R/GA

Lazy-Loading

Summary

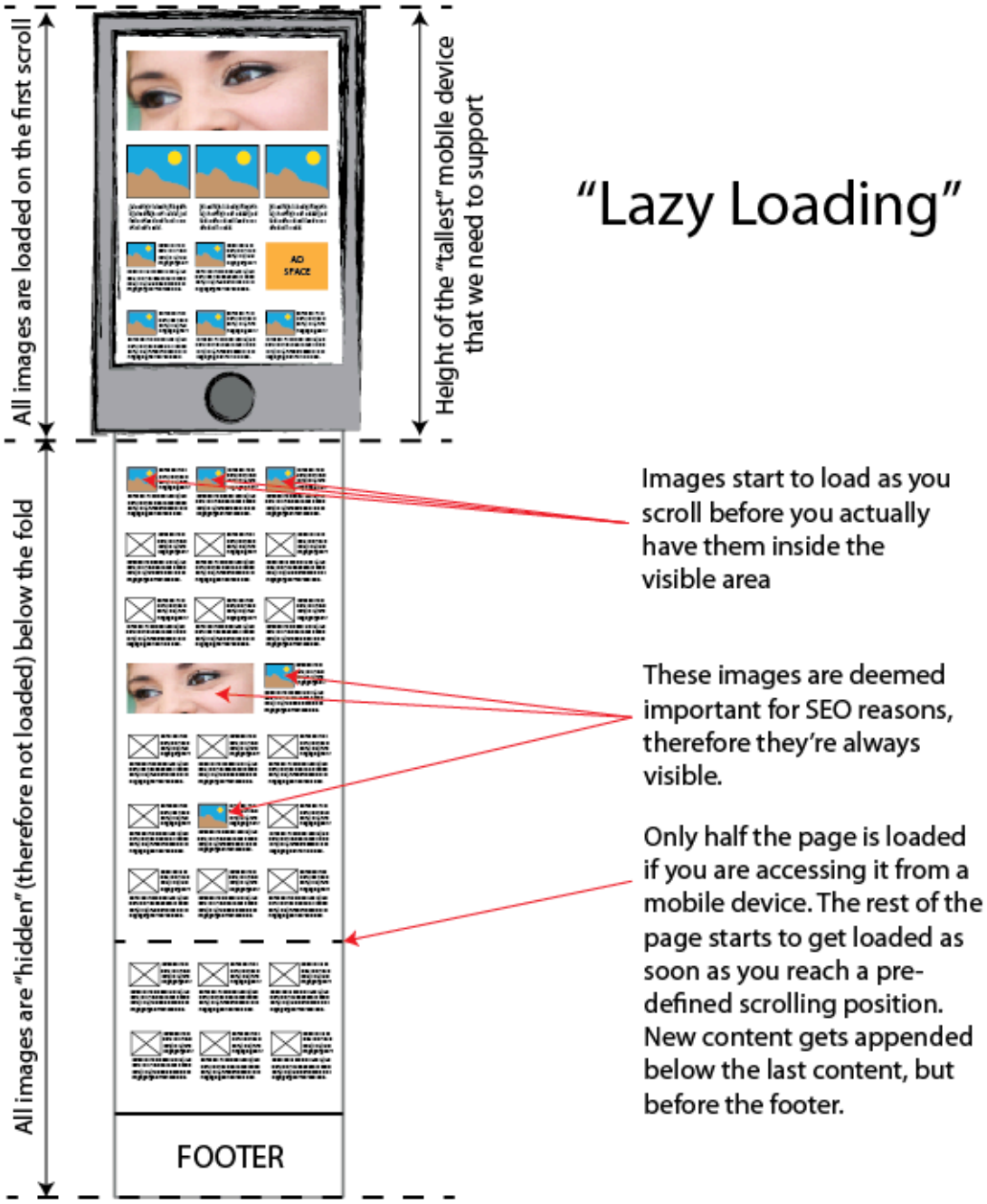
In essence, lazy-loading is a technique to load content dynamically as you scroll the page -or in other words- to load content on-demand. Loading content on-demand has several benefits among which one can mention that it can:

- Reduce the number of unnecessary download of content that is never consumed, saving bandwidth and transfer costs in the long term to both the consumer as well as the provider of the service
- A by-product of the reduction of downloads is an increase in the transfer speed of content, as the number of things that need to be downloaded is smaller
- Thanks to the increased download speed of the page, it will produce a boost in the number of visits, page views and reduce the bounce rate (*although this last point is a bit debatable*)
- Finally, it will help reduce the dilution of “crawling/ranking priority” with images, allowing images located within important modules to rank higher in search engines

On mobile devices, we can use lazy-loading to load entire modules on demand. Data download speeds are particularly slow in Latin America, which is why we consider this to be a viable solution to a serious problem. With this approach, for example, mobile users would only need to download the “first” part of the website, while the rest would be downloaded on demand when (and if) they decide to scroll down.

How it works

The following graphic explains how lazy loading works:



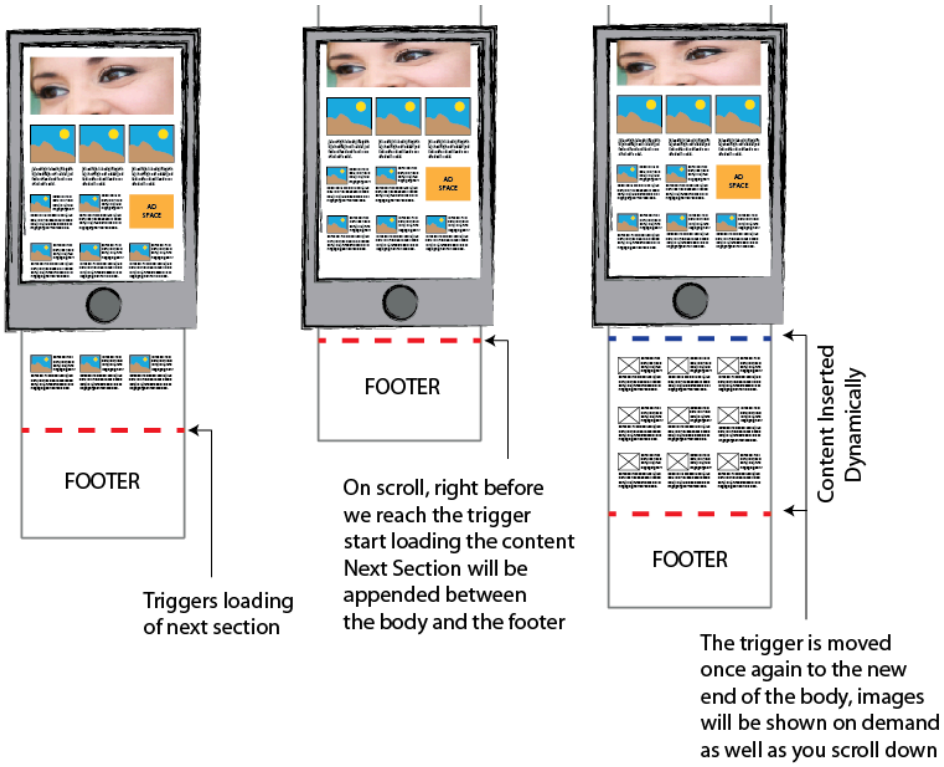
The Implementation

Implementing lazy-loading can be as simple, or as complex, as one wants it to be. A simple implementation won't be very flexible but will be fast to develop and easy to maintain, while a complex implementation will provide a greater degree of freedom, but the development and maintenance times will be longer.

Mobile Devices

A simple implementation would consist of just hiding all images below the fold and just loading them as you scroll down. While this is better than not having lazy-loading at all, it doesn't provide all the benefits of a more robust, yet more complex, implementation (*discussed below*). The main advantage of this simple approach is that it doesn't require any work at all done on the back end.

Our recommendation, a complex and full-featured implementation, involves the back end as well, because it changes the page loading process depending on which device you're using.



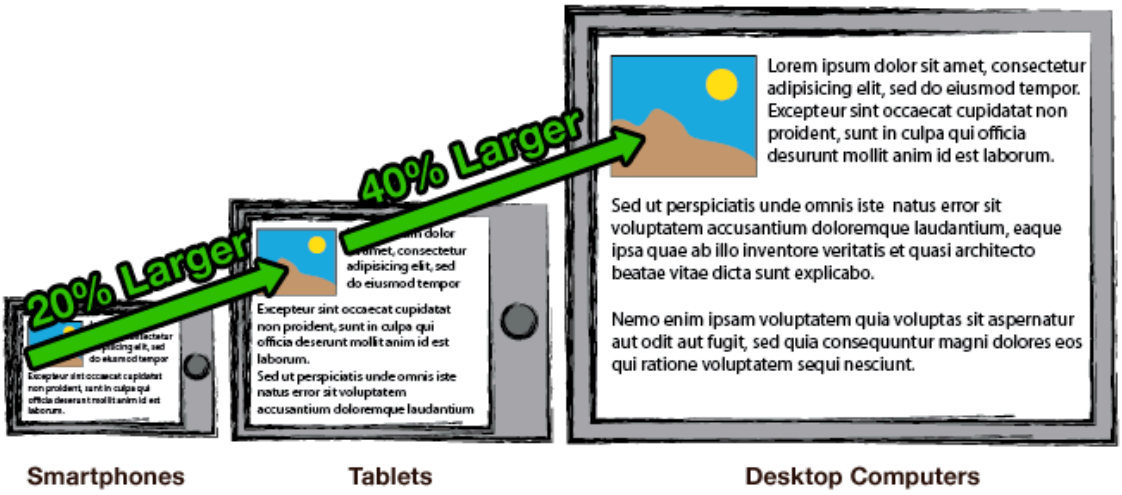
Tablet and Desktop Browsers

Bigger screens are usually connected to high-speed networks, therefore it's completely unnecessary to divide pages in "chunks" like in the mobile approach presented before.

Our recommendation for Tablet and Desktop browsers would be to download the entire page and just use lazy-loading for the images that are located below the fold (the first scroll).

Adaptive Images

It's very usual that when we implement our sites, we find ourselves in the following situation:



In order to make images look right on big screens, such as the ones in desktop computers, most developers usually decide to use image sizes that would look good on a large screen (*such as the monitor of a desktop or laptop computer*), and that, when shrunk, still look right on smaller screens. If you do the opposite, that is, to use images sizes that are optimised for mobile devices, images may look pixelated when they are enlarged to fit in larger screens.

The problem with this approach is that you'll be displaying unnecessarily large images in mobile devices, which have too much detail and therefore consume more memory and take longer to download.

The way to solve this situation is by using a technique called “Adaptive Images”. The concept behind this idea is extremely straightforward; Small screens should be served small (*and therefore lightweight*) images, while larger screens should be served heavier, larger and more detailed images that make a better use of the big screen real estate.

How it works

Basically this approach works by routing all the traffic looking up image files to a special script. On load, the page would include a small script tag that sets the value of a cookie with the dimensions of the screen – when new requests are made, the browser sends packets with data about the current browser (*including the values of those cookies well*), therefore informing the server of the size of the client’s screen. Based on this size, the server will return optimised versions of the image file for that specific screen size.

Working examples of this technique can be seen on http://adaptive-images.com/ or https://github.com/MattWilcox/Adaptive-Images