Picture deep dive

When I first looked at responsive images I thought that `srcset` and `sizes` would be enough to handle most of my responsive images needs. See Concepts and examples of responsive images and How To Use Responsive Images for reference but notice that I barely touched on `<picture>` at all in either of those articles.

When researching how to leverage new image formats in the context of HEIF and AVIF, I discovered that we needed to leverage the other part of the responsive images specification: the `picture` element.

The picture element has more applications that srcset and sizes. We will explore these applications by analyzing five use cases (adapted from Eric Portis' Responsive Images Done Right: A Guide To `<picture>` And `srcset`)

1. Our images need to be able to render crisply at different device-pixel-ratios. This is the **device-pixel-ratio** (DPR) use case
2. If we have a responsive layout, our images will need to fit it. call this **fluid-image** use case
    1. Note that we can tackle use cases one and two together
3. Sometimes we'll want to adapt our images beyond scaling, cropping or changing them in different ways for different screen sizes. This the **art-direction** use case
4. Different browsers support different image formats like WebP, HEIC and AVIF, with a JPEG fallback for browsers that don't support the new formats. We'll call this the **type-switching** use case.

The basics

At its most basic the picture element has one of more `source` children corresponding to different formats and use cases and an `img` child that will only be used if the browser can't handle any of the `source` children.

```
<picture>
  <source srcset="apartment.webp"
          type="image/webp">
  <img  src="apartment.png"
        alt="Apartment Building"
```

```
        loading="lazy">
</picture>
```

We''ll explore the 5 use cases in the following sections. We'll use `img` elements to illustrate DPR and Fluid Images. When we begin user pictures, these cases will be used inside `picture` elements.

Device Pixel Ratio

The first use case is using responsive images to server the correct images for high-dpi devices (Mac's retina displays and high density monitor or devices).

I would normally handle this with a `srcset` attribute indicating what images to use for different densities.

The `src` attribute is the default for when none of the conditions match or when the browser doesn't understand `srcset`.

```
<img  src="my-image.jpg"
      alt="My image"
      loading="lazy"
      srcset="my-image.jpg 1x,
              my-image2x.jpg 2x,
              my-image3x.jpg 3x">
```

Fluid images

There may be times when we want the image we see on screen to change depending on attributes of the page or the image itself.

Using the w attribute

The first option to specify the image to use is to use the w attribute. This will tell the browser what image to use based on the width of the viewport the user is viewing the content through.

The idea in this example is that browsers will take the image closest to the size

of the viewport, over or under, and use it when asked to deliver the image.

```
<img
  srcset="large.jpg  1024w,
          medium.jpg 640w,
          small.jpg  320w"
  src="small.jpg"
  alt="A rad wolf" />
```

Combining `w` and `sizes`

Using `w` on its own doesn't tell the browser how much of the viewport to use for the image we choose and under what conditions.

We can refine the image use by also specifying a size a `sizes` attribute. This will tell the browser how much of the viewport the image will cover.

In the following example, the sizes attributes does the following:

1. If the viewport width is at least 36em then use 50% of the viewport width (indicated with the 50vw value)
2. If the view port is larger than 36em but smaller than 45em then take 1/3 of the viewport width (indicated by the 33.3vw value)
3. If neither condition are met then use 100% of the viewport width (100vw)

```
<img
  srcset="large.jpg  1024w,
          medium.jpg 640w,
          small.jpg  320w"
  sizes=" (min-width: 36em) 50vw,
          (min-width: 45em) and (max-width: 60em) 33.3w,
          100vw"
  src="small.jpg"
  alt="A rad wolf" />
```

It's important to point out that these are not media queries, they just look like them.

Using the `media` attribute in `source`

The `media` attribute specifies a condition (similar to a media query) that the user agent will evaluate for each `<source>` element.

In the example for this section we have three source children element, each with a `media` condition that the browser will evaluate. IF the condition is not true, then the browser will move to the next `source` element and fall back to the default image if none match.

```
<picture>
  <source
    srcset="small.jpg"
    media="(max-width: 36em)">
  <source
    srcset="medium.jpg"
    media="(min-width: 45em) and (max-width: 60em)">
  <source
    srcset="large.jpg"
    media="(min-width: 60em)">
  <img  src="my-image.jpg"
        alt="My image"
        loading="lazy">
</picture>
```

We can further tailor this with DPR values for each `source` element. We've modified the example so now each `source` element has a `srcset` attribute inndicating a 1x, 2x, and 3x resolution version of each image.

We've also added a `srcset` attribute to the default image. If the browser doesn't understand the attribute, it will ignore it.

```
<picture>
  <source
    srcset="small.jpg,
            small2x.jpg 2x,
            small3x.jpg 3x"
    media="(max-width: 639px)">
```

```
  <source
    srcset="medium.jpg,
            medium2x.jpg 2x,
            medium3x.jpg 3x"
    media=" (min-width: 640px)
            and (max-width: 1023px)">
  <source
    srcset="large.jpg,
            large2x.jpg 2x,
            large3x.jpg 3x"
    media="min-width: 1024px">
  <img  src="my-image.jpg"
        alt="My image"
        loading="lazy"
        srcset="my-image2x.jpg 2x,
                my-image3x.jpg 3x">
</picture>
```

Art Direction Use Case

Art direction is, to me, the most complicated use case for responsive images.

Let's assume that the small form factors we want the image in portrait mode while the image we use for medium devices will be cropped, and the image for large devices will be used in landscape mode without cropping.

Art Direction Use Case

Art direction is, to me, the most complicated use case for responsive images.

Let's assume that the small form factors, less than 36em, we will use a portrait version of the image with multiple DPI versions of each image.

This is somewhat contrived example but it shows how we can leverage responsive images to provide rich user experiences in different form factors while our content performant.

```
<picture>
  <source
    srcset="portrait.jpg,
            portrait2x.jpg 2x,
            portrait3x.jpg 3x"
    media="(max-width: 36em)">
  <source
    srcset="cropped.jpg,
            cropped2x.jpg 2x,
            cropped3x.jpg 3x"
    media=" (min-width: 45em)
            and (max-width: 60em)">
  <source
    srcset="landscape.jpg,
            landscape2x.jpg 2x,
            landscape3x.jpg 3x"
    media="min-width: 60em">
  <img  src="my-image.jpg"
        alt="My image"
        loading="lazy"
        srcset="my-image2x.jpg 2x,
                my-image3x.jpg 3x">
</picture>
```

We can add ad many sources, each with it's own combination of DPI (`x`), width (`w`), `media` and `srcset` attributes. As many as we need to cover the art directing needs.

> **It's up to us as designers and developers not to let it get out of hand.**

Different Formats for Different browsers

The final, and to me most useful, use case for responsive images is serving different file formats for browsers that support them.

It used to be the case that WebP was the only image format that wasn't

universally supported. But that's not the case anymore. According to caniuse.com Safari 14 (to be released with the new version of iOS and macOS) will support the format leaving IE and Opera Mini as the only browsers that don't support it.

However there are two new formats to consider as we future proof our content: HEIF and AVIF. These are next generation image formats based on video codecs (HEVC for HEIF and AV1 for AVIF) that produce smaller file sizes of equivalent quality.

HEIF is not currently supported by any browser but, considering that the format is supported in both macOS and iOS, it shouldn't take much for Apple to add support to Safari in the near term.

AVIF is fully supported in Chrome (85 and later) and partially supported in Firefox (behind the `image.avif.enabled` flag).

So what does mean for our responsive images using `picture`?

It means a couple things:

1. We need to be careful with the order of the formats. If the browser supports `picture` and `source`, it will use the first one it can read
2. We need to add the `type` attribute with the mime type of the image format to help the browser figure out if it can use it or not
3. We still need to provide a format for older versions of modern browsers. This is still a problem in companies where IT will restrict browser upgrades

At the very minimum our `picture` element will look like this:

```
<picture>
  <source
    srcset="wolf-medium.heif"
    type="image/heif">
    <source
    srcset="wolf-medium.avif"
    type="image/avif">
  <source
    srcset="wolf-medium.webp"
    type="image/webp">
  <img  src="wolf-medium.jpg"
```

```
        alt="Grey Wolf in the snow"
        loading="lazy">
</picture>
```

When the browser sees this it will run through the following questions

1. Do I support HEIF?
    1. If I do then display the HEIF image
    2. If not then go to step 2
2. Do I support AVIF?
    1. If I do then display the AVIF image
    2. If not then go to step 3
3. Do I support WebP?
    1. If I do then display the WebP image
    2. If not then go to step 4
4. Display the default JPG image

Using different browsers as an example:

Safari 13.1 will fall through to the default in step 4 as it doesn't support any of the formats we're using.

Safari 14.0 will stop in step 3 and serve WebP.

Chrome 85 will stop in step 2 and serve the AVIF image. However Chrome 84 will stop in step 3 and sever WebP instead.

Firefox 80 with the `image.avif.enabled` flag enabled will stop in step 2. Without the flag it will stop in step 3

We can also add other attributes and change the naming scheme to acommodate DPI or Fluid images. We can also leverage the formats with `sizes` and keep the layout control without losing the benefits of the advanced formats.

The example below adds 2x and 3x HDPI versions to all formats.

```
<picture>
  <source
    srcset="wolf-medium.heif,
```

```
            wolf-medium2x.heif 2x,
            wolf-medium3x.heif 3x"
    type="image/heif">
    <source
    srcset="wolf-medium.avif,
            wolf-medium2x.avif 2x,
            wolf-medium3x.avif 3x"
    type="image/avif">
  <source
    srcset="wolf-medium.webp ,
            wolf-medium2x.webp 2x,
            wolf-medium3x.webp 3x"
    type="image/webp">
  <img  src="wolf-medium.jpg"
        alt="Grey Wolf in the snow"
        loading="lazy"
        srcset="wolf-medium2x.jpg 2x,
                wolf-medium3x.jpg 3x">
</picture>
```

We can further tailor this with DPR values for each `source` element. We've modified the example so now each `source` element has a `srcset` attribute inndicating a 1x, 2x, and 3x resolution version of each image.

We've also added a `srcset` attribute to the default image. If the browser doesn't understand the attribute, it will ignore it.

```
<picture>
  <source
    srcset="small.jpg,
            small2x.jpg 2x,
            small3x.jpg 3x"
    media="(max-width: 36em)">
  <source
    srcset="medium.jpg,
            medium2x.jpg 2x,
            medium3x.jpg 3x"
```

```
    media=" (min-width: 45em)
            and (max-width: 60em)">
  <source
    srcset="large.jpg,
            large2x.jpg 2x,
            large3x.jpg 3x"
    media="min-width: 60em">
  <img  src="my-image.jpg"
        alt="My image"
        loading="lazy"
        srcset="my-image2x.jpg 2x,
                my-image3x.jpg 3x">
</picture>
```

To conclude, `picture`-based responsive images can be as simple or complex as we want to make them. We need to keep the KISS Principle firmly in mind as we work in developing these images particularly because HEIF and AVIF tooling is just getting started.