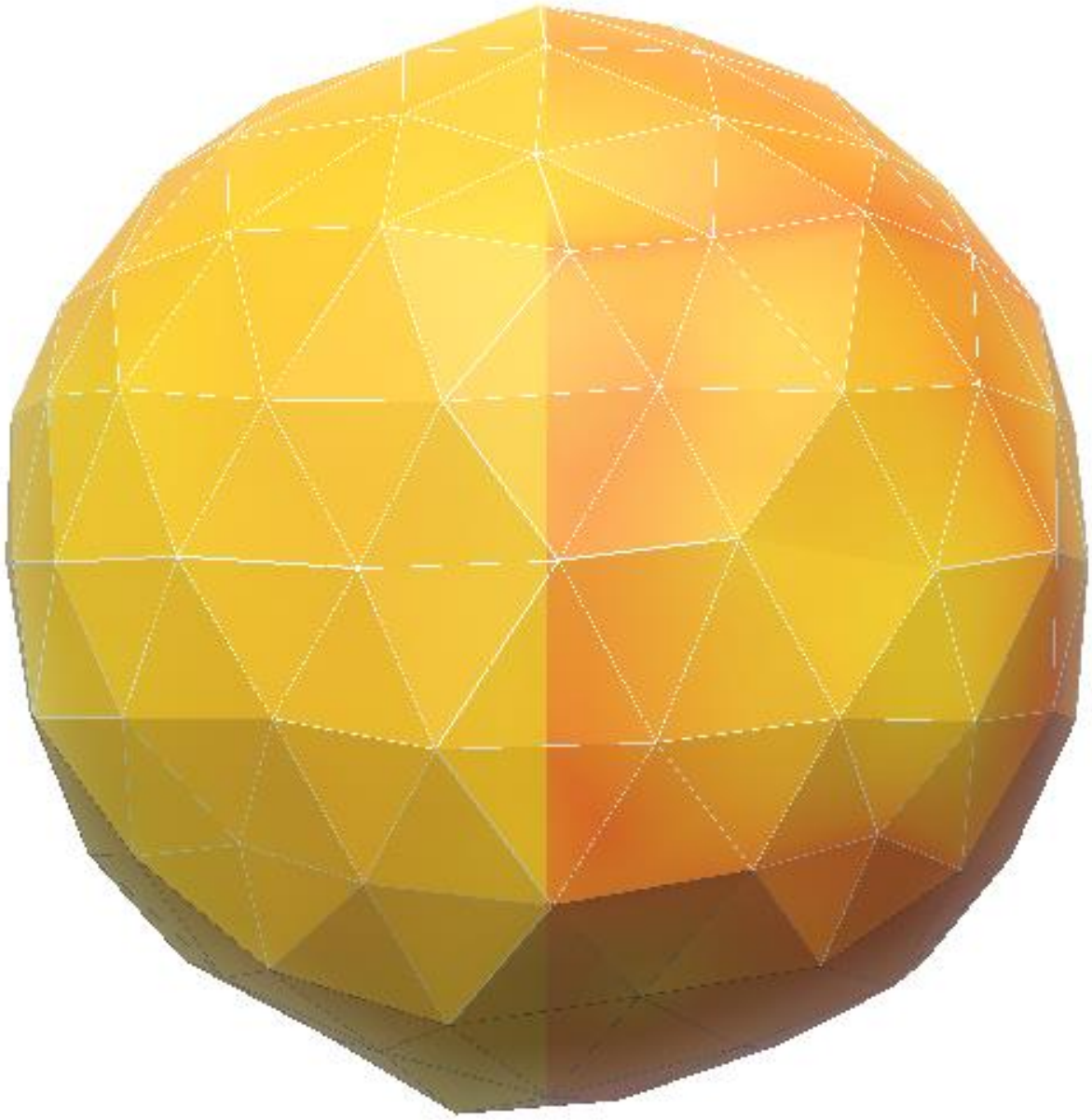
2016



[VERTEX VS PIXEL SHADING COMPARISON]

When a per vertex shaded mesh, has advantages over a per pixel shaded mesh in a WebGL application?

Martin Niehoff (326516), 17.04.2016
For the Game Creation and Producing at the Saxion University of Applied Science

Vertex Vs Pixel Shading Comparison

When a per vertex shaded mesh, has advantages over a per pixel shaded mesh in a WebGL application?

Written by

Martin Niehoff (326516)

info@martin-niehoff.de

As specialization topic in the Game Creation and Producing Course at the Saxion University of Applied Science

Supervising Teachers

Lukas Malec & Taco van Loon

17.04.2016

Summary

Web based applications are a rapidly growing market in the gaming and interactive media industry in general, since the establishment of WebGL as standard. However, as these applications should run smoothly across as many devices as possible, it's particular important to optimize them. As this optimization can greatly influence the success of an application, it's interesting to see,

When a per vertex shaded mesh, has advantages over a per pixel shaded mesh in a WebGL application?

Therefore, this paper is examining the differences between per vertex and per pixel based shading, to find the limitations and possible use cases of both technologies by the use of a practically oriented prototype, build in the Playcanvas WebGL engine. Furthermore, this paper evaluates the influence that asset sizes have on the web, to underlie the impact of their optimization, as well as the usability of both techniques regarding their creation expenses.

The gathered data shows that per vertex shaded meshes are generally taking up less VRAM, while remaining smaller than their pixel shaded fellows, as long as their vertex count stays identical. This arises from the fact, that vertex color is requiring less data per vertex, than the texture coordinates of a pixel shaded mesh. However, if additional vertices are required to accomplish the desired color detail, the advantage of the smaller asset sizes shifts to the per pixel shaded mesh. As a pixel of a texture is requiring just a fraction of the data, that a vertex with color information is requiring. Nevertheless, remains the advantage of the low VRAM consummation of per vertex shaded meshes, even if their file sizes are bigger.

Therefore, per vertex shaded meshes should be preferred in situations, where it's possible to achieve the desired color detail without the addition of too many additional vertices, or if low VRAM usage is an important factor. In addition, the build of the prototype showed, that the creation of meshes with vertex color have some advantages in a production environment, as the lack of need to unwrap meshes. Although, its usability stays a case to case decision, based on the used art style and personal preference due to its own limitations and advantages.

Preface

First off I want to thank my colleagues at C4real, for the experience, which I gained there during my internship. As I probably wouldn't be that interested in web based applications like I am at the moment without them. Furthermore, I want to thank the great team behind the Playcanvas engine, as they are building a great tool, without that I probably couldn't make the prototype for this paper in the extend it is now. Of course not to forget, my teachers who guided me throughput the creation of this report with some useful input.

Table of Content

Introduction

With the current trend to allow users to watch, play or interact with media content directly within their browsers, WebGL is a rapidly growing technology in the gaming and interactive media industry in general, as its not requiring any additional plug-in, as it was required in the past (Unity Technologies, 2015).

Content that is accessible in the browser is a giant target market, as not everybody has the knowledge, or the rights to install additional drivers, or software on their devices. But nowadays every device, no matter if it's a powerful desktop pc, or a Smartphone has an integrated browser. And most of these devices are supporting WebGL already out of the box (Deveria, 2016).

In the past users had to install Adobe Flash, to watch videos, or play simple flash games in their browser. The same was the case for game engines like Unity, which required a custom plug-in to run. By now this isn't really necessary anymore, as native support for HTML5 and WebGL is replacing these third party plug-ins. Nevertheless, what has not changed is the requirement to optimize applications, which should run smoothly on a variety of target devices and platforms.

With these requirements in mind, this report is going to compare two texturing techniques with each other, to find out in which use-cases one technique could be preferred over the other. More specifically, this report is comparing per vertex vs. pixel shaded meshes for the use in an WebGL application.

As usual, there is no perfect, or the only way to solve a problem, but this report is trying to compare both techniques based on their asset sizes, VRAM consummation and creation expenses in a variety of different use-cases, to find a nice usability guideline.

> *When a per vertex shaded mesh, has advantages over a per pixel shaded mesh in a WebGL application?*

Reason

I was doing the internship for my study in the interactive department of visual 3D Design company and worked on a few web based projects during that time. One thing, that I noticed while developing these interactive applications in WebGL, were the different requirements between native applications and those who run in your browser.

As WebGL isn't reaching the same performance as native applications yet (Unity Technologies, 2014), it requires some additional optimization for developers. Further optimization is required if the applications should run smoothly across all devices, due to the compared to a pc, low performance of current Smartphone's.

But the run time-performance isn't the only thing, that is important for the development of WebGL applications. Due to the fact that a WebGL application is, like its name already suggests running in the web, it usually has to be downloaded by the user. Today's PCs usually have at least hundreds of GB´s, or even a few terabyte of space on their drives. That's why nobody really complains, if they have to download 50gb of data once, before they can play a game. Certainly this is a different story, if users are in their browser, or even on their mobile phones.

In this case applications have to start quickly, like the websites the users are used to. Because users expect to get quick, if not even instant feedback and appearing content, if they type an URL into their browsers. Studies have shown, that websites which take more than one second to load are negatively affecting the user experience (Google, 2015). Although it's not always possible to keep applications, as small as normal websites, it's admittedly more important for the development of a Web based application to think twice about the used asset sizes.

With these requirements in mind, I was interested, if I could somehow optimize my assets further, then just packing their textures together as I was used to. After I read some stuff about per vertex shading, I was interested, if this technique could be a useful alternative for some WebGL projects. That's why I want to compare per vertex vs. pixel shaded meshes for the use in an WebGL application in this report.

Preliminary Problem statement

Therefore the following preliminary problem statement was built:

> When has a per vertex shaded mesh advantages over a per pixel shaded mesh, or vise versa?

Theoretical framework

The theoretical framework is used to build a foundation for the further research done throughout this report.

Definitions

Before getting started, lets clarify some definitions, which are going to be used in this paper, to avoid misinterpretation.

What means per vertex shading?

Per vertex shading refers to the method of texturing a model, via color information, which are stored in the vertices of the mesh.

This means that the RGB, or RGBA values, are stored like the UV-coordinates, or normal's in a per-vertex basis, so that the color information is independent of any texture coordinates, or texture map. But this also means, that a model can only have exactly one color for each vertex.

Although some model formats and engines support to store multiple colors, by adding extra vertices, which are only used to store the additional color information.

(For further details have a look at the Technique Comparison "Vertex Color")

What means per pixel shading?

Per pixel shading refers to a texturing method, were the color information for the mesh are stored in a separate image file (texture).

This means that the RGB, or RGBA values, are stored in a separate image file, which is wrapped onto the model by the shader, based on texture coordinates (UVs), that are stored on a per vertex basis in the mesh (Polycount, Texture Coordinates - polycount, 2016).

WebGL

WebGL (Web Graphics Library) is a cross-platform, royalty-free JavaScript based API for rendering interactive 2D and 3D content native within any compatible browser. It allows browsers to display interactive applications with physics, image processing and effects in a canvas element as part of a web page. WebGL is based on OpenGL ES 2.0 and uses GLSL as shading language.

Technique Comparison

After the basic term definitions are clarified, it's time to compare both techniques, *"per vertex shading & per pixel shading"* a bit more in depth. As it's necessary to know the requirements and limits of both techniques, to be able to find the most ideal and non-ideal use-case for both. Admittedly, it will also help to know the typical usage of both techniques in the industry, to find these ideal use-cases.

Vertex Color

Usage

Vertex color can be used to color a mesh without any UVs, or as additional input for a shader. For example, to colorize, or darken the diffuse map of a mesh, or as color mask to blend multiple tiled diffuse maps onto a mesh. Some nice examples are foliage lighting (Polycount, Foliage Vertex Color - polycount, 2015), texture mapping (Polycount, MultiTexture - polycount, 2016) and procedural animations (Sousa, 2007).

Image by Eric Chadwick

Limits

Generally, Vertex color is quite cheap to render and does not require much extra storage render. Depending on the used game engine and model format it's possible to store multiple color values on one vertex, although this will cause the game engine to render this vertex twice and slightly increasing the memory cost (Polycount, MultiTexture - polycount, 2016).

Pros

- *Vertex color does not require any UVs on the mesh*
- *Works well on meshes with tiled textures (Useful to add effects like Ambient Occlusion to models* (Polycount, Ambient occlusion vertex color - polycount, 2015)*)*
- *Does not have any texture compression artifacts* (Trümpler, X:Rebirth – Geometric Lensflares | Simon Schreibt, 2015)

Cons

- *Depended on the amount of Vertices on the mesh*
- *Each Vertex requires more data, than a pixel of an uncompressed texture*

Pixel Color

Usage

Per Pixel based color information is commonly used to control a wide variety of different parts of a shader, or to supply diffuse, normal or specular color information to the shader. As pixel based color data is stored as a two dimensional image, a model with texture coordinates is required for the shader, to be able to wrap the texture onto the model.

Although the creation of the texture coordinates (UVs) for a mesh is an extra step, which is required before artists can start to texture a model, this is the most used technique, to texture models for games. Reasons for that are, that the texturing and modeling process can be easily separated and it's possible to increase the resolution of a texture without modifying the mesh itself.

Some standard techniques for creating this textures are to draw them in a separate texturing application like Photoshop, or Substance Painter, or to bake information from a mesh onto a texture (Polycount, Category:TextureTechnique - polycount, 2014)

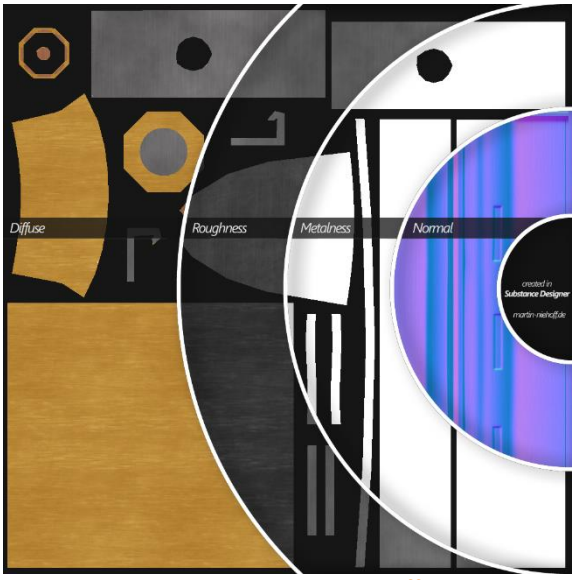


Image by Martin Niehoff

Limits

Pixel based color allows a high flexibility to change parts of the texture, or to increase its resolution, without the need to adjust anything on the mesh itself.

Pros

- *Mesh independent*
- *Allows multiple color information per vertex and model*
- *A pixel of a texture requires less data, than a vertex of a mesh*

Cons

- *Requires UV´s on the mesh*
- *Requires extra space for the textures*

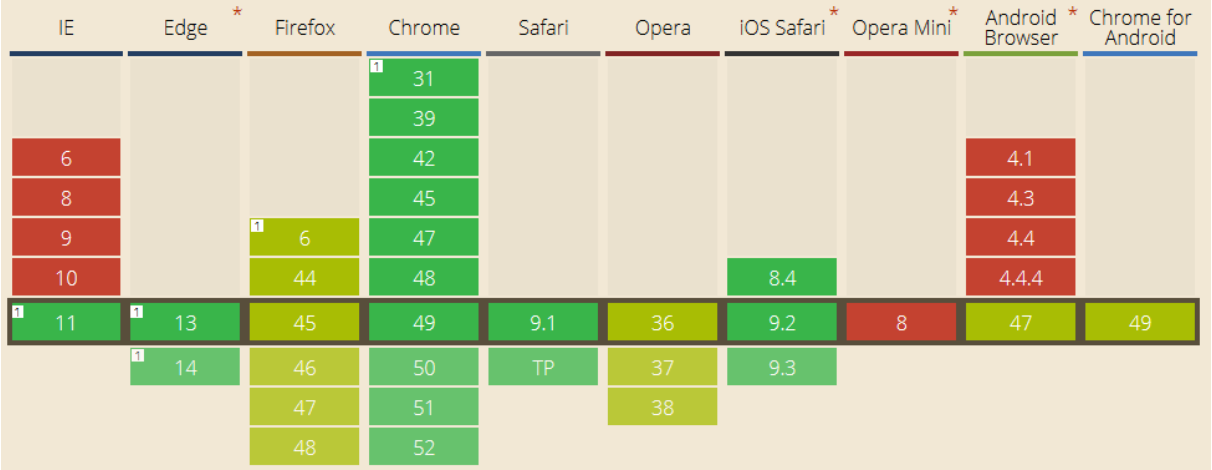
Background Information

At this point the basic definitions and used techniques should be clear to everyone, but to actually compare both techniques, its needed to gather some more background information as evidence for the importance of certain aspects. For example, why it's so important to lower the asset size for web based applications, or on which devise WebGL is actually running.

What are the requirements of WebGL?

The only thing that is required to run a WebGL based application is a browser with WebGL support and as all Major browser vendors are members of the WebGL Working Group, its widely supported (Khronos Group, 2011). At this date around 83.86% percentage of all browsers are supporting WebGL based on data from "StatCounter GlobalStats" for March 2016 (Deveria, 2016).

Although this value is depending on the browser usage per country, for example is the support in Germany with 88.17% better than in China with 71.57% (Deveria, 2016).



Are there different requirements between the two shading methods in WebGL?

As this report is comparing two different shading methods, its rational to check the availability and support of both, before starting with their implementation. Luckily WebGL is using the OpenGL shading language and offers therefore the familiarity of the standard OpenGL API, so that implementation of both techniques is similar to the implementation in native OpenGL applications (Khronos Group, 2011).

Admittedly is the list of available GLSL commands quite short compared to modern OpenGL applications, due to the old standard used. OpenGL ES 2.0 is built to run on as many devices as possible, so it's only requiring GLSL Version 1.0 (Khronos Group, 2014) instead of the current version 4.5 (Wikipedia, 2015).

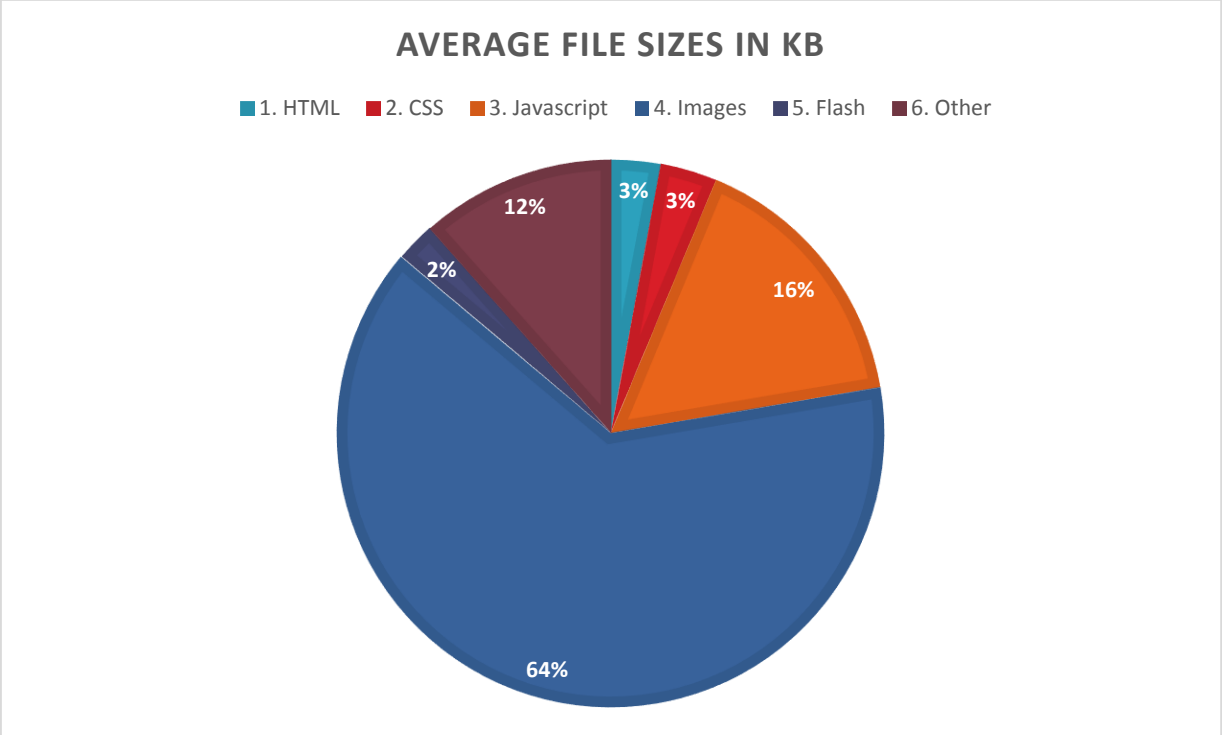
As the goal of this report is not to figure out how to program shaders for these techniques in WebGL, but rather a general comparison of their usability. A WebGL engine called "Playcanvas" with pre-integrated shaders for both shading techniques is used as reference engine throughout this paper.

What is the average size of a website in 2015?

Too measure the importance of the asset size for Web based applications, some kind of reference is needed. A good value for that is the size of an average website nowadays, as these applications will be accessed by users in the same way, as normal websites.

A good source for this data is the "HTTP Archive Report", which collects technical information from half a million of the most popular websites (Souders, 2016). The report analyzes web pages, that are publicly available and stores technical information about them.

It shows that the average web page was around 2.2mb big in 2015, this is an increase of 16% compared to 2014.



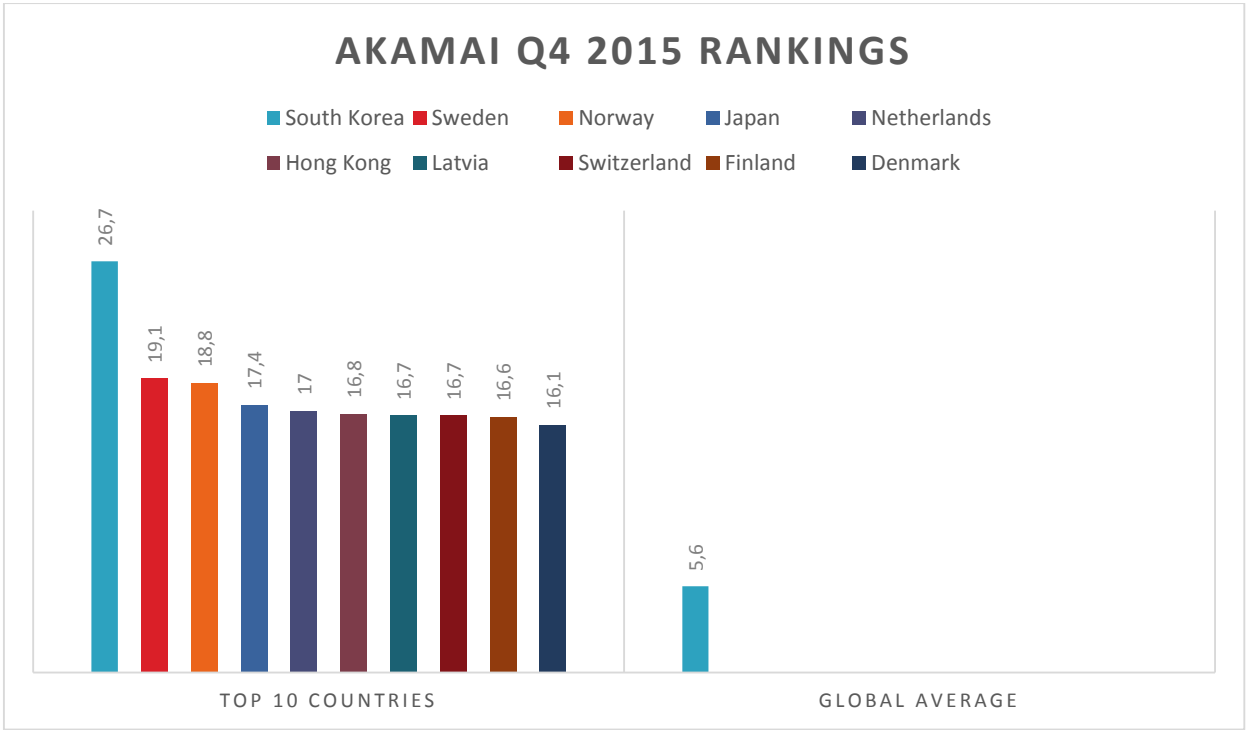
What is the average bandwidth of users?

After knowing the average size of websites, it's time to get the average bandwidth of the users, so that their average loading times can be roughly estimated. Although this could be taken into a separate research topic on its own, due to the many factors influencing the actual end user connection. It's enough evidence for this topic to have a rough value for a basic comparison, as the real value will differ based on the geographic and demographic target group, which is out of the scope of this paper.

For this purpose, the "State of the internet report Q4 2015" (Akamai Technologies, 2016) from the Akamai network will be used. Akamai is the global leader in Content Delivery Network (CDN) services with more than more than 2 trillion Internet interactions each day (Akamai Technologies, 2016).

Although their data should not be taken as a reflection of the actual end-user connection speeds, but more as an indication of the performance of their CDN and their connected ISPs. It is providing an interesting view on the general Internet speed and this is enough information for this paper.

The statistics show that the country with the highest average internet connection speed during the fourth quarter of 2015, was South Korea with an averaged a connection speed of 26.7 Mbps, while the global average connection speed was just 5.6 Mbps.



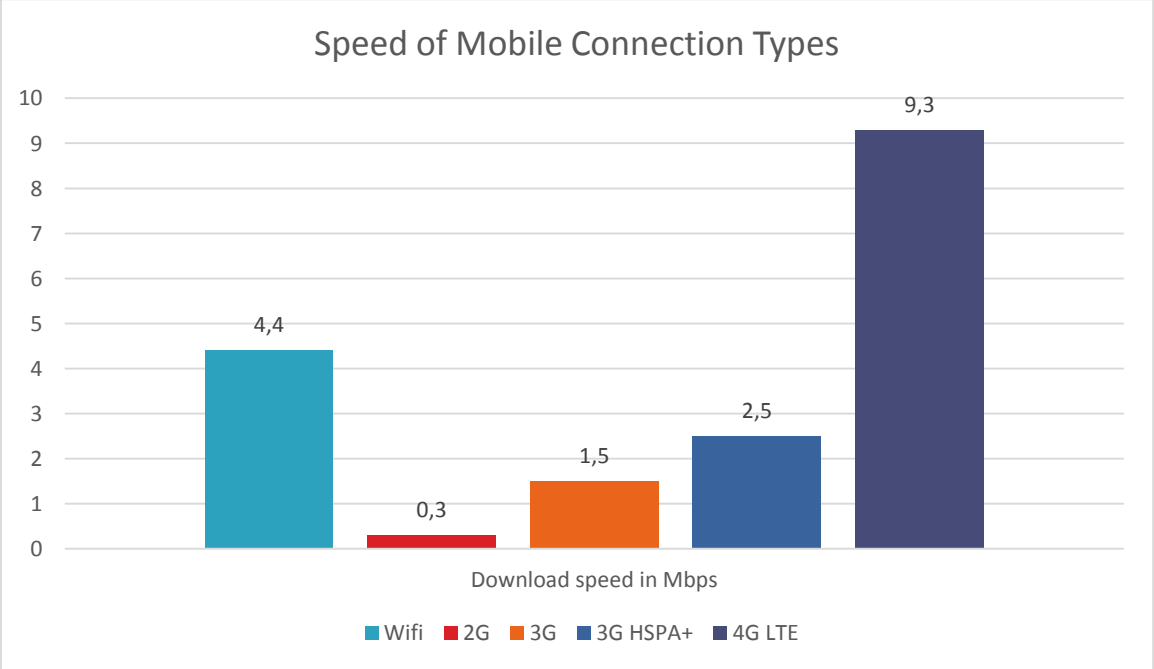
What is the difference between mobile and static internet bandwidth?

As WebGL applications are generally build to run on as many devices as possible and also on as many locations, the overall bandwidth is not the only thing to keep in mind while developing these applications.

If a user is visiting an application from his or her mobile phone, the chances are high, that they are doing this through their mobile connection. So it's an interesting question what bandwidth differences are there between users, which are browsing through their mobile connection, or via a static connection.

For this let's take a look at a study from "OpenSignal" a leading wireless network tracking service. OpenSignal is determining data speeds and accessibility via an app, with currently over 15 million downloads (OpenSignal, 2016), so more than enough data as baseline.

One of their recent studies about the state of LTE from March 2015 (OpenSignal, 2016) taken from 11 million users with an LTE plan offers a valid answer to this question. It shows that Europe is the world's quickest LTE region with an average of 18 Mbps, while the U.S. have just an average download speed of 7 Mbps. The global average is in-between on 9.3 Mbps. In addition to that, the report also offers a nice insight about the average download speeds of other mobile connections, which offers some valuable data to compare the impact of different connection types on the download time of an application.



What is the maximum average time, that a website should take to load?

After defining some baseline values for the average download speeds of static and mobile internet connections, it's easy to calculate the time that users have to wait for the download of their WebGL applications. But the question is, what download times are accepted by users?

As WebGL applications itself are a relatively new trend on the internet, there are hardly any statistics about that. In addition to this, the "accepted" time which are users willing to wait for something, is based on their own mindset and that is hard to measure. Although a user is most likely willing to wait longer, if they know they are loading some fancy 3D application, or a game instead of their local newspapers webpage. WebGL applications have many familiarities with normal websites, like their access, that happens in the same way through the browser of the user. This allows a comparison based on the average accepted loading time for websites, as plenty of statistics are available for this.

A recent speed experiment run at the financial times website shows nicely how the speed of a website can affect the user engagement. They tested how the speed of their website is influencing the amount of articles, which is read by their users and by that its impact on their revenue.

"The speed of the site negatively impacts a user's session depth, no matter how small the delay." (Lahav, 2016)

"The data suggests, both in terms of user experience and financial impact, that there are clear and highly valued benefits in making the site even faster."

(Lahav, 2016)

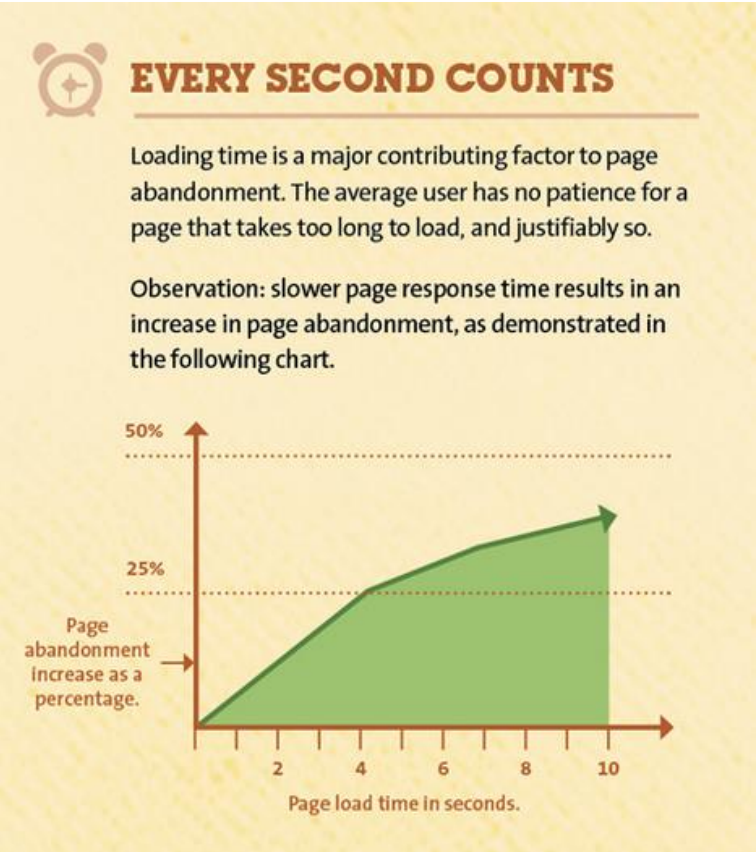


Image from Shaun Anderson

What costs is this traffic producing?

Another aspect to measure the accepted size for web based applications, are the costs that their download produces on an average user's data plan, if its downloaded via a mobile connection.

Though these data plans are highly dependent on the user's country and provider, as well as underlying frequent changes, this paper is not going through any statistics to avoid blowing up the scope of this research.

Instead a fictive user from the Netherlands with a data plan, which charges 12 cents per MB (Maemo, 2016), if he or she out of their inclusive volume is going to be used. As the actual prices are not for any relevance for the conclusion of this report, they are merely to delineate their influence in the user's acceptance of application sizes.

Definition of the problem

The gathered data shows the importance of asset optimization regarding their file sizes, for the use in web based applications, especially if the application should be widely accessible by mobile users and from areas with low download speeds. Based on this foundation, this paper is comparing the differences between per vertex and per pixel shaded meshes to find out, if one of these techniques can be used to lower the file size of certain assets for the use in web based applications.

Main Question

Therefore, the following main question can be formed:

When a per vertex shaded mesh, has advantages over a per pixel shaded mesh in a WebGL application?

Sub Questions

To be able to answer the main question of this paper, while staying in the scope of this report, two things have to be clarified upfront.

- **When are per vertex shaded meshes resulting in smaller asset sizes?**
- **When have per vertex shaded meshes a lower VRAM consummation?**

With answers to both sub questions it's possible to conclude which technique can be preferred on a given use case. Although as the previous research showed, that both techniques have their own limitations and ideal use cases, this paper is not going to give a fixed rule, but rather recommend certain use cases in which one technique should be preferred over the over.

Scope

The scope of this report is bound to the research, that is required to recommend one of the listed shading techniques, for certain use cases, or project requirements.

Factors which are influencing this recommendation are the technical limitations and advantages of both techniques, as well as their usability for the artist. In addition to that the importance of file sizes and its impact on the user experience is briefly investigated, to emphasize the importance of file size optimization for web based applications.

Points that are out of the scope of this report, but could have influenced the recommendation of this report are performance comparisons of both techniques, on different hardware setups, as well as different shader implementations and asset creation workflows with addition software, or different asset pipelines. As well as a more in depth look at a certain target user group, or location, to gather more in depth statistics about the download speed influences.

Research and design

The main research method, which is used to gather the data required for the recommendation of this paper is a praxis oriented prototype. This prototype is built to answer the questions, which raised while collecting the background information needed for this research.

While the collection of these background information helped to figure out the more and less important aspects of the compared shading techniques, they give no clear answer to the main question. Therefore, the prototype is used to verify the collected theoretical data and also to give a more in depth answer to the given main question. In addition to that the prototype is offering a good possibility to re-check, or extent this research topic, due to its open nature on the web.

The prototype contains only a basic scene with an environment map and light setup to avoid, that the research results are influenced by any unneeded influences. Meshes with a variety of different polygon counts and texture resolutions will be spawned separately, to measure the differences in their asset sizes and VRAM consummations. These information is used to define the asset size and VRAM usage of both shading techniques, which allows to draw a line between both. This allows therefore to define a point, where one technique could be preferred, to reduce the download size, or VRAM consume of an asset.

A positive side effect of building the prototype is that the creation of required assets allows a basic comparison of their usability in a production environment. As its generally not feasible to use a technique, even if it offers slightly better results, if it’s too hard to use, or too time consuming. Therefore, this is another important aspect, that should be taken into account for a recommendation.

List of test cases:

For the comparison of the asset sizes and VRAM usage, a low poly styled model of a sun is used, as this style can be easily achieved with both techniques, while being quickly adjustable to every use case. This ensures, that the focus of this research if on the user shading techniques, rather than the used art style.

1. Case: Low Poly Vertex Shaded Mesh vs. Low Poly Pixel Shaded Mesh

In the first test case a mesh with a low polygon count is used for both techniques. The only difference is, that the vertex shaded mesh has vertex color information, but no texture coordinates, while the pixel shaded meshes has texture coordinates but no vertex color information assigned.

This is used as first comparison to figure out the influence, that the color and UVs information have on the file size of the assets.

Both meshes contain the same color information. The vertex color is assigned in Maya, while the pixel color is baked into a texture from the exact same vertex color.

2. Case: Mid Poly Vertex Shaded Mesh vs. Low Poly Pixel Shaded Mesh

The second test case is using the same low poly pixel shaded mesh as in the first test. However, it has a more detailed texture applied, which is baked from a vertex shaded mesh with an increased amount of detail. To be able to achieve this amount of on the vertex shaded mesh, it is twice linear subdivided in Maya.

The purpose of this test is to show the influence of the textures detail on the usability of vertex color.

3. Case: Optimized Vertex Mesh vs. Low Poly Pixel Shaded Mesh

The third and last case shows how vertex shaded meshes can be optimized with adding additional geometry only on required spots, to increase its usability. As subdivided meshes, like the ones used in the second test have most likely always more polygons, than most use cases require.

The manually created per vertex shaded mesh of this test case only contains the required polygon count to visualize the given color information. So that a mesh with a reduced polygon count can be compared with the low poly pixel shaded mesh from the first test case, which has the same amount of details stored in the texture.

This test is used to check the maximal possible data reduction with both techniques.

Research results

What is the importance of file sizes for web based applications?

In the theoretical background of this report, a variety of different statistics about average download speeds on static and mobile internet connections were gathered, to investigate the importance of asset sizes for web based applications. The gathered data underlay's the importance of file size optimizations for web based applications, to avoid negative influences on the user experience and engagement.

The statistics are showing that slower loading times of websites are negatively influencing the user experience and engagement. "Largely, the slower the site, the greater the effect." (Lahav, 2016)

It shows that an average website of **2.2MB** would require around **0.13 seconds** for the data transfer with an average connection speed of **17 Mbps** in the Netherlands. A very good result, thinking about 1 second as optimal loading time.

However, the global average download speed is only **5.6Mbps**, so the same content would require already three times more time to load. Or even longer in a different part of the world. Nevertheless, is the loading times in both cases relatively short, but web based applications will most of the times be way bigger than an average website. If the user browses to a web based applications with a size of **120MB**, like Googles Lightsaber Demo (Google, 2016), for example. The user is accessing a web based application like this in the same way, as a normal website, however the page loading suddenly needs **7seconds** on the same connection.

If an application should be usable on mobile connections as well, this trend will continue rapidly. In this case the download speed is not the only aspect to think about, but also the price its traffic could produce. Assuming a user from the Netherlands is visiting an average website of **2.2MB** via his mobile phone. At the average connection speed of LTE in the Netherlands the app still required only around **0.13seconds** for the data transfer, but in addition to that they also have to pay for the produced traffic. Thinking about a price of **12cent per MB**, they would have a bill of **26cents**.

But if the user browses to the Lightsaber demo from his mobile the page loading would suddenly need **8,5seconds** and produce a bill over **14,40€**.

Furthermore, the download speeds over LTE and their accessibility in the Netherlands are within the top 10 worldwide. Assuming the fictive user is visiting from a country with exactly the average download speed, these times will already increase to **0.24seconds** for a normal website and more than **12,9seconds** for the WebGL application. Going further and assuming the user is just connected with a 3G HSPA+ connection instead of 4G, that has an average download speed of **2.5Mbps** (OpenSignal, 2016) the user has to wait for **0,88seconds** respective **48seconds** for the WebGL demo.

Even on an average Wi-Fi connection with **4.4Mbps** download speed Googles Lightsaber demo would take already more than **27seconds** to load. In all cases the WebGL application will load longer than the time, which is accepted by a lot of users and therefore resulting into the loss of some users. This shows the importance of file size optimizations for web based applications.

Are there differences between, different mesh file types?

The creation of the assets showed a few interesting aspects of the used file types. The mesh itself is identically for both shading techniques in the first test case. However, its saved once without vertex color and once without UV information for the different shading techniques.

FBX *(FBX 2013 in Binary Mode)*

This showed a relatively big difference in the file size of the generated FBX files. As the Pixel Shaded Test Mesh with UVs is **47,4KB** big, while the same mesh with additional Vertex Color, but without the UVs is only **34,5KB** big. *That's a file reduction of* ***27percent****, just by removing the UVs.* To cross check that the same mesh without UVs and Vertex Color is only **29,7KB** big.

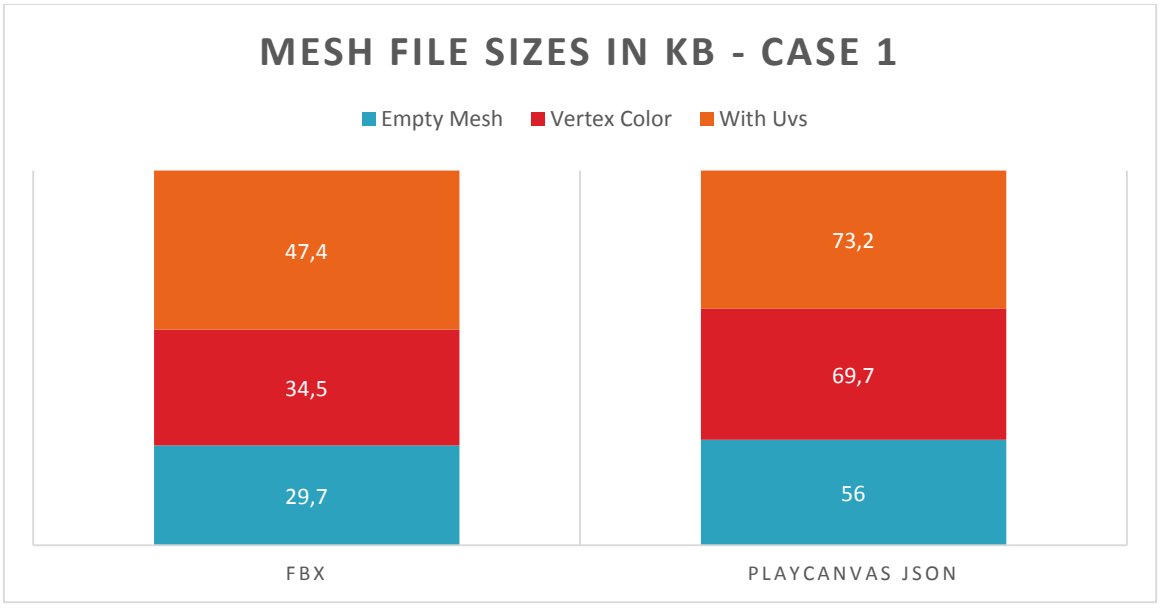
This shows that the extra vertex color information in this case, is increasing the file size of the mesh by around **16percent**. As different colors will result into different stored numbers, this value can slightly change depending on the stored colors. *(See "Can different Vertex Colors influence the file size of the mesh?" For further details)*

In comparison the file size of the same mesh is blown up by almost **60percent** by its UV coordinates. This illustrates that it's possible to reduce the file size of meshes by great numbers, while using vertex color instead of per pixel color. As a per pixel shaded mesh would require not just the UVs, but also an additional texture file.

Playcanvas JSON

The for this prototype used game engine is using its own file format, based on the JSON file format (Playcanvas, 2016). As all uploaded FBX files have to be converted into this format, to be readable by the browser, it's interesting how this file type is affecting our file sizes.

The converted meshes are increasing in their overall size, due to the textual format, but the size differences stay in the same order. The Mesh with UVs has still the biggest file with **73,2KB** followed closely by the Vertex Colored Mesh with **69,7KB,** while the mesh without any additional information is only **56KB** big. Although the margin between the mesh with vertex color and the mesh with the UVs is smaller in this format, the vertex colored mesh is still around **4,8%** smaller.



Can different Vertex Colors influence the file size of the mesh?

As the color information, which is stored in the vertices, is like the color information stored in textures also just a number, it was interesting to see, how different color values can influence the file size of a mesh.

To check this, the per vertex colored mesh of the first test case was taken again and slightly modified, to test different color setups.

1. *Just one Black Color Applied*
2. *Multiple Different Colors Applied*
3. *Smoothed Colors of the second test to produce gradients*

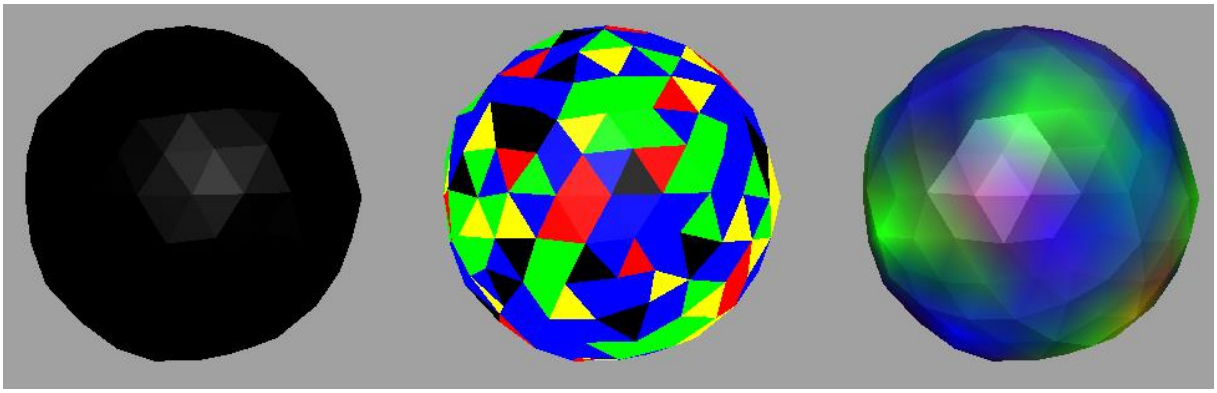
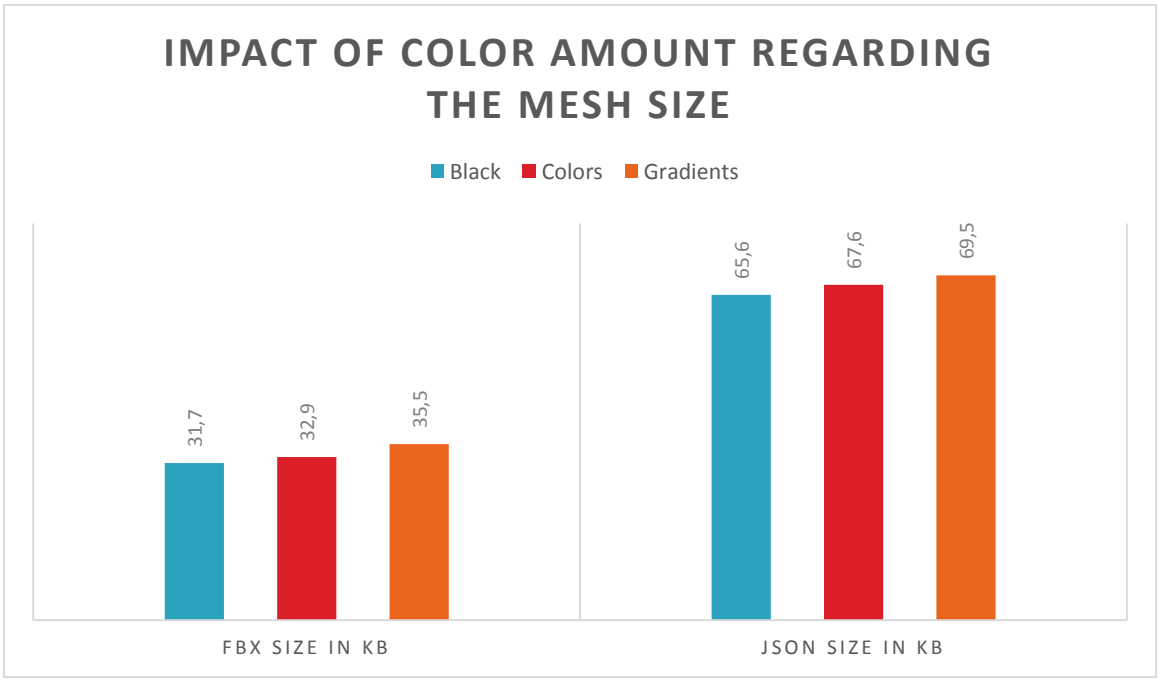


Image showing the compared meshes

The results are showing clearly, that the amount of color information is directly influencing the size of a mesh with vertex colors applied. This means, that a vertex shaded mesh with a single color will be smaller than a mesh with multiple applied colors.



Prototype Results

After the examination of the file types already showed some useful information about the different file sizes, between both shading techniques, the next step is going a bit further with a comparison under different circumstances, as well as the impact of the textures applied to a pixel shaded mesh.

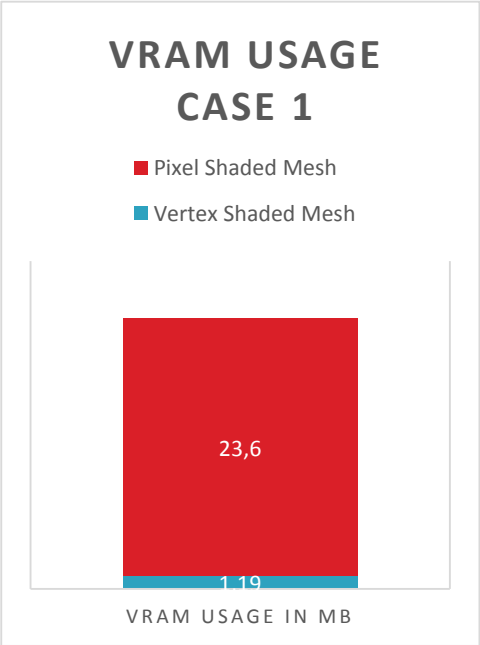
The comparison of the different test cases with the prototype has shown the following data.

1. Case: Low Poly Vertex Shaded Mesh vs. Low Poly Pixel Shaded Mesh

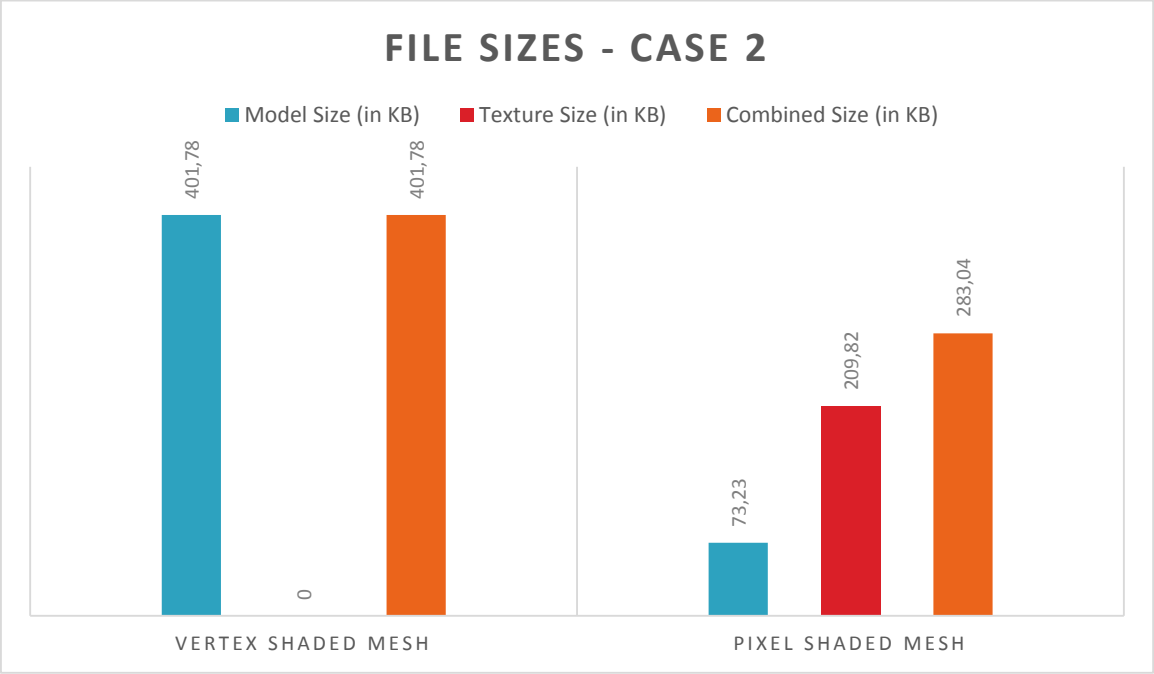
The first test case shows clearly that per vertex shaded meshes can result in way smaller file sizes under the given circumstances. Even if the texture would be downscaled and used in a lower resolution, the overall asset size of the per pixel shaded mesh, will always result in a bigger asset size. As the pixel shaded mesh itself is already bigger, than its vertex shaded counterpart.

Another interesting fact that the prototype has shown is the difference in the used VRAM, between both shading techniques.

While the entire scene with the vertex shaded mesh only required **1.19MB** of VRAM, the same scene with the pixel shaded mesh already required **23.6MB** of VRAM. This can be useful for optimizing applications to run on mobile devices with low VRAM capabilities.



2. Case: Mid Poly Vertex Shaded Mesh vs. Low Poly Pixel Shaded Mesh



The second test case shows nicely, that vertex shaded meshes, are not always resulting in smaller asset sizes. As more color information will require more vertices, to store the additional color information, there is quickly a point reached at which the combined size of the pixel shaded mesh and its texture is resulting in a smaller asset size.

This can be explained by the following:

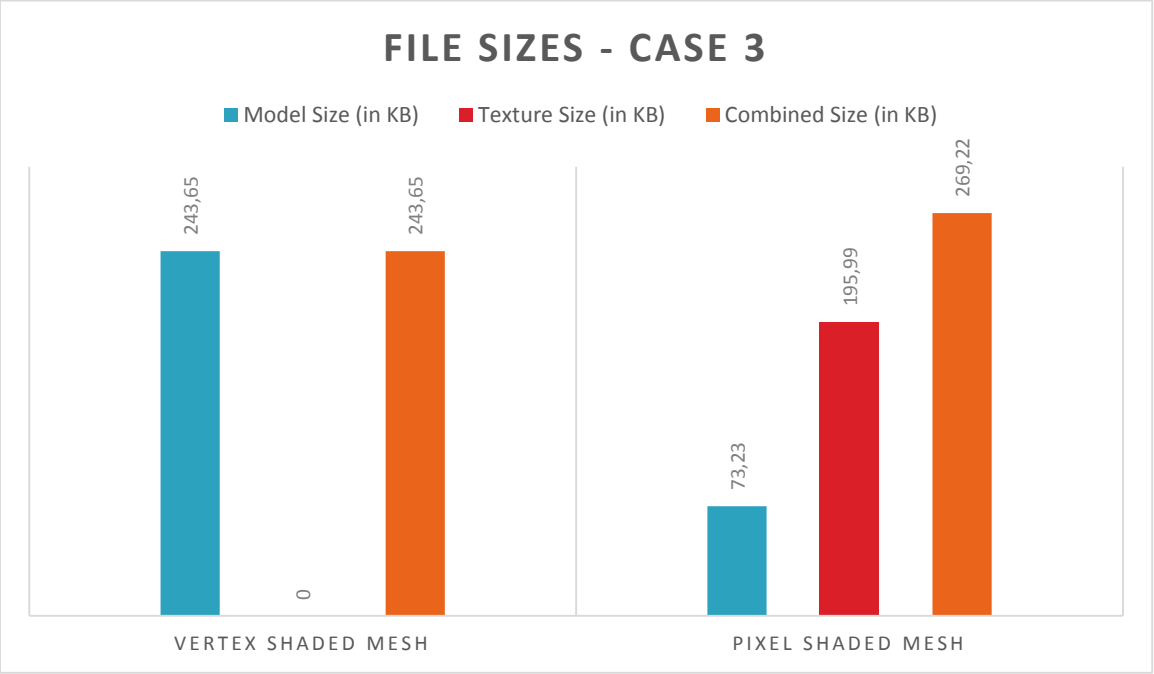
- *Each Pixel of a texture needs 24/32bpp (8Bit for each channel), resulting in **3Bytes per pixel***
- *But each Vertex needs a position, a normal and the color, resulting in **38Bytes per vertex***

Therefore, additional vertices which have to be added to a vertex shaded mesh, are only reducing the overall asset size, if their additional data is not exceeding the data required for an additional texture.

As the actual file size of a mesh is dependent of many factors, it's not practical to define a general mathematical formula, at which a per vertex shaded mesh will be bigger than a pixel shaded mesh with its texture. However, looking at the first two test cases, a rough estimate could be that meshes with less than the double amount of additional vertices, added for vertex color information, can result into smaller asset sizes. A more detailed estimate would require a lot of additional base information, like the used file type (and therefore the possible compression used), the amount of vertices, their normal & UVs (if existing), the assigned colors, as well as the amount of triangles. Similar dependencies apply to the texture. To gather all these data of an asset, before it exists is not possible, and after the assets are already created the formula would be unnecessary.

But even with the increased asset size, the vertex shaded mesh still remains lower on VRAM usage. In this case at **1.19MB** for the Vertex Shaded Mesh and **23.6MB** for the pixel shaded mesh. This makes the vertex shaded mesh still to a valid alternative for applications with the requirement of low VRAM usage.

3. Case: Optimized Vertex Mesh vs. Low Poly Pixel Shaded Mesh



The third test case shows, that vertex shaded meshes can be further optimized to increase the region where they are resulting in smaller asset sizes. Meshes can have additional vertices to add a greater amount of color details, while keeping the asset size small. As long as the required additional vertices are not requiring more data as a texture would do.

The manually created mesh of this test case only contains the required polygon count to visualize the given color information, without any unneeded extra geometry. Therefore, it is still a valid alternative to reduce the asset size. However, what is important on this method is to carefully think about the amount of extra vertices, that will be required for the additional color information, to ensure its not ending up in a higher data consummation than intended. In addition to that, this method is requiring more manual work as the default creation of vertex shaded meshes, as the model has to be adjusted.

Usability

The asset creation for the prototype showed, that vertex shaded meshes can be created quicker, than pixel shaded meshes, as the additional step of unwrapping the model falls away.

Furthermore, most modeling applications support vertex color painting out of the box, so that no additional software is required, for the creation of vertex shaded meshes. Moreover, the separate step of transferring an asset into a texturing application falls away.

Another interesting technique is the conversion of textures into geometry, a technique that was amongst other things used in Homeworld (Trümpler, Homeworld 2 - Backgrounds | Simon Schreibt, 2013).

Conclusion and discussion

The optimization of assets regarding their files sizes and VRAM is even more important for web based applications, which should run smoothly across a wide variety of target devices and locations.

As textures usually take up a lot of memory and thus can drastically increase the download size and VRAM usage of web based applications, the use a vertex color offers a tempting alternative to the standard method of per pixel shading.

Web based applications like for example Googles Lightsaber Demo are requiring way more data than normal websites do, consequently their download times can be quite long. Depending on the location and device of the user this can greatly influence the user experience. While five MB in additional asset size are probably unnoticed in countries above the average download speed, they can increase the loading time of a user connected with an average 3G mobile connection by more than 3seconds. As the average internet user expects websites to load in under 3 seconds, the reduction of asset sizes gets more important, the lower the average download speed of the target area gets.

A prototype created with the Playcanvas WebGL engine and its JSON based model format showed the influence, that the different mesh data has on the actual file size. While a vertex shaded mesh is only requiring around 38Bytes of data per vertex, a per pixel shaded mesh is claiming two additional bytes per vertices in this engine *(See Appendix "Calculation of Playcanvas Model Sizes")*. Similar results are observed with native FBX files. This makes vertex color always the more lightweight shading technique, regarding the required file sizes, as long as the mesh does not require additional vertices, which only store color information.

However, if more vertices are required to color a mesh with the desired detail resolution, than its pixel shaded alternative would take up, it's depended of the amount of additional vertices. This is caused by the fact that each pixel of a texture takes up way less bytes, than a single vertex does. Its therefore only cheaper to add additional vertices for supplementary color information to a mesh, if their additional data is not exceeding the data required for an additional texture.

When exactly this point is reached is however dependent of too many factors to define a general mathematical formula, which still remains practical to use. Thought a comparison of different models in the given prototype indicates, that a rough estimate could be: That meshes with less than the double amount of additional vertices, added for vertex color information, can still result into smaller asset sizes, than a pixel shaded mesh.

Certainly this remains only a very rough estimate, which can be used as recommendation for further research, on a case to case basis, as a more detailed estimate would require way more data like the used file types of the model and texture, their compressions, as well as the amount of vertices, normals, UVs, triangles and used colors.

Another advantage of per vertex shaded meshes is their compared to per pixel shaded meshes very low VRAM usage. The prototype showed, that the VRAM usage of vertex shaded meshes is taking up only a fraction of their pixel shaded opponents, even if their asset sizes are bigger. This makes vertex shaded meshes particular interesting for applications with the requirement of low VRAM usage.

As this paper only investigated the general possibility of using vertex colored meshes as alternative in some cases, based on a prototype build in the Playcanvas game engine. Further research could examine the influence of different shaders, file types and compression methods, as well as their implementation in different engines, to see how this can affect the asset sizes and VRAM usage of both shading techniques.

Vertex shading offers a tempting alternative to reduce asset sizes and VRAM consummation, to some extend and therefore allows to optimize web based applications to download faster and run smoother on a greater amount of target devices, which can positively influence the user experience.

In addition to the question, when vertex colored meshes have advantages in their asset sizes and VRAM consummation the question of their usability remains a personal decision. While the creation of meshes with vertex color is generally quicker, due the lacking requirement of unwrapping the mesh as well as the reduced number of production steps, it is quite different method of texturing, which might not fit to everybody and every project.

Recommendations

The use of per vertex colored meshes is valid solution to reduce the asset size and VRAM consummation of WebGL applications, as long as some limitations are taken into account.

- If the art style allows to color meshes via vertex color, without the addition of too many additional vertices, use vertex color. As this will reduce the asset sizes and VRAM consummation of the application. However, if this is not the case, a per pixel shaded mesh is cheaper regarding the asset sizes.
- If vertex shaded meshes are requiring too many additional vertices, evaluate the advantage of the lower VRAM consummation for the target group. If that's a limiting factor for the application, consider the use of vertex color, even if that increases the asset sizes.
- If low VRAM usage and tiny asset sizes are key requirement for the target group, consider using a low poly art style, as used in the prototype, because this offers an ideal use case for vertex color.

Graduation Products

The prototype created for this paper can be found under the following link:

http://playcanv.as/p/IFYx958a

References

Akamai Technologies. (2016, 03 22). *Content Delivery Network (CDN) and Cloud Computing Services Provider | Akamai*. Retrieved from Akamai: https://content.akamai.com/PG5641-Q4-2015-SOTI-Connectivity-Report.html

Akamai Technologies. (2016, 04 12). *Content Delivery Network (CDN) and Cloud Computing Services Provider | Akamai*. Retrieved from Akamai: https://www.akamai.com/us/en/our-thinking/state-of-the-internet-report/index.jsp

Alkemi. (2013, August 15). *A game of Tricks II – Vertex Color | Alkemi.* Retrieved from Alkemi Games: http://www.alkemi-games.com/a-game-of-tricks-ii-vertex-color/

Anderson, S. (2016, April 12). *hobo - internet marketing*. Retrieved from http://www.hobo-web.co.uk/your-website-design-should-load-in-4-seconds/

Deveria, A. (2016, 04 11). *Can I use... Support Tables for HTML5, CSS3 etc.* Retrieved from Can I use: http://caniuse.com/#feat=webgl

Google. (2015, April 8). *Mobile Analysis in PageSpeed Insights | PageSpeed Insights | Google Developers*. Retrieved from Google Developers: https://developers.google.com/speed/docs/insights/mobile?hl=en

Google. (2016, April 12). *Lightsaber Escape*. Retrieved from Lightsaber Demo: https://lightsaber.withgoogle.com/

Khronos Group. (2011, April 10). *Getting Started - WebGL Public Wiki*. Retrieved from Khronos.org: https://www.khronos.org/webgl/wiki/Getting_Started

Khronos Group. (2014, October 27). *WebGL Supported GLSL Constructs*. Retrieved from https://www.khronos.org/registry/webgl/specs/1.0/#4.3

Lahav, M. C. (2016, April 4). *A faster FT.com - Engine Room*. Retrieved from Engine Room: http://engineroom.ft.com/2016/04/04/a-faster-ft-com/

Maemo. (2016, 04 13). *Data Plans - maemo.org Wiki*. Retrieved from Maemo Wiki: https://wiki.maemo.org/Data_plans#Netherlands

OpenSignal. (2016, 04 12). *About Us - OpenSignal*. Retrieved from OpenSignal: http://opensignal.com/about/

OpenSignal. (2016, 03 1). *The State of LTE - OpenSignal*. Retrieved from OpenSignal: http://opensignal.com/reports/2015/02/state-of-lte-q1-2015/

Playcanvas. (2016, 04 14). *File Format - Learn Playcanvas*. Retrieved from Playcanvas Manual: http://developer.playcanvas.com/en/user-manual/graphics/file-format/

Polycount. (2014, September 26). *Category:TextureTechnique - polycount*. Retrieved from Polycount Wiki: http://wiki.polycount.com/wiki/Category:TextureTechnique

Polycount. (2015, November 03). *Ambient occlusion vertex color - polycount*. Retrieved from Polycount Wiki: http://wiki.polycount.com/wiki/Ambient_occlusion_vertex_color

Polycount. (2015, 8 12). *Foliage Vertex Color - polycount*. Retrieved 3 13, 2016, from Polycount Wiki: http://wiki.polycount.com/wiki/Foliage_Vertex_Color

Polycount. (2016, April 15). *MultiTexture - polycount*. Retrieved from Polycount Wiki: http://wiki.polycount.com/wiki/MultiTexture

Polycount. (2016, January 28). *Texture Coordinates - polycount*. Retrieved from Polycount Wiki: http://wiki.polycount.com/wiki/Texture_Coordinates

Souders, S. (2016, 04 11). *About the HTTP Archive*. Retrieved from HTTP Archive: http://httparchive.org/about.php#mission

Sousa, T. (2007, December 1). *GPU Gems 3 - Chapter 16. Vegetation Procedural Animation and Shading in Crysis.* Retrieved from Nvidia Developer Zone: http://http.developer.nvidia.com/GPUGems3/gpugems3_ch16.html

Trümpler, S. (2013, March 45). *Homeworld 2 - Backgrounds | Simon Schreibt*. Retrieved 04 2016, 10, from Simon Schreibt: https://simonschreibt.de/gat/homeworld-2-backgrounds/

Trümpler, S. (2015, 8 23). *X:Rebirth – Geometric Lensflares | Simon Schreibt*. Retrieved 3 13, 2016, from Simon Schreibt: https://simonschreibt.de/gat/xrebirth-geometric-lensflares/

Unity Technologies. (2014, October 7). *Benchmarking Unity performance in WebGL - Unity Blog*. Retrieved from Unity Blog: http://blogs.unity3d.com/2014/10/07/benchmarking-unity-performance-in-webgl/

Unity Technologies. (2015, May 28). *Web Publishing Following Chrome NPAPI Deprecation - Unity Blog*. Retrieved from Unity Blog: http://blogs.unity3d.com/2015/05/28/web-publishing-following-chrome-npapi-deprecation/

Wikipedia. (2015, September 12). *OpenGL Shading Language - Wikipedia, the free encyclopedia*. Retrieved from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/OpenGL_Shading_Language#Versions

Appendices

Calculation of Playcanvas Model Sizes

The calculation is based on the official documentation (Playcanvas, 2016).

Each Vertex Shaded Mesh is storing the following data per vertex:

- Position Vector: 15bytes
- Normal Vector: 15bytes
- Vertex Color: 8bytes

Resulting in **38 Bytes per vertex.**

Comparing that with a pixel shaded mesh, which is storing:

- Position Vector: 15bytes
- Normal Vector: 15bytes
- Texture Coordinates: 10bytes

Resulting in **40 Bytes per vertex** with an additional texture with 24/32bpp: **3 bytes per pixel**.