Utrecht University for Applied Sciences

Digital Communication & Media

Faculty for Communication & Journalism

WebGL Product Configurators

Towards Understanding Current Technical Limitations

Author:
R.W.J. Peelen

Supervisor:
Dhr. K. Winkel

January 10, 2017

Abstract

Abstract will go here

Contents

Chapter 1

Introduction

Peppr is a company specialising in building photo-realistic visualisations. In 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 customised 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. A reason why most web-based product configurators are splitting the final images into several layers. But even with a smart division, Peppr still needed to render a staggering amount of images.

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

In their research, Peppr stumbled upon a technology called OpenGL. The first version released March 2011 (**(author?)** [3]). WebGL technology renders from the video processor and opens up the web to a whole new way of using 3d models. Using 3d models instead of images could make configurators easier to build and maintain. Unfortunately, the adoption rate of WebGL has always been low. Only the latest browser versions would have the technology. Next to that, there are hardware limitations on mobile phones. Anno 2016 though, the playing field has changed. With more browsers supporting the technology, timing might be perfect to start using it.

Peppr has done these projects and states that the usual way of business consist of two parts. One being the images, the other, the software. In the outline below follows a brief description of how the process is currently done.

For this thesis, we will use the following example. Please note that this process is the way Peppr handles projects like this. This is not necessarily the best way to handle a project like this one.

“ A chair manufacturer wants a product configurator for one of their most popular chairs. It consists 25 different colours and has 4 different subframes. Two of the subframes are steel and can be either black or plain. The other two have wooden elements and have four wood colour options. ”

1-1 Visuals

1-1-1 Image Planning

Product configurators may have to deal with an exponentially growing set of options. Peppr always checks if these image sets can be split into separate components. This may make the set smaller and easier to maintain. Below is a summary of the option set, split per item so we can start calculating how many images will be necessary.

- 25 colours - **(α)**
- 2 steel frames - **(β)**
- 2 frame colours - **(γ)**
- 2 wood frames - **(δ)**
- 4 wood colours - **(ϵ)**

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:

$$f(x) = (\alpha \cdot \beta \cdot \gamma) + (\alpha \cdot \delta \cdot \epsilon)$$
$$f(x) = (25 \cdot 2 \cdot 2) + (25 \cdot 2 \cdot 4)$$
$$f(x) = 300$$

Going with just 1 colour more adds 12 extra renders. While this does not seem like much, these renders are build from several different files. This means an artist has to open 12 different files, and do the edit 12 times. 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. Unfortunately, while this works here, in some cases this is not possible. Sometimes, one of the objects is both in front and behind other objects. This makes it difficult to 'mask'. Also, creating new layers does add complexity and constraints. Adding new options and layers might prove difficult when they 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 the layers as one image (**(author?)** [5]), or by stacking the images client-side. The user will notice this option though, as it slows things down. This is fixed with the adoption of HTTP 2.0. This pipelines the http-requests, reducing latency (**(author?)** [7]).

1-1-2 Modeling

Step two is the modelling proces where an artist makes a virtual model. Either from scratch, or by 3d scanning real-life objects. If the choice to go for layers in the renders has been made, the modeller make sure he keeps this in mind.

1-1-3 Lighting & Shading

Once in the lighting department, the model is placed into a suitable environment. Most of the time a studio setup. There it is lit and shaded (the process of creating a life-like material) to perfection. The artist makes sure that when there are layers involved, the shadows do not overlap the layers.

1-1-4 Render

This is where things get together and the calculation from 3d model to actual image start. The actual rendering proces differs. Sometimes it is easier to render out the whole sequence with every configuration. Other times, it is easier to render just one point-of-view at a time.

1-1-5 Post Production

Every 3d model needs a bit of post production to make it look better. If the model has layers rendered as masks (an option to overcome overlapping images). This is the step where they would split them into the different layers.

1-1-6 Compression

Using images on the web, especially on bigger resolution displays, small images are important. Current mobile data speeds and capped contracts are difficult. So laying a 50mb burden upon the user when opening a site is not a good idea. This is were compression comes in. Every images gets compressed for a fast download, while remaining a good quality.

1-2 Software

1-2-1 Requirements

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

Functional Requirements

Functional requirements help map out required functionality of an application. In an agile (**(author?)** [9]) working environment, these are 'user stories'. 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, these requirements are not the implementation details of the functional requirements. They are the requirements the system itself. The technical requirements handle things like performance, availability and security. Some put these in a list **(author?)** [10], but Peppr uses the same format as the user stories.

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

Peppr uses this to differentiate between different types of systems. An example of a technical requirement can be found below. This specific one has to do with limiting an API's response time (**(author?)** [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 (like user management, file handling). It does however, require the developer to work in the way the system intended. The other option is to go with a system that is written from the ground up. It will be much leaner and quicker when deployed, but will not be as mature as a popular CMS. So if the system is not built right, it may result in a buggy experience for the end user. In Peppr's case, a CMS called Magento was used during implementation. One thing to keep in mind; In case of a 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. It is a broader term to describe how users undergo the experience of customising a product. Itis how they learn to use the product configurator. How the application can help the user in the best possible way to do so. Don Normal (**(author?)** [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. It is to go beyond that, and make the experience joyful.

1-2-4 Back-End Development

Techopedia (**(author?)** [13]) states that a back-end developer's task is to develop and maintain a logical or computational back-end for a website. Focussing on C++, C#, Java, or another high-level programming language.Another Definition (**(author?)** [14]) states back-ends are mostly developed using either Ruby or Python. In reality, any programming languages that can interact with a databases can act as a back-end. For instance, the 'Dollar Shave Club' uses 6 languages in their infrastructure (**(author?)** [15]). 'Uber', uses a completely different stack (**(author?)** [16]). Peppr generally uses a microservices architecture for their apps (**(author?)** [17]). This pattern allows the use of different applications. So, Peppr can build each application using a specific language and with a specific target in mind. So imagine a configurator needs to stack images but also needs a realtime chatbot. Peppr will be able to handle the high concurrency using something like Elixr. The realtime chatbot will then made with Firebase, a realtime service (**(author?)** [19]).

1-2-5 Front-End Development

According to Wikipedia, a Front-End Developer (**(author?)** [20]) produces the client-side interface. Wikipedia states that they use HTML, CSS and Javascript. However, anno 2016, Front-End Development has become quite a bit broader (**(author?)** [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 many ways. The most convenient way is to build a Single-Page-Application (SPA). This gives the user a desktop-like experience (**(author?)** [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 an 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 (**(author?)** [23]). Peppr 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 goal of this thesis is to find whether using WebGL is a viable option when building configurators. This objective has two sections, the development and the user 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 be even harder. The following three questions should be answered in this thesis.

1. Is there a way to compare development costs and if so, what is the difference compared to existing configurators?
2. How difficult is it to build a product configurator whereby the user navigates using a 3d model instead of a set of images?
3. Is the end-result flexible enough to be re-used for other clients?
4. How easy is it so maintain the application compared to existing configurators?
5. How easy is it to add new options to the configurator compared to the existing configurators?

2-2 User Experience

Apart from the development cycle, the user experience is a huge part of whether using these types of configurators is viable or not. Even if the development is way cheaper, if users cannot use the application, problems will arise and the old method may prove to be a better option.

1. How is the bandwidth usage compared to existing configurators?
2. How is the speed compared to existing configurators?
3. Is the technology compatible with the users browsing preferences?

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

Conslusion

Chapter 7

Attachments

Chapter 8

Bibliography

[1] Andreas Anyuru, *Professional WebGL Programming: Developing 3D Graphics for the Web*, John Wiley and Sons Ltd, West Sussex, 2012

[2] *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 2017

[3] 2017 *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* http://blog.irayrender.com/post/112595883086/bugaboo-configurator-making-of 2015

[6] 3Dimerce, iRender blog, *Bugaboo configurator Making of* https://www.bugaboo.com/NL/nl_NL/strollers/create?sId=CAM3STD 2015

[7] Wikipedia *HTTP/2* https://en.wikipedia.org/wiki/HTTP/2#Differences_from_HTTP_1.1 2015

[8] Jesse James Garrett *Ajax: A New Approach to Web Applications* https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf 2005

[9] Scott W. Ambler *2. Initial User Stories (Formal)* http://www.agilemodeling.com/artifacts/userStory.htm 2014

[10] Scott W. Ambler *Figure 1. Technical requirements (summary form).* http://agilemodeling.com/artifacts/technicalRequirement.htm 2014

[11] Jakob Nielsen *Response Times: The 3 Important Limits* https://www.nngroup.com/articles/response-times-3-important-limits/ 1993

[12] Don Norman & Jakob Nielsen *The Definition of User Experience* https://www.nngroup.com/articles/definition-user-experience/ 2016

[13] Techopedia *Back-End Developer* https://www.techopedia.com/definition/29568/back-end-developer 2017

[14] Bloc *A Comparison of Frontend and Backend Web Development* https://blog.bloc.io/frontend-vs-backend-web-development/ 2016

[15] High Scalability *The Dollar Shave Club Architecture Unilever Bought For $1 Billion* http://highscalability.com/blog/2016/9/13/the-dollar-shave-club-architecture-unilever-bought-for-1-bil.html 2016

[16] High Scalability *How Uber Manages A Million Writes Per Second Using Mesos And Cassandra Across Multiple Datacenters* http://highscalability.com/blog/2016/9/28/how-uber-manages-a-million-writes-per-second-using-mesos-and.html 2016

[17] microservices.io *Pattern: Microservices Architecture* http://microservices.io/patterns/microservices.html 2016

[18] Elixer Programming Language *Elixer Programming Language* http://elixir-lang.org/ 2016

[19] Firebase *Firebase* https://firebase.google.com/ 2016

[20] Wikipedia *Front-End Web Development* https://en.wikipedia.org/wiki/Front-end_web_development 2016

[21] Hackernoon *How it feels to learn javascript in 2016* https://hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f#.9tsvqyql0 2016

[22] Wikipedia *Single-Page-Applications* https://en.wikipedia.org/wiki/Single-page_application 2016

[23] NoeticForce *Best javascript frameworks for single page web applications* http://noeticforce.com/best-Javascript-frameworks-for-single-page-modern-web-applications
2016

[24] Angular *Angular* https://angular.io/ 2016