

UTRECHT UNIVERSITY FOR APPLIED SCIENCES

DIGITAL COMMUNICATION & MEDIA

FACULTY FOR COMMUNICATION & JOURNALISM

---

# WebGL Product Configurators

TOWARDS UNDERSTANDING CURRENT TECHNICAL LIMITATIONS

---

*Author:*

R.W.J. PEELLEN

*Supervisor:*

Dhr. K. WINKEL

January 10, 2017



---

# Abstract

Abstract will go here



---

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1-1	Visuals . . . . .	8
1-1-1	Image Planning . . . . .	8
1-1-2	Modeling . . . . .	9
1-1-3	Lighting & Shading . . . . .	9
1-1-4	Render . . . . .	9
1-1-5	Post Production . . . . .	9
1-1-6	Compression . . . . .	10
1-2	Software . . . . .	10
1-2-1	Requirements . . . . .	10
1-2-2	CMS . . . . .	11
1-2-3	UX Development . . . . .	11
1-2-4	Back-End Development . . . . .	11
1-2-5	Front-End Development . . . . .	12
<b>2</b>	<b>Aims</b>	<b>13</b>
2-1	Development . . . . .	13
2-2	User Experience . . . . .	13
2-2-1	Size & Bandwidth . . . . .	13
2-2-2	Perceived Speed . . . . .	13
2-2-3	Compatibility . . . . .	13

<b>3</b>	<b>Methodology</b>	<b>15</b>
<b>4</b>	<b>Results</b>	<b>17</b>
<b>5</b>	<b>Discussion</b>	<b>19</b>
<b>6</b>	<b>Conslusion</b>	<b>21</b>
<b>7</b>	<b>Attachments</b>	<b>23</b>
<b>8</b>	<b>Bibliography</b>	<b>25</b>

---

# Chapter 1

---

## Introduction

Peppr is a company specialising in building photo-realistic visualisations. Late 2014, Peppr built a product configurator for SlimFitted, a company that makes tailored shirts. They wanted a platform on which their customers would be able to order customized shirts.

The customer requested 25 colors, 2 perspectives, 7 types of collars, 6 types of sleeves and 3 different plaids. This would amount to 6300 possible shirt combinations. Creating all the combinations is massive undertaking, which is why most web-based product configurators are built by splitting the final image into several configurable components. But even with the smart division of layers, Peppr still needed to render a staggering amount of images.

Apart from the images, Peppr encountered a problem that needed to be solved with regards to software. For every configurator, the end-user needs to be able to view and edit a configuration in a user-friendly way. Unfortunately, this means something different for all configurators. Combined with branding, creating a universal and more importantly, re-usable configurator is extremely difficult.

After some research, Peppr stumbled upon a technology called OpenGL which was released in March 2011 ([3]), an implementation of OpenGL technology for the web. WebGL technology renders directly from the video processor and it opens up the web to a whole new way of using actual 3d models. Being able to use the 3d models instead of the vast amount of images would make the configurators easier (less expensive) to build and maintain. Unfortunately, the actual adoption rate of WebGL has always been low because only the latest browsers would integrate the technology and there were hardware limitations on mobile phones. Anno though, the playing field has changed. With more-and-more browsers supporting this new type of technology, the timing might be perfect to bring it to the masses.

Peppr has done these projects in the past and states that the usual way of producing configurators consists of two sub-projects; the images, and the software. In the outline below follows a brief description of how the process is currently done at Peppr and some advantages that

can be made when using OpenGL. For this introduction, the following example project shall be used. Please note that the following process is the way Peppr would handle a project like this. " A chair manufacturer wants a product configurator for one of their most popular chairs. It can be delivered in 25 different colours and has 4 different subframes. Two of the subframes are fully made from steel and can be delivered in either black or plain steel, the other two have wooden elements and have four different wood colour options. "

## 1-1 Visuals

### 1-1-1 Image Planning

Product configurators possibly have to deal with an exponentially growing set of options. Peppr believes it is good practice to see if these images can be split into separate components that make the image set smaller and easier to maintain. Below is a summary of the full option set, split per item so we can start calculating how many images will be necessary.

- 25 colours -  $(\alpha)$
- 2 steel frames -  $(\beta)$
- 2 frame colours -  $(\gamma)$
- 2 wood frames -  $(\delta)$
- 4 wood colours -  $(\epsilon)$

In this case, any seat has a pick of several frame options, and several colours. But steel frames have two colours and wood frames have four colours. To calculate the full amount of options  $(x)$ , we can use the following formula:

$$\begin{aligned} f(x) &= (\alpha \cdot \beta \cdot \gamma) + (\alpha \cdot \delta \cdot \epsilon) \\ f(x) &= (25 \cdot 2 \cdot 2) + (25 \cdot 2 \cdot 4) \\ f(x) &= 300 \end{aligned}$$

Going with just 1 colour more adds 12 extra renders. While this may not seem like much, more often than not, these renders need to be build from several different files. This means an artist has to open 12 different files, and do the edit 12 times. Fortunately, in some cases, you might be able to get around it using layers. In this case, splitting up the chair into a seat and frame layer will get us the following formula:

$$f(x) = (\alpha) + (\cdot \beta \cdot \gamma) + (\cdot \delta \cdot \epsilon)$$



$$f(x) = (25) + (2 \cdot 2) + (2 \cdot 4)$$

$$f(x) = 37$$

This decreases the amount of renders massively. Unfortunately, while this may work in this case, in some cases this is not possible because the object is being obstructed in its view, making it extremely difficult to 'mask'. Next to that, creating new layers does add complexity and constraints. Adding new options / layers later on might prove extremely difficult if they were to overlap with existing layers.

Apart to difficulties in the 3d process, these layers need to be build into the software as well. Either by creating an API that serves those layers as one image (Like the Bugaboo configurator [5] ) or a front-end that can layer multiple images in a correct way. In the lather case, the user will directly notice this option as it slows things down, this should be resolved with the adoption of HTTP 2.0 and the pipelining of http-requests ([7]), although some overhead will exist.

## 1-1-2 Modeling

Step two in the chain is the modelling proces whereby a virtual model is being created, either from scratch, or by 3d scanning real-life objects. If the choice to go for layers in the renders have been made, the modeller make sure he keeps this in mind.

## 1-1-3 Lighting & Shading

In the lighting department, the model is being put into a suitable environment (mostly studio like setups), where it is lit and shaded (process of creating a life-like material) to perfection. The lighter must make sure that when there are layers involved, there is no unwanted shadow casting because when hiding certain layers for rendering.

## 1-1-4 Render

This is where things get together and the calculation from 3d model to actual image start. Depending on the configuration and setup, one renders the entire sequence, if  $< 100$  renders and only one point of view for instance, one might swap out the model at certain frames. If one needs to render 360's, rendering one file at at time is better suitable.

## 1-1-5 Post Production

Every 3d model needs a bit of post production to make it look better. Also, if the model was split out into different layers but these were rendered as masks (an option to overcome overlapping images), this is where they would be split out into the different layers.

## 1-1-6 Compression

Using images of the web, especially on @2x or @3x resolution devices, mostly phones and tablets which get their data through mobile networks, compression is extremely important. With current mobile data speeds and capped contracts, laying a 50mb burden upon the user when opening a site is not a good idea.

## 1-2 Software

### 1-2-1 Requirements

Peppr generally starts out a software project by working out both functional and technical requirements.

#### Functional Requirements

Functional requirements are used to map out the required functionality of the application. In an agile [9] working environment, these would be referred to as 'user stories', but Peppr prefers the term requirements. They do generally use the same notation.

*"As a <type of user>, I want <some goal>, so that <some reason>"*

In the context of the configurator specified above, a requirement could be:

*"As a **User**, I want **to be able to save my configuration**, so that **I can later continue were I left off with my configuration**"*

#### Technical Requirements

Contrary to popular belief, technical requirements are not the technical implementation details of the functional requirements, but they are the requirements the system itself needs to fulfil with regards to things like performance, availability and security. Some simply put these in a list [10], but Peppr handles the same format as the user stories.

*"As a <type of **system**>, I want <some goal>, so that <some reason>"*

This way, Peppr is able to differentiate between different systems like an API or a Front-End web application. Below follows an example of a technical requirement where we want to limit

the API's response time ([11])

*“As an **API**, I want **to respond within 300ms**, so that **the user feels in control of the application at all times**”*

## 1-2-2 CMS

A content management system has basic functionality built in (user management, file handling, basic front-end), but it does require to work in the way that system is meant to be worked in. The other option, going with a from the ground up written system, will be much leaner and quicker when deployed, but will not be as mature as a popular CMS, which, if the system is not built properly, might result in a buggy experience for the end user. In Peppr's case, a CMS called Magento was used for the implementation. In case of a fully custom application, a back-end needs to be developed as well.

## 1-2-3 UX Development

While this term gets thrown around a lot, the UX (User Experience) covers not only the UI of the application, but is a broader term to describe how users undergo the experience of customising a product. How they learn to use the product configurator, and how the application can help the user in the best possible way to do so. Don Norman ([12]) says:

*“The first requirement for an exemplary user experience is to meet the exact needs of the customer, without fuss or bother. Next comes simplicity and elegance that produce products that are a joy to own, a joy to use. True user experience goes far beyond giving customers what they say they want, or providing checklist features.”*

In other words, UX is not only helping the user to meet his or her goal, but to go beyond that, and make the experience joyful.

## 1-2-4 Back-End Development

According to Techopedia ([13]), a back-end developer's primary task is to develop and maintain a logical or computational back-end for a website. Typically focussing in on C++, C#, Java, or another high-level programming language. Another Definition ([14]) states back-ends are mostly developed using either Ruby or Python. In reality however, any programming languages that can interact with a databases and serve that data can act as a back-end. For instance, the 'Dollar Shave Club' uses 6 different languages across their back-end infrastructure ([15]), while Uber uses a completely different stack ([16]). Peppr generally uses a microservices architecture for their apps ([17]). This pattern allows the use of multiple applications, so Peppr can build each application using a specific language and with a specific target in mind. For example,

Pepr can opt to handle real-time data (like a real-time help chat in a configurator) using a service called Firebase ([19]), which easily implements directly into front-ends, while handling other tasks that need high concurrency and scalability (like stacking the layers in a back-end to a complete image and delivering them to the front-end) with an Elixir application ([18])

## 1-2-5 Front-End Development

According to Wikipedia, a Front-End Developer ([20]) produces the client-side interface using HTML, CSS and Javascript. However, anno 2016, Front-End Development has become quite a bit broader ([21]). For relatively complex projects like configurators, building a front-end can be a daunting task. More often than not, you will want to serve the changes to the user instantly, without a page refresh. Resulting in an asynchronous application. Building an asynchronous application can be done in numerous ways. By far the most convenient way is by building Single-Page-Application (SPA), this might give the user a desktop-like experience ([22]). As stated on Wikipedia, there are some caveats to this, like search engine optimisation, speed and browser history. Fortunately (and unfortunately), there are frameworks to help build a SPA. Anno 2016, the field of Front-End Development is clouded with frameworks to help build these types of applications. MeteorJS, React, Angular, Vue and many more (). Pepr develops most of their applications using Angular. They found that the code is easiest to maintain and the separation of concerns using a MVC type setup works well with Angular.

---

## Chapter 2

---

### Aims

The main aim of this thesis is to find whether or not using OpenGL / WebGL is a viable option for building product configurators. This objective can be split up into several sections, the main two being the development and experience of the application. Below is a brief summary of each section, ending with an aim for that specific property.

#### 2-1 Development

Developing a product configurator is by any means no easy feat. Building it using 3d technology may possibly be even harder. The following three questions would ideally be answered in this thesis.

1. How difficult is it to build a product configurator whereby the user navigates using a 3d model instead of a set of images?
2. Is the end-result flexible enough to be re-used for other clients?
3. How easy is it so maintain the application compared to existing configurators?

#### 2-2 User Experience

##### 2-2-1 Size & Bandwidth

##### 2-2-2 Perceived Speed

##### 2-2-3 Compatibility



---

## Chapter 3

---

# Methodology





---

## Chapter 4

---

# Results

Here I'll state the findings on the research per domain objectives, with a research conclusion in the end. This will contain a go / no go moment, in which shall be decided if the building section shall be finished within this research.



---

## Chapter 5

---

# Discussion



---

## Chapter 6

---

# Conclusion



---

## Chapter 7

---

# **Attachments**





# Bibliography

- [1] Andreas Anyuru, *Professional WebGL Programming: Developing 3D Graphics for the Web*, John Wiley and Sons Ltd, West Sussex, 2012.
- [2] Alexis Deveria, *Can I use" provides up-to-date browser support tables for support of front-end web technologies on desktop and mobile web browsers.*, [caniuse.com](http://caniuse.com)
- [3] *This is the official OpenGL Website, where they've stated a lot of information on the history of OpenGL*, <https://www.opengl.org/about/>
- [4] Sparsh Mittal and Jeffrey S. Vetter *A Survey of CPU-GPU Heterogeneous Computing Techniques*, Oak Ridge National Laboratory and Georgia Tech, 2015.
- [5] Bugaboo *Bugaboo configurator*, Bugaboo website, Oct. 2015. <http://blog.irayrender.com/post/112595883086/bugaboo-configurator-making-of>
- [6] 3Dimerce *Bugaboo configurator Making of*, iRender blog, Oct. 2015. [https://www.bugaboo.com/NL/nl\\_NL/strollers/create?sId=CAM3STD](https://www.bugaboo.com/NL/nl_NL/strollers/create?sId=CAM3STD)
- [7] Wikipedia *HTTP/2*, Oct. 2015. [https://en.wikipedia.org/wiki/HTTP/2#Differences\\_from\\_HTTP\\_1.1](https://en.wikipedia.org/wiki/HTTP/2#Differences_from_HTTP_1.1)
- [8] Jesse James Garrett *Ajax: A New Approach to Web Applications*, Feb. 2005. [https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax\\_adaptive\\_path.pdf](https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf)
- [9] Scott W. Ambler *2. Initial User Stories (Formal)* 2003 - 2014 <http://www.agilemodeling.com/artifacts/userStory.htm>
- [10] Scott W. Ambler *Figure 1. Technical requirements (summary form)*. 2003 - 2014 <http://agilemodeling.com/artifacts/technicalRequirement.htm>

- [11] Jakob Nielsen *Response Times: The 3 Important Limits* January, 1993 <https://www.nngroup.com/articles/response-times-3-important-limits/>
- [12] Don Norman & Jakob Nielsen *The Definition of User Experience* July, 2016 <https://www.nngroup.com/articles/definition-user-experience/>
- [13] Techopedia *Back-End Developer* <https://www.techopedia.com/definition/29568/back-end-developer>
- [14] Bloc *A Comparison of Frontend and Backend Web Development* Dec. 2016 <https://blog.bloc.io/frontend-vs-backend-web-development/>
- [15] High Scalability *The Dollar Shave Club Architecture Unilever Bought For \$1 Billion* Sep. 2016 <http://highscalability.com/blog/2016/9/13/the-dollar-shave-club-architecture-unilever-bought-for-1-bil.html>
- [16] High Scalability *How Uber Manages A Million Writes Per Second Using Mesos And Cassandra Across Multiple Datacenters* Sep. 2016 <http://highscalability.com/blog/2016/9/28/how-uber-manages-a-million-writes-per-second-using-mesos-and.html>
- [17] microservices.io *Pattern: Microservices Architecture* Dec. 2016 <http://microservices.io/patterns/microservices.html>
- [18] Elixir Programming Language *Elixir Programming Language* <http://elixir-lang.org/>
- [19] Firebase *Firebase* <https://firebase.google.com/>
- [20] Wikipedia *Front-End Web Development* [https://en.wikipedia.org/wiki/Front-end\\_web\\_development](https://en.wikipedia.org/wiki/Front-end_web_development)
- [21] Hackernoon *How it feels to learn javascript in 2016* <https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f#.9tsvqyql0>
- [22] Wikipedia *Single-Page-Applications* [https://en.wikipedia.org/wiki/Single-page\\_application](https://en.wikipedia.org/wiki/Single-page_application)
- [23] NoeticForce *Best javascript frameworks for single page web applications* <http://noeticforce.com/best-Javascript-frameworks-for-single-page-modern-web-applications>
- [24] Angular *Angular* <https://angular.io/>