made.”
Alternates between local installation and documentation.
“Haha [it says] if it doesn’t work the first time, try one more time.”
Follows the setting up guide now, installing.
Sighs.
“Ok, this was somewhat pedagogical actually.”
“Seriously I just want something that works.”
Runs npm run build and gets module not found error.
“Well, if it had worked right away it had been too good.”
Searches in the documentation after Webpack.
“i am sometimes pretty bad at reading from the book.”
Installs webpack dev-server, but webpack-cli has some problems, googles error message.
Performs advanced google search connected to this year, finds that it has been a problem with webpack-cli in october 2020.
Goes back to npmjs.com and webpack-cli package, locates older version, installs that.
Runs npm run start .
“Moment of truth.”
The browsers shows a 3D model.
“There were not that many lines to make it work.”

Three.js
Looks at link one, laughs when seeing the instruction to copy the entire framework and put it in a file.
“It hurts a bit, this will be bad, should I have all of it in git then?”
Goes to installation page via the menu field, not via the second link provided.
Reads aloud “For mos users installing with npm is the best choice.”
Runs npm install on the command line.
Runs into troubles when their repo is called three and the dependancy package is also called three, changes the name.
“It was good that the other was called babylonjs.”
“Gee, 0.1.2, have they not dared release a 1.0, that is like saying you haven’t committed to your community”
Sighs.
“Now I’m ready to create some kind of index.html.”
Copies the code from the guide.
“Now I have something I’m comfortable with and should theoretically work.”
Copies som JavaScript from the guide and adjusts.
“I want to make sure it works before I check… Ok it just creates some kind of camera and world.”

49
Reads aloud from the guide “If you started now you would see that it doesn’t work.” and says “This thing with early feedback, the first thing you want to see is that something works.”
Adjusts and tries different things from the guide.
“Here you need some kind of webpack but it doesn’t say, I’m a little bit interested in just copying the files from babylon and just hit it til it works but...”
Reads more under documentation on How to run things locally from that menu option.
Cuts and pastes from Babylon.js, the guide on Three.js and an index.ts file from another unrelated project.
“Frankenstein programming at its best!”
Erases chunks of code from index.ts, changes file ending to .js.
“I have not done a lot of thees we need, only a little, this is good for me.”
Now has one index.js, Express server in server.js file.
“I have not seen how to pipe an html file in express.”
Performs google search and finds you can do it with a res.sendFile.
Tries to run node server.js on the command line.
Gets error messages regarding TypeScript and file paths.
Uses devtools in Chrome to investigate.
“I’m getting a little tired, it serves now but I can’t see what is there.”
Goes back to creating a scene guide on three documentation, double checks with their own code.
“It feels like it should work, it could be that I don’t know how the source tag works.”
Tries to ask the author for help, gets no reply.
“The server and the page works but there is nothing in the dom tree.”
Performs google search and finds someone who has forgotten the camera.
“But I have a camera.”
Logs a message in chrome to make sure it works and fins error message that their box.js file not found.
Tries to put the js code between the script tags in the html file, gets error message regarding importing module.
Reverts back to having html and js code separate.
Tries to export the animate function as a module and run it in res.send in the server script.
“Now I’m doing things I don’t know how.”
Gets error messages that function can’t be found.
“Nope, now I’m inventing stuff on my own, this doesn’t work, a little more stringency.”
Goes back to the guide and reads more about WebPack.
“Maybe it wasn’t such a bad idea to pull files from the other [Babylon.js] project I made.”
Performs google search express serve static files.
“That’s not what I want, I want dynamic, this feels like it’s more complicated than it has to be.”
“Did I break my path?”
Changes back to sendFile in server, now the path works.
The code is visible in the web browser.
Restarts the server.
The code is visible in the web browser.
Adds index.html in sendFile and restarts server.
Console shows not found.
Performs google search on express path script.
Copies a solution from Stack Overflow.
Restarts server, gets syntax error, cannot use port outside of scope.
“I need something to build it all together.”
I do understand now why they wanted you to cut and paste the entire source file because then they don’t have to write a lot about how hard it is to make it work in a good way.”
Reads more on the installation page.
“Ok so they recommend you don’t use it in Node.js, but installing via NPM was still listed as the best option?”
Test concludes without sucess.

Choice: Babylon.js
Why: It gave some kind of feedback. The documentation was not excellent, but compared to three it was awesome. Both of them could have gotten to feedback earlier and provided clear instructions for different levels of previous knowledge.
Participant B

Hardware + OS: PC, Windows 10
Languages + techniques: T-SQL, C#, Python
Education: MSc Economics, self-taught, in-service training, work experience

Three.js

“There are two links, huh?”

"Now I'm reading this to see what it says" Starts reading from link two.
Runs npm install --save three in the terminal in a repo.
Tries to run import * as THREE from 'three' in the terminal, does not work.
Creates a file in the text editor, selects javascript file.
Writes everything by hand, does not copy.
Tests that the code can be run in the visual studios terminal.
Reads through everything on link two at a glance.
Checks out link one.
Copies the html code there.
Pastes it in a document.
"Can I open this in visual studio maybe? Yes, I could."
Goes back to the guide and borrows the code from there, pastes the JavaScript code between the script tags where it says in the example 'javascript code goes here'.
Follows the guide, works calmly and slowly.
"Okay so these stuff is basic settings to be able to do anything whatsoever."
Copies JavaScript.
"This function looks a little weird but I may not be so used to it?” Points to animate ();
Chooses to attach in Visual Studio.
Checks test1.html in chrome, it's white.
Checks devtools.
"It's something I missed then."
Runs attach in visual studio against Chrome.
Compares the end result of the guide with own code, adds a semicolon.
Goes back to link two and highlights a paragraph about WebPack.
Reads about examples and checks the Three.js examples on github that are linked.
Highlights the script tag and looks in the folders, seems to understand that there is something wrong with the path and how it connects to Three.js.
"Wonder if there's something wrong with my path here.”
Searches the manual for module path.
"It should be in another folder or file, that import."
Opens the three.js file located in the node_modules/three folder where the framework is located.
Tries to comment out the rotating functions, to see if it come from another module not correctly placed in the path.
Goes to 'How to run things locally' via the menu in the manual, reads about servers.
Checks the FAQ via the menu.
Tries to write an alert with hello in it to see if it comes up in the HTML file.
"There is something about my porting that is not right, but it also says there’s nothing wrong."  
Tries via Visual Studios Server, it is still white.
"Do you need to compile this in any way?"
Looking in the Three.js file again.
Looks back to FAQ about servers.
Then to installation and highlights 'script type module' under a section on CDN.
"This thing is going somewhere, where is it?"
Tries to add script type module in HTML document.
"How are we doing with time by the way?" 13 min left.
Tries to load with CDN instead.
Adds import * as THREE from 'unpkg[...].
Checks the console log in devtools.
"Here we have something."
Tries again to solve the path.
Selects 'copy as path on' the Three.js file in the file manager and inserts it as a script source but has extra double quotes.
"It still understands that it is here"
Refers to being able to write in visual studio and choose THREE. → and then you can choose from the parts that are possible.
Checks the devtools console.

"Mhmm! It says it must be forward slash."
Changes to shorter path.
Absolute or relative path, tries different variations.
"There is something about that import that makes it not understand where it is."
Test concludes without success.

**Babylon.js**

"I'll start reading the starting thing."
"It was a nice introductory text."
"Here you jump right into the code itself." not directly any setup thing
Goes to link two and creates a new repo for Babylon.js.
Starts a shell from that repo.
Runs npm install.
"Npm init, good"
"This is for TypeScript all this or? Yeah... I think so."

Goes back to the start page link 1.
Creates a test1.html in repo.
Scrolls and reads in the guide on first scene and model.
Copies the JavaScript code.
"Feels like this does not have as much code/info in the guide."
"You can not just start with this right away."
"Those instructions actually came from the Three framework" Reuses the setup from Babylon.js by copying and pasting.
"Feels more reasonable like having a js file."
Finds 'Using Visual Studio Code' in the documentation, but nothing of use there, mostly keyboard shortcuts.
Copies the HTML file and changes to .js file ending.
"It takes the import there, it seems", double checks that it is possible to get the files through Visual Studio.

Finds the playground under 'Getting started first scene' in the Babylon.js documentation.
Clicks and rotates the cube there in the overlay that appears.
Finds the section 'First model on a web page' and reads more.
Reads about it's about CDN and the babylon elemen <babylon>.
Checks out their viewer example and what the elements look like in devtools.
"Okay this is a slightly different approach on how to think, maybe I should go back here and do this step and save."
Reopens the playground.
"This is exactly the code I have."
Tries to export the scene as .glb.

Goes back to the viewer example.
"So that file is just a link inside that tag then."
investigates how the scripts and imports work there.
Tries to create a babylon model element based on viewers example.
Reads more about how the script works with CDN.
Copies the script tag with CDN.
Reads about first import of a model.
"A little unsure of how to assemble..."
Finds script tag with CDN in viewer example.
Creates an HTML file and pastes in the script tag similar to that example.
"Feels like it creates a little more in there."
The Babylon.js logo is displayed in Chrome, like a spinning loader.
Removes some extra parts in the HTML file that was left.
Removes everything except the script tag and babylon element.
The logo still spins.
Moves the script tag to head and reloads.
Nothing appears, console log says 'No url provided'.
Compares the viewer's source code with own code.
Copys and pastes in another part and reloads.
Still error message 'No url provided' and additional error on the audio context.
Performs a google search on the error messages.
"Is it because I took it from them [Babylon]?
Peruses the documentation.
"I have this, .glb format."
Closes all files in the text editor.
Tries to open the HTML file in Chrome.
Performs a google search on 'babylon viewer js 2098'.

Gets very few hits.

"Is that a line reference, okay…" [about 2098].

"Is it the glb file it is missing then?"

Looks for the error at the line tries to interpret the error message.

Tries to import in the playground and then export everything to a .glb file and then import this locally.

Reloads the page, it's white.

No error messages are visible in the console.

"It does not seem to be better in any case."

Test concludes without success.

**Choice:** Babylon.js

**Why:** The viewer function and the compiled file, as it seems easier to use. It seems that the source code would be obfuscated, which might prevent scripting attacks.

---

**Participant C**

**Hardware + OS:** Macbook Pro / macOS 10.14.6

**Languages + techniques:** Javascript, PHP, p5.js, React, TypeScript (no 3d experience)

**Education:** Bachelor in Web Programming

**Babylon.js**

"Let's see."

Checks the first link.

"Chapter one seems to be it."

Performs a google search to find npmjs.org.

"I still think there are no instructions."

Switches between visual studio and the guide on npmjs.org.

Creates a test.js file and pastes example from guide.

Adds import " as BABYLON; at the top of the js file.

Compares the guide instruction with the guide code under the first scene and model in. Babylon.js documentation.

Clicks on a playground examples.

"Oh, it's their thing to test stuff, let's skip it".

Checks out the viewer news example and copies the entire source code in the background.

"Ok copy this so we do not have to think so much."

"Yeah exciting babylon", refers to the <babylon> element

"Never seen such a custom tag before, exciting."

Reads under first app on the web.

"Well of course, I have to have a canvas."

Comments out the babylon element.

Pastes in code from an example under first app on the web in documentation.

Brings up index.html in Firefox. No luck.

Finds more code on the engine and the loop and copies this into the JS file.

"The first example was a bit strange because it jumped right into the code."

"Do we have everything? We have canvas, engine, stage, camera attached, box… but it is not used anywhere."

Reloads the page in Firefox, does not work.

The script tags from the copied examples are with CDN, this could be related to the problem.

Tries to add more cdn scripts with loaders.

"This feels stupid that I should need… because I already have babylon... have to retrace my steps a bit"

Reads from the beginning of the documentation.

Notes that it says 'to be usable in a playground'.

"But playground is just their online tool or..."

"Nope we'll skip it."

Reads a section with the text 'now we will give you an html template to do just this'.

"Maybe I should use npm then."

Heads to to npmjs.org.

"This looks better."

Copies code from npmjs.org under section 'usage'.

The npmjs example is from version 4.1, not 4.2 which is the lastest.

"This is kind of the same as what I had."
Comments out all CDN script tags.
Reloads.
Still empty in the browser.
Types in run npm init in the terminal.
“So package.json.”
“God what I feel stupid, there’s something simple I do not understand.”
“But I may have to run npm start.”
Reads readme.md fil in node_modules.
“Aha here is all I need.”
“A list of extra packages, do I need it or not?”
Checks out babylon.js at npmjs.org again.
“It should work…”
Creates a div with the text ‘hello’ in index.html and reloads.
Hello shows up.
Checks Firefox devtools.
“Aha canvas is there.”
Calls on the scene object in console.log().
“I shouldn’t need anything else.”
“Oh god, I never load the .js file in the index.html, good thing my boss can’t see me doing this.”
Corrects this mistake and reloads.
“Aha, import declarations may only appear at the top level of the module…”
Module and import problems show in console.log in Firefox.
“I feel there is something very basic here that I…”
Goes through all local files, tries different setups, tweaks.
“Haven’t they got any examples?”
Looks at Babylon.js repo on Github.
“Then we’ll have to start googling basic stuff here.”
Performs google search on ‘import mdn’ and reads up.
Sighs.
Checks package.json.
Goes to mdn again
Tries to import with import() and reloads page.
Get typeerror when checking in devtools.
“I don’t get this at all.”
Tries variations of the imports.
Gets CORS cross origin req blocked.
“Reading the remote resource? Oh god.”
Performs a google search on ‘script mdn’.
“This has nothing to do with Babylon, this stuff.”
“That’s how I did it”
“Now I will make it much easier for me”
Moves the JavaScript code into the HTML file.
Adds ‘script type = module’.
Removes this and reverts the file.
Goes to mdn ‘use our module guide’.
Looks at package.json.
“Should I not have som kind of npm start? how to connect…”
Performs google search on ‘npm getting started’.
Finds a beginner’s guide on Npm.
“I use this every day.”
“How do I make Babylon work with npm?”
Goes to the documentation on Babylon.js.
Checks the source code in viewer news example.
“No it is not relevant.”
Checks npmjs.org again.
Finds examples there, clicks through and ends up in the documentation in the Babylon.js documentation.
Heads to ‘Guided learning’.
Goggles ‘getting started with babylon.js npm’ comes to techshard.cwpcomstaging.com
“Okay codepen, this is everything, no.”
Realises the example uses CDN.
“This does not use npm either.”
Goes back to google results.
"An absolute beginner [guide]! No, it was at Npm... although that’s where I am now.”

Reads about npm at nodesource.com.

"This was not that fun."

Goes back to local installation.

"What was the problem now, I forgot."

Looks at devtools and tries to decipher the CORS request error.

Checks the repo.

Switches to Chrome instead of Firefox.

"No it did not help. Hmm."

Tries to change privacy to get around local file security, it works. But new type error.

"How difficult is this going to be?"

"Is there something I should run then?"

Tries to run node test.js in the terminal.

Renames test.js to main.js.

Reads about npm again a getting started guide on w3resource.

Noticeably frustrated.

Googles ‘npm import not working’.

Googlar ‘node getting started’.

"I do not understand why I can’t make this work."

"Should I seriously need to create a…” [server]

Creates a simple server.

"This is probably what I should have done all along, oh god."

Googles ‘import module’.

"There is something here I do not understand…”

Test ends, unsuccessful.

Three.js

Reads aloud from link 2 “For most users installing with npm is the most… okay this seems promising.”

Creates new environment with repo.

Runs run npm init in terminal.

Gets error when the repo name is three and the npm package as well. Renames.

"I get a bad feeling about this.”

Reads about bundling tools in the Three.js docs (link 2).

"Aha Webpack."

Clicks through to Webpack getting started guide.

Also clicks through to Eloquent JavaScript on ES modules, leaves fast.

“Okay let’s use Webpack.”

Runs npm install --save-dev webpack.

"I feel like going back to the other and trying that with the same [babylon]"

"Okay so we should have index and source [folder] with index.js."

Copies and pastes sample code from the docs into index.js and index.html.

Runs npm install --save lodash.

Read about creating a bundle in the Webpack guide.

Creates a dist folder.

"They have no finished file here at the end” References the Webpack guide.

Runs run npx webpack.

Compilation successful.

Opens the HTML file in the browser and sees “hello webpack”.

"Okay then I think this will work."

"Okay examples.”

Has not checked link 1, only link 2 so far from the provided links.

Adds some sample code from Three.js docs link 2 and recompiles, nothing happens.

Goes further in the docs and finds ‘Creating a scene’, reads the HTML file in the example.

“Oh well! This looks good.” About the JavaScript code.

Incorporates the JS code in JS file, reloads and sees a canvas in the browser.

"Cool, let’s fill it with something."

Copies code for the cube in the docs, reloads, recompiles, reloads.

"Mhm! No box.”

"Ah, rendering.”

Finds the animate() function in the docs, uses this and reloads.

The green box animation is visible in the browser.
Choice: Three.js
Why: Because I got it to work :) No but seriously it included some idiot-safe instructions, which I needed, regarding using webpack. Also, I have heard three.js being mentioned more so that is also why I would try that one first.

Participant D

Hardware + OS: Ubuntu on Asus laptop
Languages + techniques: Python, java, c#, c++, javascript. Not familiar with 3d
Education: Data science, engineering and economics

Three.js
Starts by copying in npm install --save three in the terminal.
Gets error ‘no such file or directory’.
"Do I have to stand in a certain place?"
Googles the error message.
"Ah I probably have to run an [npm] init"
Runs npm init.
Runs npm install --save three.
Does not check the other link but reads directly from Npm install page.
"Now it seems to work better."
"Okay now I seem to be finished, am I done?"
Observer: "Do you see a 3D model in your browser?"
"No, I do not, that’s right."
Creates a test.js and enters some import statements from the docs.
Tries to run the file with node in the terminal, gets error messages.
Googles ‘run node js file’.
Tries to add the string ‘hello world’ to log it. Gets error.
Googles the error ‘unexpected token import’.
Copies --experimental -modules -test.js and tries to run the file with those flags.
Hello world shows in the terminal.
Re-adds the import statements again, does not work.
Selects import option number 2 in the docs, separate modules.
Error message: ‘unexpected token’.
"Is it the extension it’s complaining about?"
Switches to file ending to .mjs.
Error message: THREE.Scene is not a constructor.
Googles the error message.
Finds that it is the wrong three.js file.
Also sees more code for what a standard installation might look like in the Stack Overflow post.
Investigates the three.js files for a build folder.
"Maybe it does not like my path, if I run an npm init here then?"
Moves the location of JS file.
Operates everything via a terminal.
Does not notice that more code is needed than just importing a scene.
"Nope, its not a constructor."
Reads more on SO.
Copies a CDN link, changes mind, reads more on SO in that thread.
"Okay so it does not like my file it seems."
Checks the folder again.
"The question is whether I should actually replace them."
Moves around the file tree.
Googles ‘path npm’.
"Let’s see if we need a path, add a path maybe?"
Ends up on nodejs.org/docs
Sighs.
Googles ‘add path to module’.
Goes into the js file again and writes three = require (‘three’) and changes the import.
Runs the file again and gets ‘Unexpected identifier’.
Googles the error message.
Googles ‘node print debug’.
Goes to node js docs.
Googles ‘node print error stack trace’.
Copies console.trace(); into the .mjs file.
Same error messages.
Reverts back. Gets new error ‘Require is not defined…’ and googles it.
Finds that it should not work in the browser due to Node.js specific.
"Yes, but why does it work inside the terminal then?"
"It seems to work so far."
Googles ‘node js list submodule’.
Googles ‘list node js packages’.
Finds a function on SO.
"Let’s try to print all the packages to begin with."
Tries to do so but gets ‘Require is not defined’.
"Require it does not like require."
Reverts to importing instead.
Gets ‘Syntaxerror unexpected token import’.
"Then we got rid of the first mistake at least."
Reads up on the reference error.
"The module type was causing the issue."
"Feels like I’ve gotten side-tracked."
Goes back to Three.js’ installation page.
Tries out option two listed for imports.
Gets error message concerning scene.
"So it does not have a scene."
Looks through Three.js file for scene.
Searches for scene in the file via the terminal with grep.
Finds that there is a scene that is exported from there.
"Yeeees…"
Googles ‘node.js list module exports’.
Reads about how modules are connected.
Goes back to ‘Creating a scene’ in Three.js docs on three.js for a second, clicks back to the installation page on npm.
Tries to delete all code and redo from the beginning.
Gets the same error message.
Logs (Object.keys (THREE)); in the terminal console.
Runs the file, gets ['default'].
Tries to log (Object.keys (THREE.default));.
Looks through the list.
"Aahh… this was not so smart."
Changes scene to be three.default.scene.
Runs the file in node and receives no error messages.
Finds a code section on orbitcontrols from Three.js npm doc page, link 2.
Tries to use the code, nothing happens.
Reads more about ‘CDN or static hosting’ in the docs.
Reads more about orbit controls.
"No such file, wonderful."
"Ah, of course"
Changes to node_modules in the path import.
"Mhm, can not find module, and why not then?"
Tries to do according to the second syntax stated in docs: import * as TEST from "path to orbitcontrols but gets error message.
"Can not find and why can you not find it then?"
- Test concludes unsuccessfully.

Babylon.js
"Let’s take the first here, Chapter 1."
Observer says explicitly that the second link is about Npm.
"Okay."
Reads under chapter 1.
"Say hello to your first world, it was straight on here."
"Okay so then I will install Babylon first"
Googles ‘install babylon.js ubuntu’.
"Now it sounds like [baby’s name] is waking up."
Ends up in the documentation.
"Oh that’s right this was the link you sent."
Reads at the top of the page and runs `npm install --save babylon`.

Goes further down the page, scrolls to setting up the project.

"Building your project, checking your project. This was quite detailed."

Pastes some code from 'first scene and model' section in js file.

Receives error message about engine.

Scrolls up and down in the npm file, reads about engine, canvas, scrolls frantically.

"Ah we can create an Engine then."

"Here it seems that you can use the engine directly."

Enters BABYLON.Engine in the code and tries to run it.

Gets 'document is not defined'.

"Can you leave it empty maybe?" Refers to Babylon.Engine(null, true).

Gets error message that it is not a constructor.

"Is it default again or?"

Adds default between like so BABYLON.default.Engine().

Gets error 'canvas is not defined'.

Checks out 'First scene and model' again.

"Hmmmmmmm."

Scrolls through the npm support page again.

"Our index.ts will show..."

Copies 'npm install webpack'.

"I probably jumped a little too fast I think."

Reads about the webpack file.

Runs the npm command.

Gets 'missing required argument', googles the error message.

"Using an old version of Npm."

Checks version, has 3.5.2.

"Now [baby] has woken up."

-Test concludes without success.

Choice: Babylon.js

Why: Better and more explicit documentation.

Participant E

Hardware + OS: OS: Pop_OS! Stationary PC: Intel i7 6700 Nvidia GTX 980 Ti 32 GB RAM

Languages + techniques: JavaScript, TypeScript, Scala, C++, Unreal engine, Unity.

Education: University program in software engineering and self projects

Babylon.js

Clicks link 1.

"I'm looking for something with installation or install, quick start, but this looks like a book or something."

"Maybe I can create a project somewhere on my computer."

Creates a folder, calls it babylon-proj.

Opens a terminal window in the folder location.

Scrolls through link 2 on Npm support "This was not so clear to me, I have to read properly."

"I guess you have to run npm init."

Looks for template project.

Opens VS Code.

"Then we have our package.json here."

"Now I think I can follow the instructions."

"JavaScript or TypeScript, looks like you can do both."

"I have a main file here, an index.js."

Pastes import * as BABYLON etc at the top on js file.

Reads about the Webpack configuration in the docs.

"This seems like Webpack and TypeScript."

Copies an index.html file snippet from the docs.

"Though here is their output." Refers to the dist folder in the Webpack config.

"No this is not the way I usually do."
Checks through other menu options.
Checks getting started again, first scene link 1.
Clicks on playground.
Goes to babylonjs.com front page.
Finds ‘Get’ in the menu and ends up back in the docs.
"Yep, we have been here before."
Finds a menu choice in the guided learning section with a game tutorial.
"Maybe I should check it out."
Finds an example of a file tree there.
"Okay so we are working with TypeScript now."
Follows the guide with ‘npm install babylon’ etc, sets up the files like so.
"Inspector, do not know what that is, but we’ll install it anyway."
Also runs ‘typescript init’.
Goes to npmjs.org to read up on TypeScript installation.
"Okay, it was a bit confusing if I really installed it so quickly."
Copies tsconfig.json from the guide.
"Aha here they install TypeScript, I react to the fact that they don’t have npm install typescript during configuration, but I am not an expert on it."
"Looks like they forgot to make a new line here." Points to a missing line break in the guide.
"Okay here we have our app.js file."
Pastes in a class provided called App.
Then runs ‘npm run build’, then ‘npm run start’.
Gets error, "Oh maybe I have something on port 8080 already."
The error is "cannot found module webpack-cli / bin / config-yargs."
Searches the guide for Webpack again.
"Setting upWebpack… have we done that?" Reads that if it does not work, you can try again.
"I will try to run them again."
Compares Babylon.js github repo package.json with own file.
"Yes they have much more."
"Yep I’ll run it again."
Runs ‘npm install --save webpack’ etc.
"Updated four packages, okay."
"We’ll continue."
Re-runs all npm installs.
"I do not know if it does anything. Just to check I’m not mistaken."
Runs ‘npm run build’-
Googles the error message ‘clean webpack plugin. options output path not defined’. Runs ‘npm run start’.
Same error as before plus ‘cannot find module webpack-cli’. Googles error messages and finds an open issue about it on github.
"Should just try to change the port, no I do not think it will work."
Reads about versions of Webpack and Webpack server.
Checks own version and which ones are available today.
"Well that’s the latest all right."
Tries to run ‘npm install webpack webpack-cli’. Is unsure whether to change from webpack devserver to webpack serve but tries to change the script in package.json to it. Runs ‘npm run start’
Checks file in browser and sees a sphere.
"All in all I want to start, not to learn at first."
"I want things to stand out, I have ADHD, I don’t have the energy to read a lot."

**Three.js**
"This was much easier is the first impression."
"Larger text, very clear."
"Not really clear where you are." Refers to the menu option not having a visual indicator.
"Aha so they"… Refers to placing Three.js directly in a folder.
Reads more about Npm in the docs.
"Yep yes I’ll do as they say."
"Now it feels like they are mixing two tutorials here."
"Normally I would have installed this so and then connected via package.json but now I do exactly as they say because it is the first time."
Pastes the HTML code and more into the correct file structure, also Three.js in a folder.
Looks for live server in vscode to check it out.
Now sees the rotating cube in the browser.
"Am I done?"
Observer: "No, you have not installed with Npm."
Deletes everything in the project repo.
Runs ‘npm init’ and then creates an index.html.
"I don’t have the energy to read this." Refers to a lot of text at the bottom of the docs page link 2 on node.js.
Finds an example code at the bottom and copies it.
Highlights a short passage on bundling and Webpack and reads it thoroughly.
"Too bad they had a good start here though."
Searches the manual for Webpack.
Looks at other menu options on how to run things locally.
Copies the JS code from the example in the docs into the index.js file.
Creates index.html.
Investigates more about paths.
"Feels like you still need Webpack, I do not think I can do just that, but it says nothing…"
Checks agains.
"Usually I would have googled and checked out another tutorial."
Looks through some examples, threejs.org/examples.
Finds the source code for the example on Github
"Aha, they use script type module... Holy fuck…" Reacts to a lot of extra files there.
"Should I run with Webpack or can I run with live server?"
The observer: "It does not matter."
Finds a paragraph on Webpack in the docs, skips it, reads on Npm via link in the docs (Eloquent JavaScript).
Runs ‘webpack serve’ in the terminal, does not work.
Runs ‘npm install webpack webpack-cli webpack server’. 
Troubles with installing webpack server, googles ‘live server npm module’.
"Nah, I’ll have to go back and read."
Clicks on ‘Download’ menu option. "Oh no it just downloads directly."
Ends up in docs again and getting started link 1.
Clicks on the Webpack link in the docs.
"Oh my god." Reacts to a lot of information.
Creates a webpack.config.js according to the guide there on Webpack’s website.
Reads about entry points and finds the right entrance, copies into config file.
"Well then we’ll have to install it too."
"It feels cheating to check how I did in Babylon."
Goes into Babylon’s documentation and copies that Webpack configuration from there.
Adds build and run scripts in package.json.
"This was much harder, Three I mean."
"Oh I forgot to install it"
Installs webpack, webpack-cli.
Checks out the model in the browser.
"To be honest, I used Babylon’s documentation to run Three"
"I got disappointed because Three was a little more readable and clear in the beginning."
"The impression was nicer with Babylon, shading, etc. That you could spin the model."

Choice: Babylon.js
Why: It gave me a better initial impression of its capabilities as well as the depth of documentation that is available on their official site.

Participant F
Hardware + OS: Win10
Languages + techniques: C#, C, C++, Java, Python, PHP... (but not a lot of JS). Know how 3D works on a mathematical level.
Education: Master of science in computer science

"I usually come to a framework knowing what to do."
The observer: “You have to install it locally, see a 3D model, and you have two possible links on how to solve it.”

“Oh right yes.”

Follows the links and reads.

The phone rings. “Excuse me, I'll just see who's calling.”

Goes to link 2 on Npm.

“It presupposes a lot of things here, that I have a project folder, a terminal instead of a command prompt... hmm.”

Creates a repo and runs `npm install --save three` in the command prompt.

“I'm trying to understand where they want this to be.” Refers to index.html and three.js files.

“Somewhere there should be some kind…”

Creates a test.htm file.

“I think there should be some kind of web server here but since I have not worked with this stuff before, it's a bit hard to know.”

Pastes HTML code from the docs.

“Feels like one step is missing.” Refers to the docs.

Pastes some JS code in between the script tags.

“That's going to be weird, it's not there at all” Refers to where Three points to in the index.html example and the path compared to where Npm installed three.

Reads about the installation via link 2.

“Somehow I have to be able to run this stuff, it could be very obvious but…”

Deletes the path in the sample file.

“For lack of better knowledge.”

Copies more javascript code between the script tags.

Opens test.htm in the browser.

Checks inspector in Firefox devtools if the code is run, the code is gray.

“Noo.”

Pastes in the code for the cube as well as the animation function from the docs.

Reloads browser.

Scrolls through the docs and sees an example files there at the bottom.

Checks the file path manually in file browser.

“Whyy.”

Copies that path and puts it into the script tag.

Changes slash direction.

“I hope it's obvious that I don't have the faintest idea what I'm doing.”

Tries to open in browser again, it is empty.

“I feel I need to get it in, in some other way.”

Comments out the import and reloads, no difference.

Checks js-fiddle example in the docs and compares with own setup.

Finds another path to the build folder.

Pastes it into sample file.

Reloads, black background visible in browser.

“Aha mhm.”

Copies rest of the code from js-fiddle example, reloads and the animation shows.

Babylon.js

Creates a new repo for Babylon.js.

Reads instructions.

Runs `npm install --save babylon.js`

“Funny that they have different save and install [Three and Babylon]. Oh I read it wrong, it was actually very similar.”

Goes to link location ‘Getting started, say hello to your first world’.

Checks playground.

“It’s now you get to take advantage of having just created another thing”

Copies a bit from the three.js index.htm and .js file setup.

Checks the folder with babylon.js to find the correct path.

Copies it from the file manager into the script tag.

Saves and opens in browser, it shows a white empty page.

“Ah it was too easy.”

Changes to max.js in the path, reloads and still white.

Goes back to the first scene instruction.

“Mm, of course.”

Pastes in code with scene and more from the playground. Reloads, white.

Changes the path back, deletes max.js. Reloads, white.

Goes to the first scene and model again in docs.
Tries to use a self-executing function from the playground example, does not work.
Saves the code from the playground and opens in text editor
"Nope."
Gooogles ‘babylon.js html’.
Finds a basic demo on Babylon.js on Mdn.com
Goes to babylon.js-guide.github.io/begins/introducing_Babylonjs and read there.
Copies the HTML template there to local installation.
Reloads and the 3D model is visible.
"Again I do not know how I did"
"Usually you want to see that something works, before you get into it."

Choice: Babylon.js
Why: Because the test script resulted in a cooler/prettier image/animation.

Participant G
Hardware + OS: HP EliteBook G3, Windows 10 Pro
Languages + techniques: ASP, PHP, VBScript, SQL, JavaScript, haven’t worked with 3D before
Education: 1 year of web programming (upper secondary school level), otherwise self-taught & worked with IT & programming

Babylon.js
Reads info on both links, some out loud.
Gooogles ‘what is npm’.
"Okay so you can use it to install nice libraries."
"Ah okay so it's a pretty new thing, oh damn 2011 okay I'm a little behind."
"Wow okay, what you can do… " Refers to information on 3D via link 1.
"Okay I need a scene” Clicks on that link and reads more.
"Okay have to take this in properly.”
Gooogles what a facet is and reads.
"Shit now I'm doing something else, no it seems like part of a greater whole."
"Normally I would not have read so much but jumped right in.”
"It's a little confusing that they first show some code and then ‘okay but first you have to do this’" Clips on playground.
"Okay it's this demo I guess then created by that code”
"Aha can you change here, can you write cone?”
Changes to cone instead of box.
"How do you run this?”
Finds that you can change to createSphere.
"Cool.”
"Okay now I'm getting side-tracked here.”
"I do not understand how to install.”
"I do expect some kind of download.”
Gooogles ‘babylon js’
"Hmm nothing appears with download”
"A bit annoying, not sure if I'm thinking wrong in some way.”
goes on github and reads there, does not see npm
"But I want them locally”
Goes to link two.
"Oh that was where I was before, sorry hehe.”
"Gui module what is it, exciting, check this out.”
"Why can't you just go somewhere to get it.”
Reads more on link 2.
"Okay then I’ll do it' Writes ‘npm install --save babylon.js’.
"Then I’ll consider it installed?”
Checks in node_modules.
Reads again on the npm support page on babylon.js, scrolls a bit.
Reads again on getting started page.
"It's just that it starts with a lot of code stuff, it's not what I call getting started, it's like they are so eager to show how easy it is to write code, but you need to have the framework somewhere."

Searches the documentation for npm, comes to a top level page for npm support.
"God that is a lot of bla bla, come to the point, you can't write this much on a web page."
"It's like I'm missing an intermediate step."

Scrolls around the npm support page.
"Aha import… is it JavaScript or npm?"
Tries to write import in the terminal.
"Nope, use import where?"
Gogles import.
"What is the context?"
Finds a page on html5gamedevs.com.
"Aha TypeScript, maybe something I missed…"
Reads on TypeScript on Wikipedia.
"It's unclear if I should dive further into this or if this is something else."
"Normally around now, I would go to YouTube for ‘wtf is TypeScript’"
"I'll have to retrace my steps a bit."

Goes to the terminal and tries ‘npm -?’. 
Tries to open Babylon.js locally.
"Trying to understand."
"I'm so lost, when I don't get something I usually look around completely random"
"Okay it's a kilometre long file." Refers to Babylon.js file.
"I'm so mad that I'm not getting it."

Creates a new HTML file.
"But no I will read a little more first."
Gogles ‘import javascript keyword’
"Okay it might be javascript"
"Okay this may be how you do these things now, how the hell is this supposed to work."
"I'll create a JavaScript file and HTML file."

Tries to write import directly between script tags in the HTML file.
"But hell, it can't be called just babylonjs, it's like this is written in a context I do not understand."
"I'm going to YouTube!"

YouTube search for ‘how to run bablon js locally’.

Watches 14 minute video - ‘babylonJS Tutorial Series - Part 1: Getting started’
"Canvas, but it's still not… well I still can not, I do the same."
"Noo he runs remotely." Refers to the video tutorial’s use of CDN.
"Well I'll have to go back to the documentation and see if I understand."

Reads about ‘npm init’ on the npm support page, link 2.
Never scrolls all the way down the page.
Moves local HTML file.
Runs ‘npm init’. 
"Not quite sure what this will do."
"Okay what have I done, god I'm so fucking lost, I understand it's not a problem for you but."
Runs ‘npm install webpack’ etc.
"I don't like to not have control over exactly what I do."
Gogles ‘webpack project’
"Okay compile, not compile but probably build."

Reads about concepts in a guide on Webpack found via Google.
"There are so many things here that I don’t understand. But where and what should I do before?"
Does not create a webpack.config.js.

Searches the entire computer for that file that is listed in the npm support page.
"Aah maybe I put stuff in the wrong place."
"No I have not."

"Need to test some stuff."

Sighs.
"Crazy shit."

Notices that dist is in the path "Is this how you’re supposed to write or just an example?"
"Now I guess I should put it… no."
Tries to figure out where to put the HTML file.
"It's a bit lost in translation."
"I think it should be the absolute easiest thing to just get started, but I might be missing something."

-Test concludes without success.
"I could be the one at fault here."
"Oh they must have worked really hard on this and then here I am not understanding." Genuinely sad.

Three.js
"How nice, I already like it more, it explains the different ways."
"Now I have practiced a bit."
Copies previous repo.
Runs 'npm install --save tree'.
"I don’t understand." Refers to where to add imports.
Creates a new HTML file and places the import there between script tags.
"I guess this will not work but I will still try it."
Opens the file in the browser and checks the console.
"It says that you can not use import like that."
"God, this is what I need to understand."
"Script type module this is new to me, never seen before."
Googles it.
"Fuuuck, I do not get it."
"Let’s read around, really I do know JavaScript."
Reads about modules on Mdn.
"Thank you! Someone who understands me here."
"Cool, I did not know this."
Googles ‘canvas element’.
"Okay then I know, this is really cool, you can do so much nowadays."
Tries to write the script tag differently with type = "module".
"Need to understand this thing with the path."
Searches the file manager, enters a path in the script tag.
"No wait this won’t work, no I’ll place the file in the same folder."
Puts the html file in node_modules.
Reads the npm support page via link 2, has not read link one closely yet.
Searches ‘local’ in doc, ends up on a page ‘How to run things locally’.
"Aha mhm."
"I already have a web server."
"I’ll try to upload the file there."
Tries to access localhost in browser, gets ‘not found’.
"Come on computer you can do it!"
Configures a bit.
"Right now my file is running on my web server."
"Okay then we have a web server running"
"But how do I get this damn thing there."
Reads about creating a scene via link 1.
"This is so nice."
"Okay let’s do this, let’s freak out."
"Now we install it here."
Tries to install npm in a folder being served.
Gets admin rights problems, fixes that.
Runs ‘npm insall --save three’ again
"Okay seems to work."
Tries to find the right path.
Reads more about the import in the docs.
"I know I have the right page up here but I just have to check…"
Places the path in the script, reloads, opens browser.
Gets an error that devtools could not load one specific thing, but it seems to work.
"Let’s pretend everything works now."
Replaces the HTML code with the example from ‘Getting started’ in the docs.
"And what JavaScript is supposed to be here?"
Scrolls down and finds a complete file.
Tries to put it between the script tags, saves, reloads.
"Aha nothing is visible."
Gets error ‘three is not defined.’
Changes to script type module.
"Now I'm just playing by ear. God this will not work."
Shuffles around among the script tags.
"Now I'm so just guessing."
"I have to read up some more."
Checks a jsfiddle example in the docs.
Sees that it says that you can put the entire framework in a file locally and tries to do that.
"Now I expect it to work."
Pastes in the code example, reloads and the cube is visible.
"You get a little peace of mind when you see some kind of result, even if [it is not with npm]"
Reads again in the npm support page.
"It's this import that I do not understand."
Clicks on the Eloquent JavaScript link referenced in the docs.
"Sometimes you just have to take your time and read this stuff."
Looks around in three under node_modules.
Reads about package.json on Eloquent JavaScript.
Runs 'npm init' and thus creates a package.json.
"Now I'm trying stuff again without a plan."
Reads again on Eloquent JavaScript.
"I probably have to read up lot more, what’s happened the last ten years [with JavaScript]"
Goggles import statements and finds a forum post.
"Okay so you have to do a build."
Follows a link that mrdoob suggests with basics for modules on exploringjs.com and reads.
"Why do they have a lot of fucking code without saying WHERE to write this code, just dive in without understanding the context."
"Damn this is confusing."
"It's like JavaScript has evolved but not the basics."
"I guess you write a .js file where you write import first and then the JavaScript below that."
Goggles 'JavaScript keyword import'.
"So much to learn."
"There are a thousand examples but I don’t understand exactly WHAT is happening."
"I understand that I should import and that this is the part I should load."
"I'm so confused."
"Should you do it manually? I do not understand where this should go."
"Lost in translation, there’s something absolutely basic I don’t understand."
"It just says 'ready to import it into your code', should I do it once or every time?"
Reads about node.js in the docs via link 2.
Reads creating a scene again link 1.
"This is not that complicated, so it's not this thing that's difficult."
"Must think."
Clicks on FAQ menu item in the docs.
"How often do you click on FAQ and find what you are looking for, never happened to me! Nope, not now either."
Checks in the three folder locally under node_modules.
"Do not know if this really makes me wiser."
Opens a random file to look.
-Test concludes without success.

Choice: Three.js
Why: Easier to follow the documentation.

Participant H

Hardware + OS: HomePC - Ubuntu 20.04
Languages + techniques: I've used JavaScript, TypeScript, Webpack, NodeJS, Python, Bash, Powershell, PHP, Docker, Kubernetes, Mithril, React, ReactNative, AngularJS, VueJS. I do not have any 3d experinces within the web but have worked and designed in autocad before.
Education: BoS within Web development

Three.js
Follows link 2.
Starts to run `npm init`.
Runs `npm install --save three`.
Creates an HTML file.
"Wonder if they have a dist folder you can import directly? Yes they have a build folder, great."
Do they have an HTML page you can use? No."
Tries to get the path, finds the way to the build folder.
Copies the import code from the docs to the HTML file.
Scrolls a bit in the npm guide in the docs.
Goes to link 1 and copies the html code there.
"Let's take this instead."
Changes the script path to the module and creates a scene.
"Webpack is not needed? No, just straight on."
"Too bad they don't have something to copy and paste on the spot, aha yes they did, I was a little too fast."
Copies this, saves and reloads.
Sees the cube in the browser.

**Babylon.js**
Follows link 2.
Runs `npm install --save babylonjs`.
"Npm init I forgot."
Uses the same setup where Three.js is and the same index file.
Reads about scene and examples via link 1, then returns to the npm support page.
Tries to import directly into the HTML code.
"They have no HTML page you can..."
"Hmm, they need a canvas element anyway or? No."
"Okay this was not as easy to get started with, I can tell."
Tries to reload.
Googles 'getting started babylon js'"n
"Aha, it was the same, great."
Scrolls and reads on the npm support page.
Reads properly under import then changes script-type module.
Copies some code from first scene and model.
"Engine, where does it define engine?"
"Babylon model."
Reads about the special element that babylon made, pastes it into the body of the HTML file and links that path to the main.js file.
"Then you still need Webpack, that's sad."
Runs `npm install --save babylonjs-materials`
Imports materials in main.js.
Checks if has TypeScript installed, no.
"It was not in the requirements either if I did not miss it completely."
"Oh, good times."
"Good thing they put this so far down the page."
Runs `npm install loader and webpack` etc.
Creates a webpackconfig.js.
Modifies it to use a main.ts entry.
Deletes a bit in main.js and replaces according to instructions under ES modules on npm support page.
"What is this, no thanks."
Reverts some changes to Webpack config.
"There are more things needed, ok."
Copies the index.ts file example with example sphere.
Tries to add the right build script in package.json.
Tries to build.
"No? What is it complaining about then?"
"Save the file of course."
Saves, reloads, still doesn't work.
Googles 'getting started babylon js.
Starts following a medium article instead of npm support page.
Deletes everything and starts from scratch.
"Something with source classes..."
Creates all files via the commandline.
Copies HTML file boilerplate, CSS, JS.

"I have no Webpack file." Creates one.

Copies basic server setup, game example, scene.

"Render loop .." Gets type error on when runs.

"Something else it wants." Where do we have the build script?

Creates it, runs 'npm run build'. Gets 'unexpected token'.

Tries to remove <doctype>.

"It does not want to cooperate."

Switches tutorial, now on dzone, reads.

Opens Babylon.js npm support page again.

Reads more about requirements.

Ends up under the 'first web app'. Selects the CDN script.

"We can delete everything and start over."

Copies that HTML code example under 'first web app'.

"What's going on now?" Vscode seems to have frozen. Thaws up.

Removes the CDN script tags from the HTML file.

Runs 'npm install --save babylonjs'

"No loaders or... no, guess they are not needed."

Adds path to babylon.js in the script tag.

Opens index.html.

Console log says 'create scene is not found'.

Goes back to getting started first model, own scene code with this.

Reloads, the background is purple which mean the scene works.

Gets 'no camera defined'.

Reads more about camera and lights.

Compares the example in the guide with own code.

"Does not look quite the same."

Reads more.

"Is there no code for this?"

Copies some code from the npm support page.

"Okay let's take a shot."

Adds camera, sphere and engine by typing by hand.

Reloads and gets 'engine is not defined'.

"Okay..."

Finds more in the docs.

Copies, saves, reloads.

"Is it never imported?"

Pastes in the canvas as well.

Tries to defined more variables, reloads, get reference error engine not defined.

Reads more about imports, tries to paste in, reloads, gets syntax error.

Tries another alternative.

Gets 'arcRotateCamera not defined'.

"Okay, that was good."

Adds BABYLON. after the camera is instantiated.

Does the same with vector3 and more objects.

Reloads and the sphere is visible in the browser.

"I would have liked everything more grouped together in Babylon, the information was in so many places."

**Choice:** Three.js

**Why:** My first impression tells me that it is easier to use.

**Participant I**

**Hardware + OS:** macbook pro, ubuntu

**Languages + techniques:** c++,c#,html,js have used 3d before. not used webpack/node in a while so didn't
remember how it works and how I remembered how it works was wrong and made me confused

Education: self-taught, worked, courses

Babylon.js
Runs `npm install --save babylonjs`
"Well now it went wrong."
Deletes everything and creates a test folder and then runs `npm init` first and then `npm install`. 
Reads about imports via link 2 then reads under ‘getting started’ via link 1.
“So we can test an example then to see if things work.”
Copies some code from getting started.
“It’s been a long time since I’ve done this but we’ll see if we get things to work”
Gogles `npm run package.json`
Ends up on micahel-kuehnel.de guide about npm, follows it a bit.
"Aha maybe it is not an imported script, but why doesn’t it want to, now I lost the second link.”
Pastes import * as BABYLON at the top of a .js file.
Gets syntax error, cannot import module.
"Ouch ouch ouch.”
Tries require instead of import.
Reloads, gets ‘engine is not defined’. 
"I suppose I have to go back and check out how the heck this works.”
Gogles ‘npm tutorial’.
Reads over at nodesource.com.
Double-checks that guide with own installation.
Goes to sipeoint.com/npmguide.
Gogles ‘npm tutorial run script’
Back to the michael-kuehnel.de guide.
Runs `npm --help` and reads a bit about it.
"Yes it was an error message,” Refers to it being connected to babylon.js not npm.
Reads more in the Babylon.js docs.
Copies selective import {engine} etc scene from babylonjs.
Tries to use import instead of require.
Reads throughout the npm support page.
"Need to do all these things as well.”
Runs `npm install --savedev typescript webpack tsloader webpack-cli`.
Creates a webpack.config.js and pastes the sample code there.
Also adds an tsconfig.json.
"Should I really need to include…?"
Creates HTML file index.html and pastes code from npm support page Babylon.js docs.
"This seems to be for ts, that’s not what I’m doing here.”
Finds guided learning getting set up in the documentation, scrolls through it and reads.
Opens getting started again, link 1.
Big sigh.
Clicks on the ‘first scene and model’ and getting started playground, it comes up a cube.
Scrolls more in ‘first scene and model’ section.
Gogles ‘babylon nodejs’, clicks on a local webserver guide, goes back to babylon.js.
Peruses documentation, homepage, diving deeper, api, babylon native. Clicks on learn and then documentation, getting started.
"I have already looked at this.”
"Well, they could have been a little clearer but they seem to have their own thing that they test all their examples in.” Refers to the playground.
Deletes everything in the test folder.
Creates an index.html.
Now follows guided learning create a game tutorial.
run sudo apt-install node-typescript.
"Do you need all that?”
Runs everything anyway.
Runs ‘npm run build’ but gets error that it does not resolve some dependencies with webpack, cli and more.
Checks through its terminal history.
Creates a public folder and moves index.html to it.
Clean webpack plugin seems to have problems.
Runs ‘npm run start’ and gets ‘webpack-dev-server not found’.
"Here we see that it is installed."
Installs it again anyway.
Does not seem to work.
Runs the installations of typescript and ts-loader etc.
Gets several deprecation warnings that fs.events will break and chokidar will break on nodev14.
The computer crashes and the participant drops out of zoom-call, back again two minutes later.
Gooogles ‘set up babylon.js’
Comes to the a page on tree shaking on Babylon.js docs.
Tries ‘npm run build’.
Reads more on the tree shaking page.
Goes to stack overflow.
Runs ‘npm run start’, does not work.
Deletes entire node_modules.
Copiaes webpack-dev-server again from the guide to start a game within Babylon.js docs.
Runs ‘npm run build’.
Runs ‘npm run start’.
Gets error ‘cannot find node_module webpack-cli’
Tnstalls it.
Runs ‘npm run build’.
Opens index.html in the browser, sees a sphere.
"But if I do as they write there I don’t get it to work. Only like this."

**Three.js**
Creates a new project repo.
Reads on installation npm page link 2.
Reads on getting started link 1, jumps back to link 2 quickly.
Runs ‘npm init’
Runs ‘npm install --save three’
Reads a bit on Eloquent Javascript which is linked in the Three.js docs.
Creates index.html.
Pastes HTML code from the docs.
Creates a three.js file, creates a js folder and places it there.
Copies JavaScript from the docs example and.
Scrolls down and sees a whole file with HTML and JavaScript in ‘getting started’.
"Aha..."
Clicks on ‘testing with npm’ in the menu, goes back to installation.
Copies import statements.
Reads more quickly about bundling tool on Eloquent Javascript.
Reads about CDN, how to run things locally, browser support, typescript support, faq, useful links, all menu options in the docs.
Modifies script tag in index.html to connect to js / three.js also inserts script type module
Comments out the first script tag, only has the one on module now that is not connected to three.
Pastes JavaScript where it says in the sample file "our js goes here".
Opens index.html in the browser and checks the console log, ‘three is not define, cross origin request blocked’.
Runs ‘npm install http server’.
Starts it, the command is not found.
run sudo npm install http-server -g
Runs it again the server.
Clicks on the link, nothing is visible in the browser, checks in the console log.
Same: ‘three is not defined, loading module, cross-origin request blocked’
"It was not that easy to get it started then."
Reads more under npm support in the docs.
Pastes in import alternative.
Gets syntax errors, it is not possible to declare the module at the top level. In HTML file.
Adds ‘import three from unpkg’, then it's CDN though.
Checks in browser, same error with import declarations.
Adds script type module.
Console: ‘three.scene is not a constructor’
Sighs.
Gooogles ‘three.js webpack’.
Clicks on ‘threeJS webpack es6 boilerplate’.
Checks in the forum.
Best practice webpack.
Back to installation.
Clicks through to guide on Webpack from the docs.
Reads about getting started Webpack.
Installs webpack and webpack-cli.
Renames the js folder src.
Creates main.js file there.
Changes script src = "src / three.js"
Adds import * as Three from 'three'; at the top.
Reads more in the Webpack guide.
Switches script source to main.js.
Searches for npx in the Webpack guide.
Inserts a build script in package.json.
Runs 'npm run build'.
Gets 'can’t resolve ./src'.
Renames index.js to three.js.
Runs 'npm run build'.
Changes the script tag to the dist folder.
Opens HTML file in browser, it is black, no cube is visible.
"So now it almost seems to be up and running."
Copies the cube code from the wizard.
Rebuilds and checks, still black, inspects the console, rechecks the docs.
Finds rendering the scene and copies that and the animate cube as well.
"Aha ok."
Rebuilds and the animation shows in the browser.
"I have used some API:s before, both these frameworks presupposes a lot, they assumes the user knows a lot. If you’re new it can be very confusing. Both node.js and Webpack. I don’t think they even linked to Webpack in Babylon. At least Three had a link to how it can be installed."

Choice: Babylon.js
Why: Not entirely sure, it felt like it might have more help functionality. But in the end I need to research both more to pick one.
Appendix F: Research 3D Frameworks

A list of 3D frameworks collected during the decision phase, with their exclusion criteria. They had to adhere to the following: based on JavaScript, actively maintained (have commits the last three months), used by professionals, used in both large and small productions, often mentioned in comparisons, built on top of WebGL, focused on 3D in the browser and not be work in progress, they had to be released and have been around for at least a few years.

- A-Frame: primarily focused on AR/VR
- Away3D: not actively maintained
- Clara.io: not a web framework, more like Blender or other 3D software
- CopperLicht: not often mentioned in comparisons, unclear maintain status, no official up to date open repo on GitHub
- JanusWeb: VR/XR focus, the tech team quit in 2019
- Kubity: not JavaScript
- LayaAir: not often cited in comparisons, also tutorials are in Chinese
- OSG.JS: discontinued as of April 2020.
- PlayCanvas: matches all but some features require a subscription
- Sketchfab: more a 3D assets management platform than 3D framework
- Unity: written in C#
- Verge3D: some features requires a subscription, also first stable release was in October 2019, less than a year from the start of the thesis
- Wonderland Engine: VR and AR focus, no 1.0 version as of yet

To list all references that support this information would take several pages, however they are all on one Wikipedia page as well: