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

February 20, 2017

Abstract

Abstract will go here
[**?**]

Contents

Chapter 1

Introduction

Peppr is a company that is specialised in building visualisations and animations. They were founded in 2007 and from the start, focussed on creating high-end, photorealistic imagery. They try to stay up-to-date with new technologies and are always up for a challenge. The company consists of 4 people that bind themselves using a partnership agreement.

In 2014, Peppr was asked to build a product configurator for one of their customers (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 the configurator had 25 colors, 2 perspectives, 7 types of collars, 6 types of sleeves and 3 different plaids. This would amount to 6300 possible shirt combinations. Even with a smart division in layers, making all those combinations is a massive undertaking.

Apart from the actual images needed, the end-user needs to be able to view and edit a configuration in a simple way. This means a piece of software had to be built to show the configuration to the user. Unfortunately, every customer wants their own branding and with the nature of layered images (and the technical difficulties they create), creating a universal and re-usable configurator is difficult.

Peppr feels there must be a better way to serve their customers, so they will end up with a cheaper, more maintainable configurator. Current technologies allow the viewing 3d models directly in the browser. In 2015, Peppr got a second request to built a configurator and started researching their possibilities. When they did, they stumbled upon a piece of technology called OpenGL (/ WebGL) and a javascript based binding to use it on the web called "three.js". This type of technology means that a 3d model can be shown directly in browser (hardware accelerated) and it could be the next big thing when it comes to configurators.

The second clients order fell through, but Peppr felt the need to research this further and wants to know wether or not a WebGL product configurator that directly shows 3d models instead of images is a viable alternative. Both in development, as in user experience.

Chapter 2

Theoretical Framework

2-1 OpenGL & WebGL

Peppr stated the technology that possibly holds the answer is OpenGL and its web counterpart; WebGL. The first version of OpenGL was released March 2011 ([OpenGL, 2016]). It is a software interface to allow programmers to interact with graphics hardware ([Mark Segal, 2010]). It is essentially an API acting as a middle man between software and hardware. In practice, this means that programmers will need only little (or no) hardware knowledge wile still being able to tap into the hardwares calculation capacity. Normally, starting up an OpenGL application window, which ties into the graphics hardwares framebuffer ([fra, 2016]). That application window then holds a GL context, and the OpenGL commands can be used to modify and interact with the contents.

To use OpenGL on the web, one needs an API that interacts with OpenGL. This can be done server-side, with the result of the graphics hardware framebuffer being sent to the user ([Brian Paul & Southard, 2008]). This means that any language parse-able on a server that can send data out (and has an OpenGL enabled graphics card), can be used. Even though this article stems from 2008, the problems stated with latency still exist today, even while considering their suggested and tested optimisations. Next to that, servers that have graphics cards are relatively expensive. Another option is client-side rendering. With the increase in speed for customer technology and the rise of smartphone usage, the amount of devices that has a graphics card and can use this technology increases. Unfortunately, it is not possible for (most) webbrowsers to directly communicate with the OpenGL framebuffer. Fortunately, there is a solution; WebGL.

"WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES 2.0, exposed through the HTML5 Canvas element as Document Object Model interfaces." [site, 2016]

WebGL technology renders from the graphics hardware (just like OpenGL). The key difference is that the API is exposed to an HTML5 canvas element. This means the framebuffer is not opened into an application window, but into an HTML5 canvas element. While this solves the issue of running GL content client-side, GLSL (the language used to interact with WebGL & OpenGL) does not make it into the Stack Overflows Annual Survey ([sta, 2016]). This does imply it would be hard (and expensive) to find competent developers that can built an application using this technology. Fortunately, Peppr found a library for this. It uses the most popular language in the survey (Javascript). "Three.js", as the library is called, filled the last gap in client-side OpenGL / WebGL content. The question remains, while javascript is adopted in most modern browsers ([jav, 2016]), will it work in the by Peppr required context.

Research subquestions:

1. What is the development cost difference both in time and money? *While the technology uses a Javascript wrapper, it might not mean that any javascript developer will know how to work with it. Next to that, for a site or Vanilla Javascript application, development cycles and workflows are known. It might mean that reinventing the wheel for these sort of applications because of the use of WebGL might lead to more development time.*

2. How easy is it to maintain the application compared to existing configurators? *For current web technology, a lot has been done to make it easy to maintain (package managers, dedicated servers), but going with a proprietary system that uses OpenGL via WebGL via a Javascript wrapper, might mean that some of these existing solutions will not work. This might mean that the application will be very hard to maintain*

2-2 Technology Usage

(Bij de probleemanalyse wordt gebruik gemaakt van relevante theorieën en werkmodellen.) Technology Acceptance Model Hypecycle

2-3 Workflows

To properly determine needed requirements and possible research angles, and in-depth review of the current workflow and new workflow (as suggested by Peppr) will be stated below. Peppr has done these projects and states that the usual way of building consist of two main 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 an example to show the differences. Please note that this process is the way Peppr handles projects like this, and, while they did try to optimise it, they in no means imply this is the perfect way to tackle such a project.

“ 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. ”

2-3-1 Graphics

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' and makes proper reflections tricky. 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 ([bug, 2016]), 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 ([Wikipedia, 2015]).

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.

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.

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.

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.

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.

2-3-2 Software

Functional Requirements

Functional requirements help map out required functionality of an application. In an agile ([Ambler, 2014a]) 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 [Ambler, 2014b], 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 ([Nielsen, 1993]).

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

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.

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 ([Nielsen, 2016]) 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.

Back-End Development

Techopedia ([Techopedia, 2016]) 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 ([Techopedia, 2016]) 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 ([back end, 2016]). 'Uber', uses a completely different stack ([Scalability, 2016]). Peppr generally uses a microservices architecture for their apps ([microservices.io, 2016]). 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 ([eli, 2016]) . The realtime chatbot will then made with Firebase, a realtime service ([fir, 2016]).

Front-End Development

According to Wikipedia, a Front-End Developer ([Wikipedia, 2016a]) 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 ([Hackernoon, 2016]). 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 ([Wikipedia, 2016b]). 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 ([NoeticForce, 2016]). 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 3

Aims

3-1 Introduction

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.

3-2 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. What is the development cost difference both in time and money?
2. How easy is it to maintain the application compared to existing configurators?
3. Is the end-result flexible enough to be re-used for other clients?
4. How easy is it to add new options to the configurator compared to the existing configurators?

3-3 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. Is the technology compatible with the users browsing preferences?
2. Is the technology small enough?
3. Is the technology quick enough?

Chapter 4

Methodology

4-1 Introduction

4-2 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. What is the cost differences for development?
2. How flexible is the end result compared to previous generation configurators
3. How easy is it so maintain the application compared to existing configurators?
4. How easy is it to add new options to the configurator compared to the existing configurators?

4-3 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. Is the technology compatible with the users browsing preferences?
2. Is the technology small enough?
3. Is the technology quick enough?

Chapter 5

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 6

Discussion

Chapter 7

Conclusion

Chapter 8

Attachments

Chapter 9

Bibliography

[bug, 2016] (2016). Bugaboo website.

[jav, 2016] (2016). Comparison of webbrowsers.

[sta, 2016] (2016). Developer survey results.

[eli, 2016] (2016). Elixer programming language.

[fir, 2016] (2016). Firebase.

[fra, 2016] (2016). Framebuffer.

[Ambler, 2014a] Ambler, S. W. (2003 - 2014a). 2. initial user stories (formal).

[Ambler, 2014b] Ambler, S. W. (2003 - 2014b). Figure 1. technical requirements (summary form).

[back end, 2016] back end, D. S. C. (2016). The dollar shave club architecture unilever bought for $1 billion.

[Brian Paul & Southard, 2008] Brian Paul, Member, I.-S. A. M. I. E. W. B. M. I. E. B. R. C. J. D. K. L. J. O. & Southard, D. (2008). Chromium renderserver: Scalable and open remote rendering infrastructure.

[Hackernoon, 2016] Hackernoon (2016). How it feels to learn javascript in 2016.

[Mark Segal, 2010] Mark Segal, K. A. (March 2010). The openglr graphics system: A specification (version 4.0 (core profile).

[microservices.io, 2016] microservices.io (2016). Pattern: Microservices architecture.

[Nielsen, 2016] Nielsen, D. N. . J. (2016). The definition of user experience.

[Nielsen, 1993] Nielsen, J. (1993). Response times: The 3 important limits.

[NoeticForce, 2016] NoeticForce (2016). Best javascript frameworks for single page web applications.

[OpenGL, 2016] OpenGL (2016). Opengl website.

[Scalability, 2016] Scalability, H. (2016). How uber manages a million writes per second using mesos and cassandra across multiple datacenters.

[site, 2016] site, W. (2016). The webgl website.

[Techopedia, 2016] Techopedia (2016). Back-end developer.

[Wikipedia, 2015] Wikipedia (2015). Http/2.

[Wikipedia, 2016a] Wikipedia (2016a). Front-end web development.

[Wikipedia, 2016b] Wikipedia (2016b). Single-page-applications.