Web Performance Review

Differences Between Real and Perceived Performance

- **Perceived Performance**: This is how users perceive the performance of a website. It's how they evaluate it in terms of responsiveness and reliability. This is a subjective measurement, so it's hard to quantify it, but it's very important, since the user experience determines the success or failure of a website.
- **Real Performance**: This is the objective and measurable performance of the website. It's measured using metrics like page load time, server response time, and rendering time. These measurements are influenced by multiple factors related to the network and to the code itself.

Techniques for Improving Perceived Performance

- **Lazy Loading**: This technique reduces the initial load time as much as possible by loading non-essential resources in the background.
- **Minimize Font Delays**: If your website has custom fonts, you should also try to minimize font loading delays, since this may result in flickering or in showing the fallback font while the custom font is being loaded. A suggestion for this is using a fallback font that is similar to the custom font, so in case this happens, the change will be more subtle.
- **Use of Loading Indicators**: Showing a loading indicator for a long-running process as soon as the user clicks on an element can help the user feel connected and engaged with the process, making the wait time feel shorter.

Core Performance Concepts

Source order: This refers to the way HTML elements are structured in the document. This determines what loads first and can significantly impact performance and accessibility.

Some best practices for source order include:

- Placing critical content such as headings, navigation or main text higher in the HTML structure.
- Deferring non-essential scripts such as ones for analytics, or third-party widgets, so they don't block rendering.
- Using progressive enhancement, to ensure the core experience works even before styles and scripts load. Progressive enhancement is a way of building websites and applications based on the idea that you should make your page work with HTML first.

Here is an example of good source order, using the best practices we just went through:

```
<h1>Welcome to FastSite!</h1>
<p>Critical information loads first.</p>
<script src="slow-script.js" defer></script>
```

- **Critical Rendering Path**: This is the sequence of steps the browser follows to convert code into pixels on the screen.
- **Latency**: This is the time it takes for a request to travel between the browser and the server. So in other words, high latency equals slow pages.

Some ways of reducing latency include:

- Using CDNs, or Content Delivery Networks, to serve files from closer locations.
- Enabling compression using things such as Gzip to reduce file sizes.
- Optimizing images and using lazy loading.

```
<img src="placeholder.jpg" data-src="real-image.jpg" loading="lazy">
```

Improving INP

Definition: INP (Interaction to Next Paint) assesses a page's overall responsiveness by measuring the time from when a user interacts, like a click or key press, to the next time the browser updates the display. A lower INP indicates a more responsive page.

Here are some ways to improve INP:

- Reduce main thread work by breaking up long JavaScript tasks.
- Use `requestIdleCallback()` for non-critical scripts. This will queue a function to be called during a browser's idle periods.
- Defer or lazy-load heavy assets which were covered earlier.

- Optimize event handlers. If these handlers run too frequently or perform heavy operations, they can slow down the page and increase INP. The solution for this is debouncing. Debouncing ensures that the function only runs after the user stops typing for a short delay - so for example 300ms. This prevents unnecessary calculations and improves performance.

How Rendering Works in the Browser

How Rendering works: First the browser parses the HTML and builds the DOM. Next, the browser processes the CSS, constructing the CSS Object Model, or CSSOM. This is another tree structure that dictates how elements should be styled. Finally, the browser paints the pixels to the screen, rendering each element based on the calculated styles and layout. In complex pages, this might involve multiple layers that are composited together to form the final visual output.

How Performance Impacts Sustainability

Background Information: The internet accounts for around 2% of global carbon emissions—that's the same as the airline industry! Every byte transferred requires electricity, from data centers to user devices. Larger files and inefficient scripts mean more power consumption. A high-performance website isn't just faster, it also reduces unnecessary processing and energy use.

Ways to Reduce Page Loading Times

- **Optimize Media Assets**: Large images and videos are common culprits for slow load times. By optimizing these assets, you can significantly speed up your site. This includes things like compressing images, using modern formats like WebP and using lazy loading for assets.
- **Leverage Browser Caching**: Caching allows browsers to store parts of your website locally, reducing load times for returning visitors.
- **Minify and Compress Files**: Reducing the size of your files can lead to quicker downloads. This includes reducing the size of transmitted files and minifying CSS and JavaScript files.

Improving "time to usable"

Definition: "time to usable" is the interval from when a user requests a page to when they can meaningfully interact with it. To improve "time to usable" you can lazy load your asset or minimize render-blocking resources.

Key Metrics for Measuring Performance

- **First Contentful Paint or FCP**: It measures how quickly the first piece of content—text or image—appears on the screen. A good FCP is regarded as a time below 1.8 seconds, and a poor FCP is above 3 seconds. You can check your FCP using Chrome DevTools and checking the performance tab.
- **Total Blocking Time**: This shows how long the main thread is blocked by heavy JavaScript tasks. If Total Blocking Time (TBT) is high, users experience sluggish interactions. To improve TBT, break up long tasks and defer non-essential scripts.
- **Bounce Rate**: This is the percentage of visitors who leave without interacting. If your site has high bounce rates it might be because your page is too slow.
- **Unique Users**: This metric tracks how many individual visitors come to your site. To view the Bounce Rate and Unique Users, you can use Google Analytics. It will allow you to monitor these metrics and improve engagement.

Common Performance Measurement Tools

- **Chrome DevTools**: Chrome DevTools is a built-in tool inside Google Chrome that lets you analyze and debug performance in real-time. DevTools will show loading times, CPU usage, and render delays. It's especially useful for measuring First Contentful Paint, or FCP, is how fast a user sees the first visible content. If your website feels slow, DevTools will help you spot the bottlenecks.
- **Lighthouse**: This is an automated tool that checks performance, SEO, and accessibility.
- **WebPageTest**: This tool lets you test how your site loads from different locations and devices. This tool gives you a detailed breakdown of your site's Speed Index, Total Blocking Time, and other key performance metrics. If you want to know how real users experience your site globally, WebPageTest is the tool for that.
- **PageSpeed Insights**: This tool analyzes your website and suggests quick improvements for both mobile and desktop. It will tell you what's slowing your site down and give specific recommendations like optimizing images, removing render-blocking scripts, and reducing server response times. PageSpeed Insights is a fast and easy way to check how Google sees your site's performance.
- **Real User Monitoring**: RUM tools track actual user behavior, showing how real visitors experience your site. Popular RUM tools include Google Analytics, which tracks page load times and bounce rates, and New Relic or Datadog, which monitor real-time performance issues. If you want data from actual users, RUM tools are essential.

Working with Performance Web APIs

- **Definition**: Performance Web APIs let developers track how efficiently a webpage loads and responds directly in the code. These APIs allow you to measure page load times, track rendering and interaction delays and analyze JavaScript execution time.
- `performance.now()`: This API gives you high-precision timestamps (in milliseconds) to measure how long different parts of your site take to load.

```
const start = performance.now();
// Run some code here
const end = performance.now();

console.log(`Execution time: ${end - start}ms`);
```

Performance Timing API: This API gives you a breakdown of every stage of page loading, from DNS lookup to `DOMContentLoaded`.

```
const timing = performance.timing;

const pageLoadTime = timing.loadEventEnd - timing.navigationStart;
console.log(`Page load time: ${pageLoadTime}ms`);
```

`PerformanceObserver`: This API listens for performance events such as layout shifts, long tasks, and user interactions.

```
const observer = new PerformanceObserver((list) => {
  list.getEntries().forEach((entry) => {
    console.log(`Long task detected: ${entry.duration}ms`);
  });
});

observer.observe({ type: "longtask", buffered: true });
```

Techniques for Improving CSS Performance

CSS Animations: Animating certain CSS properties, such as dimensions, position, and layout, triggers a process called "reflow", during which the browser recalculates the position and geometry of certain elements on the page. This requires a repaint, which is computationally expensive. Therefore, it's recommended to reduce the number of CSS animations as much as possible or at least give the user an option to toggle them on or off.

Techniques for Improving JavaScript Performance

- **Code Splitting**: Splitting your JavaScript code into modules that perform critical and non-critical tasks is also helpful. This way, you'll be able to preload the critical ones as soon as possible and defer the non-critical ones to render the page as fast as possible.
- **DOM Manipulation**: Remember that DOM Manipulation refers to the process of dynamically changing the content of a page with JavaScript by interacting with the Document Object Model (DOM). Manipulating the DOM is computationally expensive. By reducing the amount of DOM manipulation in your JavaScript code will improve performance.

Assignment

Review the Web Performance topics and concepts.