Raster images

Raster images can be thought of as a set of pixel-by-pixel instructions for rendering a two-dimensional grid. Common raster image formats include GIF (.gif), JPEG (.jpg), PNG (.png), and WebP (.webp). The way each image format compresses and encodes these instructions differs, resulting in a huge variance between file sizes: a photographic image encoded as a JPEG might only be a few hundred kilobytes, while the same image encoded as a PNG might be several megabytes, without any discernible difference in quality to the end user.

A raster image source scaled beyond its inherent dimensions will appear distorted, blocky, or blurred:

https://codepen.io/web-dot-dev/pen/ZEjBQqB

For artwork containing real world levels of detail, raster images are the right tool for the job.

Just like choosing between raster and vector images, choosing the appropriate type of raster image ultimately comes down to the use case. When we break raster images down into their encodings, what we're really talking about are the methods used to describe their contents, and the compression methods (or lack thereof) we're applying. Remember that a server doesn't send *an image* over the wire to a browser, but a stream of bytes describing the pixel grid that makes up that image for the client to recompose.

So, to better visualize the process of encoding a grid of pixels as bytestream data, I want you to imagine that you're acting as the web browser. You have a sheet of millimeter graph paper and a specific pack of brand name crayons. I, as the web server, have the exact same things—but I've already used my crayons to fill the graph paper in with a source image. If I were to send you a plain text message, I couldn't send you the image itself, but I could convey information about an image source in a language we both understand using our shared standard for our “pixel” grid and colors:

> “Start at the top left. Row one, column one is blue. Row one, column two is blue. Row one, column three is blue. Row one, column four is red.

Using this textual information, you'd be able to perfectly recreate the image I have on my sheet of graph paper.

Three horizontal blue boxes followed by one red box.

Differences in image formats and the way they're encoded as data can be loosely thought of as the way this information has been formatted. For example, the information I've sent you could be just

as readily expressed as:

> Start at the top left. Row one, columns one through three are blue. Row one, column four is red.

Either one of these descriptions will result in the same image, but the second one manages to describe the same image with fewer characters. This is a *lossless* method of compressing image data: all the same information—and thus, no reduction of visual fidelity—but fewer bytes transferred over the wire from me to you—from the server to the rendering engine. This is the plain language equivalent of “run-length encoding” for image data, where data is encoded as the value to be repeated and a count, rather than repeating the full value multiple times.

The inverse, *lossy* compression, might sound like a non-starter at face value—why would you ever want your images to look *worse*? That isn't strictly the case, though, and it's worth keeping in mind that our eyes don't have perfect fidelity either. Choosing the correct format and settings for image compression is an exercise in finding the balance between the level of visual detail we're able to perceive and the amount of data sent to the browser. Both of these factors are determined by the content of our source image.

The raster image formats are those you are likely most familiar with as a developer—GIF, JPEG, PNG, WebP, and more. You'll learn about the capabilities of each in the next few modules.

□□□ #3
□ □□□ □□□ 2026-10-31 22:53:55 UTC
□ cocobook-sync □□□ 2026-11-04 17:39:26 UTC