Vestibular Disorders, Reduced Motion Media Query and Video Backgrounds

I wasn't aware of Vestibular Disorders being an issue with animations on the web but it's a big part of the disabilities we need to consider when working on the web. It's important to remember that...

> People with vestibular disorders have a problem with their inner ear. It affects their balance as well as their visual perception of their world around them.
>
> Sometimes the sensation lasts only a short while, but others can suffer it for years. Walking becomes a challenge and they have a constant risk of falling. Concentration is diminished leaving the sufferer unfocused and often unproductive. It is often viewed as a "hidden" disability because it has no outward showing symptoms.

So we really want to avoid that kind of problems for our users.

Currently we can use the `preferes-reduced-motion` media query to test if the reduced motion settings are enabled (Safari only) and reduce or disable animations from your page. If the browser doesn't understand the query it'll skip the query and its content altogether.

```
@media (prefers-reduced-motion) {
  .background {
    animation: none;
  }
}
```

Val Head's Desgigning Safer Web Animations For Motion Sensitivity she outlines some examples of sites that cause issues for users who experience Vestibular

Disorders, some of the issues that trigger Vestibular Disorders and some solutions to address these problems.

Perhaps the most important thing to learn about learning about accessibility and the web is the closing quote on Val's article:

> On the web, more than in any other medium, the flexibility and control are there for you to design creatively and responsibly at the same time. We absolutely can innovate and push the web forward designing kick-ass interface animations while still being responsible designers. As a web animator, you can have your animation cake and eat it too—with a little creative thinking.

The `playsinline` attribute and iOS

Moving on we'll look at a (sort of) new attribute for iOS video playback: `playsinline`.

Older versions of iOS required you to tap the video before playback would begin. This was done to prevent unnecessary battery usage and to avoid random, and sometimes multiple, video playback on the page if autoplay was enabled for the videos.

iOS 7 introduced `webkit-playsinline` as a way to relax the requirement for video playback with a gesture.

With iOS 10 Apple has further relaxed the requirements for automatic video playback. The short version of the new video policies for iOS 10 is as follows:

- `<video autoplay>` elements will now honor the autoplay attribute, for elements which meet the following conditions:
    - `<video>` elements will be allowed to autoplay without a user gesture if their source media contains no audio tracks
    - `<video muted>` elements will also be allowed to autoplay without a user gesture
    - If a `<video>` element gains an audio track or becomes un-muted without a user gesture, playback will pause
    - `<video autoplay>` elements will only begin playing when visible on-screen such as when they are scrolled into the viewport, made

visible through CSS, and inserted into the DOM
        - `<video autoplay>` elements will pause if they become non-visible, such as by being scrolled out of the viewport
- `<video>` elements will now honor the play() method, for elements which meet the following conditions:
    - `<video>` elements will be allowed to `play()` without a user gesture if their source media contains no audio tracks, or if their muted property is set to true
    - If a `<video>` element gains an audio track or becomes un-muted without a user gesture, playback will pause
    - `<video>` elements will be allowed to play() when not visible on-screen or when out of the viewport
    - `video.play()` will return a Promise, which will be rejected if any of these conditions are not met
- On iPhone, `<video playsinline>` elements will now be allowed to play inline, and will not automatically enter fullscreen mode when playback begins
    - `<video>` elements without playsinline attributes will continue to require fullscreen mode for playback on iPhone
    - When exiting fullscreen with a pinch gesture, `<video>` elements without playsinline will continue to play inline

So, now that we know how to auto play a video in mobile (at least some of them) and desktop we'll dive into background videos.

Chrome for Android

As of version 53 Chrome for Android supports muted autoplay on mobile. This means that the background video will work in Chrome as wel as in Firefox and UC Browsers where it has worked without a problem (Chrome was the only browser that restricted video autoplay in Android devices) but now that Chrome plays the same game we get wider support.

Some things to remember when testing the feature with Chrome in Android:

- From an accessibility viewpoint, autoplay can be particularly problematic. Chrome 53 and above on Android provides a setting to disable autoplay completely at the OS level
- Autoplay for audio is disabled on Chrome on Android, muted autoplay doesn't make sense for audio

- There is no autoplay if Data Saver mode is enabled. If Data Saver mode is enabled, autoplay is disabled in Media settings
- Muted autoplay will work for any visible video element in any visible document, iframe or otherwise
- To take advantage of the new behaviour, you'll need to add muted as well as autoplay

Background videos

The idea of using a video as a background to text is intriguing. I like the idea of providing additional context using motion video rather static images. At the same time we also have to be careful and mindful of how we use the video so as not to trigger vestibular disorders and keeping in mind that the video will not have audio so we can't rely on an audio content for the background.

Given those constraints we can still do video background.

We build the video element with playsinline, autoplay and muted attributes. These attributes will make sure that the video will autoplay in iOS by respecting the requirements for autoplay in iOS 10. I guess if I wanted to be absolutely sure I'd also include `webkit-playsinline` to account for older versions of iOS.

```
<video poster="images/tron-bg.jpg" id="bgvid"
       playsinline autoplay muted>
  <source src="video/tron-bg.webm" type="video/webm">
  <source src="video/tron-bg.mp4" type="video/mp4">
</video>
```

This is half the magic, the other half happens in CSS.

CSS

In CSS we style the content to make sure that it works reliably across browsers. We do a quick margin reset and set the background for the page to white. Users should not see the white background under any circumstance so the big red flag appears when they do.

```
body {
```

```
    margin: 0;
    background: #fff;
}
```

We then set the video in the page and make it full width and full height. This is a combination of methodologies to center content:

- We center the content using absolute positioning and then translate the content up and to the left
- We set width and height to auto and constrain it to 100% of the width and height of the window

It makes the content fixed so that it won't matter how large the content is the video will not scroll.

It gives the video a negative z-index lower than any other content on the page.

finally we give the video a background element and make it the same as the poster and the first frame of the video.

```
video {
    position: fixed;

    top: 50%;
    left: 50%;
    transform: translateX(-50%) translateY(-50%);

    min-width: 100%;
    min-height: 100%;
    width: auto;
    height: auto;

    z-index: -1000;

    background: url('../images/tron-bg.jpg') no-repeat;
    background-size: cover;
    transition: 1s opacity;
}
```

`stopfade` is a class that only holds opacity. When we get to Javascript we'll toggle the class to produce an animation like effect of transforming opacity.

```
.stopfade {
    opacity: .5;
}
```

The remaining selectors control the content area laid over the video.

```
#content {
    font-family: "Roboto""Roboto"Sans", sans-serif;
    font-weight:100;
    background: rgba(0,0,0,0.3);
    color: white;
    padding: 2rem;
    width: 50%;
    margin:2rem;
    float: right;
    font-size: 1.2rem;
}
h1 {
    font-size: 3rem;
    text-transform: uppercase;
    margin-top: 0;
    letter-spacing: .2rem;
}
#content button {
    display: block;
    width: 80%;
    padding: .4rem;
    border: none;
    margin: 1rem auto;
    font-size: 1.3rem;
    background: rgba(255,255,255,0.23);
    color: #fff;
    border-radius: 3px;
    cursor: pointer;
```

```
    transition: .3s background;
}
#content button:hover {
    background: rgba(0,0,0,0.5);
}

a {
    display: inline-block;
    color: #fff;
    text-decoration: none;
    background:rgba(0,0,0,0.5);
    padding: .5rem;
    transition: .6s background;
}

a:hover{
    background:rgba(0,0,0,0.9);
}
```

Finally we use two media queries to control what happens when the width of the window hits certain break points. If the window is smaller than 500 pixels wide we change the width of the content area to 70% of the width of the window.

When the width of the device is no more than 800 pixes then we remove the video from the page and add the poster image as a background element for the root element of the page (HTML)

```
@media screen and (max-width: 500px) {
    div{width:70%;}
}

@media screen and (max-device-width: 800px) {
    html {
      background: url('../images/tron-bg.jpg')
                  #000 no-repeat center center fixed;
    }

    #bgvid {
```

```
      display: none;
    }
}
```

Javascript

The Javascript handles interactivity and the special case of reduced motion.

After defining constants for the video and the play/pause button we handle the reduced motion media query match. `window.matchMedia` is the programmatic way to test if a media query matches the current environment. if it does then we remove the `autoplay` attribute of the video element; we don't want the video to play automatically if it may cause problems for our users; we then set the button's text to paused to indicate the video is not playing.

```
const vid = document.getElementById('bgvid');
const pauseButton = document.querySelector('#content button');

if (window.matchMedia('(prefers-reduced-motion)').matches) {
  vid.removeAttribute('autoplay');
  vid.pause();
  pauseButton.innerHTML = 'Paused';
}

function vidFade() {
  vid.classList.add('stopfade');
}
```

The rest of the script sets up events for the user to interact with.

When the video ends we want to pause the video and call the `vidFade()` function to change the opacity of the video by toggling a CSS class on and off.

```
vid.addEventListener('ended', function()
{
// only functional if 'loop' is removed
  vid.pause();
```

```
// to capture IE10
  vidFade();
});
```

Next we register a click event handler for the video and toggle between play and paused states using the button at the end of our content session. We also register a `keypress` event to handle keyboard pausing using space and enter.

```
pauseButton.addEventListener('click', () => {
  vid.classList.toggle('stopfade');
  if (vid.paused) {
   'stopfade');
    pauseButton.innerHTML = 'Pause';
  } else {
    vid.pause();
    pauseButton.innerHTML = 'Paused';
  }
});


'Pause'// Event handler for keyboard navigation
window.addEventListener('keypress', (e) => {
  switch (e.which) {
    case 32:
    case 13:
      e.preventDefault();
      if (vid.paused) {
        vid.play();
        pauseButton.innerHTML = 'Pause';
      } else {
        vid.pause();
        pauseButton.innerHTML = 'Paused';
      }
      break;
  }
});
```

Final thoughts

This is a nice way to enhance the the content of a page however there are a few things we need to keep in mind:

- Don’t just use this technique because you can. The video should enhance the message of your content or it’s just a distraction
- The video will autoplay, but it should be muted by default; ideally, it should not include sound at all
- Be mindful of mobile devices: many phones and tablets disable autoplay on videos to save bandwidth and battery. See the section on `playsinline` for a discussion on how this requirement is relaxed in iOS 10 and `Chrome for Android` for what Chrome supports
- Consider the video’s length is important
    - If it’s too short a video can feel repetitive (as most such videos will be set to loop)
    - If the video is too long it becomes a narrative unto itself, and deserves to be a separate design element
- Accessibility is essential: Make sure that any text you place on top of the video has a high contrast ration to the video
    - Users should have easy access to a UI control to pause the video
    - Ideally, the video should play through only once.
- Bandwidth is a big deal. The video needs to be small, and compressed as effectively as possible
    - At the same time, it needs to scale across different devices and their associated screens
    - For high end experiences you may consider (unencrypted) DASH video with multiple bitrates to serve different devices

Links and resources

- Understanding Vestibular Disorders
- Designing Safer Web Animation For Motion Sensitivity
- An Introduction to the Reduced Motion Media Query
- The A11Y Project