

PNG vs JPEG Compression

Brady Jacobson and Samuel Teare

Image compression is data compression that allows us to transfer and store images at a lower cost. Many methods of image compression exist, which allow us to compress images with different results. Lossless compression is where an image is reduced, but the method is reversible, resulting in an accurate image. Examples of Lossless image compression includes PNG, GIF, and TIFF. Lossy compression is where an image is reduced by eliminating some data and using inexact approximations. The output is an image close to the original but not exactly the same. In many cases the difference is almost unnoticeable. The primary example of Lossy image compression is JPEG. In this paper, we will explore PNG compression first and then look at JPEG compression.

PNG Compression

One form of lossless compression for images is PNG compression. PNG, or Portable Network Graphic, was designed to be an open source replacement of the GIF (Graphics Interchange Format). Two main reasons for wanting a replacement for GIF is that it was patented and only supported up to 256 colors. PNG is ideal for images that you don't want to lose any part of the original image while compressing, such as icons and logos. Images such as photographs are more suited by JPEG compression. PNG

compression can be broken down to 3 steps: Filtering, LZ77 Compression, Huffman Coding. The last two steps are known as DEFLATE Compression, or just DEFLATE.

Filtering of an image in PNG Compression is to make the compression easier and more efficient. The five options for filtering are explain below:

Filter	Method
None	No filtering. This uses the raw bytes of the original image
Sub	Calculates the difference/distance between this byte and the byte to its left $\text{Sub}(x) = \text{Original}(x) - \text{Original}(x - \text{bpp})$
Up	Calculates the difference/distance between this byte and the byte directly above it. $\text{Up}(x) = \text{Original}(x) - \text{Above}(x)$
Average	Calculates the difference/distance between this byte and the average of the byte to the left and the byte above. $\text{Average}(x) = \text{Original}(x) - ((\text{Original}(x - \text{bpp}) - \text{Above}(x))/2)$
Paeth	Uses the bytes to the left, above, and above left to calculate the Paeth predictor. It then calculates the difference/distance between this byte and the Paeth predictor. $\text{Paeth}(x) = \text{Original}(x) - \text{PaethPredictor}(x)$

It is important to note that filtering is a byte comparison and not a pixel comparison. If the image uses 8 bit coloring, then it would be a comparison of pixels as each pixel is one byte. For higher colors, the comparison would be based off of the blue (one byte) of this pixel with the blue (one byte) of the other pixel(s). This comparison is helped by the bpp, or bytes per pixel. The value of bpp is always a whole number and is rounded up to 1 if it is less than 1. The calculation of the PaethPredictor is explained below:

```
PaethPredictor (x) {  
    estimate = Original(x - bpp) + Above(x) - Above(x - bpp);  
    distanceLeft = Absolute(estimate - Original(x - bpp));  
    distanceAbove = Absolute(estimate - Above(x));  
    distanceAboveLeft = Absolute(estimate - Above(x - bpp));  
    if distanceLeft is less than distanceAbove and distanceAboveLeft  
        return distanceLeft;  
    else if distanceAbove is less than distanceAboveLeft  
        return distanceAbove;  
    else  
        return distanceAboveLeft;  
}
```

After the filtering is completed, PNG compression uses LZ77 Compression. LZ77, or Lempel-Ziv 1977, was created by Abraham Lempel and Jacob Ziv in 1977. LZ77 uses a *sliding window* technique that keeps track of a certain number of previous bytes. The larger the window the easier and more efficient the compression. Using the window, LZ77 looks for repeated sequences in the pixels. The format of the

compression is the set of pixels leading up to a repetition, then designate the distance back that the repetition starts and the length of the repetition. For example:

(Using W to represent a white pixel, R to represent a red pixel, D for distance, and L for length)



This line of pixels could be compressed to W [D=1, L=4] R [D=1, L=4] [D=10, L = 5].

Since the last set of 5 white pixels has already been expressed earlier in the line, the compression merely denotes the distance back to the beginning of the first set of 5 white pixels and then states that the following 5 pixels should be copied. Another example:

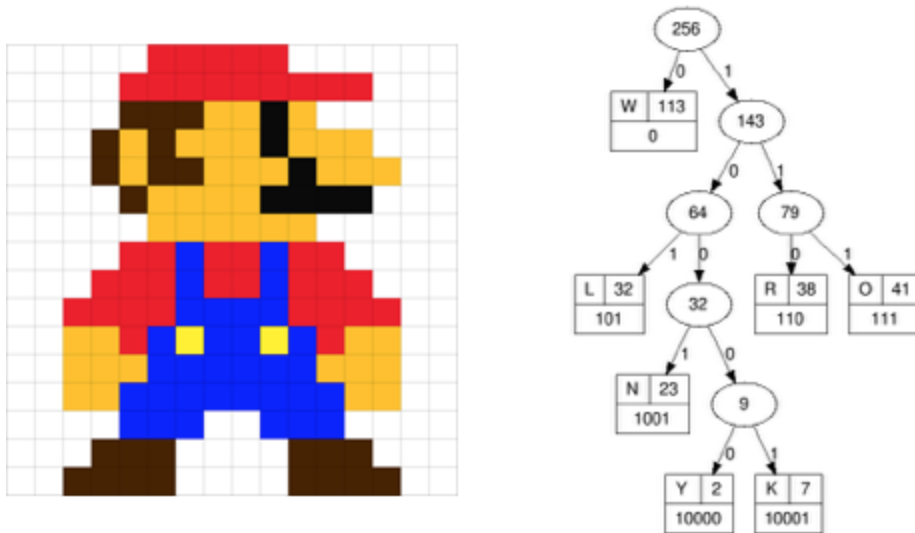


This line would be compressed to WRW [D=3, L = 9].

After the LZ77 compression is complete, PNG compression then uses Huffman coding to finish the job. Huffman coding is used to look at the frequency of literals, characters, numbers, or bytes, in a file and then compress the file by using a few bits to represent those literals. A literal usually consists of a whole byte to several bytes, using only one bit to a few bits to stand in for the literals greatly reduces the size of the file. The literals that are most frequent in the file are represented with the fewest number of bits, while literals that are less frequent or only appear a few times in the file are

represented with more bits. The literals and their frequency are used to construct a Huffman tree.

A Huffman tree is constructed by ordering the literals by their frequency in ascending order. A recursive method is then used to combine the two smallest items and then place them as one item back in the list. Once this process is completed a Huffman tree allows the proxy bits to be calculated. A Huffman tree is read by how many times you move left or right from the nodes, starting from the head node, to get to the desired literal. The head node is the node usually represented at the top center of the tree. Every left move is a 0 and every right move is a 1. An example of a 16x16 8-bit color image and a Huffman tree created from the color of the pixels can be seen below.

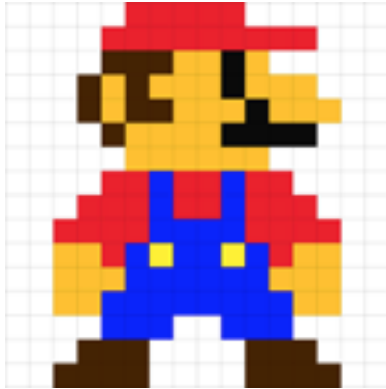


The generator used for the Huffman tree flipped all of the nodes from “64” down to fit the tree in a smaller dimension image. Normally, all the nodes under “64” would be shown on the opposite side as the smaller weights are placed on the left and larger weights on the right.

The Huffman code used to represent the white pixels would be just one bit, '0', as white is the most common color in the image. On the other hand the Huffman code used to represent the yellow would be five bits, '10000'.

PNG compression has two options for using the Huffman tree. It can read the file, calculate the frequencies of the literals within the image and then generate a Huffman tree specific to the image or it can use a standard Huffman tree. If a Huffman tree is generated specifically for the image, its information must be included in the compressed file. If a standard Huffman tree is used, then the compressed file must only reference which standard Huffman tree is being used. In the case of PNG, the literals are the output of the LZ77 compression, or the sequence of bytes with distances and lengths. In the example above, if WRW [D=3, L = 9] is the most frequent literal in the image after LZ77 Compression, what would normally be represented as 7+ bytes, might be represented with maybe 7 bits. If this pattern is repeated 100 times within an image, what would take up 700 bytes normally would be compressed to only 88 bytes.

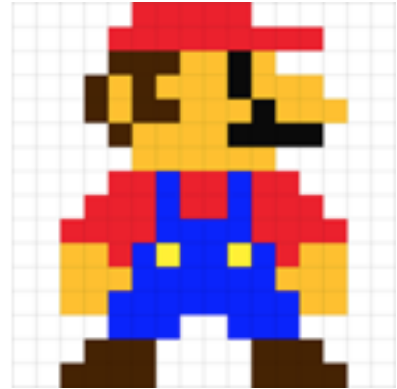
**Comparison of Image File Size:
8-Bit Mario (800x800)**



**JPEG: 167KB
(Lossy)**



**TIFF: 125KB
(Lossless)**



**PNG: 35KB
(Lossless)**

Sources:

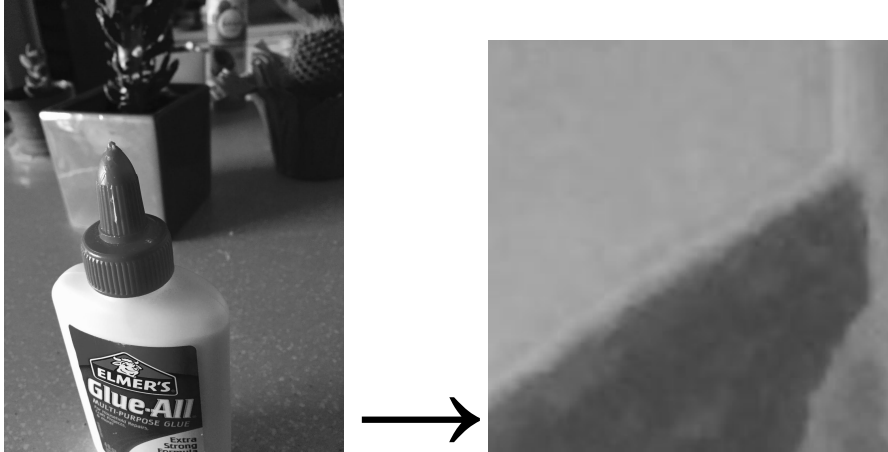
www.libpng.org, "PNG The Definitive Guide", Chapter 9.

JPEG Compression

One form of Lossy Compression for images is JPEG compression. This compression method allows for the user to alter the degree of compression by changing the quantization table, which will adjust the storage space and image quality. This quantization table is the reason that JPEG is lossy compression, as certain information will be irreversibly lost. JPEG compression also uses Discrete Cosine Transformations, which essentially takes a number of patterns created from different cosine lines and adds them together to form a specific image. JPEG takes many different steps, which include changing the color format to YCbCr, separating the image into groups, performing DCT, performing quantization, and finally organizing with huffman coding.

First the user should convert the image to the color format YCbCr. This makes compression easier due to what each symbol represents. Y is the brightness while Cb and Cr represent the chroma components of blue and red respectively. The human eye relies on brightness for the perception of an image, so we can get better compression by working with the brightness component Y. The Cb and Cr components are also used to assist with compression. If you wanted the greatest possible quality, however, you can use the RGB color format. However, this does not allow for a smaller file size, which is one of the main reasons to use JPEGs.

The next step is dividing the image into groups. Each group is made up of 8 by 8 pixels. The rest of the compression method is performed on the Y, Cb, and Cr components separately.



For each component we take the component's value of each pixel and place it in a table respective to where it is. These values range between 0 and 255, but in order to use discrete cosine transforms we need to alter it to closely resemble a cosine line. This involves subtracting every value by 128, making the range between -128 and 127. With these values, we apply the Two Dimension Discrete Cosine Transform 2, which will give us the Coefficients.

Input Table Example:

**Two Dimensional
Discrete Cosine Transform 2
equation (2D DCT-II):**

-22	-24	-27	-22	-22	-22	-18	-2
-22	-22	-24	-24	-22	-22	-22	-13
-29	-27	-27	-27	-27	-22	-18	-13
-29	-33	-27	-27	-22	-9	36	-3
-33	-31	-27	-27	32	41	45	5
-33	-33	14	41	50	48	50	-2
-22	32	43	48	48	52	43	18
34	45	45	50	39	52	7	16

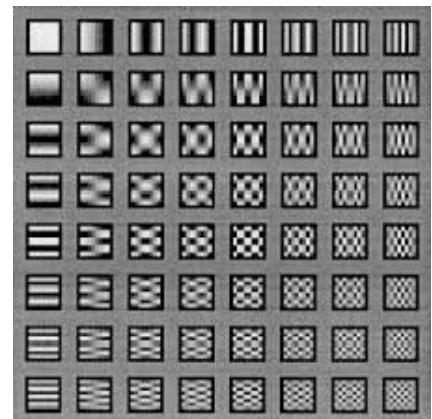
$$G_{u,v} = \frac{1}{4} \alpha(u) \alpha(v) \sum_{x=0}^7 \sum_{y=0}^7 g_{x,y} \cos \left[\frac{(2x+1)u\pi}{16} \right] \cos \left[\frac{(2y+1)v\pi}{16} \right]$$

We create a table similar to the previous and place each coefficients in it. This table is special, as each spot does not correlate with the original image, but rather the pattern table that shows the possible patterns different cosine lines can create. Each value in the coefficient table represents the influence of each pattern. The patterns in the top left are very simple while the patterns in the bottom right are highly complex. This often results in more influence being placed in the top left rather than the bottom right. The idea is that by adding enough of these patterns together, and taking into account influence, we can recreate an image.

Coefficient Table Example:

21	14	4	A	B	C	D	E
11	5	7.5	F	G	H	I	J
4	8.5	3	K	L	M	N	O
P	Q	R	S	T	U	V	W
X	Y	Z	AA	BB	CC	DD	EE
FF	GG	HH	II	JJ	4	1	1.8
KK	LL	MM	NN	OO	1.2	3	2.2
PP	QQ	RR	SS	TT	1	1.5	1

DCT Table:



Using the current coefficient table would recreate the exact image we started, but because we're attempting to compress the image we begin the next step, which is quantization. We take a quantization table, which can differ between programs and attempts. Changing the quantization table will cause a change in the balance between image quality and reduced file size. We take our coefficient table and divide each value by its respective quantization table and round to the nearest whole number.

Quantization Table Example:

2	1.9	2	a	b	c	d	e
1.5	2.3	4	f	g	h	i	j
2	2	3	k	l	m	n	o
p	q	r	s	t	u	v	w
x	y	z	aa	bb	cc	dd	ee
ff	gg	hh	ii	jj	26	26	25
kk	ll	mm	nn	oo	23	30	27
pp	qq	rr	ss	tt	28	30	30

Values in the top left of the quantization table are often smaller than values on the bottom right. This allows us to keep the necessary coefficients while eliminating the less influential ones. To demonstrate If we take 21 from the Coefficient table and divide it by 1 from the quantization table, we round to 11. This value still has influence over the image. On the other hand, if we take 1 from the coefficient(the bottom right) and divide it by 30 from the coefficient table we get .033, which rounds to 0. This means that the pattern no longer has influence on the image. All of the least influential values are eliminated while leaving the ones with greater influence. The resulting image should be close to the same as the old one, with some differences that aren't always apparent.

Coefficient table:

21	14	4	A	B	C	D	E
11	5	7.5	F	G	H	I	J
4	8.5	3	K	L	M	N	O
P	Q	R	S	T	U	V	W
X	Y	Z	AA	<u>BB</u>	CC	DD	EE
FF	GG	HH	II	JJ	4	1	1.8
KK	LL	MM	NN	OO	1.2	3	2.2
PP	QQ	RR	SS	TT	1	1.5	1



Quantization table:

2	1.9	2	a	b	c	d	e
1.5	2.3	4	f	g	h	i	j
2	2	3	k	l	m	n	o
p	q	r	s	t	u	v	w
x	y	z	aa	<u>bb</u>	cc	dd	ee
ff	gg	hh	ii	jj	26	26	25
kk	ll	mm	nn	oo	23	30	27
pp	qq	rr	ss	tt	28	30	30

Quantized Table Example:

11	7	2	0	0	0	0	0
7	2	2	0	0	0	0	0
2	4	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

We calculate the quantized table by dividing the coefficient table by the quantization table and rounding to the nearest whole number.

We have gotten rid of unnecessary information, but the recreated image will be largely the same. This is what makes JPEG compression Lossy instead of

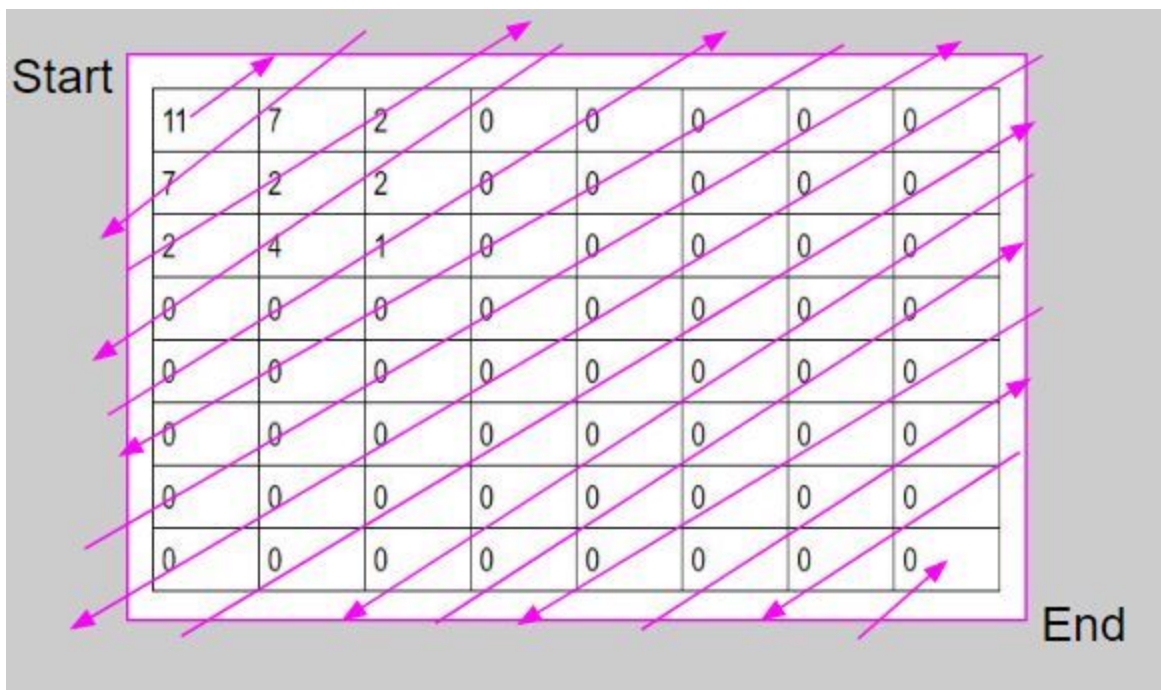
lossless. Now we need to take the quantized coefficients and encode them. We can use

Huffman Coding, a lossless method, to reduce the

list. This will allow us to keep the same values we have now. If we take the quantized

table and go in a zigzag pattern to organize the values for Huffman Coding then we will

end up with several 0's in a row. This will allow us to compress more efficiently.



Conclusion:

Both Lossy and Lossless compression methods have their place. They both effectively compress an image, and the result still looks like the original. PNG keeps the original image mostly in tact. While JPEG does not result in the exact same image, the result is very close to the same and takes up much less space. This key difference highlights the main appeal of both. PNG is used to keep as much of the original as possible and works great for sprites and logos. JPEG is used to save space depending on the preference of the user, and is great for more detailed images because it should result in a smaller file size. Both image compression methods have their places where they shine and struggle.