

Image Optimization

by Addy Osmani

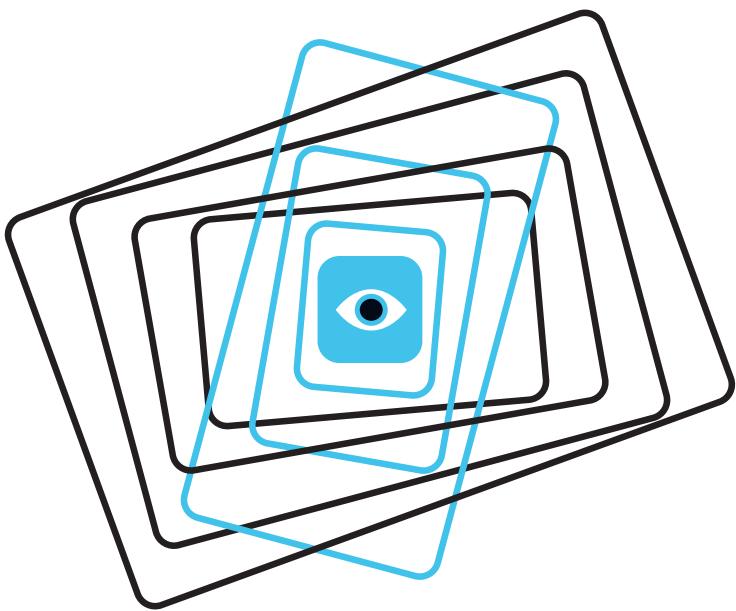


Image Optimization

by
Addy Osmani

Published 2021 by Smashing Media AG, Freiburg, Germany.

All rights reserved.

ISBN: 978-3-945749-94-4

Technical editing: Milica Mihajlja and Colin Bendell

Copyediting: Owen Gregory

Cover and section illustrations: Espen Brunborg

Interior full-page illustrations: Nadia Snopek

Book design and indexing: Ari Stiles

Ebook production: Cosima Mielke

Typefaces: Elena by Nicole Dotin, Mija by

Miguel Hernández and Andalé Mono by Steve Matteson

Image Optimization was written by Addy Osmani.

Reviewers and contributors include Colin Bendell,

Kornel Lesiński, Estelle Weyl, Jeremy Wagner,

Tim Kadlec, Nolan O'Brien, Pat Meenan, Kristofer Baxter,

Henri (Helvetica) Brisard, Houssein Djirdeh, Una Kravets,

Ilya Grigorik, Elle Osmani, Leena Sohoni, Katie Hempenius,

Jon Sneyers & Mathias Bynens.

This book is printed with material from

FSC® certified forests, recycled

material and other controlled sources.



Please send errors to: errata@smashingmagazine.com



Contents

<i>Foreword by Colin Bendell</i>	vi
<i>Introduction</i>	xix
1 The Humble Element	29
PART ONE ■ IMAGE QUALITY AND PERFORMANCE	
2 Optimizing Image Quality	43
3 Comparing Image Formats	54
4 Color Management	68
5 Image Decoding Performance	80
6 Measuring Image Performance	100
PART TWO ■ CURRENT IMAGE FORMATS	
7 JPEG	115
8 PNG	144
9 WebP	170
10 SVG	200

PART THREE ■ IMAGES IN BROWSERS

11	Responsive Images	223
12	Progressive Rendering Techniques	238
13	Caching Image Assets	256
14	Lazy-Loading Images	290
15	Replacing Animated GIFS	314
16	Image Content Delivery Networks	337

PART FOUR ■ NEW & EMERGING IMAGE FORMATS

17	HEIF and HEIC	382
18	AVIF	398
19	JPEG XL	410
20	Comparing New Image Formats	433

PART FIVE ■ FURTHER OPTIMIZATION

21	Data Saver	457
22	Optimize Images for Core Web Vitals	471
23	Case Study: Twitter	490
	Conclusion	505

CHAPTER ONE

The Humble Element

The humble element has gained some super-powers over the years. Given how central it is to image optimization on the web, let's catch up on what it can do.

The Basics

To place an image on a web page, we use the element. This is an empty element – it has no closing tag – requiring a minimum of one attribute to be helpful: `src`, the source. If an image is called `donut.jpg` and it exists in the same location as your HTML document, it can be embedded as follows:

```

```

To ensure our image is accessible, we add the `alt` attribute. The value of this attribute should be a textual description of the image, and is used as an alternative to the image when it can't be displayed or seen; for example, a user accessing



75% The speed it takes to load the page

66% How easy it is to find what I'm looking for

61% How well the site fits my screen

58% How simple the site is to use

25% How attractive the site looks

HOW IMPORTANT IS SPEED?

Users rated speed highest in the UX hierarchy according to Google's Speed Matters Vol. 3

your page via a screen reader. The above code with an `alt` specified looks as follows:

```

```

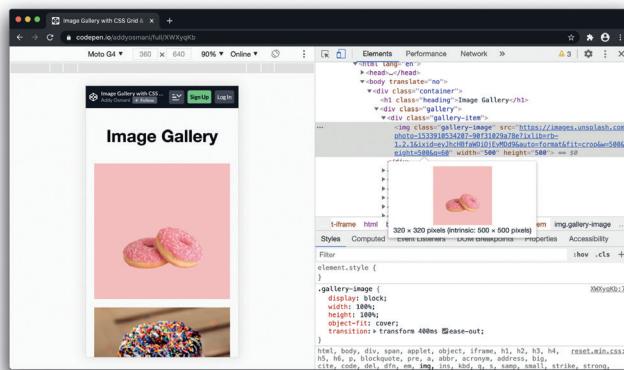
Next, we add `width` and `height` attributes to specify the width and height of the image. The dimensions of an image can usually be found by looking at this information via your operating system's file explorer (**Cmd + I** on macOS).

```

```

When `width` and `height` are specified on an image, the browser knows how much space to reserve for the image until it is downloaded. Forgetting to include the image's dimensions can cause layout shifts, as the browser is unsure how much space the image will need.

Modern browsers now set the default aspect ratio of images based on an image's `width` and `height` attributes, so it's valuable to set them to prevent such layout shifts.



Hovering over an image in the Chrome DevTools **Elements** panel displays the dimensions of the image as well as the image's intrinsic size.

Swapping Out Images

What about switching image resolution? A standard `` only allows us to supply a single source file to the browser. But with the `srcset` and `sizes` attributes we can provide many additional source images (and hints) so the browser can pick the most appropriate one. This allows us to supply images that are smaller or larger.

```

```

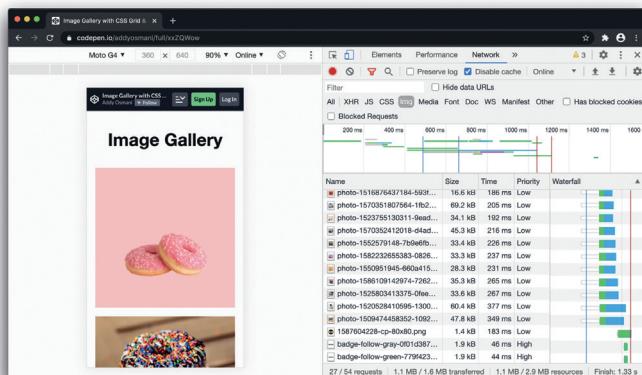
The `srcset` attribute defines the set of images the browser can select from, as well as the size of each image. Each image string is separated by a comma and includes: a source filename (`donut-400w.jpg`); a space; and the image's intrinsic width specified in pixels (`400w`), or a pixel density descriptor (`1x`, `1.5x`, `2x`, etc.).

The `sizes` attribute specifies a set of conditions, such as screen widths, and what image size is best to select when those conditions are met. Above, `(max-width: 640px)` is a media condition asking “if the viewport width is 640 pixels or less,” and `400px` is the width the image is going to fill when the media condition is true.

Even those images which are responsive (that is, sized relative to the viewport) should have width and height set. In modern browsers, these attributes establish an aspect ratio that helps prevent layout shifts, even if the absolute sizes are overridden by CSS. (Chapter 11 covers responsive images.)

Image Loading

What about offscreen images that are not visible until a user scrolls down the page? In the example below, all the images on the page are “eagerly loaded” (the default in browsers today), causing the user to download 1.1 MB of images. This can cause users’ data plans to take a hit in addition to affecting performance.



An image gallery eagerly loading all the images it needs up front, as shown in the Chrome DevTools **Network** panel. 1.1 MB of images have been downloaded, despite only a small number being visible when the user first lands on the page.

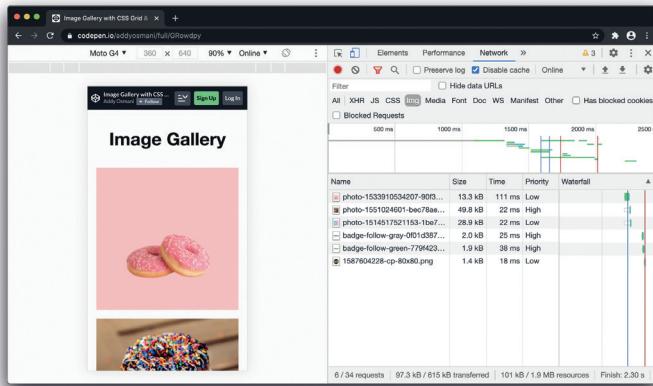
Using the `loading` attribute on ``, we can control the behavior of image loading. `loading="lazy"` lazy-loads images, deferring them loading until they reach a calculated distance from the viewport. `loading="eager"` loads images

right away, regardless of their visibility in the viewport. `eager` is the default and can be ignored (that is, just use `` for eager loading).

Below is an example of lazy-loading an `` with a single source:

```

```



An image gallery using native image lazy-loading on images outside of the viewport. As seen in the Chrome DevTools **Network** panel, the page now only downloads the bare minimum of images users need up front. The rest of the images are loaded in as users scroll down the page.

With native `` lazy-loading, the earlier example now downloads only about 90 KB of images! Just adding `loading="lazy"` to our offscreen images has a huge impact.

Lazy loading also works with images that include `srcset`, as `` is what drives image loading:

```

```

We'll cover lazy loading in full in chapter 14.

Image Decoding

Browsers need to decode the images they download in order to turn them into pixels on your screen. However, how browsers handle deferring images can vary. At the time of writing, Chrome and Safari present images and text together – synchronously – if possible. This looks correct visually, but images have to be decoded, which can mean text isn't

shown until this work is done. The decoding attribute on allows you to signal a preference between synchronous and asynchronous image decoding.

```

```

decoding="async" suggests it's OK for image decoding to be deferred, meaning the browser can rasterize and display content without images while scheduling an asynchronous decode that is off the critical path.

As soon as image decoding is complete, the browser can update the presentation to include images. decoding="sync" hints that the decode for an image should not be deferred, and decoding="auto" lets the browser do what it determines is best. (There's more on the decoding attribute in chapter 5.)

Placeholders

What if you would like to show the user a placeholder while the image loads? The `background-image` CSS property allows us to set background images on an element, including the `` tag or any parent container elements. We can combine `background-image` with `background-size: cover` to set the size of an element's background image and scale the image as large as possible without stretching the image.

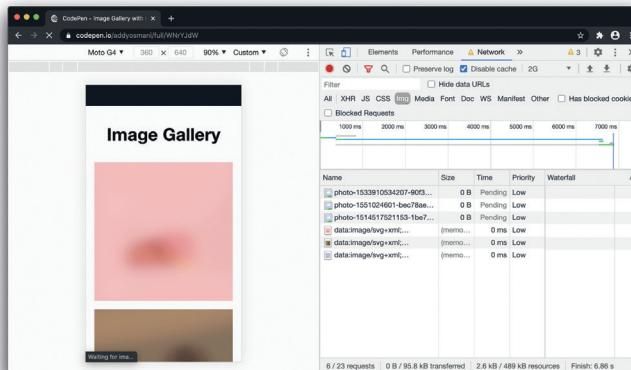
Placeholders are often inline, Base64-encoded data URLs which are low-quality image placeholders (LQIP) or SVG image placeholders (SQIP). This allows users to get a very quick preview of the image, even on slow network connections, before the sharper final image loads in to replace it.

```

```

Note: Given that Base64 data URLs can be quite long, [svg text] is denoted in the example above to improve readability.

With an inline SVG placeholder, here is how the example from earlier now looks when loaded on a very slow connection. Notice how users are shown a preview right away prior to any full-size images being downloaded:



Images loaded on a simulated slow connection, displaying a placeholder approximating the final image as it loads in. This can improve perceived performance in certain cases.

Chapter 12 has much more on progressive rendering techniques, including placeholder images.

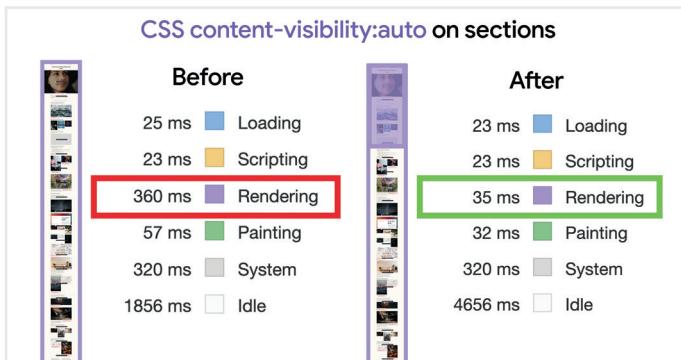
Lazy-Render Offscreen Content

Next, let's discuss the `css content-visibility` property, which allows the browser to skip rendering, layout, and paint for elements until they are needed. This can help optimize page load performance if a large quantity of your page's content is offscreen, including content which uses `` elements.

```
section {  
  content-visibility: auto;  
}
```

The `content-visibility` property¹ can take a number of values; `auto` is the one that offers performance benefits. Sections of the page with `content-visibility:auto` get containment for layout, paint, and style. Should the element be offscreen, it would also get size containment.

¹ <https://web.dev/content-visibility/>



When chunking up a page into sections with content-visibility: auto, developers have observed a 7-10x improvement in rendering times as a result. Note the reduction in rendering times above of 937ms to 37ms for a long HTML document.

Browsers don't paint the image content for content-visibility affected images, so this approach may introduce some savings.

```
section {  
  content-visibility: auto;  
  contain-intrinsic-size: 700px;  
}
```

One option is to pair `content-visibility` with `contain-intrinsic-size`, which provides the natural size of the element if it is affected by size containment. The `700px` value in this example approximates the width and height of each chunked section.

Maintain a Consistent Aspect-Ratio

The aspect ratio of an image is the ratio of its width to its height. This is often represented by two numbers separated by a colon (such as `4:3` or `16:9`). Maintaining a consistent aspect ratio can be important in responsive web design where the dimensions of images can vary and introduce layout shifts depending on how much space is available in the page.

In our image gallery, we might wish to create responsive space for images that vary by dimension, are in more complex elements like cards, or require a placeholder container to avoid layout shifts when the images load and occupy space.

Historically, developers have used the padding-top hack to maintain aspect ratio using an image's width. This involves

using two containers: a parent container, and a child container that gets absolutely positioned. The aspect ratio is then computed as a percentage for the padding-top value.

For example, a 16:9 aspect ratio = $9 \div 16 = 0.5625$ = CSS padding-top: 56.25%. For the following container:

```
<div class='container'>
    <img style='position: absolute; top:0;'>
</div>
```

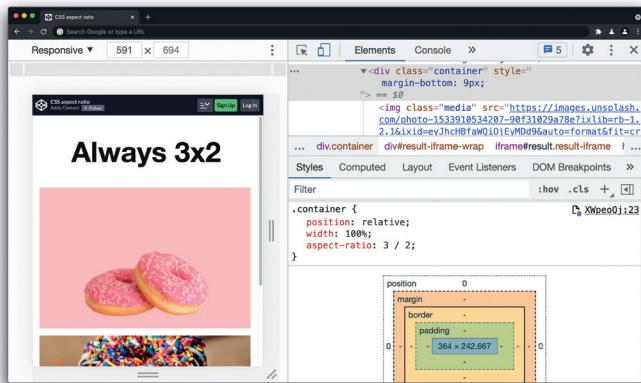
This is the css for the padding-top hack to maintain aspect ratio:

```
.container {
    position: relative;
    padding-top: 56.25%; /* Aspect ratio of 16:9 */
    width: 100%;
}
```

Thanks to the new CSS `aspect-ratio` property,² a more intuitive alternative to the padding-top hack is now available.³ This enables replacing `padding-top: 56.25%` with `aspect-ratio: 16/9` to clearly specify the width to height ratio.

² <https://web.dev/aspect-ratio/>

³ <https://css-tricks.com/aspect-ratio-boxes/>



The new `css aspect-ratio` property, available in modern browsers, is clearer than the `padding-top` hack and doesn't involve more manual calculation for positioning. In the example above, a 3:2 aspect ratio = $2 \div 3 = 0.66666$ = `padding-top: 66.67%`. Thanks to the `CSS aspect-ratio` property, this can just be defined as `aspect-ratio: 3 / 2`.



Throughout this book, we will cover advanced image optimization techniques, as well as how to best use elements like `` and `<picture>` to make your images on the web shine. Now that we've covered the foundations of the modern `` tag, let's turn our attention to understanding image quality and how it affects web performance.

Index

- 7-zip. 158
- accessibility. xii, 206, 226, 348, 359
- adaptive
 - code loading 463, 465
 - media loading 461–462
 - predictor 429
 - quantization 424, 427
- Adobe
 - Illustrator. 202, 213
 - Lightroom 395
 - rgb color space. . . 71–73
- Advanced Video Coding (AVC). . . 384
- advdef 165
- AdvPNG 158, 509
- Akamai. 242, 508
- Aleksandersen,
 - Daniel 398–399
- aliasing. 161–162
- Alliance for Open Media (aomedia) 399
- alpha transparency . 148, 152, 154
- Amazon S3 190
- Android 78, 122, 187, 301, 303, 333, 394–395, 397, 414, 458, 495–497, 499, 503
- Android Pie. 394
- Animated GIFs v, 174, 222, 399, 507
 - replacing
 - Chapter 15: 314–335
- animations vi, xi, 337, 340, 345, 403, 435, 461
- anti-aliasing. 152, 157, 161–163
- API 364
 - fetch 269
- Apple 131, 382, 391, 393, 397, 405, 414, 494
- art direction. xxii, 223, 231–233, 354, 486, 515

artifacts	43–44, 51, 128, 135, 161–162, 172, 178, 184, 196, 324, 351, 353, 407, 409, 425, 428, 502	bandwidth.	xix, 119, 194, 202, 261, 298, 314–316, 354, 379, 432, 461, 466
aspect-ratio	42–44, 484	savings	45
attributes		Base64-encoded . . .	38
decoding	37, 96	Bash.	186, 197
poster	327, 333	Bazel	50
sizes	32	British Broadcasting Company (BBC) .	313
srcset.	33, 224, 230	Beamtic	148
Atwood, Jeff	155	Bendell, Colin	ii, xvi, 130, 336
Authoring Features .	447	Better Portable Graphics (BPG).	133
av1 video codec	131	Blink	98, 188
avc	384, 389, 400	blur filter, CSS	244, 253, 369
AVIF.	Chapter 18: 398–415 v, xiii, xxiv, 131, 199, 283, 378, 380–381, 430–432	Book of Speed	241
background image .	38, 476	Brunner, Gunther .	239
Backups	348	Butteraugli	47, 49–51, 54–55, 133, 138–139, 142, 424, 505
Baldauf, Tobias	245	Bynens, Matthias	ii, 290, 307

- C++ 50
cache 98,
193, 256–258, 260–261,
263–274, 276, 278, 287, 373
duration. 260
enabler. 193
hit ratios 265
lifetime 266
miss 265
offline caching . . 269–271,
274
opportunities . . 268
Cache-Control 257–260,
262–263, 467
Caching Image Assets
 Chapter 13: 256–289
Calibre 106–107
Camera Raw 395
Cascading Style Sheets
 (CSS) ix, x, xx,
33, 40, 43–44, 81, 92, 201,
204, 207–208, 210, 218, 226,
230, 246–247, 251–253, 271,
273, 277–278, 281, 297, 302,
310, 476–477, 483–485
 aspect ratio 481
backgrounds 38, 251,
288
blur filter 244, 253,
369
dimensions. viii
display. 208, 249
pixel volume viii
pseudo-selector . . 252
sprites 278
Working Group . . 484
Chocolatey 319
chroma 55,
57–58, 124–131, 159
subsampling 57, 61, 67,
87, 124–128, 130–131, 135,
178, 193–196, 432
upsampling 90
values 195
Chrome on Android 78, 458
Chrome DevTools ...
 audits panel 46
 elements panel . . 32, 487
 lighthouse panel . 100, 236,
288, 478
 network panel . . . 34–35,
188, 251, 260, 514
 performance panel 84, 93,
329, 478–479, 488, 514

CIELAB	71, 129, 426	Adobe RGB	71–73
client-side	94, 281, 477, 502	ProPhoto RGB . . .	72
Clipping Path Zone.	. 73	SRGB.	71–74, 77–78, 132, 143, 509
Cloudinary	46, 54–55, 59, 119, 142, 242, 333, 338, 343–346, 350, 356–357, 361–362, 392, 396, 418, 420, 437, 508	compression client-side.	502–504
content management system (CMS) .	340–341, 348	modes	115, 398
CodePen	325	photographic images comparison.	437
color management Chapter 4:	68–79	non-photographic images comparison.	438
color models	68–69, 71, 129	Consistent Aspect-Ratio	42
additive	69	constant rate factor (CRF)	320–324
cmyk	69	content delivery network (CDN)	xxiv, 192–193, 265–267, 276, 315, 415, 492
rgb	69, 71	conversion	192
subtractive	69	Image CDN. Chapter 16: 337–379	Chapter 16: 337–379
color profile	76–78, 172	performance	358
color space	71–72, 74, 76, 78, 87, 89–90, 159, 175	CopyTrans.	395
Core Web Vitals	v, xxv, 456, Chapter 22: 471–489		

- CPU xxiii, 80, 122, 141, 173, 244, 329, 331, 507
- CSS Tricks 277
- cumulative layout shift (CLS) 307, 472–474, 479–480, 482, 487–488
- cyclic redundancy code (CRC) 145
- Data Saver Chapter 21: 457–470
- Data Saver mode. xxv, xxvi, 333, 465–467, 496–497, 499
- decoding. iv, xvii, 39, 42, 80–81, 83–85, 87, 89–99, 121, 131–132, 143, 178, 199, 315, 331, 344, 384, 405–406, 410, 415–418, 421, 429
- asynchronous 37, 95–97
- attribute. 37, 96
- decode() method . 95, 97–98, 416
- developer-controlled image performance Chapter 6: 100–113 94
- Deflate 158, 165
- Delivery Features comparison. 442–443, 452–453
- Derivations 387
- device pixel ratio. viii–ix, xxvi, 104, 225–227, 230
- digital asset management (DAM) 348
- discrete cosine transform (DCT) 53, 87–90, 177, 424–425
- display:none 249–252
- distance threshold. 303, 310
- DOM tree 82
- dots per inch (DPI) 493–494
- Drasner, Sarah 79
- DSSIM 48, 54, 67, 350, 406
- Edge. 78, 83, 96, 130–131, 151, 234, 261, 279, 299, 301, 333, 343, 354, 380, 407, 409, 457, 465
- Efficient Compression Tool (ECT) 165–166

element	extensible markup
img	language (XML) 39, 136,
Chapter 1: 29–40	200, 207, 210–211, 244
picture	Facebook 118, 120,
189–190, 231–235, 249, 297,	179, 346
308, 402, 431, 486, 509	fallback image 234, 284
source	Fetch api 269
189–190, 231–234, 308, 323, 327–328	FFmpeg 317–320,
video	323, 372, 396, 411–412
325–328, 411, 476	Figma 202
Elements panel	Firefox 78,
encoding	96–97, 131, 188, 279, 299, 301,
xiv, xxi, 47, 51, 75, 85, 88, 104, 111,	312, 401, 405, 411, 485, 507
132, 134, 137, 139, 171, 175,	First input delay
177, 183, 194, 196, 320, 325,	(FID) 472–473
331–334, 344, 371, 384, 397,	free lossless image format
403, 407, 410–411, 415–420,	(FLIF) 53,
425, 429–430, 507	420–421
Entropy Coding	free universal image format
Erdmann,	(FUIF) 418
Christoph	freshness 261–263,
ETag	276
Everts, Tammy	Furnspace 362
EXIF	gamma correction 75
117, 129,	
172, 385, 389	

- gamut 57
 72–74, 78, 130–131, 178, 346,
 399, 403, 417, 425, 432
- Gao, Nancy 460
- Gaussian blur. 246
- Gifsicle. 334
- GIMP. viii, 187,
 351, 395
- Giphy. 174
- GitHub. 139, 190,
 245, 409, 411
- Gmyr, Chris 357
- Google xiii, xx,
 49, 51, 131, 133, 138, 170,
 175–177, 181, 187, 206, 211,
 217, 238–239, 274, 278, 306,
 339, 380, 394–395, 399,
 405, 407, 418, 424, 491
- Core Web Vitals . . v, xxv,
 456, Chapter 22: 471–489
- Cloud Platform . . 374
- doodle 157, 176
- Offline Cookbook 272
- Pixel 394
- Workbox Recipes 272
- gradient image placeholders
 246–247
- graphics 75,
 81, 111, 133, 144, 147–148,
 162–163, 223, 244, 278, 315,
 417, 435
- graphics processing unit
 (GPU) 80, 244,
 246
- portable network graphics
 (PNG) Chapter 8: 144–169
- raster. 111, 162
- scalable vector graphics
 (SVG) Chapter 10: 200–219
- vector 111, 162,
 202, 518, 525
- Greenstein, Ben . . 460
- Group Splitting . . 428
- Guetzli 51, 111,
 134, 138–143, 379, 507, 509
- gulp. 123, 136,
 140, 166–167, 184–185
- Gumby 243
- gzip 158, 211,
 217, 265
- Hansen, Patrick . . 153–154

- hashing 263
- high dynamic range (HDR)
130–131, 398–399, 403, 407,
417, 425, 502
- High Efficiency Image File
Format (HEIF) . . v, xxiv,
61, 64–65, 67, 131, 133, 356,
378, 403
- Chapter 17: 382–397
- HEIC v, xiii,
356, 378, 382–385, 387, 389,
391–397, 403, 427
- HEIC Converter . . 393–395
- high efficiency video cod-
ing (HEVC) . . . 133, 382,
384, 389–391, 396, 400, 403
- Hidayat, Ariya . . . 143
- histogram 159
- .htaccess 190–192
- Huffman coding
algorithm 88
- International Color
Consortium (ICC)
77–78, 172, 424
- color profiles 424
- Image Formats
- Comparing Chapter 3: 54–67; Chapter 20:
433–453
- ImageMagick 47, 53,
122, 136, 187, 237
- imagemin 122–123,
136, 140, 167, 183–185, 507
- ImageOptim 43, 77,
135–136, 140, 164–165, 407,
505, 508
- iv, xxvi,
44, 98, 189, 202–204,
206–207, 224, 231–232, 235,
243, 281, 283, 285, 298, 300,
306–308, 325, 329, 332, 396,
475, 482, 486
- element . . . Chapter
1: 29–40
- imgix 243, 357,
361–362, 508
- Instagram 232
- interlaced display . . 154
- Intersection Observer
... 294–295, 298

- iOS 118, 120,
122, 130–131, 327, 380, 382–
384, 393, 414, 495–497, 499,
503
- ISO BMFF 389, 403
- jank 91–92,
97–99, 329
- JavaScript x, xx,
xxiii, 81, 84, 94, 97, 201,
218, 243, 247, 249, 251–253,
260, 269, 271, 274, 279,
281, 283–285, 293, 295,
298–299, 303, 305, 311, 396,
459–461, 463, 465, 477
- libraries xxiii,
274, 295, 298–299
- Jetpack 193
- Jobs, Steve 494
- JPEG chapter
7: 115–143
2000 xxiii, 61,
63, 65, 67, 130, 133, 380, 398,
426, 491
compression
modes 115
decoding 131
- encoders 58, 61,
89, 119, 134, 138, 142
- jpeg2png 353
- standard. 89
- XL v, xxiv,
132, 199, 254, 378, 380–381,
Chapter 19: 416–432
- XR xiii, 130,
380
- XT 428
- whitepaper 416
- Kayser, Frédéric . . . 126
- KeyCDN 193
- Kobes, Steve. 82–83,
91
- Kodak. 407
- largest contentful
paint (LCP) 279,
282–283, 288–289, 309,
472–476, 478–479
- Last-Modified 257–258,
260, 268
- latency 92, 120,
122, 262, 358, 431, 491–492,
494–496, 508

lazy loading	xii, xv, xxiii, 36, 290–291, 293–295, 297–298, 302–303, 305, 307, 310–311, 459	Linux	165, 319, 414, 505
images, Chapter 11:	223–237	Liquid Web	240
Lempel-Ziv-Welch algorithm	158	live photos.	380, 382–383, 391–392, 403
Leptonica	187	lossy files	171
Lesinski, Kornel	48, 142, 407	low quality image placeholders (LQIP) 38, 242–243, 245–246, 252–255, 295, 417, 497	
libaom	410–411	luma values	195
libHEIF.	396	LZ77.	158
libjpeg-turbo	58–61, 67, 119, 128, 135, 351, 428, 491	macOS	31, 130, 164, 179–180, 182, 187, 319, 383, 393–394, 413–414
libvips	53, 507	Magento.	348
licensing.	133, 354, 356, 360, 400	McAnlis, Colt	160, 176
Life of a Pixel	83, 91	McComb, Glenn	254
Lighthouse panel	100, 236, 288, 478	Median Cut	166
Lightroom.	72, 395	metadata.	159, 164, 172, 183, 210, 334, 385–386, 389–390, 413, 507
limits	106, 427, 441, 469	Microsoft Paint	405
comparison tables	440, 452	MKV container.	412

Moving Picture Experts Group (MPEG)	.384
mpeg-4	.319–325, 328–329, 331, 383
Mozilla	.51, 135, 399, 405
mozjpeg	.51–52, 58–61, 67, 111, 117, 119–120, 128, 133–138, 142–143, 186, 371, 379, 432, 505, 509
mp4box	.411
natural pixels	.viii
Navbharat Times	.194
Netflix	.175, 399, 405, 407–409
Netlify	.348
Nginx	.192, 349
Nine Degrees Below	78
Node.js	.48, 101, 183
Nokia	.356, 384, 390, 396
Offscreen Content	.40
opacity	.144, 148–149, 152–153
OpenGL	.81
optimizing	
Image Quality	. Chapter 2: 43–53
JPEG	.134
largest contentful paint (LCP)	.279
reoptimizing	.352
SVG	.219
Thumbor	.341, 361, 363–365, 367–375, 508
Optimus	.193
optipng	.165–167
Orback, Vincent	.190, 192
padding-top hack	.42–44
PageSpeed Insights	.46, 289, 473, 478
Parallelization	.417
performance	
AVIF	.405
budget	.106–107
CDN	.358
impact	.xx, 347, 359
page load	.40, 269
panel	.84, 93, 329, 478–479, 488, 514

- perceived 39, 115, 117, 238, 252, 523
- photographs xxii, 115, 417, 419, 434
- Photoshop. viii, 47, 72–73, 76, 123, 128, 144, 160, 187, 212, 351, 414
- PIK 418
- Pingo 167
- Pinterest. 120–121, 238–239
- pixel
 - density. viii, 33, 229–230, 493–495
 - fitting 162–163
 - hinting 162
- Pixelmator 187, 395
- Pixels per inch (ppi) 493
- jpeg 115, 118–119, 121
- placeholder 38–40, 42, 93, 98–99, 239, 245, 247–248, 252–253, 291
- Portable Network Graphics (PNG) 144, 518
- Beamtic 148
- Chapter 8: 144–169
- compression 158
- palette modes 146
- png-8. 146–147, 150, 153, 156–157, 163, 499–502
- png-24. 146–148, 153–154, 156, 163, 500, 502
- png-32. 146, 148, 153–154, 156, 500, 502
- posterizer. 166
- PNGcrush. 164, 166–167
- PNG Optimization
 - Tools 164
 - pngOptimizer 168
- PNGout 164, 168–169
- PNGquant. 164, 168, 371, 507, 509
- PNGwolf. 169
- polyfill 310–311, 344
- Portis, Eric. 142

- Posterization 160–161
Potts 313
Preloading. 256,
 279–281, 284, 288
progressive v, 40,
 98, 115–118, 120–124, 131,
 135, 154–155, 222, 270, 293,
 337, 362, 415–417, 420–421,
 430–431, 496
decoding 178, 417
enhancement . . . 401–402,
 431, 463
image rendering . 155, 239,
 252, 255
jpeg. xv, 115,
 117–118, 123, 241, 491–492, 505
png 155
scans 421
rendering techniques
 Chapter 12: 238–255
ProPhoto RGB 71–74
Proxying pages. 459
Peak signal-to-noise ratio
 (PSNR). 54
quality index 44, 47,
 321
quantization 52–53,
 57–58, 88, 135–136, 353,
 424, 427–428
QuickLook. 180, 413
Quick Sync Video . 331
RGB
 color model. 69, 71
 transparency. 172
raster 83, 111,
 144, 161–162, 201, 212–214,
 218
rasterization 82–85
real user monitoring
 (RUM) 466, 488
recompression 52,
 352–353, 424
responsive
 breakpoints. 235
 design xxii, 203,
 420
 images. Chapter
 11: 223–237
 image techniques 234
Retina ix, 223,
 230, 494–495
density. 495
display. 494

- revving 263, 267
- Rigor 334
- Rogers, Philip. 82–83, 91
- royalty-free 381, 398, 403, 418, 449
- comparison. 448, 452
- Safari 36, 78, 97, 130–131, 241, 279, 301, 312, 332, 380, 396, 485
- scalable vector graphics (SVG)
Chapter 10: 200–219
- screen pixels viii
- security 53, 204, 340–341, 345, 373–374
- sequences 131, 386, 389, 403
- service workers 269–271, 274–276, 461
- Shopee 98–99
- Shopify. 468
- Sketch 43, 124, 144, 187, 202, 212, 219
- Sneyers, Jon. ii, 52, 416, 420, 422–423, 431, 437
- SOASTA xx
- Software Support . . . comparison. 450, 452
- speed comparison. 438, 452
- SpeedCurve 106–107
- spriting. 277–279
- SQIP 38, 244–246, 252–255, 295
- Squoosh 43, 164, 181, 197, 410, 412, 430
- srcset 32–33, 36–38, 93, 189, 203, 224–225, 227, 229–236, 249, 297, 308, 402, 431–432, 486, 508
- SRGB 71–74, 77–78, 132, 143, 509
- SSIM 47–49, 51, 133, 137, 143, 350, 355, 505
- SSIMulacra. 54–55, 59, 67
- stale-while-revalidate
(SWR) 262–263
- Stefanov, Stoyan 241, 247

- SVG
 Chapter 10: 200–220
 icons 205–206,
 219, 270
 maps 207, 209
 svgo 186, 212,
 214–216, 507
 svgomg 212–213
Thumbor. 341, 361,
 363–365, 367–375, 508
TIFF. 53
Tinder 469
transitional features
 comparison. 448, 452
transparency 55, 67,
 144, 159, 163, 166, 168–170,
 172–173, 184, 199, 344–345,
 382, 388, 390, 499, 501
alpha. 148, 152,
 154
support 131, 149,
 168, 380, 399
Tribune. 470
Tumblr. 174
Twitter. v, xxv,
 xxvi, 93–94, 120, 174, 456
Image Pipeline . . Chapter
 23: 490–504
Lite 93,
 496–497
URL 39, 101,
 180, 193, 205, 251, 263–264,
 266–267, 278, 325, 344,
 364–365, 368–369, 476
Vary header. 264
vector. 161–162,
 200–202, 214, 218
graphics. 111, 162,
 202
versioning. 263,
 266–267
video vi–xi,
 xv, 65, 75, 92, 125, 133, 171,
 242, 278, 314–328, 331–335,
 342, 346, 356, 359, 371–372,
 382, 384, 389, 392, 396, 398,
 400–401, 411, 426, 462, 476,
 481
autoplay. 326–327,
 469, 499
decoding 131, 331
Quick Sync Video 331

VLC media player . . .	405, 413	WebPShop.	187
VoxMedia	175	Website Speed Test	.46
W3C.	202	WhatDoesMySiteCost.com	
Wagner, Jeremy . . .	ii, 114,	· · · · ·	X
	186		
Web Almanac.	vi, ix, xii	wide color gamut (WCG)	131,
WebM	317–319,	399, 425	
	322–325, 328–329, 331, 507	width descriptor	. . . 229–230
WebP		Wikipedia	287, 341
browser support.	179	Wiltzius	91
Chapter 9: 170–199		WordPress.	193,
encoding	175	348–349	
lossless	172	Workbox Recipes . .	272
lossy files	171	XnConvert.	182–183,
serving	170, 175,	186	
	188, 192–193	YCbCr.	87, 90,
		129, 178, 426	
		Yelp	58, 120
		zopfli	158, 169
		ZopfliPNG.	165, 169



Smashing Library

Expert authors & timely topics
for **Smashing Readers**.



smashed.by/library



More Smashing Books

Crafted with care for you, and for the Web



TypeScript in 50 Lessons

by Stefan Baumgartner



Inclusive Components

by Heydon Pickering



The Ethical Design Handbook

by Trine Falbe,
Martin Michael Frederiksen
and Kim Andersen



Form Design Patterns

by Adam Silver



Art Direction for the Web

by Andy Clarke



Click! How to Encourage Clicks Without Shady Tricks

by Paul Boag



The world is a miracle. So are you.
Thanks for being smashing.

“An incredibly comprehensive overview of image optimization. This book will teach you everything you need to know about delivering effective and performant images on the web.”

—**Katie Hempenius, Google**

“Optimizing image delivery is key to building high-performance web apps. This book explains everything developers should know about choosing the right image format, compressing image assets — and more!”

—**Mathias Bynens, Google**

“Images are the heart and soul of the web; they help create that emotional connection with humans. Yet, it is really easy to ruin that experience through slow loading or worse, over quantizing the pixels and distorting images. Understanding how images work is essential for every engineer; the last thing we want is to deal with open bugs from bad creative or performance experiences.”

—**Colin Bendell, Shopify**



Addy Osmani is an engineering manager working on Google Chrome. His team focuses on speed, helping keep the web fast. Devoted to the open-source community, Addy's past open-source contributions include Lighthouse, Workbox, Yeoman, Critical, and TodoMVC.