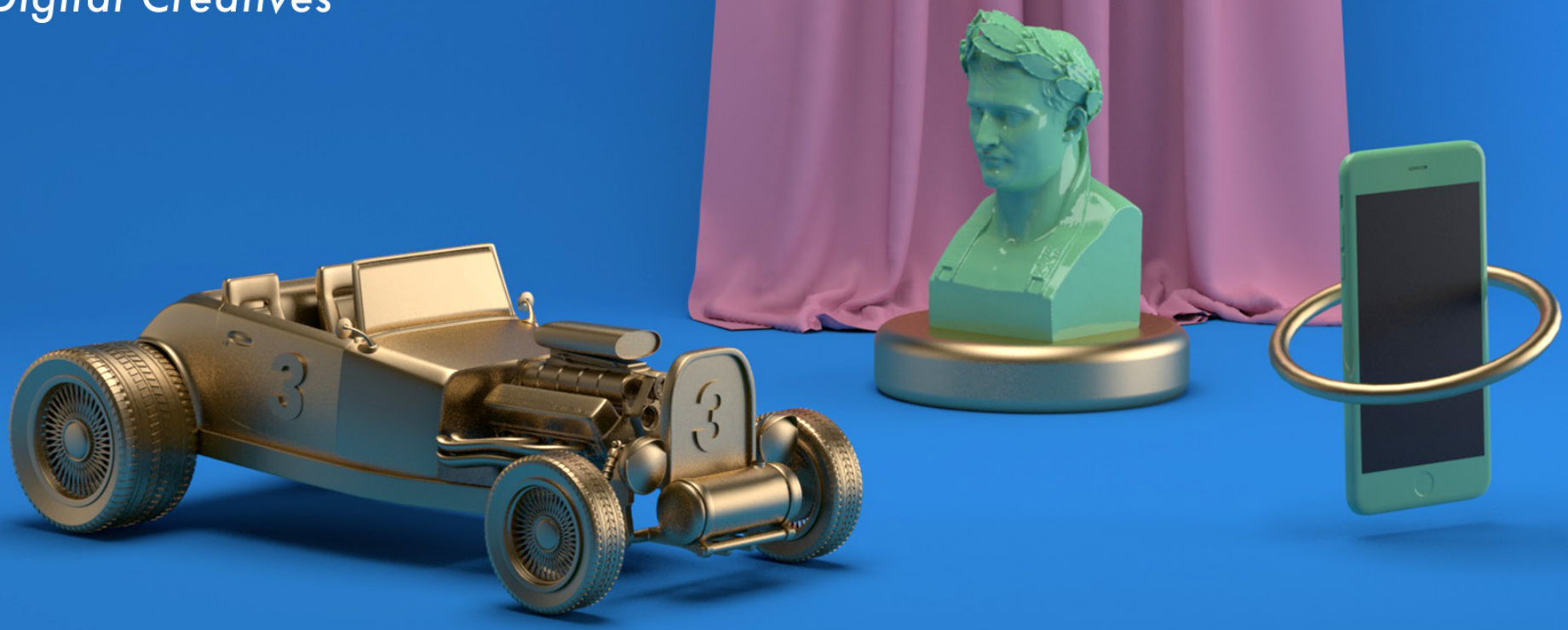
BRAIN FOOD!

for Digital Creatives

SPEED MATTERS

Designing for Mobile Performance



VOL.3

Google AWWWARDS®

Speed matters on the mobile web, but perception of speed is just as important.

In collaboration with:

Google

Whether you're a web developer, web designer or web marketer, you probably care about the end user of your product more than anything else. If you don't, well maybe we need another eBook for that!

When we look at internet users today, there is one thing that is new and striking: their level of expectations. Since mobile has become the dominant way to view the web, users access content and services on the go, and expect to be able to do that with their smartphone, anywhere, anytime. They want this to happen fast: load time is now rated highest and most requested criteria in what users expect from a site. In addition to this, speed can have a massive impact on businesses, knowing that 53% of mobile site visits are abandoned if pages take longer than 3 seconds to load.

In this context it becomes very clear that pretty isn't enough. If you're building websites, and particularly mobile websites, you need to make sure they are fast as well. But, speed itself isn't the only thing that matters, how we as human beings perceive speed and reaction times of a website is of utmost importance. This is at the core of what this eBook will cover, exploring how speed perception impacts user behaviour on your website, as well as sharing tips, tricks and techniques to better aid you in crafting highly performant websites that appeal to your audience.

Mustafa Kurtuldu & Lionel Mora

Index.

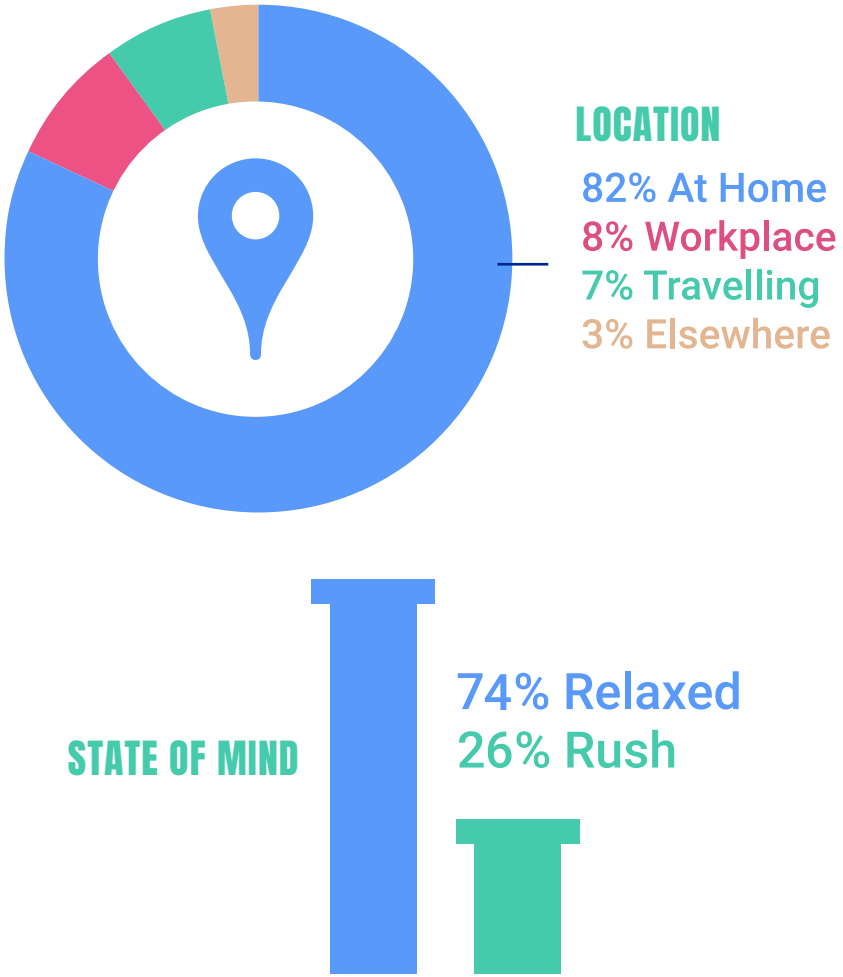
The Need for Speed

Is some new research carried out by Google, based on a survey which studies the relation between real and perceived performance by users. The study reaches a series of conclusions which will prove very useful when it comes to optimising mobile sites.

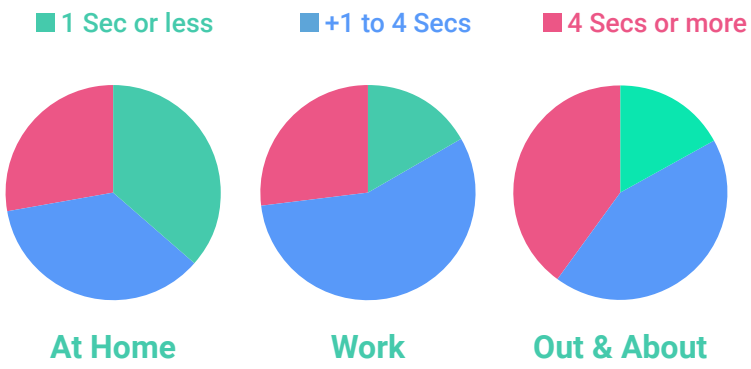
1.0

Mobile Browsing in Context

Mobile web browsing mainly happens in the home by people that are relaxed and calm, in this environment page load speeds are measured as being faster. This is probably due to that fact that Wi-Fi is prevalent and the user feels comfortable and less anxious about the time it takes to attain the desired information.

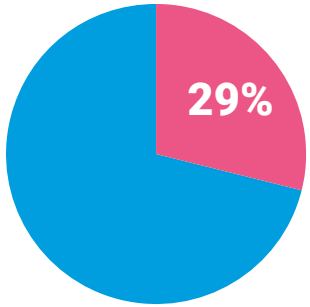


LOAD TIMES



In this environment goal based browsing activities are successful, with over 8 in 10 users achieving what they wanted to do.

PERCEIVE AS FAST



Even with fast loading sites (less than 4 seconds) 29% of people still don't actually perceive this as fast. Users have become demanding and building better sites must be a priority.

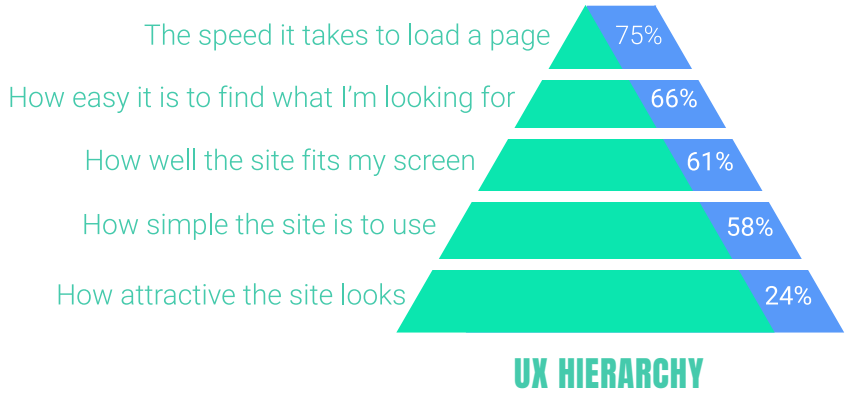
Speed Matters!

53% of mobile site visits are abandoned if pages take longer than 3 seconds to load.

1.2

How Important is speed for Users?

At a conceptual level, people want pages to load quickly - speed is important and is rated the highest in the UX hierarchy, this is not surprising as nothing can happen until the page is loaded (or at least assumed to be loaded).



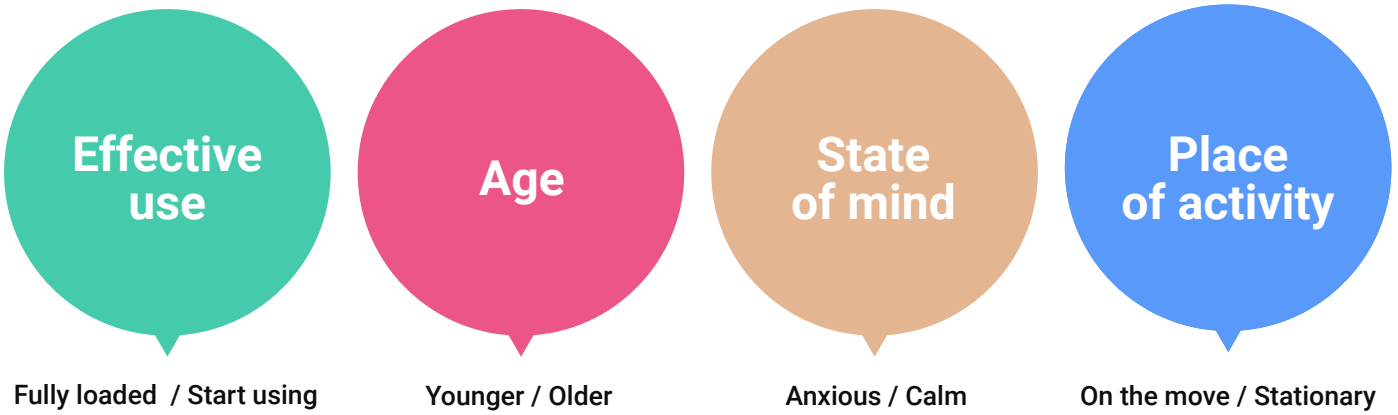
In general the real downloading speed is quite fast. The data collected in the study shows that 7 out of 10 sites loaded in less than 4 seconds and around 33% load in less than 1 second. In respect to the perception of the users, in general, they perceive the speed of the download as quite fast with figures very similar to the real measured speed.

However, within the users whose perceptions were different, there are noticeable discrepancies. For some, the perceived speed was quicker than the actual speed, and for others the perceived speed was much slower than the actual speed. This study aims to uncover the external factors that can affect this perception.

So what's going on?

Why is there this lack of correlation and what factors could have caused it?

EXTERNAL FACTORS THAT AFFECT PERCEIVED SPEED

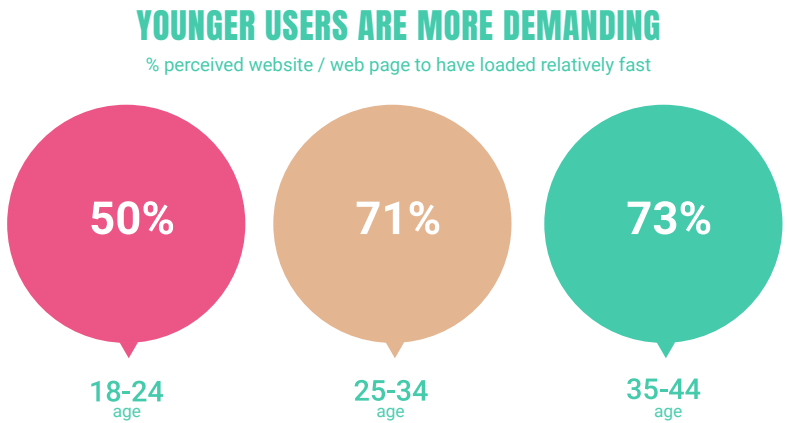


1.3

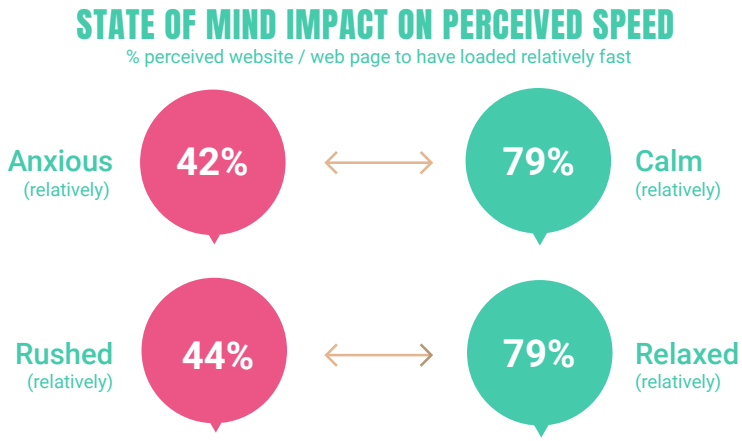
Is Speed Fluid?

We explored what might create a distortion between perceived speed and the reality, by considering four key areas: **Effective Use of the Site, User Profile, State of Mind and User Situation.**

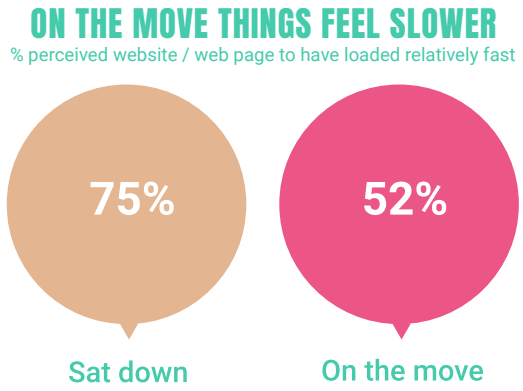
Effective use of the site – We found that some web visits where actual speed was slow but felt to be fast, tended to be retail sites. We know that long scroll sites (such as used by retailers) are designed to populate the page 'above the fold' giving the impression that the page is complete, even though loading is still ongoing 'below the fold'. We hypothesise that this could explain some of the distortion between the reality and the perception, the difference between the time after which a site can effectively be used, and the time that it takes to fully load. This is a prime example of design thought out to optimize the perception of performance, often used in an effective way to present the information on the main retail sites.



The user profile – When comparing profiles of mobile web users we found that the younger audience (18-24 yr. olds) tended to be more demanding of load times whilst their older counterparts were more relaxed and perceived speed as being fast, whether true or not.



State of mind – When users are calm and relaxed, speed gets faster, or at least it is perceived as such. On the other hand, when users are feeling rushed or anxious, the perceived speed slows down.



Situation – In a similar vein, web visits that are made 'on the move' are also less likely to be perceived as fast (even though they are), suggesting that when we are not at home and need the information ASAP, things feel slower.

Speed is Fluid!

Although actual time doesn't register that strongly, the external factors outlined on the left can in fact influence and distort user's perceptions.

1.4

Outcomes of Speed

Positive perceptions of speed can have a positive impact on how people feel about the visit outcome, future visit behaviour, and the the brand as a whole.

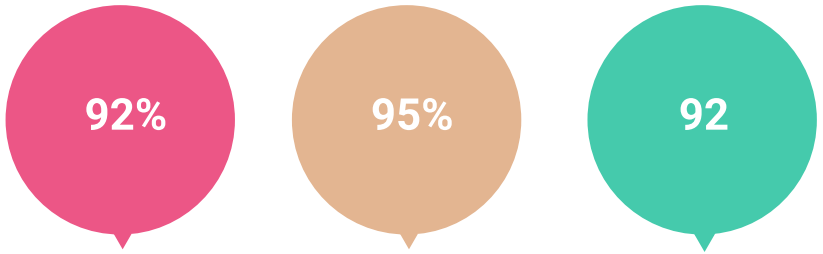
Achieved Goals – When perceptions of speed were thought to be fast, a higher percentage of users reported that goals were achieved. Of course, we can't unpick cause and effect here and it could be the case that when goals are achieved, as an afterthought users then perceive speed to have been faster. It remains though, a virtuous circle, with an intrinsic link between perceived speed and goal achievement.

Return visits – Users who sensed page load times as being faster were significantly more likely to predict they would return to the website again at 95%, on the other hand, a smaller figure of 62% of users predicted they would return to a site that was perceived as loading slowly.

Brand NPS - Net Promoter Score (a management tool used to gauge the loyalty of a firm's customer relationships) We also saw a major uplift in NPS where speed perceptions were 'fast'. We can use this as an indicator that the immediate browsing experience and sense of speed can offer positive results for the brand. How sustained that will be, we can't say, but some residual positivity would be expected.

Percieved speed = Results!

Percieved speed drives goal achievement, revisits and improved brand image



Achieved goal
(vs. 73% for sites with relatively slow perceived loading times)

Likely to return
(vs. 62% for sites with relatively slow perceived loading times)

Brand NPS
(vs. 44 for sites with relatively slow perceived loading times)

Improving the Perception of Speed

2.0

Perception Of Speed

As we have seen in the study, the perception of speed that the user has, does not always correlate with the real dowloading figures. In his talk "Speed Matters" (Awwwards Conference LA 2017) Paul Bakaus described that on a conscious level, we perceive a delay on loading which is 80 milliseconds more than the reality, which can be added to a determinable list of mental states, environmental factors and other contexts which can affect the reality perceived.

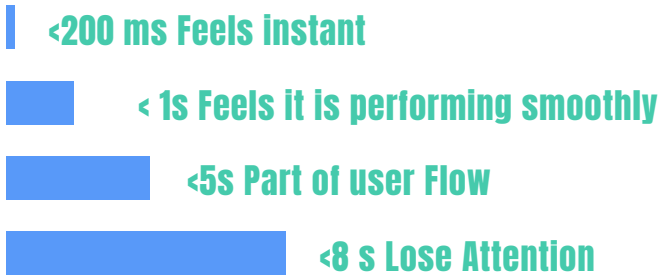
> ***"We perceive a delay on loading which is 80 milliseconds more than the reality."*** Paul Bakaus

The same user considers a long waiting time acceptable in some circumstances, but under other conditions a shorter waiting time actually causes them to abandon the task immediately.

2.1

Interface Response Times

Obviously it’s not only the waiting time during loading that’s important, but the time that the interface takes to respond to each interaction with the user - which has also been measured in our study. Less than 200ms is viewed as an instant reaction and if it takes more than 8 seconds the task is abandoned. But we must bear in mind that users have less patience when they first enter a site than when they make successive interactions within the same site.. A recent study by Google shows that 53% of users abandon a mobile site that takes more than 3 seconds to load.



2.2

First Meaningful Paint and Time to Interactive

First Meaningful Paint

FMP is when there is something useful on the screen to engage the user.

Time to interactive

This is when the browser finishes building the DOM and the user can begin to interact with the application.

As we have seen in the Google study, many retail sites with excessive loading times are perceived as fast because they use a very common TIP, they prioritize the loading “above the fold” so that the user can download the most relevant content and interact quickly while the rest of the site is loading. In other words a First Meaningful Paint is produced quickly and the loading of the scripts necessary is prioritized so that the content functions.

More Info

- Leveraging the Performance Metrics that Most Affect User Experience
- Time to First Meaningful Paint
- Measuring Perceived Performance

Tools

- Paint Timing API

Remove the 300-350ms tap delay

Without doubt a way to start the interaction more quickly is to remove this delay. For a long time mobile browsers used a 300-350ms delay between touchend and click while they waited to see if this was going to be a double-tap or not, since double-tap was a gesture to zoom into text.

To remove the 300-350ms tap delay, all you need is the **following in the** `<head>` **of your page:**

```
<meta name="viewport" content="width=device-width">
```

In new Chrome CSS rule “touch-action: manipulation” also eliminates click delay. For Old browsers, FastClick by FT Labs uses touch events to trigger clicks faster & remove the double-tap gesture.

2.3

Put the User in Active Mode

We can identify two forms of waiting - passive and active. In passive waiting the user simply watches a progress bar loading without doing anything, in active waiting the user could be interacting with a game or answering a little questionnaire about their profile.

“People in passive waiting mode overestimate waiting times by 36%.”

Use the Right Loader

Use spinners for very short waiting times, they are probably not necessary but they help to maintain continuity before showing the new element. Use progress loading bars for longer waiting times.

SPINNER
Short waiting times

PROGRESS BAR
Long waiting times

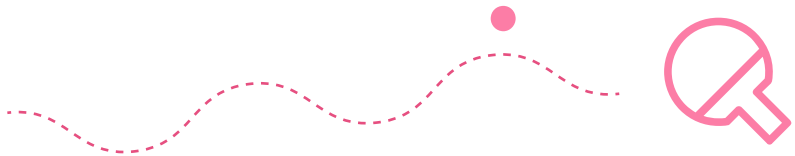
Create Instant Interactions

Instant feedback from the interface makes the experience feel fluid, generating the illusion of continuity. Microinteractions are a good example which support the dynamism of an interface. You can see some good examples here in our collection. "UI Animation and Microinteractions".

Tiny instant animations provide feedback and entertain the user while the "real action" is pushed into the background on a second thread.

Active Waiting in Loading

Put the user in a kind of "**Active Mode**" like playing mini-games during loading, answering some questions about their user profile, etc.



UX Patterns

To improve perception, time in passive wait must be minimized, to do this we can use the UX patterns that Paul Bakaus describes in his talk as well as improving the dynamism of our animations and transitions.

Preemptive Start

A preemptive start is the realisation of the beginning of a task before the user demands it. For example by showing the user a small questionnaire the user is distracted while other assets are loaded. Other examples are:

Loading Assets on Login Screen

Preload critical assets on the Login Screen. The process of preloading is started by taking the users to the login screen.

Preload Assets on Rollover

When rolling over elements in the menu, the preload of the assets from this sections starts.

Preload Content on Previous States

Preload critical content for the next screen when the users is in previous states.

Multi-Step Forms

Preload assets while the user is on a multi-step form.

Early Completion

In this case, the opposite as before, we partially show content that is not yet complete, the most common example is video streaming.

Progressive Images

A progressive image is created using compression algorithms that load the image in successive states. The user can quickly see a low resolution representation of the image that continues downloading.

Placeholders or Low-res Blurred Images Like Medium

Low-resolution images are used to preload the high-res ones, applying to them a blur filter to emulate a real progressive image.
continue navigating. It’s not bad practice as long as the user is informed of the state of the upload, alongside possible failures and interruptions.

Optimistic UI

A trick in which an incomplete task is presented as complete.

Instagram Likes

In Instagram, the like action appears immediate to the user, but in reality it is stored in the database at a later stage.

Upload File in a Second Thread

In this example the user uploads a heavy file to the server and a message appears in the interface saying that the task is “Processed” The task will actually be sent to a second thread using technologies like Web Workers. Thus giving the sensation of great efficiency and speed as well as allowing the user to continue navigating. It’s not bad practice as long as the user is informed of the state of the upload, alongside possible failures and interruptions.

2.4

Smooth and Optimized Animations

The main rule in Web Animations is to maintain 60fps. To get a smooth motion, each frame needs to be rendered in less than 16ms. There are a lot of animation techniques that aren't directly related to performance, but if they help to create a smoother and more fluid motion, they will surely help to improve the perception of performance too.

Easing functions

Choosing the right "easing" is decisive to produce a smooth animation that feels natural and pleasant. Easing Functions are curves that describe the acceleration or deceleration of a motion in a period of time.

1. Linear:

Linear motion describes a continuous acceleration. It doesn't feel natural.

2. Ease-in:

The animation begins slowly and accelerates

3. Ease-out:

It begins quickly and decelerates, it's useful for initial interactions when the user expects the quickest response time possible.

4. Ease-in-out:

The animation is slower in the entrance and exit, it produces a very fluid effect.

The Right Easing

Choose the right easing for fluid animation and a quick response

FOR USER INTERFACE

User Input

Ease-Out
Requires Instant reaction, Menus, Buttons,

Display Info

Ease-In
Prompt Windows

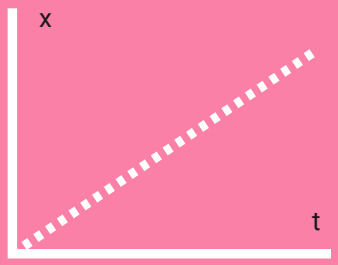
FOR SMOOTH ANIMATIONS

Short Times

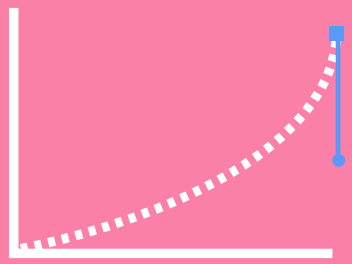
Ease-In / InOut
These curves are perceived as too slow in longer times

Motion Duration

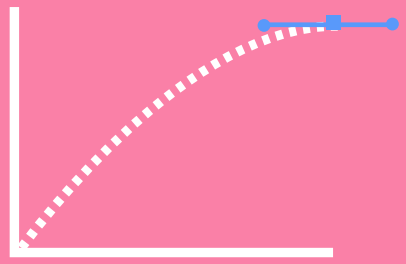
200 - 500 ms
For Bounce and Elastic Effects, use 200 - 800 ms



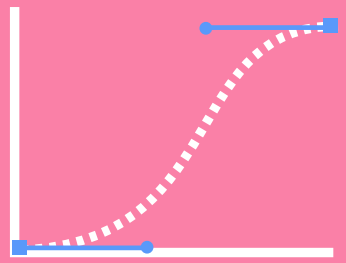
Linear
Constant Motion



Ease-In
Acceleration Curve



Ease-Out
Deceleration Curve



Ease-InOut
Smooth Motion

Choosing the Right Easing

1. **Use Ease-out for UI elements**

   This type of animation starts quickly and then slows down, which gives your animations a feeling of responsiveness with a nice slowdown at the end.

2. **Avoid Ease-in and Ease-in-out Animation with long times**

   This can be too slow for the user.

3. **For Fast Ease-out effects use Quintic Equations**

4. **Duration**

   Set the animation duration for Ease-outs and Ease-ins around 200ms-500ms.

5. **For Bounce or Elastic effects**

   800ms-1200ms. You need to allow more time for the elastic bouncing part of the animation.

More info:

- Choosing the right easing
- The Basics of Easings

Tools:

- http://cubic-bezier.com/
- Easing Functions

2.5

Animation Optimization

Animation Performance

This tip is very important, because every interface animation must be designed in terms of performance, especially in mobile. When we produce an animation in CSS or Javascript, depending on how the different properties are rendered by the browser, the 60 fps can be optimized for the fluidity of our animation.

Understanding Browser Rendering

We recommend that you have a look at this course by Paul Lewis and Cameron Pittman to better understand how animations are rendered by the browser.

CSS/JS Animation Performance

CSS Animation

Use CSS for simple animations, transitions and to animate DOM elements. Declarative animations are optimized by the browser.

1. **Use Hardware Accelerated Safe Properties:**

   Opacity and Transform (Rotate,Translate, Scale)

2. **Avoid animate styles that affect Layout:**

   Width, height, padding, margin.

3. **Avoid animate styles that affect Paint:**

   Color, background, background-image, border.

Javascript Animation

Use Javascript for complex animations, animate sequences, independent transformations, Canvas, CallBack functions, animate paths.

1. **Use RequestAnimationFrame**
2. **Avoid setTimeOut, setInterval**
3. **Avoid changing inline styles on every frame**
4. **Decouple events from animations**
5. **Avoid reflow and repaint loops**
6. **Use GPU acceleration with Matrix3D Transformations**

More info:

- MicroTip: Flip Animations
- Rendering Performance
- CSS GPU Animation: Doing It Right
- Browser Rendering Optimization Free Curse - Udacity

Animate the correct properties

By **Surma**, Google Developer Advocate

Whenever using CSS Transitions or CSS Animations, try very hard to only use the **opacity** and **transform** properties. All other CSS properties (like width or top) are mostly very slow to animate and will make your animation janky, especially on mobile devices.

Designing for Mobile Performance

3.1

Content Strategy

Traditionally we tend to perceive performance as a task of compression of assets, optimizing server architecture and DB, cache etc, things that, in general, escape the understanding of a designer.

However, the biggest part of the task of optimization begins in the analysis phase of a project. Establishing viable goals, designing the contents, understanding the user path and the way in which the users interact with the app.browser to retrieve files needed for the site)

“Performance is an essential design feature.” Brad Frost

In this first phase, a very useful TIP is to always create a Performance Budget which imposes a series of limitations which we can use from the start to avoid an excess of unnecessary resources.

The Performance Budget

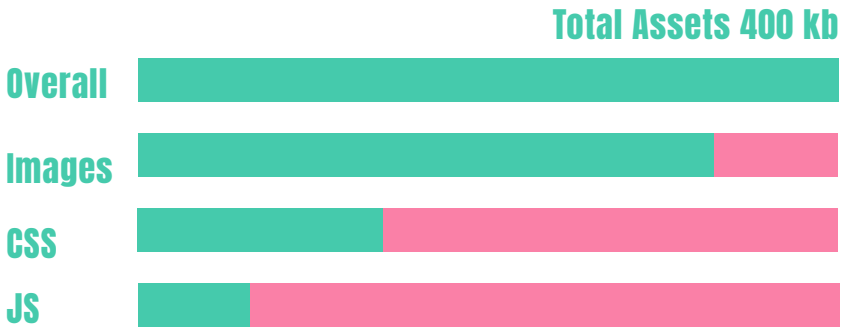
This is composed of a series of limitations which we establish depending on the analysis of our target, their devices and possible access conditions to our site in terms of connectivity. The Performance Budget is composed of the following data.

Page weight, the site size in kilobytes including all assets (HTML documents, CSS images, styles, scripts, videos, fonts, etc.)

Number of HTTP requests (a request made by the user's browser to retrieve files needed for the site)

Sample:

Overall 400 kb /page
80% images
15 request
7% Fonts
Max 7 secs load time over 3G connections



Tools

- Performance Budget Builder
- Performance Budget Calculator

More Info

- Setting a Performance Budget
- Performance As Design
- Design Through the Lens of Performance

3.2

Image Strategy

As we have just tested, a very high percentage, around 70% of a site is composed of images. A big part of the optimization process centres on compression, adapting and managing loading images, but there is a lot of work that must be done beforehand in the area of content design, that is to decide carefully what graphic material is really useful or significant for the brand.

On the next page, a TIP that takes us to the content strategy in relation to images.

Designing an Image Strategy

According to a study by httparchive, a majority of data that makes up the weight of a website is images.

Ewa Gasperowicz developed a 4 point plan for images when designing a site that looks at images according to function;

1. Navigation and action
2. Branding and priority
3. Decorative
4. Informative

Navigation and action images are very important and always need to be present, so circling them in your design shows that they need to be present at all times.

Branding and priority images are images that effect the experience in a way that relates the product to the user. If your brand has a distinct look, images that fulfil that look are quite important so they may need to also be present at all times.

Decorative image are the opposite of the branding images, they merely give a feeling and emotion but are not critical to a design. So when working with a developer consider that these could be replaced with block colours or an alternative design.

Informative images convey a message to the user, perhaps sponsorship of a site or clients and partners that you work with. These are important also but may not be as critical as your own branding and priority images so provide an alternative design that would show the developer an alternative with text or an image / logo that can be repeated and cached by the browser.

More Info

WomenTechmakers.com

3.3

Borrowing Performance Ideas from Native Apps

Use Progress Indicators in Standalone Mode

By **Ewa Gasperowicz**, Developer Programs Engineer

“Use progress bars and widgets when running the app in an app shell or in standalone mode.”

When an app is launched in the standalone mode (e.g. from a link in a homescreen), it does not show the default browser UI, like the URL bar or the menu button. This includes progress bars - if the user performs an action that causes the app to make a (sometimes lengthy) request to the server, there is no default way for them to know that the request was sent and that the app is waiting for a reply. The app might seem “frozen” and it could discourage the user from further interaction. You need to proactively include progress indicators to keep the user informed as to what is happening.

Never Block Page Transitions on the Network and Provide Continuity

By **Owen Campbell-Moore**

When a user taps a button or link in a web app, especially a server rendered web app, they are often made to wait staring at the current screen before suddenly jumping to a whole new screen of content.

Aim to provide the perception that the whole “app” is stored locally on the phone and that only the content is being dynamically loaded, use skeleton screens as placeholders as content is loading, but be sure to remember to reuse any data you already have from the previous page (e.g. the article title, thumbnails etc) to get the illusion of continuity and create a fluid transition into the new content.

Prevent Content Jumping as the Page Loads

By **Owen Campbell-Moore**

All img tags on a page should proactively include the dimensions of the image. This allows the browser to layout the screen correctly even before the image is loaded.

If dimensions aren’t specified on the tag then content will jump when the image is downloaded, which makes for a poor user experience.

Tip: show a placeholder where the image will go, such as a grey square, or a blurred thumbnail of the image if it is available on the client due to being shown on a previous screen. Fading in the image when it is downloaded is also a nice touch of polish.

More Info

- Designing Great UIs for Progressive Web Apps
- Creating UX that “Just Feels Right” with Progressive Web Apps, Google I/O ‘17

3.4

Optimize & Prioritize CSS / Scripts

Deliver the minimal code to make a page useful upfront

By **Addy Osmani**, Engineering Manager at Google

Most websites ship all of the JavaScript a website needs upfront. Think of this as your framework, all your plugins and widgets which can be incredibly costly to load; **not only are you shipping 100s of KBs of script to the user, but the major hidden cost is how long this JavaScript takes to be parsed, compiled and executed by the browser.**

On the powerful desktop machines we build sites on, this JavaScript can boot-up relatively quickly. On underpowered mobile devices however, it can often take 4-5 times as long for JavaScript to boot-up. This can leave a user waiting a long time before any UI can be interacted with. In our studies, anywhere up to 19 seconds.

To work around this problem, consider splitting your JavaScript up and only serving down the minimal code needed to make the current page useful to the user. Non-critical code can be lazily loaded in or added on the routes or pages that absolutely need it. This can shave seconds off your load time, allowing your user to interact with your experience more quickly.

Preload Critical Resources

By **Addy Osmani**, Engineering Manager at Google

Browsers like Chrome load resources like JavaScript, Web Fonts and images with different priorities. We usually make a best guess of what we think you'll need first. As a web page author however, you know more about the assets that are critical to load early on in your page than we do.

"As a web page author however, you know more about the assets that are critical to load early on in your page than we do."

To improve the load performance of specific resources, you can take advantage of `<link rel="preload">` to preload assets that might be discovered late by a browser that are important to load sooner than others. Examples might be Web Fonts that are critical for your main content, JavaScript bundles critical for a page to get interactive or images that might be a large part of your user experience.

`<link rel="preload">` is supported in Chrome and Blink-based browsers like Opera.

Inline Critical CSS

By **Prateek Bhatnagar**, UX Engineer

While serving markup from server, try to put the CSS that you expect the current page to use (aka Critical CSS) inside a style tag in the header section. This allows the browser to paint the page without making any network call and thus is super quick. This boosts the first paint time of your app. With this technique while the browser renders the DOM for you, it already has the respective styling for it.

If you sent these styles in a css file instead, the browser will make a network call and wait for it before it can apply this CSS to your page.

Use Passive Event Listeners

By **Prateek Bhatnagar**, UX Engineer

When you want to add a touch event listener to your page and don't want to use preventDefault on it, make sure you add Passive event listeners. This helps browsers know that you have no intent of calling preventDefault. If not done like so, browsers will wait for the added listeners to finish before they can scroll the page and hence your users might feel a jank while scrolling your web page.

Eliminate render-blocking resources in above-the-fold content

By **Luciano Borromei**, UX Engineer at Ingamana

The term “above-the-fold” is derived from the print media industry and describes the upper half of the front page of a newspaper, the one that is visible with the paper folded.

In web development it refers to the portion of a webpage that is visible without further user interaction like scrolling.

Since nowadays screen sizes varies drastically, there is no fixed definition for the number of pixels that determines the fold: it just make reference to the initially visible contents of the viewport size.

Loading contents that are not directly involved in the rendering of this prioritized visible content causes a delay that is proved impacts directly in conversion rates and in the user’s speed perception of the website.

Optimization Tips:

1. **Load the critical above-the-fold content of your page first.**
2. **Defer or asynchronously load ATF blocking resources, or inline the critical portions of them.**
3. **Reduce the amount of data required to render your page by minifying resources and enabling compression for every HTTP request.**

Once this initial part of the page is loaded you can instruct the browser to load the rest of the CSS, JS, media and other assets required to render the complete page

More Performance Optimization TIPS

4.1

Accelerated Mobile Pages

The Accelerated Mobile Pages (AMP) provide a new standard, built on top of existing web technologies, to enable blazing-fast page rendering and content delivery.

1. Speed - Pages load instantly which means more engagement.

2. Openness - It works everywhere, cross platform.

3. Control - It's just HTML - you control how your content looks and feels.

The way it works is by executing all AMP JavaScript asynchronously, preventing JS files from blocking the HTML page from loading. It then loads the layout of the page without waiting for any resources, such as images, to download. There are many more features AMP uses to beef up the speed of a HTML page which you can learn about here.

Some successful examples of executed AMP sites include The Washington Post, which saw a 23% increase in returning users when they switched to a AMP. They also saw a 88% improvement in load time for the AMP site over their traditional mobile web experience.

Also news site Mynet increased the speed of their mobile site by 4X, increasing revenue by 25% with AMP-based PWA. They also saw a 43% longer average time on site and reduction of bounce rate by 24%.

You can learn more about AMP on our tutorials page here and also read success stories with AMP at Web Fundamentals.

4.2

Progressive Web Apps

A Progressive Web App (PWA) is a responsive web app that uses modern web capabilities to deliver an app-like experience to users. The look and feel of a PWA is very similar to a native mobile app, in terms of reaction times, animation, transitions, etc.

Checklist before you begin migrating to PWA

1. **Implement HTTPS:** Just do it.
2. **Enable browser caching:** Set up correct response headers.
3. **Avoid blocking JavaScript:** Move script elements to the bottom of the page and/or add async or defer attributes.
4. **Remove links to unused JavaScript libraries:** Don't link to JavaScript you don't need.
5. **Avoid including JavaScript libraries more than once:** Check that you don't link to both minified and 'full-fat' versions, or multiple versions of the same library.
6. **Remove unused CSS and unused JavaScript:** Pay down technical debt!
7. **Reduce JavaScript dependencies:** With simple refactoring, you might not need to use JavaScript libraries.
8. **Use the right image format:** JPEG or WebP for photos, SVG for PNG for logos or icons.

9. **Compress images:** Check for images that should be saved with greater compression: anything over about 10KB for small images, 100KB for large images. Resave then optimise with tools such as ImageOptim. Hero images are the most likely offenders.

10. **Size images correctly:** Check for images with pixel dimensions too large for display size. Resave at appropriate sizes.

11. **Optimise video resolution:** If you use video, check that resolution is no larger than required for the display size. Re-encode as necessary.

12. **Include a meta viewport tag:** `<meta name="viewport" content="width=device-width,minimum-scale=1.0">`

1. HTTPS

Every site should deliver all assets over HTTPS.

Why

- HTTPS is fundamental for security, content integrity and authentication.
- Many APIs won't work without it.

Check

- Use the Chrome DevTools Security panel to check that all assets for your site are delivered via HTTPS.

Fix

- Use a simple service like CloudFlare (though, strictly speaking, this does not implement end-to-end encryption).
- Purchase a security certificate (or get a free one) and enable HTTPS on your server.

Learn

- Why HTTPS matters
- Enabling HTTPS on Your Servers
- Secure your site with HTTPS

2. Enable browser caching

All cacheable resources should be delivered with the appropriate Cache-Control headers.

Why

- Cache-Control headers enable browsers to load previously downloaded resources from the local cache rather than from the network.
- Requests to the network use battery, incur data cost and add to server load.
- Responses from the local cache are much faster and more reliable than from the network.

Check

- Run PageSpeed Insights for your site or check response headers from the Chrome DevTools network panel.

Fix

- Ensure your server sets an expiry date or a maximum age in HTTP headers.

Learn

- Leverage browser caching

3. Avoid blocking JavaScript

Put all script elements at the bottom of the page and/or add a defer or async attribute.

Why

When the browser encounters a script element without a `defer` or `async` attribute, it has to pause page rendering to parse and execute JavaScript.

- Blocking JavaScript is the main reason for slow load times for many sites.
- 53% of users abandon sites that take longer than three seconds to load.

Check

- Run PageSpeed Insights for your site.

Fix

- Add an async or defer attribute to every script element and/or move all script elements to the bottom of the page, just before the closing `</body>` tag.

Learn

- Remove Render-Blocking JavaScript

4. Remove links to unused JavaScript libraries

Remove links to JavaScript libraries you don't use.

Why

- Every JavaScript file you include incurs data cost and a resource request.
- All JavaScript must be parsed and executed — even if it's not used elsewhere.

Check

- Use the Chrome DevTools network panel to check for for redundant library downloads.

Fix

- Remove links to JavaScript you don't use, or code that programmatically loads unused scripts.

Learn

- Analyze Runtime Performance

5. Don't include JavaScript libraries more than once

Make sure not to link to both minified and unminified versions of JavaScript libraries, or both new and old library versions.

Why

- A surprising number of sites link to the same JavaScript library more than once.
- Linking to multiple versions of the same library can cause unexpected behaviour.
- Every resource request adds to data cost, power usage and server load.
- All JavaScript must be parsed and executed, whether or not it's used, thereby adding to page load time.

Check

- Use the Chrome DevTools network panel to check for JavaScript resources.

Fix

- Remove links to redundant old or unminified versions of JavaScript files.

Learn

- Analyze Runtime Performance

6. Remove unused code

Remove CSS or JavaScript that's no longer used by your site.

Why

- CSS and JavaScript code incurs a download and parsing cost even if it's not used.
- Unused code adds to project complexity and maintenance cost.

Check

- Check for unused CSS: use the uncss Node module, record CSS coverage from Chrome DevTools, or use the Chrome DevTools Audits panel (or Legacy Audits from Chrome 60).
- Check for unused JavaScript: used the Chrome DevTools coverage tool.

Fix

- Remove unused code — but be careful to avoid unwanted side effects!

Learn

- CSS and JS code coverage

7. Reduce JavaScript dependencies

With simple refactoring you may be able to remove dependencies on one or more JavaScript libraries.

Why

- You may be loading JavaScript libraries that you don't really need, such as outdated polyfills, utilities with minimal usage, or unused UX frameworks.
- Every resource request adds to code complexity, data cost, power usage and server load.
- All JavaScript must be parsed and executed, whether or not it's used, thereby adding to page load time.

Check

- Use Chrome DevTools to check for JavaScript library downloads.
- Check caniuse.com to find out if polyfills are necessary for your target platforms.

Fix

- Where possible, refactor code to remove dependencies, then remove code or links that load unnecessary libraries.

Learn

- Analyze Runtime Performance

8. Use the right image formats

Use the appropriate format for images, depending on their content: WebP or JPEG, SVG or PNG.

Why

- Incorrect use of image formats is a major cause of data bloat.
- In terms of file size, WebP and JPEG are much more efficient for photographic images than PNG.
- SVGs are inherently responsive (they can shrink or expand to fit) and far more efficient for vector graphics than other formats.
- Inline SVGs can significantly reduce file requests.

Check

- Use browser tools to check image formats.
- Look for large image files in the Network panel of your browser tools.
- Check for PNG images such as icons or logos that could be replaced with inline SVGs.

Fix

- Photos should be WebP or JPEG; vector images such as logos or icons should be SVG or PNG.
- If you use PNG for transparency, try WebP instead. JPEG with an opaque background works well enough in many cases.
- Try using PNG-8 instead of PNG-24.

- Remove redundant images! Avoid images wherever possible.

Learn

- Image Optimization
- Udacity Responsive Images course (it's free!)

9. Compress images

Compress images as much as possible. Pay particular attention to 'hero' images.

Why

- Images constitute by far the most weight and most requests for most web pages.
- For many people high data cost is often a greater barrier to access than poor connectivity, especially for users on capped data plans.

Check

- From the Network panel of your browser tools, check for the largest image file sizes.
- As a rule of thumb, large images over 100KB and small images over 10KB can probably be resaved with higher compression without visible quality degradation.

Fix

- Use the highest possible compression value (lowest quality) and work up until you get an acceptable result.
- Use your browser tools to check for heavy image files, and individually optimise static images such as banners and backgrounds.
- For dynamically saved images, try increasing compression (reducing quality) with your workflow tools.
- Optimise images with lossless tools such as ImageOptim.
- Optimise SVGs with a tool such as as SVGOMG.

Learn

- Image Optimization
- Udacity Responsive Images course (it's free!)

10. Size images correctly

Save images using pixel dimensions appropriate for the display size and pixel density.

Why

- Saving images with minimum possible pixel size can significantly reduce data cost.
- Small increases in pixel dimensions result in big increases in memory usage. With images on mobile — especially on low-spec devices — memory can become the new bottleneck.

Check

- From your browser tools, check the natural (saved) size of an image compared with the display size. If you're using a 1x display, such as most desktop monitors, these two

sizes should match. For a 2x display, such as on high-spec phone or laptop, the image dimensions should be double the display size — and so on.

- If you inspect an img element from the browser console, you can compare naturalWidth and naturalHeight properties (saved size) with clientWidth and clientHeight (displayed size).
- Avoid stretching or squashing images. You can check this with the Chrome Image Checker extension.

Fix

- Resave individual images using the correct dimensions.
- Consider using responsive images. The widely supported srcset attribute makes this extremely simple:

```
<img src="small.jpg" srcset="medium.jpg 1000w, large.jpg 2000w" alt="My image">
```

Learn

- Image Optimization
- Udacity Responsive Images course.

11. Use the right video resolution

Ensure that encoded video frame dimensions are no larger than necessary for the display size.

Why

- There is no point in delivering video at a higher resolution than the largest size it will be displayed on a web page.
- A video with a resolution even slightly larger than required will result in a significant number of wasted bytes, adding to data cost and increasing the likelihood of video buffering.

Check

- If a `video` element from the browser console, and compare `videoWidth` and `videoHeight` properties with `clientWidth` and `clientHeight`.
- Don't squash or stretch videos: check for videos with aspect ratio different from the video.

Fix

- Deliver videos using the correct resolution.
- If necessary use media queries or adaptive streaming techniques to deliver the correct resolution

Learn

- developers.google.com/web/media
- Video on mobile

12. Include a meta viewport tag

Add a valid meta viewport tag to every page on your site.

Why

- Without a meta viewport tag, web pages will be displayed at a typical desktop screen width, scaled to fit the viewport — which is probably not what you want.

Check

- Check for a valid meta tag in the head element.

Fix

- The meta viewport tag for your site should look like this:

```
<meta name="viewport" content="width=device-width,minimum-scale=1.0">
```

- Make sure not to use a maximum-scale value, since that disables zoom.

Learn

- Configure the Viewport
- Responsive Meta Tag
- Using the Viewport Meta Tag

Maybe

Avoid monolithic JavaScript files

- Bundling JavaScript can be a good way to reduce the number of resource requests, but it can also result in huge files that take time to parse and execute.

Avoid blocking CSS

- nline CSS manually or during your build process
- Separate out CSS not required for initial page load, then retrieve additional CSS later

THANKS!

We would like to thank the following authors for their collaboration on this eBook:

Ewa Gasperowicz, Developer Programs Engineer at Google
Owen Campbell-Moore, Product Marketing at Google
Addy Osmani, Engineering Manager at Google
Prateek Bhatnagar, UX Engineer at Google
Sam Dutton, Developer Advocate, Chrome at Google
Luciano Borromei UX Engineer at Ingamana
Paul Bakaus, Developer Advocate, AMP at Google

Special thanks to:

Lionel Mora, Product Marketing at Google
Mustafa Kurtuldu, Design Advocate at Google


Google
AWWWARDS®