Dustin Jowsey

Project Report

Table of Contents

1. **Introduction/Purpose of This Project**

We have all ran into the issue where you try to open a file in a specific software, but the software does not support the file type you are trying to use. There are some work arounds to this by using a different file of the correct format, say for example you wanted to edit a JPEG file, but the software you are using only supports PNG or BMP. Maybe you decide to choose a different file of the correct format, but it isn't the file you wanted to use, or maybe you decide to use a less-than-reputable website to convert the file for you, which involves uploading the file you want to convert and then downloading the converted file. The converted file you download could be anything; for all you know, you could be downloading a virus or some other form of malicious software. This brings me to the purpose of this project. I am striving to provide users with a free open source Python program that will convert these types of image files for you. This would relieve you from using worrisome websites or

alternative files you didn't want to use in the first place.

2. **My Expectations of The Project**

My expectations for this project were much higher than I ever could have imagined. In the beginning I wanted my program to be able to convert common image formats such as PNG,BMP and JPEG as well as audio formats like MP3 and WAV. Furthermore, I wanted to include video conversions such as MP4 and MOV. By the midterm update I realized just how high my expectations were. I was not able to complete the JPEG decoder I was working on by the deadline and this was the first conversion I was working on. As you read on you will see just how complex even the most basic of image file formats is and why I resorted to conversions between strictly PNG and BMP for the project.

3. **The PNG File Format**

3.1 **The Structure of The PNG File Format**

The main structure of PNG consists of chunks. Each chunk in PNG begins with the length of the chunk as a 4 byte integer. Following the length is the chunk type, which again is represented as a 4 byte integer. There are a minimum of 3 required chunks which include IHDR which is the header, IDAT which contains the image data, and IEND to indicate the end of the file. These chunks must appear in the file in the order listed above. Preceding the chunk type is the data of the chunk which can vary in length depending on the chunk type. Lastly each chunk ends with a cyclic redundancy checksum, which is also 4 bytes long and is used to check and correct corruption in the chunk. Now that we have a basic idea of how PNG is storing data, we can move onto the encoding steps of a PNG image.

3.2 **Encoding/Decoding The PNG File Format**

For a basic encoder we will need to know some basic information. We will need to know what width and height in pixels the image will be. We also need to know the bit depth i.e. the number of bits

that will be used to represent a one pixel of a colour channel. Furthermore it is important to know what colour type we will be using. PNG supports several colour types each with varying allowed bit depths. These colour types include truecolour (RGB), truecolour and alpha (RGBA), grayscale, grayscale and alpha, and indexed. It is important to note that if indexed is used we will need to include the palette chunk right after the header chunk so the decoder will know where the indexed colours are stored and can retrieve them.

With this information we can begin the actual encoding steps. To begin with PNG uses a filtering algorithm which allows the image to be compressed more efficiently. To begin filtering we first need to split the image data into the correct number of colour channels based on the colour type (Ex. RGB has 3 colour channels). These filtering methods are applied to each each row of each colour channel individually. There are 5 filtering types with the first being no filter. The second filter type is called Sub

and involves subtracting the byte one ahead from the current byte and replacing the current byte with this value. We must apply modular 256 to the final value to ensure that the resulting value will fit in one byte and this applies to all the other filter methods. The third filter type is called Up and is similar to Sub, except we subtract the value directly above the current value and row. Fourthly we have Average which takes the current value minus the average of Sub and Up rounded down. Lastly is the Paeth filter which is the most complex of the filtering methods. To begin this filter uses a value p which consists of adding the upper byte and the previous byte, then subtracting the byte that is both above and before the current byte. Using p Paeth calculates three other values a, b, c where a is p - the byte before the current byte, b is p - the byte above the current byte, and c is p - the byte that is both above and before the current byte. After this we subtract the current byte from the byte before, above, or above and before the current byte based on the lowest value of a,b or c. For edge cases such as using the up

filter method in the top row of bytes, always assume bytes outside the dimensions of the matrix are just 0.

To pick which filter to use on each row the encoder must try all these filter methods on every row of every colour channel. After which we will pick the method that results in the sum of the row being the smallest. Furthermore, we will need to include a way for the decoder to identify which filter type is being used for each row, so we add the filter type as the first byte to each row.

Now we can store the data in a list consisting of each row starting with the first colour component. It is important to note that for bit depths less than 8 or if using a colour palette, the None filter is the best choice of filter to use.

The next step to encoding a png image involves using the Deflate compression scheme. This scheme consists of first performing the LZ77 compression algorithm which uses a sliding window to

compress reoccurring information within the current search window into offset and length pairs. This offset will represent the offset in the search window and the length will represent the length of bytes to read starting at the offset within the current sliding window. If the current read data is not in the sliding window, the data is not compressed and is added to the output as is. To avoid further redundancy, this algorithm does not allow for offset length pairs of length 2 or less. Otherwise we would be encoding 1 symbol using 2 which is not ideal. This will result in a list of offset length pairs as well as raw data that has been filtered.

The second part of the Deflate compression scheme consists of using a specialized form of Huffman encoding to allow for offset length pairs, as well as regular image data. This algorithm is rather complex and can be read about in reference **[1]** of this report.

The result of these steps will give you an encoded PNG image. To decode the image you can

simply follow these steps in reverse order.

3.3 **My Implementation of The Encoder/Decoder**

As you can infer PNG is by no means a simple file format. There are many options to consider when creating an encoder or decoder for PNG. Because of this my implementation focused specifically on the truecolour colour type as well as restricting the bit depth to 8. This simplified the encoder and decoder greatly.

Due to the complexity of the DEFLATE compression scheme, I also used the zlib library to help with this. Although this library helped me save some time, I was still unable to get the encoder and decoder to output a similar looking image that we started with. This was rather disappointing as I have put in a lot of work to get the encoder and decoder to where they are now. Similar to the other image file types in my project I discovered that one of the biggest issues I ran into was trying to fully grasp

how these file types are storing the data. In conclusion, I was able to get an output, but it was quite different from the original and only included full red, green, blue, white or black pixels.

4. **The BMP File Format**

4.1 **The Structure of The BMP File Format**

BMP approaches storing data in a vastly different way than PNG. The beginning of each BMP file consists of the bitmap header. This header is always 14 bytes in length and contains basic information about the file such as the bitmap identifier BM to verify the file is indeed a bitmap file, the file size, and the offset of where the start of the image data is.

Following this header is the DIB header which can vary in length depending on the type of bitmap file. This header begins with 4 bytes for the length of the header, following with the image width in pixels and the image height in pixels. Lastly is the number of bits per pixel which is the same as bit

depth in PNG. There are a few optional headers that BMP offers as well, but the last required field is the pixel array which contains all the image information. It is important to know that BMP stores the pixels in an array where all colour channels of each pixel are listed together. Furthermore each row of pixels must be a multiple of 4 bytes, so each row may have padding at the end. Lastly, BMP stores this pixel array starting at the bottom left of the image. When considering an encoder or decoder the image will be processed from the bottom up, left to right.

4.2 **Encoding/Decoding of The BMP File Format**

Encoding a BMP file is a lot simpler compared to PNG. This is mostly due to the fact that BMP in general does not compress the image. This makes the encoding and decoding of BMP files very straight forward.

To begin the encoding step we will need some basic information regarding the image. This

includes the image width, height, number of colour channels, and the bit depth. There are many options for BMP so we will focus on the 16 byte DIB header as well as only using the RGB colour scheme with 24 bits per pixel. After getting this information, we can move to the final step.

Lastly, we want to read the image data by beginning at the bottom row of pixels. For each pixel we will store the data in big endian format. Once we reach the end of the row we want to ensure the row occupies a multiple of 4 bytes of space, otherwise we will add 0s until this is the case. After doing this we can build the headers and add the pixel array after the DIB header. To decode a BMP file simply follow these steps in reverse.

4.3 **My Implementation of The Encoder/Decoder**

As we can see the BMP format is very simple. For my encoder and decoder implementation, I stuck to using the 16 byte DIB header and only considering RGB values. To incorporate all the options

for BMP would take a considerable amount of time which is outside the timeline for this project. That is all there is to a basic BMP encoder or decoder.

5. **The JPEG File Format**

For the JPEG file type I will provide a very basic idea of the format as well as the encoding and decoding steps. JPEG is very complex and could easily take up the whole scope of this project.

5.1 **The Structure of The JPEG File Format**

The JPEG file format is slightly different from BMP, but includes a lot more information for decompressing the data. Each section of information begins with a JPEG marker. Following these markers will be the length of the information for the specific marker. A basic JPEG image must begin with the start of image marker which contains no information and just the marker. After this marker is the default header which lets the decoder know the type of JPEG as well as the version of JPEG the

image was compressed with. Next is the quantization table markers. The information stored at these markers will include the quantization table itself and whether or not it is an AC or DC table. Following the quantization tables will be a marker for the start of frame. This section will include information specific to the image such as the image width and height, as well as the number of colour channels. It is important to note that JPEG uses the YCbCr colour space when dealing with coloured images. Furthermore, this marker includes the precision of the image so the decoder knows that the image was encoded using quantization tables of size 64 or 128. The next marker will be the Huffman tables. Each entry will contain information to tell the decoder whether the table is a AC or DC table as well as the number of symbols with lengths of 1 to 16 respectively. The last part of the Huffman information is the list of symbols themselves. Finally we reach the start of scan. This marker has information to determine which quantization table and Huffman table each colour channel should refer to. Now that we have all

the information required for decoding the image the encoded image data begins right after the start of scan information without a marker. JPEG images should also have an end of image marker which has no length and is simply used to indicate the end of the image.

5.2 **A Quick Idea of The Encoding/Decoding Steps of The JPEG File Format**

To encode a JPEG image it can get very involved. We first need to know some information about the image such as the width and height in pixels as well as the color type being used i.e. the number of color channels or components in the image. Lastly, we need to decide how precise we want to be with our encoding so we know to use a quantization table of size 64 or 128. After we acquire this information we can begin encoding the image. For this encoder we will assume there are 3 color channels as well as quantization tables of size 64 are being used.

The first step to encoding a JPEG will be to convert the RGB components to YCrCb. After this

step it is important that each row of each colour channel be a length divisible by 8 in our case, so we may need to add padding to the end of the rows. This is so we can effectively apply our quantization tables that are 8x8 matrices to the image data. Once we have converted the channels we want to split all the data of each component into 8x8 chunks. This will allow us to apply the quantization matrices to each chunk. Now we need to convert each chunk to a list for compression purposes. To do this we will apply the zig zag matrix of size 8x8 to correctly order the data into a 1D list. This will allow us to more efficiently encode the data using Huffman encoding. Lastly we need to encode the data using the Huffman encoding scheme.

As you can see this process is very complex and difficult to understand. See **[2]** for a great video serious discussing the decoding steps of JPEG.

5.3 **My Implementation of The Decoder**

For my implementation I strictly focused on decoding images with 3 colour channels. Furthermore I also used quantization tables of size 64. While working on this decoder I ran into many bugs and was not able to get past the first main stage of decoding which involved decoding the Huffman data. At around 800 lines of code just to reach this step it is clear to see that the JPEG decoder is out of scope for the time frame of this project. Unfortunately I spent more than half my time working on this decoder and attempting to understand the complex steps involved to decode a JPEG image. This left me with little time to complete the PNG and BMP encoders and decoders.

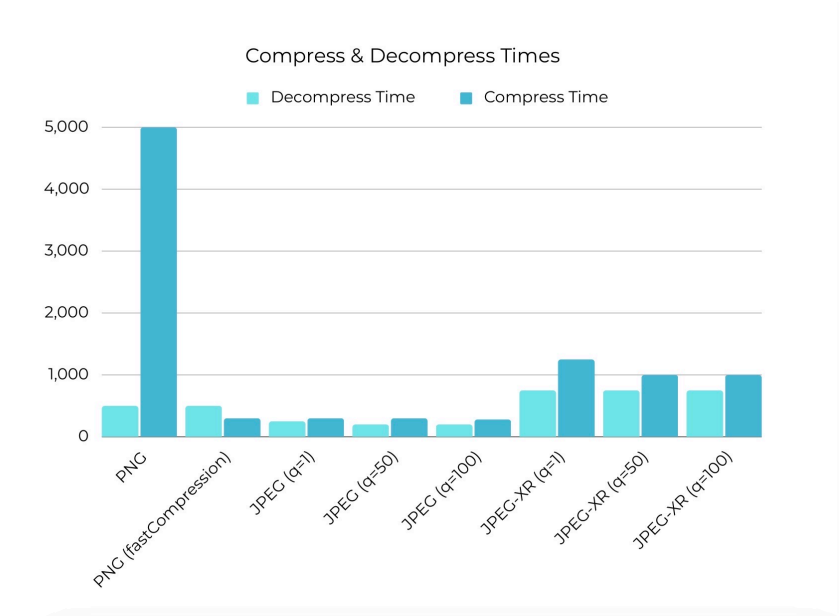
6. **Comparisons Between The PNG,BMP, and JPEG File Formats**

It is important to briefly discuss a few differences between JPEG, PNG, and BMP image types. Firstly both PNG and BMP are lossless file types, meaning the image data is fully recoverable when you store an image in these formats. JPEG on the other hand is lossless and only maintains the

important data of an image such that when seen by humans looks nearly identical to the original.

Next I would like to discuss the encoding and decoding speeds of each file type. An important thing to consider when comparing these speeds is that BMP does not do any compression generally, so it should not be compared with PNG and JPEG. One of the draw backs to PNG is just how long it takes to compress the image. The encoding speed for PNG is vastly greater compared to JPEG and can be seen in the diagram below.

[3]



As you can see in terms of decoding both PNG and JPEG are quite quick with JPEG being faster. For

BMP we are essentially just rearranging how the image data is stored. Therefore you could consider

BMP encoding and decoding to run in linear time.

Laslty, I would like to discuss the best use cases of each image. To easily represent this the

image below shows what each image type offers.

[4]

	BMP	JPG	PNG
Compressed	FALSE	TRUE	TRUE
Lossless	TRUE	FALSE	TRUE
Transparency	FALSE	FALSE	TRUE
Translucency	FALSE	FALSE	TRUE
Recommended for photographs	FALSE	TRUE	FALSE
Recommended for static graphics/icons	FALSE	FALSE	TRUE
Recommmended for animated graphics/icons	FALSE	FALSE	FALSE

so based on your needs you would want to choose one over the other. For example if you wanted to

have a transparent background, you could not use BMP as it does not support transparency and

neither does JPEG. This leaves just PNG.

7. **Project Conclusions**

In conclusion of this project, I have learned a great amount about these file types. The complexity of encoding and decoding these file types is much greater than I initially imagined going into this project. Because of this I set the scope of my project to be way too large for the time frame and as a result did not generate a working algorithm to convert even PNG to BMP. Much of my time was spent attempting to understand the steps involved to encode and decode these file types and as a result I was rushed to complete my implementation which explains the bugs I am encountering. However, It was exciting to at least get an output of some sort from the conversion between PNG and BMP. In the end it may be worth spending money on software to do these conversion for you as it is not quick and easy to implement these conversions. All in all I enjoyed working on this project and with the effort put into it I would like to continue to develop this program in my free time. In writing this report I have come up with some new ideas of where there could be bugs in my implementations, so I would like to

attempt to work these out by the due date of the website and include a short demo video of successful results.

Thank you for taking the time to read through my project and I hope you found it as interesting as I did to learn about these file types.

8. **References For Project Report**

[1] Explanation of PNGs Huffman encoding: http://www.zlib.org/feldspar.html

[2] Understanding JPEG video series: https://www.google.ca/url?

sa=t&rct=j&q=&esrc=s&source=video&cd=&ved=2ahUKEwiw8Z6Q1f6CAxV8MjQIHQPSDPEQtwJ6BAg

NEAI&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DCPT4FSkFUgs&usg=AOvVaw35

8f-uYgYcTBihuK-wKlEU&opi=89978449

[3] Source of image comparing JPEG and PNG compression times: https://

www.jacksondunstan.com/articles/2117

[4] Source of image comparing BMP,PNG, and JPEG use cases: https://superuser.com/

questions/53600/jpeg-vs-png-vs-bmp-vs-gif-vs-svg