DUG: Image Compression Taken Way Too Far

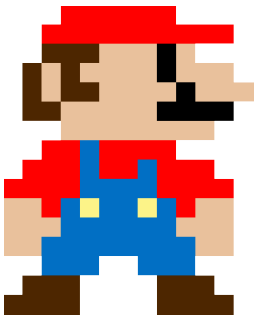
Imran Khaliq
Computer Science Researcher
Tamalpais High School
ikhaliq15@yahoo.com

Abstract

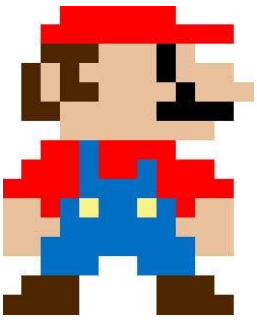
Images are being used more in more using technology and the more we use images, the more there seems to be need for having smaller image sizes. I conducted an project to see if I was capable of creating my own file type, such as GIF, JPEG, or PNG. The file type's goal is to have a smaller file size than the other 3 major ones. I created an algorithm that used Base64, then get rid of redundant pixels, and then compress the whole thing with bzip2. I called the new file type DUG. DUG was on average 16.37% the size of its closest competitor, the PNG.

1. Introduction

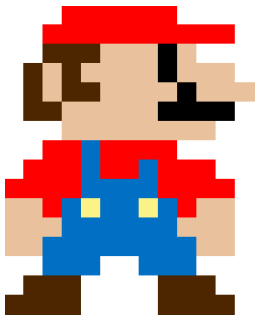
As society imbraces the use of computers more and more, the more we need a way to digitally store images. There are many filetypes for images out there, including but not limited to ".jpeg", ".png", and ".gif" [1]. Each of these methods have their own benefits. JPEG is great for images taken on a camera [2] because of there is often few sharp changes in color. PNG is often used when you don't want to lose any quality in the image [3], especially in applications like games. GIF is great because it is the only one of the 3 major file types that supports animation[3]. However, when it comes to the compression of an image, I noticed that PNG was substantially lower than JPEG or GIF. Below is the same image but in the different file types. Below each image is its file type and size.



GIF - 8 KB



JPEG - 11 KB



PNG - 460 Bytes

As you can see the PNG beats the other two by so much more.
The size of the PNG is 5% of the size of the GIF and 4% the size of the JPEG. And do note, that you may not be able to see it, but the JPEG version also has some bluriness

around the edges, near changes of color. The PNG and GIF have a similar quality to each other. However the fact that PNG is less then 1 KB is a huge benefit of using it.

2. Background

After conducting some more examples with more images, I found that PNG consitently delivered a smaller or similar file size to JPEG and GIF. However, I believed that I could create my own compression algorithm that would create a smaller file size. A *compression algorithm* is a method of representing information but using less bytes then the original information. For example, you could have the sentence "I ate a apple". Lets say that the compression algorithm uses the character '%' to represent apple. The new sentence becomes "I ate a %". The sentece went from 13 characters to 9 characters. When you compress an image you do something similar. I decided to name my compression method DUG, which stands for Don't Use GIF.

3. The DUG Algorithm

The DUG algorithm takes in a 2D Array of pixels. Each pixel is represented by its color in hexadecimal. DUG would like the image to be as scaled down as possible before being input into the algoritihm, as it will be easier to reverse the scale back to proper size. After receiving the inputed image, the algorithm requires that each pixel's color be converted into Base64.

3.1 Convert Pixels to Base64

It doesn't matter what library you use to convert Hexadecimal to Base64 as long as it follows the standard Base64 "alphabet". [4] is a good website for converting between Decimal, Hexadecimal, and Base64. Look in *Appendix A* for a chart of common colors in Decimal, Hexadecimal, and Base64. Hexadecimal takes 6 characters to represent one color while Base64 takes only 4. This will save us 2 bytes per pixel. In a 100 px by 100 px, this will save us 2 x (100 * 100) bytes, which equals 20,000 bytes or 19 KB. This alone saves us 33% of the original.

3.2 Counting the Number of Simultaneous Pixels

After all the pixel colors have been converted to Base64, the Algorithm goes through all the pixels and if there are any of the same color pixels in a row

For example lets say that this was the input:

```
1111111111111111
1111100000111111
1111111111111111
```

The same thing could be represented as:

```
18:1
6:1 6:0 6:1
18:1
```

The first line means 18 1's. The second line means 6 1's, then 6 0's, then 6 1's. The last line is the same as the first. As you can see, we repsented the whole rectangle in a very few amount of characters. The goal is to the same thing with colors. The format for this would be `# of pixels:color of pixels`. The amount of bytes this would save will depend on the image. If the image has a lot of detail or color changes then it will actually make the file size bigger but if the image uses the same colors then it will save a gigantic amount of space.

3.3 Compressing the Whole image

Our image will look something like this write now:

```
4://// 3:/w== 2:AAAA
9:/w==
5:AP8A 4://8A
```

which looks like this:

The next step is to compress the whole file. The DUG algorithm uses the bzip2 [5] compression method. The method has a really good comrpession ratio. The compressed Version of the above example is:

```
BZh91AY&SY`?   □@@?r
```

```
@?
1L?=L?C?C?R??4i(??]Sg?&t]??BA?#
```

The PNG version above is 236 bytes. My DUG version below is 73 bytes.

Do note that the above image is a screenshot in the form of a PNG. This is because my word processor does not know how to display a raw DUG file. As you can see, there is no difference in quality and the DUG file is 25% of the size of the PNG.

3.4 The Algorithim As A Whole

You will find the psuedo-code for the whole algorithim in *Appendix B.* I wrote my actual code in Java but you can apply the psuedo-code to many languages. The algorithm is simply put here into 3 steps.

1. Receive input as 2D array with hexadecimal colors
2. Convert colors from hexadecimal to Base64.
3. Remove redundacy by counting the number of times a pixel appears in a row.
4. Compress the image using bzip2.

4. Results

Because I did not write any software to easily create a DUG image, I created them all by hand. Therefore I was not able to create to many examples. If you look in *Appendix C,* you will find the images listed here. Here is chart for images for their sizes.

Image Label	PNG Size	DUG Size	Compression Ratio
Image #1	460 Bytes	188 Bytes	40%
Image #2	9 KB	141 Bytes	1.52%
Image #3	1.3 KB	102 Bytes	7.6%

As you can see, all 3 times, DUG beat PNG by a considerable amount. Especially Image #2, where the DUG file was 1.5% the size of the PNG. The average compression ratio was 16.37%.

5. Conclusions

Even though these examples beat PNG by so much, it does not say how other images would face. We would need a much bigger sample size with 1000's of images to tell if DUG really beat PNG. However from these small results, I would like to give my conclusions.

5.1 Which Images Compress Best

The images that compressed the best are ones that have a limited color pallette. Also, images that had the same colors for long streches. Images that have sharp color changes were not good because if for time there is only 1 of a colored pixel, we lose space rather than just having the base64. For example, if "1:////" happens to many times then our image will not be compressed. We want the no ones in our file, as that will mean it actually compressed a bit.

5.2 Why Has No One Done This Or Something Similar?

Honesetly because it requires a lot of energy to compress and image like this. An JPEG or PNG take very little time to display because there are few things it needs to do to decode the image. However for DUG, the decoder needs to decompress the bzip2, then expand all the pixel data, and convert Base64 to Hexadecimal. For a larger image, it might take up to 3 seconds to display which is a very noticeable and not effective delay. That is why you would never use DUG for game development as you need to access and display images fast.

5.3 Future Development of DUG

Some future things I would like to do for DUG is add even more ways to shorten the data. I could have special codes for common colors. I could have 999 special colors and it would compress the image greatly. Also, I want to add a way for images that have patterns. Finally, I want to create a easy to use converter for PNG, JPEG, and GIF to DUG. This would do everything it needs to and also have a way to scale images to as small as they can get without losing data. Then at the top of the file, I would add "s:3" meaning to scale the image 3 times larger.

5.4 Final Comments

I did some groundwork and now someone hopefully will be able to DUG to a higher level. It will hopefully become more efficient and faster to decode. The goal is for it to one day become an ISO Standard Image Type (I can dream, can't I?).

References

1. "Table: Common Image File Formats." Table: Common Image File Formats. Cornell University Library/Research Department, 2002. Web. 18 July 2016.

2. "JPEG - JPEG." JPEG - JPEG. JPEG, n.d. Web. 18 July 2016

3. Bourque, Brad. "The Differences between a GIF, a JPG, and a PNG Explained." Digital Trends. Digital Trends, 27 Apr. 2014. Web. 18 July 2016.

4. Binary-Hexadecimal-Decimal-Base64 Converter. DarkByte, n.d. Web. 18 July 2016. <https://conv.darkbyte.ru/>.

5. http://www.bzip.org/

Appendix A

Color Name	Decimal	Hexadecimal	Base64
White	255 255 255	FF FF FF	////
Black	0 0 0	00 00 00	AAAA
Red	255 0 0	FF 00 00	/w==
Green	0 255 0	00 FF 00	AP8A
Blue	0 0 255	00 00 FF	AAD/
Orange	255 165 0	FF A5 00	/6UA
Yellow	255 255 0	FF FF 00	//8A

Appendix B

```
Input:      2D array p, where p is full of pixels in hexadecimal
            format
Output:     file f, where f is a file containing the file compressed
            form
Method:     turn all colors into Base64 (3.1). then count the number
            of simultaneous pixels (3.2). finally compress the whole
            image using bzip2 (3.3).

            // Get the 2D array any way you like
            input(p);

            // Convert all hexadecimal numbers to Base64(3.1)
            for r = 0 to size(p)
                  for c = 0 to size(p[0])
                        // You will need to find a library for
                        // the convertHexToB64() method.
                        p[r][c] = convertHexToB64(p[r][c]);
                  end
            end

            // Create a string for the results from the next
            // for loop.
            data = "";

            // Next we find all the simultaneous pixels (3.2)
            for r = 0 to size(p)
                  for c = 0 to size(p[r])
                        // 'Count' will keep track of number of //
                        same pixels in a row.
                        count = 1;
                        // 'start' will keep track of the original
                        // color
                        start = p[r][c]
                        // Loop through pixels until they are // not
                        the same as the first one.
                        while p[r][c + count] == p[r][c] do
                              // If count + c is bigger than row
                              // size then break.
                              if c + count > size() then
                                    break;
                              end
                              count = count + 1;
                        end
                        // Add the new data to the string 'data'
                        data = data + count + ":" + start + " ";
                        // Have the 'count' skip the pixels we just
                        // checked.
                        c = c + count;
                  end
            end
```

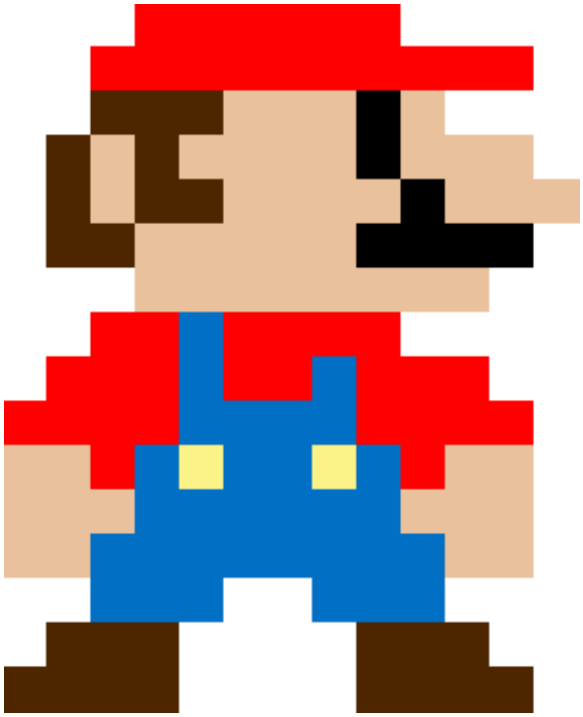
(Continued on the Next Page)

```
// The final step is to convert the 'data' using bzip2.
// We also need to write the write that data to a file.
// You'll need to find a bzip library for your language
compressedData = bzipCompress(data);
f = new File("filename.dug");
f.write(compressedData);
return (f);
```

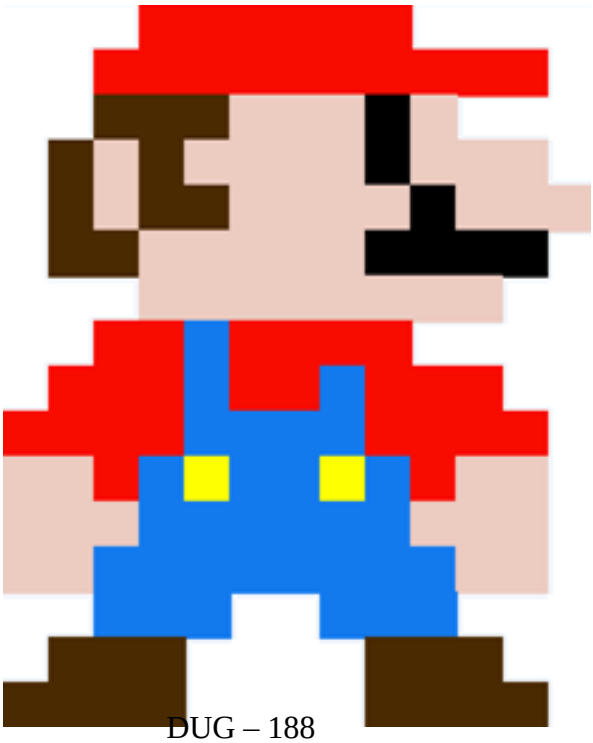
Algorithm 1. Compress image using DUG algorithm

Appendix C

Image #1 - Mario



PNG – 460 Bytes



DUG – 188 Bytes

Image #2 – Smiley Face

PNG – 9

DUG – 141

Image #3 – Pokéball

PNG – 1

DUG – 102
Bytes

PLEASE NOTE: Since word does not know how to display DUG files, I had to take screenshots from a viewer that I created. Therefore the quality may be affected as the screenshots were in PNG.