

[Edit](#) [New issue](#)

[Jump to bottom](#)

SPIKE: Retina image support & picture tag #449

[Closed](#) [3 of 7 tasks](#) **brynwhyman** opened this issue on Jun 22, 2021 · 30 comments

Assignees



Spike to investigate the method and estimate in adding retina image support AND the picture tag to images published from the CMS.

My thinking is while these can and should be implemented separately, it could make sense to spike them at the same time.

Timebox:

0.5 days

Acceptance criteria


- Supporting and implementation documentation is reviewed and briefly summarised in the issue
- Competing CMS implementations have been investigated (reading blog posts/ changelogs should be enough)
- Investigation findings are captured in the issue
- TSP have been briefed on this feature and are comfortable with the implications
- High-level development tasks are outlined
- All areas in the CMS where this feature could be used have been called out (similar to what was done for the lazy-loading spike)
- An hour estimate range has been provided





 **Dependencies** 0



 **brynwhyman** changed the pipeline from **New Issues** to **Backlog** in **CMS Squad** 3 years ago

 **brynwhyman** changed the pipeline from **Backlog** to **Ready** in **CMS Squad** 3 years ago

  **brynwhyman** assigned **maxime-rainville** on Jun 23, 2021

 **brynwhyman** added this to **Sprint: Jun 23 - Jul 7** 3 years ago

emteknetnz commented on Jun 24, 2021 • edited ▾

space



👤 **maxime-rainville** changed the pipeline from **Ready** to **In Progress** in **CMS Squad** 3 years ago

brynwhyman commented on Jun 24, 2021

Author

Related Slack thread: <https://silverstripeltd.slack.com/archives/CB95TE3MF/p1619145369113800>



maxime-rainville commented on Jun 24, 2021 • edited ▾

Implementing retina images

There's two ways we could implement out-of-thebox retina support:

- using the `srcset` attribute with an `` tag
- switching to using the `<picture>` tag

Outputted HTML

srcset

```

```



<picture>

```
<picture>  
  <source srcset="elva-fairy-320w.jpg, elva-fairy-480w.jpg 1.5x, elva-fairy-640w.jpg 2x">  
    
</picture>
```



Either way, implementation would probably involve outputting slightly different HTML that reference multiple variants.

The `ImageShortcodeProvider` logic is currently a bit disjointed from the `Image` rendering logic. This makes this work a tiny bit more difficult because you have to fix it in two places instead of one.

Some key edge-cases/questions to consider are:

- what do you do if the image provided by the user is too small to be used as a high density image
 - Maybe we just provide the `src` attribute and nothing else
 - Maybe we give a warning to the user in the CMS
- what do you do if your source file is smaller than your high-density image would be?
 - probably render the original image, but we need a bit of logic to detect that.
- What pixel density options do we want to use?
 - I've seen `1.5x`, `2x` and `3x` in the various doc articles I've read. `2x` was the most common by far and `3x` the least common one.
 - Maybe we just pick a sensible density like `2x` and make the other configurable. So out of the box you get `1x` and `2x`. If you want `1.5x`, or `3x` or `10.5x` you need to add an entry in your YML config.

Optimising for slow connection

If the user is on a slow/metered connection, it could be desirable to not serve retina images.

There's an interesting [NetworkInformation JS API](#) on modern browsers that could be use for this purpose.

But that bring up a lot of other side questions like how would your CDN handle this.

My instinct would be to just let site owners decide for themselves if they want to serve retina image or not.

Potential risk

If we switch to `<picture>`, we will probably break some people CSS if they use rules like `.boom > img`. Using `` probably is the safest bet.

Managing extra disk space and CPU usage

That would be very specific to how much images you use on your site. If we implement this in such a way that we don't create retina images bigger than the original file, that would limit the number of actual file to store



maxime-rainville commented on Jun 24, 2021 • edited ▾

Other side benefit of picture tags

The `<picture>` tag as other potential uses aside from displaying retina:

- Adjusting images bases on "Art direction". (e.g.: serve a portrait image if the user is viewing the site on a portrait device)
- Offer alternative formats (e.g.: Serve a WebP version of the image, but default to JPG if not supported)
- Adjust image based on the dimension of the device (similar to media queries)

Those use cases are probably a bit more niche than displaying retina images.

Also it would be difficult to come up with a syntax that would more useful than telling people to write their own templates.

The alternative format use case could probably not be implemented unless we also implement [silverstripe/silverstripe-assets#411](#)

Possible next steps

I'm thinking a good alternative approach to implementing outright support for `<picture>` tags could be to

1. Refactor how how `Image` objects are rendered so it's better thought out (current implementation is somewhat simplistic and doesn't provide much option)
2. Refactor `ImageShortcodeProvider` to render the `Image` object instead of manually building the tag
3. Provide better doc for people who want to switch to using `<picture>` tags for their site. This could include:
 - a. How to override the `Image` template
 - b. How to implement a custom `ImageManipulation` method to simplify the process of creating custom image manipulation method.

Point 1 and 2 probably have a lot of other side benefits in terms of unifying the rendering of `Image` in and out of the WYSIWYG.



We've found this approach combined with height/width and lazy loading attributes can be a big win for performance. Especially for the Cumulative Layout Shift (CLS) score. The art direction is really useful too - eg going to square images on mobile. When combined with things like focal point cropping its great and probably not quite as niche as expected.

On the bandwidth front - I believe browsers will do some selection based on bandwith so you may not have to go to the extent of using the network API - eg on a good connection chrome will often download a larger image see <https://developers.google.com/web/fundamentals/design-and-ux/responsive/images>

Automatically generating templates might be hard as the tags/media queries etc often depend on the position of the image in the page (eg am I a full width image or constrained at some breakpoint). But there's probably a good basic case that could be covered.



emteknetz commented on Jun 25, 2021

So, would the user upload 3x lots of images (can't see much support for that)? Or the idea is they upload a single image (4x larger for the 2x density image?) and multiple versions of each are created on the server?

Looking at the [mdn article on srcset](#) - would we want to add support for dev to add the sizes attribute to serve different images based on the width? Or would we want to limit this to strictly device pixel density = higher res image

If we want to allow different ratios for different breaks points, we'd need to something pretty funky with the upload field since it becomes a one to many



blueo commented on Jun 25, 2021

we've got away with the one image and using different transforms to create the variations. So upload a high res and it gets sized down/cropped.

It would be great to be able to create different formats of the same image (eg webp versions) but I think you need the same extension for every variant atm.



emteknetz commented on Jun 25, 2021 • edited ▾

Sorry, I'm not entirely clear, could you please confirm what you did on this project:

instead only used the 1.5x and 2x in the `srcset` attribute?

b)

You have different ratios for different images and media queries in the `sizes` attribute?



blueo commented on Jun 25, 2021 • edited ▾

Maybe an example will help:

```
<picture>
  <source
    data-ratio="32-11"
    sizes="100vw"
    media="(min-width: 960px)"
    alt="Tekapo Hot Springs"
    srcset="
      /assets/Client/Christchurch-Canterbury/Tekapo-Hot-Springs-Have-You-Ever-Miles-Ho|
      /assets/Client/Christchurch-Canterbury/Tekapo-Hot-Springs-Have-You-Ever-Miles-Ho|
      /assets/Client/Christchurch-Canterbury/Tekapo-Hot-Springs-Have-You-Ever-Miles-Ho|
    "
    width="1920"
    height="660"
  />

  <source
    data-ratio="16-7"
    sizes="100vw"
    media="(min-width: 640px) and (max-width: 960px)"
    alt="Tekapo Hot Springs"
    srcset="
      /assets/Client/Christchurch-Canterbury/Tekapo-Hot-Springs-Have-You-Ever-Miles-Ho|
      /assets/Client/Christchurch-Canterbury/Tekapo-Hot-Springs-Have-You-Ever-Miles-Ho|
    "
    width="960"
    height="420"
  />

  <source
    data-ratio="3-2"
    sizes="100vw"
    media="(max-width: 640px)"
    alt="Tekapo Hot Springs"
    srcset="
      /assets/Client/Christchurch-Canterbury/Tekapo-Hot-Springs-Have-You-Ever-Miles-Ho|
      /assets/Client/Christchurch-Canterbury/Tekapo-Hot-Springs-Have-You-Ever-Miles-Ho|
    "
    width="640"
    height="427"
  />
```



```
width="640"  
height="427"  
data-iscarouselimage="1"  
>  
</picture>
```

So we've got a `<source>` per aspect ratio. Those have a `sizes` property that changes depending on where it is placed (in this case full width all the time). And they have a `srcset` with different widths.

so we're not using 2x etc at the moment - the widths seem to provide enough as the wider image can be used on the higher dpi devices.



emteknetnz commented on Jun 25, 2021

Great ta.

So how do CMS authors go about uploading the different ratios in the CMS? Are there separate fields / UploadField's fields for each ratio? Or a single one where users upload a large file and you just crop from the middle?



blueo commented on Jun 25, 2021

They just upload one image - and add a focal point - the system then creates all the variations including aspect ratios using the focal point to avoid cutting the wrong bit (elsewhere it is just from the centre). Its the same image essentially.



maxime-rainville commented on Jun 25, 2021

For the retina part, I don't think we would go as far as TNZ went.

Right now, when you add an image to tho WYSIWYG, the CMS will create a variant based on the dimensions you put in the InsertMediaModal. If/When we implement retina support, we would generate additional variants ... provided the image you provided is big enough. e.g.:

- If you upload a 300x200 image and tell to WYSIWYG to display it at 300x200, there's no point creating a retina variant.
- If you decide to display it 150x100, then we serve the original as the retina image and create

retina variant at 150x100

Something similar would happen if you output an `$Image` in your SS template.

Short answer, this would be completely transparent to the CMS users or the site visitors. After you upgrade and flush the Silverstripe cache, all your pre-existing images would start being outputted with retina variants where it makes sense. And there wouldn't be any UI to control this ... it just does it out of the box.



adrhumphreys commented on Jun 25, 2021

I would love to see some configurability for templated images OOTB, e.g. setting a profile in a config and then being able to call `$Image.Profile("FeaturedImage")` and then getting your image tag with different sizes instead of pixel densities

We've been using something similar for our current EQC project and it's been nice, holla if you want to see my *ugly beautiful* code



maxime-rainville commented on Jun 25, 2021

@adrhumphreys That's probably not within the scope of this card, although you can probably easily achieve this by applying your own extension to Image.



emteknetnz commented on Jun 25, 2021

If you upload a 300x200 image and tell to WYSIWYG to display it at 300x200, there's no point creating a retina variant.

Possibly we'd want to 'reverse' this, or at least make it an opt-in so that that the image you upload is the 2x version. Idea here is that CMS authors who opt in and know what they're doing upload large files like 2000px wide for high pixel count devices, though the "normal" desktop image is only 1000px.



maxime-rainville commented on Jun 27, 2021

the time and let the CMS do the resizing.



chillu commented on Jun 28, 2021 • edited ▼

Managing extra disk space and CPU usage

That would be very specific to how much images you use on your site. If we implement this in such a way that we don't create retina images bigger than the original file, that would limit the number of actual file to store.

I think this needs more consideration. [@brynwhyman](#) What do you think about an explicit AC on a story around "TSP have been briefed on this feature and are comfortable with the implications".

There's a few dimensions to this:

1. Worker capacity: We generate images "lazily" as part of the main page render which first needs those images. This has always been dodgy, since it introduces instability in request processing. Presumably generating larger images requires more PHP memory - will that push some worker capacity over the edge? And since this is all single threaded, it might push beyond 30-60s execution time.
2. Storage: Disk space needs will obviously increase. This is more of an issue on CCL where disk space is "soft capped" (requires explicit resizing), than in AWS where EFS can scale more easily to demand. How much increase do we roughly expect in relative and absolute terms? You could investigate some representative sites with well managed assets (not a lot of unused ones), like covid19.govt.nz. Just saying "it depends" isn't good enough, these features have real world effects.
3. Network throughput: Larger images require more bandwidth, but they're also lazy loaded now. The effect might cancel each other out. Bandwidth is more precious in CCL than in AWS, and not a big factor in CDNs. Hard to estimate across all of our platform stacks, but ideally give some indication.
4. Upgrade path: What capacity is required from a stack after they push this feature live? That's a lot of image generation requests to handle alongside web request processing, and might well introduce instability. Do we need a task for this?

I believe that retina images are table stakes, but we need to actively manage this transition on our platform. You can very easily tip stacks over the edge with the short-term cost of generating all new variants, or long-term by requiring more capacity for each dynamic page request. We might mitigate this risk by only enabling retina images by default on new installs, and making it opt-in for the rest.

I've done some [research](#) into platform usage recently when Aaron investigated static asset cache deployment. You could work with TSP to get some samples, e.g. `find assets/ -type f -name '*.jpg' -or -name '*.png' -exec file {} \;` which will give you image dimensions.

brynwhyman commented on Jun 28, 2021

Author

@brynwhyman What do you think about an explicit AC on a story around "TSP have been briefed on this feature and are comfortable with the implications"

Yes, definitely. I hadn't realised this would have such impacts on the server. **@maxime-rainville** I'll leave it to you to please get this conversation started?



emteknetnz commented on Jun 29, 2021

Are you suggesting that if you upload a 300x200 we just pretend that it's 150x100?

(Apologies if this sounds wrong, it's likely that my understanding of how retina images is poor)

The idea that if the 'normal' 1x size of the image was 150x100, then yes upload a 300x200 which is the 2x image. The idea is that there would be a site setting opt-in which is "every image that is uploaded is a 2x image"



maxime-rainville commented on Jun 29, 2021

I would be disinclined from mocking about with the size of images. That just sounds like we would be trying to outsmart image format standards or our users. That's bound to create more confusion than it's worth.

People would create an image with the expectation that it will be one size and we just decide to ignore them and do our own thing. It would also create weird issues where you have pre-existing images.

There's also the problem that you can create images with higher DPIs ... standard DPI for web is 72, but there's nothing stopping you from uploading an image with 300 DPI, which will look quite nice when rendered on a retina display.



maxime-rainville commented on Jun 29, 2021 • edited ▼

I'll create a weird page type that renders lots of images version to mock what would happen if you have loads of retina images.

Edit: [@brynwhyman](#) that page test is probably going to blow me past the .5 days timebox



maxime-rainville commented on Jun 30, 2021 • edited ▾

These questions are coming from [@AshleyJMueller](#) on Slack.

What is the value proposition for this functionality?

High pixel density device (aka [retina display](#)) became common with the release of the iPhone 4 in 2010. Because retina displays have more pixels than a regular displays, UI elements and content rendered on them appear sharper in most cases.

However, this has the downside that if an image is displayed at native resolution on a retina device, it will look noticeably less crisp than surrounding elements.

To address this problem, many website have begun serving images at a greater dimension than they expected them to be rendered, so they look crisp on all displays. Since then, standards have evolved so images can be selectively served at different dimensions based on the pixel density of the user's display. This card aims at providing a Silverstripe CMS native implementation of those standards.

Who wants it? Why do they want it?

The primary benefactors of this would be:

- content authors who don't currently have a way of displaying retina images when editing WYSIWYG content
- developers who don't currently have an elegant way of rendering a retina image

We haven't actually validated that there's hordes of people screaming for this feature. However, we are currently in the process of implementing lazy loading for our images, which creates a side opportunity for us to add retina image support at little cost.

How much is it going to cost to develop?

From TSP's perspective, probably nothing. We just need TSP to be satisfied that this new feature won't have any adverse affect once deployed to SCP/CWP.


How much is it going to cost to maintain?

The main affect of this change will be that we will be generating additional and slightly larger image variants for existing sites.

This will require some additional disk space to store those additional variants and increase the initial rendering time for some pages.


I'll be running some benchmark so we can have more precise numbers.




  **maxime-rainville** removed their assignment on Jul 5, 2021

Automatically added this to **Sprint: Jul 7 - Jul 21** based on Sprint automation 3 years ago

 **brynwhyman** changed the pipeline from **In Progress** to **Ready** in **CMS Squad** 3 years ago

 **maxime-rainville** changed the pipeline from **Ready** to **In Progress** in **CMS Squad** 3 years ago

  **maxime-rainville** self-assigned this on Jul 14, 2021

maxime-rainville commented on Jul 14, 2021

I've got a benchmark page type I'm going to deploy to SCP to see how it performs.

This video shows what it does on my local: <https://youtu.be/WjB1o3a4Evg>

Let me know if you want it to benchmark other aspects.



maxime-rainville commented on Jul 16, 2021

Here's the [raw data](#). I didn't run every possible combination imaginable. If there's scenarios you feel are worth digging more into, let me know.

You can view the [benchmark code on cms-testing-recipes](#).

How the benchmark was run

I created a special retina benchmark page. The page receives the following information.

- A folder
- Image densities (1.5X, 2X, 3X)
- A width

If a retina image would end up being wider than the original image, the original image would be outputted instead.

The retina benchmark controller also has 2 actions:

- "Stats" that tell you the total size of the images in the folder and of their variants
- "Prune" that deletes all the image variants to allow you to run the benchmark again.

For each scenario, we measured:

- The execution time (measured with `execmetric=1`)
- Memory usage (measured with `execmetric=1`)
- Variant sizes

Results analysis

The 2 biggest factors influencing executing time were:

- The total size of the original image
 - Doubling the size of the original images increases execution time by 140%. Doubling in this context is equivalent to having 4 times more pixels.
- The total number of variants to creates
 - Creating variants for 40 images is 230% slower than creating variants for 20 images
 - That parts seems to have an algorithmic complexity of $O(n \log n)$

The biggest factor influencing memory usage was the size of the biggest original image, with the size of the variant playing a lesser role. The total number of images or variants to create didn't seem to have much of an influence.

Once all the retina variants were created, there was no discernible performance difference between the retina and none-retina pages.

Hitting the limit

I started hitting timeouts when generating 40 variants of 3000x2000 images.

Concurrency

I didn't try to thoroughly test the scenario where multiple users hit the site simultaneously. Concurrent requests did seem more prone to timing out.

It's worth pointing out that generating each variant could be considered a separate task. So this scenario could easily happen

A request.

- The user B worker will probably jump straight to generating the variant the user A worker is currently working on.

Worker B will duplicate the second half of the work done by worker A. So user A and user B will probably end up getting a response at about the same time.

An interesting approach we could try could be to get each worker to establish a lock on the variant they are about to start generating. In this scenario:

- Worker B could come in and see that some other worker is already in the process of generating variant 1, so it skips it and goes straight to generating variant 2.
- When worker A is done generating variant 1, it sees that another process is currently generating variant 2, so it jumps to variant 3.
- Etc.

You probably would want the lock to be relatively short live otherwise you could end up in a scenario where one worker crashes and the variant is never generated. Or one worker could finish too soon and the end user gets a 404 because the variant they are expecting is not quite done generating yet.

Limits

- Only tested JPEG
- Only tested the default OpenGD driver. Imagick is known to be faster.
- Only tested each scenario once. Didn't see much variation when repeating scenarios.
- I don't currently have stats on what the average size of images is, what the average size of variants is, the number of images per page, etc. So I don't know how representative or not representative of the average CMS site my test was.

Problems

It's safe to say that at least some sites would have a problematic upgrade if we blindly enabled retina images. Worst case scenario would be that their CPU consumption would jump through the roof for some time as their server is busy generating new variants everywhere.

On the plus side, if your requests time outs because it takes too long to generate all the variants, the next request will not try to regenerate the variants you've already created.

Possible steps

Give an opt-out/opt-in

we could also only enable this for new installs. That would avoid migration problems.

migration tasks

We could provide a task that would pre-create all the retina variants up front. That could be done in the background. When that's done, retina images could be enabled with confidence.

That task would need to know what variants it needs to create:

- The task could run with some flag that forces the use of retina images for that specific request only. The task would go through all the pages and force them to re-render with retina images.
- If you have a controller that is not hooked to a SiteTree object, the task would not know how to find it.
 - The task could blindly create double size variants of all pre-existing images. But you could end up with a lot more image variants than what you actually need.

Limit the number of variant we create on a given request

We could try to limit the number of retina variants we create for a given request.

- We could stop outputting retina images if the request has been going on for 5 secs or more.
- We could randomly decide to not output a retina image if one doesn't already exist.

Those approaches would avoid having all the work done by one worker. It could lead to some caching problems however.

Skipping the retina variants

There's no point outputting a retina variant if it would be bigger than the original image. In this scenario, you are better off sending the original image.

It's also worth pointing out that resizing an image to be only a tiny bit smaller than the original can lead to degradation because there's not enough pixel to properly resample the downsampled image. So maybe we only create a retina image if it's 75% the size of the original and serve the original image otherwise.

I'm not an expert on image algorithms. I ran some tests locally by resizing an 3000x2000 to be 10 pixels less wide. The resized image looked noticeably worse to me, but it's probably worth validating my assumption here.

How to develop this fix

Some of those options are complimentary.

I use `SetAttribute` for the benchmark. That worked like a charm. I strongly suggest building the end results on top of that.

Refactor `ImageShortcodeProvider` to use `ImageManipulation`

With `SetAttribute` now available on `Image`, we have all the tools necessary to have a unified rendering between our shortcode and SS viewer image implementation. This would simplify `ImageShortcodeProvider` and avoid duplicating bits of code between the two.

Ship in core with opt-out/opt-in

This would get the most people using this the fastest. But there could be upgrade pain.

Ship as a side module

We could ship this as a side module that adds an extension to `Image`. We would still need to add a few tweaks to core for this to work, but that would minimise upgrade pains. We could also choose not to support this module officially. So it's the lowest risk option.

This module could also be a little bit more edgy and do things like swapping the default image provider to `Imagick`.

Other things

I'm not 100% sure about this, but I get the impression that if you generate multiple variants from the same image, it will have to reload it in memory multiple times. Could be worthwhile exploring this further to see if we could reuse the same `ImageBackend` to generate different variants of the same image.



emteknetz commented on Jul 19, 2021 • edited ▾

I've looked at a few articles online, I'm pretty sold on the idea of using `srcset` as opposed to `<picture>` since `srcset` is aimed at serving different images of the same ratio for different pixel count/density screens, while `<picture>` seems more aimed at providing images of different ratios for different breakpoints

Just having a quick look at <https://css-tricks.com/responsive-images-youre-just-changing-resolutions-use-srcset/> they had a slightly different syntax for `srcset` that used `w` instead of `x` and had the small image was used for `src`

Where `small.jpg` was a 500px image

This contrasts with the syntax provided early on in this issue which uses the `1x 1.5x, 2x` syntax and where `src` was the large image

```

```



I don't know what the pros and cons of each approach are

This page has a deep dive on the issue <https://imagekit.io/responsive-images/>



maxime-rainville commented on Jul 20, 2021

This bit of code is doing something slightly different than displaying retina images.

```

```

For example, you could have a hero image that you want to take the full width of the page no matter what.

- If the site is being viewed on a retina iMac, you'll want to serve an image that's 5000 pixel wide.
- If the site is being viewed on a retina iPhoneX in portrait mode, you'll probably not want to serve an image that's wider than 1125 pixel. The device is not physically capable of displaying more pixels than that

So `1000w` allows you to serve a different image based on the "scaled" resolution while `2x` allows you to target the pixel density.

If you are developing a responsive theme, you'll probably want to use the `1000w` syntax. If you are a content editor who wants to display an 300px wide image in your content and want it to look crisp on retina displays, you'll want to use the `2x` syntax.



maxime-rainville commented on Jul 20, 2021

Just summarising the discussion we had at our meeting today:

customise images

- We'll do a MVP module that people can install to get their CMS site to output retina images
 - That module won't be supported
- There could be some value in allowing editors to control whether a retina image is being outputted or not, but that is not essential in the first stage.

Maxime will create a follow up card with ACs for grooming.



emteknetnz commented on Jul 20, 2021 • edited ▾

OK that makes sense re the x vs the w

I'm just wondering if we'd want any module we create to allow for $w / sizes$ `<picture>` or any other type of thing, or just constrain it to x , which would be significantly easier

We'd probably also need to decide how big we're willing to go. So far we've talked about 2x, though it may make sense to go as high as 4x.

PPI of some screens:

1920x1080 24 inch monitor ppi = 92
retina (2010) ppi = 326
iphone 12 pro ppi = 458
galaxy s7 ppi = 576

Got these numbers from google + <https://pixensity.com/list/phone/>

Calculating device pixel ratio - take the PPI and divide by 150
<https://stackoverflow.com/questions/23841812/how-to-calculate-device-pixel-ratio/23958504>

1920x1080 24 inch monitor = 1x (0.6 ratio will use 1x image)
retina = 2x
iphone 12 pro max = 3x
galaxy s7 = 4x (3.84 ratio, would probably round up and use this over the 3x)

That said, there's probably very diminishing returns in terms of a humans ability to distinguish image details beyond 2x, so the increase in server load / load time is less likely to be of benefit. Though we may still wish to include this functionality and let website devs make their own decisions.



That list on <https://pixensity.com/list/phone/> kind of indicates that we probably wouldn't need the 1.5x to support ~225 PPI devices since they're not so common these days, at least according to this list



maxime-rainville commented on Jul 20, 2021

Yeah ... my thinking is give people a sensible default, which I think the 2x variant will cover. From there, give them so way to extend/customise it for their own purpose and call it a day.



  maxime-rainville mentioned this issue on Jul 20, 2021

Add support for retina images silverstripe/silverstripe-assets#461


Closed


 13 tasks

maxime-rainville commented on Jul 20, 2021

New task has been created [silverstripe/silverstripe-assets#461](#)




 maxime-rainville closed this as completed on Jul 20, 2021

  maxime-rainville mentioned this issue on May 3, 2022

Add task to pre-warm image cache silverstripe/silverstripe-assets#492

Open

 4 tasks

  maxime-rainville mentioned this issue last week

Create an advanced image rendering solution silverstripe/.github#230

Open

Pipelines

Show 3 more

C CMS Squad

Closed



Assignees



 maxime-rainville

Projects



Not inside an epic
None yet

Releases



Not inside a release
Sprint: Jun 23 - Jul 7

Sprint: Jul 7 - Jul 21

Development



[Create a branch](#) for this issue or link a pull request.

Milestone



No milestone

3 participants



Pin issue