

THE BOOK OF GIF

A Comprehensive Guide to Optimizing GIFs

Brought to you by:



The Book of Gif

A Comprehensive Guide to Optimizing GIFs

INTRODUCTION

OPTIMIZING STATIC GIFS

LOSSLESS OPTIMIZATION FOR ANIMATED GIFS

LOSSY OPTIMIZATION FOR ANIMATED GIFS

OPTIMIZING ANIMATED GIFS WITH HTML5 VIDEO

TAKEAWAYS

Introduction

Despite being nearly 30 years old, [the GIF image format](#) is a major force on the web today. Originally created as an image format for CompuServe in 1987, animated GIFs as we currently know them didn't even exist until 1995 when Netscape Navigator 2.0 added a special Application Extension to enable looping. This makes GIF the oldest image format that's commonly used on the web today.

Due to their age, simple nature, and widespread support, GIFs are widely present on the Internet. According to the HTTP Archive, 24% of all images on the web

use the GIF format. While originally used as a format for normal images, GIF has enjoyed continued success due to its ability to support animation. While GIF's animation abilities were first used on the web for banner advertising, they are now commonly used as a easy way to share short, soundless video clips. A tremendous culture has grown around [people posting animated GIFs of jokes and memes](#).

GIFs and Web Performance

Given the popularity of GIFs, it is critical to understand how they impact web performance of websites and web applications. The performance of a site is directly tied to the user experience, and the user experience is directly tied to business objectives. The primary way to optimize images like GIFs is to reduce their size. After all, having less data to send to a browser means you can deliver it faster. There are two types of optimizations you can apply to reduce the size of optimized GIFs: Lossless optimizations and lossy optimizations.

Lossless optimizations can reduce the size of the file, without altering the graphics data itself. That's why they are called "lossless" optimizations; no graphical data is lost. The original GIF and a GIF that has been losslessly optimized will appear exactly the same, and the lossless one will be smaller. However, since lossless optimizations can't alter graphical data, they tend to result in only relatively modest size reductions of 5%-20%.

Lossy optimizations actually modify the graphical data in the GIF. This means the original GIF and the lossy optimized version are not pixel-perfect copies of each other. While "modifying" graphical data sounds scary and bad, if you are smart

about how you do it, you can get much smaller files without significantly degrading the quality of the image. This is especially true for animated GIFs, as we will soon see.

In this guide, we will see how lossless and lossy techniques can be applied to GIFs to reduce their size, and ultimately, minimize their impact on the performance of your site.

Since lossless and lossy optimizations strategies both focus on reducing size, it is helpful to understand why GIFs can be so massive to begin with. This largely comes down to two issues: How we are using GIFs vs what their capabilities are. As you will see, we typically push GIFs to do tremendous things that GIFs were not originally designed to do.

Animated GIFs for Video Clips: A Terrible Hack of Convenience

Today the major use case for GIFs are simple, embedded, looping video clips. Even major publishing and news media sites use GIFs to include animated video clips within stories or for advertisements. The problem is that **animated GIFs were never designed to store video**. In fact, GIFs weren't designed for animation at all. Here is a quote from the [actual GIF89a specification](#):

"The Graphics Interchange Format is not intended as a platform for animation, even though it can be done in a limited way."

GIFs were designed to contain multiple images inside a single file as a way to reduce overhead by combining the redundant info, like common headers or color data of each image. Think something more like “a ZIP file of images” instead of an animation format. It wasn’t until 6 years after GIFs were invented that [Netscape 2.0 added an application extension](#) enabling animated GIFs as we know them today.

Unfortunately, all this means that animated GIFs are perhaps the worst possible file format to store video. We have decades of research into how to compress digital video which have yielded tremendous breakthroughs like [chroma subsampling](#), [discrete cosine transforms](#), or [motion compensation](#). Unlike actual video files, GIF uses none of these techniques, meaning **a few seconds of content in a GIF can be megabytes in size.**

In 2016, the average web page is just over 2 MB in size. **Slow, unoptimized, bloated GIFs several MB in size can significantly impact the performance of an average web page.**

Even though animated GIFs are huge, slow, and don’t look as good as traditional video, there are still several benefits:

- Universal support for playing animated GIFs on many platforms
- Easy and widely available tools for creating GIFs
- No patent or licensing concerns that apply to common video codecs

These advantages encouraged a culture to develop around creating and sharing funny animated GIFs, and this culture has spread to their use on more mainstream sites. However, we know that GIFs are not ideal for the job of embedded video.

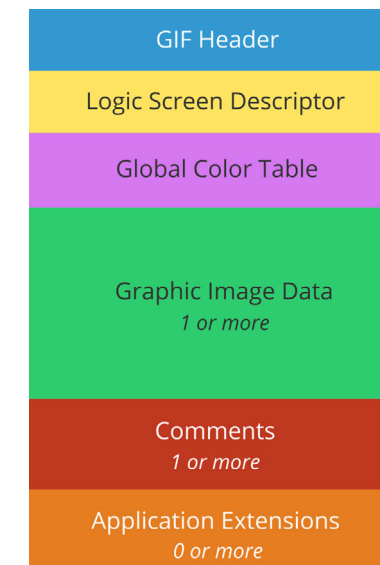
The Surprisingly Simple Format of a GIF

The other factor impacting the size of GIFs is the design of the format itself. What is this 30 year old specification capable of doing, and how might we coax it into doing those things more efficiently, resulting in less content we have to transmit to a visitor?

To understand how to optimize GIF images, let’s quickly explore [the structure of the image format](#) to identify which areas can be streamlined or optimized.

Note: While it’s not required that you understand everything in this section, a basic understanding of how GIFs are structured and how they function will help you understand the basis for the optimization techniques we’ll discuss in the rest of this guide.

The next figure shows the the layout of the data inside of a GIF image. While GIFs are composed of many different parts, we will only focus on the major ones.



GIF File Format Structure

To start, GIFs contain a **Header** and a **Logic Screen Descriptor**. These tell a program that the file is a GIF and give the dimensions of the image, as well as some additional less important properties.

The next major part of a GIF is the **Global Color Table**. This defines the color values for each of the up to 256 distinct colors which can be referenced by the graphics data.

Now we get to where the content of an image is saved inside the GIF format. This happens in the **Graphic Image Data** sections. If the GIF is a static, non-animated GIF then it will have one Graphic Image Data section. If the GIF is an animated GIF it will contain multiple Graphic Image Data sections, one for each different frame of animation.

Each Graphic Image Data section contains a few other pieces of data:

- *The graphics data* that represents an image or frame of animation
- *A local color palette* for each image or frame of animation

The graphics data inside each Graphic Image Data section is a bitmap of pixels. Each pixel is a number to an index in the color palette. The palette defines which the red, green, and blue values that represent the color. All of this pixel data is compressed using the [Lempel-Ziv-Welch \(LZW\) lossless data compression algorithm](#).

Each image can also optionally contain a local color palette specific to that piece of image data. This means that the first frame of an animation can be composed

of colors from a palette of 256 colors, and the second frame can be composed of a different palette of colors, and so on. Similar to how inline CSS takes precedence over CSS in an internal stylesheet, the local color palette data will override the palette data set in the Global Color Table. In fact, the Global Color Table is actually optional, because each Graphic Image Data section can define its own palette. When you are dealing with a static image it doesn't matter whether the palette information is stored in the Global Image Data of the single Graphic Image Data section. As we will see later, we can leverage these options for storing color palettes to optimize GIF animations.

Comments and **Application Extension** sections are both common, optional data sections that could be present. These sections can be used to supply a variety of application-specific information, for example the commonly implemented Netscape 2.0 Application Extension which enables the looping of GIF animations.

Adobe products store XMP image metadata inside of the Application Extensions. Embedded thumbnails can be stored in Application Extensions. There are also lesser-used GIF features which use additional data sections. For example, the Graphic Image Data section could contain other data sections such as a Plain Text Data section, which allows for text to be rendered on top of an image similar to a closed captioning or subtitling system for video files. With the exception of a special Application Extension which controls how animated GIFs loop, all of these sections are largely unimportant. We can re-organize or remove these to optimize GIFs.

Optimizing Static GIFs

Before we get deeper into optimizing animated GIFs, let's take a moment to cover how to optimize static GIFs. After all, animated GIFs are just static GIFs with additional images inside for each animation frame.

When we are losslessly optimizing a static GIF, we are reducing its size without altering the graphics data. Here are some things to consider:

- The LZW compression algorithm used to compress and store pixel and color data has its roots in the 1970s. While impressive for its time, its performance and compression ratios have been eclipsed by more modern lossless compression algorithms. **We can use more efficient compression.**

- Palettes may contain more color definitions than actually used by the image. **We can discard this extra data.**
- The size of palette entries can be inefficient when the image contains less than 128 colors. **We can reorganize palettes to reduce size.**
- GIF files can contain extra cruft like comments, metadata, and some Application Extension sections that don't contribute to the rendering of the graphic data. **We can remove this extra data.**

Since the majority of data inside of GIF files is the compressed pixel and color information, finding a better way to compress and store that data is the best way to losslessly optimize a GIF. We can do this by converting static GIF images into PNG images.

[PNG images](#) are a lossless image format that very closely mirror GIF images. The PNG format was defined 10 years after the GIF format, and PNG images address many of the shortcomings of GIF images, such as:

- PNG's DEFLATE algorithm achieves better compression than GIF's LZW algorithm.
- PNG images support a precompression filter step, which rearranges graphic data before compression to maximize redundancy and thus improve the efficiency of DEFLATE compression. GIF does not have this feature.
- The size of PNG palette entries can be smaller than the corresponding GIF palette entries on images with less than 256 colors.
- PNG's ancillary data sections support compression allowing their overall size to be reduced.

All these improvements add up to serious savings.

We analyzed 5500 GIF images collected by users of [Rigor's free performance report](#) and compared the difference in size of those same images when converted to PNGs. On average, static GIF images were 21% smaller when converted to PNG images. Since the PNG images are pixel-perfect copies of the original GIF images, this is an obvious and easy optimization to make!

Some GIFs are Larger as PNGs

PNG images can *sometimes* be larger than the source GIF image. PNG supports millions of distinct colors in an image, but GIF does not. Some tools could save PNGs in larger, richer formats like PNG-24 or PNG-32. Because a GIF only supports 256 distinct colors, when we're converting from GIF to PNG we only need to use PNG-8, or a PNG with an 8-bit color depth. When optimizing GIFs by converting them to PNGs, ensure you are exporting to a PNG8 image.

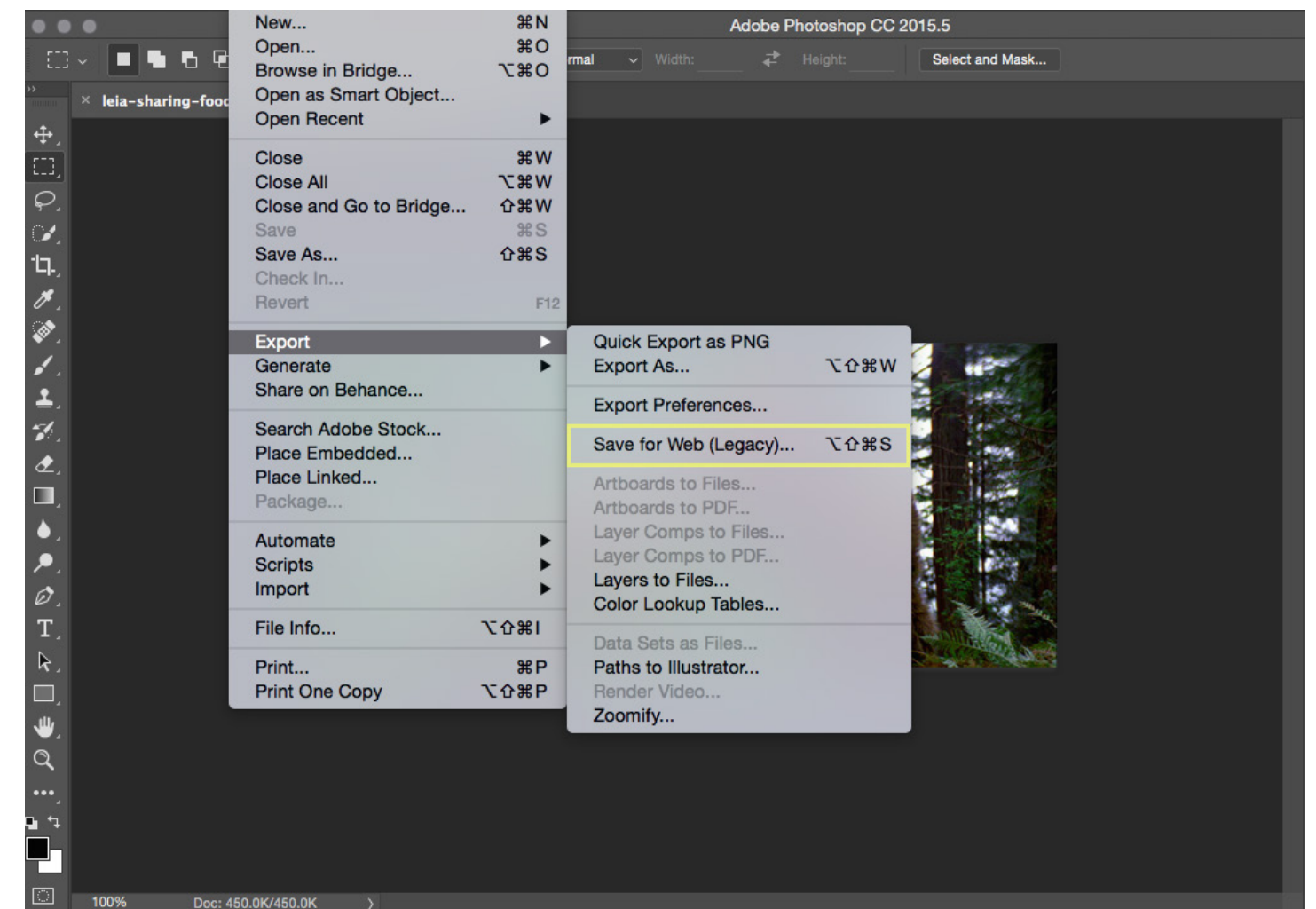
Even when using PNG-8, it is still possible for an original GIF image to be smaller than the new PNG image. This usually only occurs for extremely small images, where the actual graphical data is quite small. For very small images, the overhead of various headers and sections inside of the GIF or PNG image contribute to the overall size more than the graphics data. In these cases, very simple GIFs can be smaller than PNGs.

Typically these very small GIF images are used on websites as space images or as responses for web services beacons. These days you shouldn't be using spacer GIFs at all, and you should ideally be returning HTTP 204 No Content responses

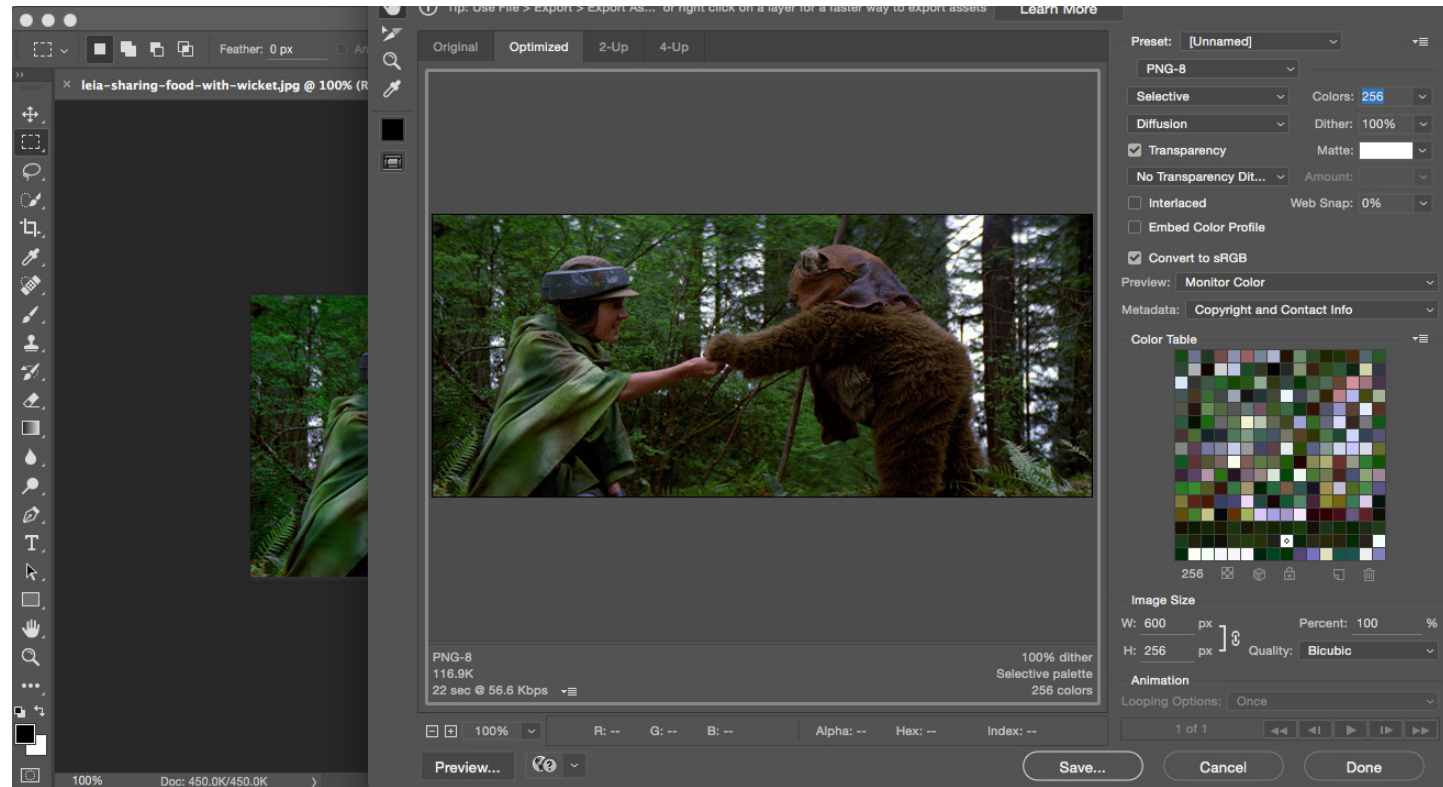
instead of using tiny images. In short, if you find any GIFs on your website that are smaller as GIFs than PNGs, you should investigate why those GIFs are smaller and how they're being used.

Replacing Static GIFs with PNGs

Any modern image editing program like Photoshop, GIMP, or even [Paint](#), should be able to open a GIF image and save it as a PNG image. These options vary from editor to editor, and usually exist under the **Save As...** or **Export As...** settings.



In Adobe Photoshop we can click **File > Export > Save for Web (Legacy)...**



In the **Export As...** window we'll find the option for PNG-8 and we can set the maximum number of colors to 256 and **Save...**

If you prefer a command line tool, the open source image manipulation tool suite ImageMagick's convert program can convert GIF images into PNG images. In the example below, `convert` creates the PNG image `smaller-cats.png` from `cats.gif`, the original GIF image.

```
convert cat.gif smaller-cats.png
```

All modern browsers, both for desktop and mobile, support PNG images. This

means that PNG can serve as a drop-in replacement for all static GIF images on a website. But does this mean you need to go through and update all tags `` into `` tags? Actually, no.

Browsers [don't care about the name of a file](#). If you are feeling lazy, you can convert your GIF images like `logo.gif` into PNG images `logo.png`, and then rename those PNG files to `logo.gif`. The file name makes will make the file look like a GIF but technically the file will be a PNG. This way you can optimize the image without needing to update all of the hyperlinks on your website.

Lossless Optimization for Animated GIFs

Animated GIFs are GIF files that store a set of still images inside, and there is a Graphic Data Section in each one. Animated GIFs also contain some extra data about how long to wait before advancing between images and how to loop the animation. You can think of animated GIFs as slightly more complex versions of static GIFs.

Animated GIFs are primarily used on the modern web for:

1. Animated UI elements, such as progress indicators or status bars

2. Video clips that have been saved as animated GIFs for embedding and sharing easily

We will see that these two different use cases have a profound impact on the types of optimizations you can make. First, let's start with lossless optimizations.

Since animated GIFs are just more complicated versions of static GIFs, the list of lossless optimizations that can be applied to animated GIFs is, in many ways, a superset of the optimizations available for static GIFs. Unfortunately PNG images don't natively support animation, so while we can't just convert animated GIFs to PNGs, there are a number of other lossless techniques we can implement:

- Remove metadata, or unused palette entries from a GIF and write a better optimized GIF
- Combine or generalize local palette information in individual Graphic Image Data sections into the Global Color Table
- Re-use existing Graphic Image Data sections as animation frames
- Minimize what is changing between animation frames, reducing the size of different Graphic Image Data sections

If this sounds complicated, don't worry! [Gifsicle](#) is a great command line tool which can, among other things, perform most of these lossless optimizations automatically. It's free, open-source, and easy to use. Because it's a command line tool, Gifsicle can be scripted for bulk optimization operations or bundled into build scripts. Using Gifsicle is super easy, as shown in the example below:

```
gifsicle -O3 -o output.gif input.gif
```

This results in an optimized GIF named output.gif.

What sort of savings can you expect to see when using Gifsicle? The answer depends on which type of animated GIF it is.

Earlier in this guide we discuss how there are two major types of animated GIFs: UI animations and video clips. Gifsicle is awesome at optimizing the first use case: UI animations, like progress spinners. This is because progress spinners are usually short, simple animations that don't change much between frames. In those cases, nearly all of the possible lossless optimizations techniques can be applied. We've seen Gifsicle **reduce the size of animated GIFs for UI animations by 20-60%**.

Unfortunately, Gifsicle is not great at losslessly optimizing for the second use case: video clips saved as GIFs. There are two reasons for this.

First, lossless optimizations cannot modify the graphical data inside an image. That's what makes them lossless! Animated GIFs of video clips can be made up of dozens or even hundreds of individual images. This means the vast majority of the data instead of the GIF is graphical data which cannot be altered. Even if Gifsicle could discard all of the non-graphical data, you would still have about 98% of all the data remaining. That means, best case, lossless optimizations for video clips could only be around 2%.

Second, the characteristics of UI animations like progress spinners (being short and simple, with few changes between animation frames) don't usually apply to video clips. As discussed above, animated GIFs are tremendously inefficient at

storing video content, and they exist mostly because of how convenient they are to share. Because of their inefficiency, there is very little that can be done to reduce the size of an animated video clip GIF without altering the graphics data.

In practice, the basic limitations of lossless optimization coupled with the nature of video clips mean that **lossless optimizations could only reduce the size of an animated GIF for a video clip by around 1%**. Lossless optimizations are not good enough for video clip GIFs. We need lossy optimizations.

Lossy Optimization for Animated GIFs

So far, we have only discussed lossless optimizations. These are the safest optimizations to make, since you end up with an optimized image that is identical to the original, only smaller. Unfortunately, animated GIFs are so terrible at efficiently storing video content, we must explore [lossy optimizations](#), which modify or discard some of the graphical data, to see any significant improvements.

While *modifying* or even *discarding* graphical data sounds scary and bad, if you are

smart about how you do it, you can get [much smaller files without noticeably or significantly degrading the quality of the image](#).

How does this work? First, it's important to note that the human eye is not very good at seeing subtle changes in color. A photograph of a rose might have thousands of shades of red. But if one pixel is a shade of red and the pixel next to it is a slightly different shade of red, your eye cannot tell the difference. An image file can take advantage of this fact and doesn't have to store two different colors for those pixels. It could use the same color for both pixels, making the file smaller, with little visual difference. This is a lossy optimization, because we have technically *lost* graphical data, but not in a visually significant way.

This approach works especially well with things like storing photographs or movies and video, since these contain so many subtle changes in colors and shades. You can throw a lot of this graphical data out and no one will notice. At a high level, this is how still image formats like [JPEG](#) and video formats like [MPEG-4](#) use lossy optimizations to reduce their size.

Since Animated GIFs of video clips should have similar properties, can we use the same lossy techniques to make a GIF? Turns out, yes we can.

A Lossy GIF Encoder

[GIFs](#) compress their graphical data using the [LZW algorithm](#). This works by keeping a dictionary of sequences of pixels that have been seen. If a sequence repeats anywhere in the image, you don't need to store those pixel values again. You can define that sequence of pixels and simply include a reference everywhere that se-

quence is used in the image.

For example, consider an image containing a sequence of five pixels **Red-Red-Red-Blue-Red**. Later in the image, that same pattern repeats. The GIF image can simply store the **Red-Red-Red-Blue-Red** pattern once in the dictionary, and then insert a reference that says *use dictionary entry X* here every time that pixel sequence appears. This process is called **encoding** and normal GIF encoders will only match exact sequences of pixels. See [this article](#) for a super technical discussion of this concept.

By reducing the redundancy that is in the file, we can reduce all of the sizes and make them flow. From a byte perspective, we are able to make the palette entries fit into a smaller area of the header format.

The more often a sequence appears, the more often you can use a reference instead of the raw pixel data, which means the GIF file gets smaller. We could make a GIF image even smaller if we are a little more liberal with how we find matching sequences. We could use an encoder that instead of using sequences of the *exact* same pixels colors will use sequences of *nearly the exact* same pixels colors.

For example, we could find a pixel sequence **Red-Red-Red-Blue-Red** in an image, and later in that image we could have a sequence of **Red-Red-Red-Blue-Slightly-DarkerRed**. We could decide that is close enough to consider them the same sequence and match them to a single dictionary entry. This would alter the image slightly; when it gets displayed, both areas of the image would be drawn as **Red-Red-Red-Blue-Red**. However, the image is smaller than if we only matched exact pixel sequences. Sounds awesome, but is there existing software that has a lossy

GIF encoder like this? Absolutely.

Lossy Optimizations with Gifsicle

Luckily there is a modified version of gifsicle that you can download that can use a lossy encoder to find these near matches, creating a smaller file than if we used a lossless encoder that only uses perfect matches. Since an animated GIF is just a series of dozens and dozens of still image frames, and this lossy encoding can make each image even smaller, these savings add up quickly.

[Here](#) is a video clip from Star Wars, optimized with a lossless encoder:



And with a [lossy encoder](#):



This lossy animated GIF is only 1.1 MB, which is a savings of nearly 65%.

Imagine how much faster your site could be if you could avoid download an extra 2 MB of content! A quick scan of animated GIF heavy sites like BuzzFeed.com shows a **median savings of around 40-50%**. Considering these GIFs are usually between 2 MB to 8 MB, that leads to truly awesome savings in size, reducing bandwidth needs and speeding up page load times.

Using this modified version of Gifsicle is easy. There is now a `lossy` parameter you can use which controls how *loose* or *strict* the matching is on pixel sequences. You can see how we use it below:

```
gifsicle -O3 --lossy=80 -o output.gif input.gif
```

In practice 80 is a good trade-off between visual quality and size savings. Feel free to adjust it and compare to see what level of lossy optimization makes sense for you.

It's pretty amazing that you can use a lossy GIF encoder and see significant savings of 40-50% with little loss of visual quality. If your site has animated GIFs you should be using this modified version of gifsicle to optimize your site.

Optimizing Animated GIFs with HTML5 Video

While reducing the size of video clip animated GIFs through lossy optimization is certainly impressive, it's really just making the best of a bad situation. The problem is not finding ways to optimize a 3 or 4 or 5 MB animated GIF of a video clip. The problem *is* the 3 or 4 or 5 MB animated GIF.

As we discussed previously, GIFs were never intended be used even as an animation format, let alone to store video content. Luckily there is another optimization you can do to make video GIFs 95% smaller: **convert them to HTML5 video.**

[HTML5 video](#) is a catch-all term for the ability for browsers to play video content via the `<video>` tag without needing to use plugins like Flash.

When HTML5 video was first standardized in 2009, there was [a lot of confusion and conflict](#) about video formats and how they would be encoded. There were some concerns about patents covering various ways to compress videos as well as what, if any, licensing costs might appear. Most of this was just a proxy argument for deeper philosophical and commercial disputes between the browser makers, kind of like when two people are having an argument about cleaning the kitchen when they are truly arguing about something else. As a result, for the first several years things were chaotic with different browsers supporting different types of video, forcing the content creators to do [a lot of work making multiple versions of the videos and doing weird tricks with HTML](#) to work around bugs and make sure the right browsers got the proper video formats.

Thankfully all of that is in the past. For a variety of reasons, H.264 encoded video stored in an MP4 container file (which I'll just simply call an MP4 video from now on) emerged as the winner. The browsers [stopped arguing](#) and all adopted support for MP4 video. Legal and [licensing concerns were resolved](#). As a result, today [over 90% of desktop browsers support MP4 video](#) as shown below. For modern mobile devices, support is close to 100%.

Support for MP4/H.264 video format

For common recent, current, and future browsers based on information available at [caniuse.com](#) as of July 2016

IE	Edge	Firefox	Chrome	Safari	iOS Safari	Android	Opera
			50			4.4	
8		46	51		9.2	4.4.4	
11	13	47	52	9.1	9.3	51	38
	14	48	53	10			39
		49	54	TP			40
		50	55				

Legend:
■ Supported (recent)
■ Supported (current)
■ Will be supported
■ Not supported

MP4 Video Instead of GIFs

Since MP4 videos are actual digital video files, they can take advantage of all sorts of advancements and optimizations such as [chroma subsampling](#), [discrete cosine transforms](#), and [motion compensation](#). This allows MP4 videos to they look much better and be substantially smaller in contrast to hacky video clip animated GIFs.

How much smaller? Let's look at that [Star Wars animated GIF](#):



This GIF is 2.9 MB in size. That is a huge size for what amounts to only a second or two of video.

Now let's convert that GIF to MP4 video. [View MP4 in your browser.](#)

This MP4 file is only 149 KB. That is 95% smaller than the animated GIF! Our research has found that **animated GIFs are usually 5 to 10 times larger** than a properly encoded MP4 video. This difference means that GIFs are not only wasting significant amounts of bandwidth, they are loading more slowly and creating bad user experiences.

In fact, converting animated GIFs to MP4 video is such an awesome optimization that it is exactly what sites like Twitter and Facebook and imgur do when you upload an animated GIF. They silently convert it to an MP4 video and display that instead. You can read more about how major publishers convert animated GIFs to MP4 video [here](#).

Converting Animated GIFs to MP4 with FFmpeg

MP4 videos have near-universal support. And they are smaller and better than animated GIFs from a performance perspective. But how easy is it to create and share MP4 videos online? Pretty easy.

We can use [ffmpeg](#), a free, open-source command line tool to convert an animated GIF into an MP4 video. Make sure you [download a pre-built package](#) unless you are nerdy enough to build from source. In the example below we use FFmpeg to convert the file `animated.gif` into an MP4 video named `video.mp4`.

```
ffmpeg -i animated.gif -movflags faststart -pix_fmt yuv420p -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" video.mp4
```

That looks intimidating, but don't worry; those extra options ensure we have a nice, fast, compatible video file that will play anywhere. Here is what they do:

- **movflags** This option optimizes the structure of the MP4 file so the browser can load it as quickly as possible.
- **pix_fmt** MP4 videos store pixels in different formats. We include this option to specify a specific format which has maximum compatibility across all browsers. This is actually the [chroma subsampling](#) we mentioned earlier.
- **vf** MP4 videos using H.264 need to have dimensions that are divisible by 2. This option ensures that's the case. Don't worry - we can still display this video at any dimensions we want.

Now that we have a video file instead of an image, our HTML markup needs to change. Our original markup to include an animated GIF looked like this:

```

```

To use our new video, we need to replace that with this:

```
<video width=400 height=300 autoplay loop>  
<source src="video.mp4" type="video/mp4">;  
  
</video>
```

This will automatically start the video, and loop it, without displaying any video controls. This gives the same experience as the animated GIF but it's faster and smaller. Notice that we still have an `` pointing at the original animated GIF inside the `<video>` tag. This way in the unlikely event the a visitor's browser doesn't support MP4 videos, the animated GIF will still be displayed and the user still experiences the content.

Takeaways

GIF continues to be in widespread use today, despite the technical limitations of the format. Because of this, it is important to understand how you can optimize GIFs as much as possible to create fast user experiences.

In this guide, we have seen that there are different lossless and lossy options that can be used to optimize GIFs, depending on your needs and use cases. These are our recommendations for optimizing GIFs:

- Do not use static GIFs at all on a website. PNG images are smaller, faster, and have universal browser support. Convert all static GIF images into PNG images. You can use [content detection to your advantage](#) and avoid renaming the files if

you want. **The expected savings are around 20%.**

- Losslessly optimize any animated GIFs used for UI animations like progress spinners. Use Gifsicle because it is the best free tool for this. **The expected savings are between 20-60%.**
- If at all possible, do not use animated GIFs for video clips on your site. While easy to share, they are massively inefficient and create bad user experiences. Convert the GIFs to MP4 videos and use the appropriate `<video>` markup to provide an autoplaying and looping experience. **The expected savings should be around 90-95%.**
- If you must use animated GIFs for video content, and if converting them to MP4 video will not work or is impractical because of the need to update the HTML, you should optimize your GIFs with the lossy version of Gifsicle. This is a half-step solution that gives you some of the benefits of true video formats like MP4 while retaining the convenience of animated GIFs. **The expected savings are between 40-60%.**

We have seen that animated GIFs were historically the easiest, most compatible way to share short video clips on the web. However GIFs are a terrible technology for storing video. HTML5 video, while initially a mess of politics and infighting, has matured and support has coalesced around MP4 video. You can convert GIFs to MP4 video, saving over 95%, and still have all of your users see the video content.

*If you are interested in improving your website's performance with optimizations like MP4 video, then you'll love Rigor Optimization. With Rigor Optimization you can test your entire website and identify **over 270 performance defects** that are slowing it down. Our continuous monitoring allows you to ensure that nothing changes that hurts your website performance. To learn more about a free trial Rigor [contact us](#) today.*

About Rigor

Founded in Atlanta in 2010, Rigor started as a team of people on a mission to help businesses make their websites faster and more reliable. Today Rigor's digital performance management platform includes both active performance monitoring and first-in-class integrated optimization. Rigor works to provide an effortless experience to website owners, digital managers, and developers.

About the Authors



Billy Hoffman is the Director of Product at Rigor where he focuses on delivering best-in-class performance software.
[@zoompf](#)



Melanie Crissey is a web performance specialist and front-end developer with a focus on user experience.
[@MelanieCrissey](#)

GET A FREE RIGOR TRIAL TODAY!

CLICK HERE