INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN 2278 – 0211 (Online)

# Encoding and Decoding of File to Image

**Souham Biswas**
B.Tech Final Year Student, Computer Science & Engineering Department
J. K. Institute of Applied Physics & Technology, University of Allahabad, India

***Abstract:***
*The past decade has seen a rise in the field of image file processing techniques. By techniques, we mean image file transfer, compression etc. The techniques pertaining to an image file are more versatile than those to normal computer files. For example, many messaging mediums allow transfer of image files but have limited support for transfer of ordinary files. Therefore, the representation of a given file as an image will allow all the technologies specialized to image files to be applied to ordinary files. Presented in this paper are methodologies to encode any file as a bitmapped image and to recover the file from a given image. This methodology has promising applications in a wide variety of fields like cryptography, file compression etc.*

***Keywords:*** *Image, Bitmap, Encoding, Cryptography, Pixel*

## 1. Introduction

With the advent of new socializing technologies, there has also been a synonymous rise in photo sharing services [1]. The ease with which an image can be shared over the internet is much greater than that of a normal file [2]. Moreover, specialized provisions are usually made in most services to deal with image files. To extend these privileges enjoyed by image files to any computer file, it is imperative that there exist a methodology which can somehow encode a computer file into an image, i.e. represent a file as an image and recover the file from the image. That way, any file may be converted into an image.

Any given computer file is a stream of binary characters. Irrespective of the file type, any file can be represented as a stream of bytes, i.e. an array of numbers where each number ranges from 0 to 255 (1 byte in each array element). A bitmap image [3] on the other hand is basically a collection of 4 2D matrices overlapped on each other. Each pixel of a bitmap image contains 4 components namely, Alpha, Red, Green, Blue where, the alpha component dictates the transparency. The 4 2D matrices correspond to these components i.e. the 1st matrix holds the Alpha component values, the 2nd matrix holds the Red component values, 3rd matrix holds the Green component values and finally the 4th matrix holds the Blue component values. These matrices are placed one on top of each other to form the complete image. Each of these components is quantified by an 8 bit number, causing a pixel to be of 32 bits or 4 bytes. Therefore, we can say that each pixel holds 4 bytes of data in a bitmap image. The methodology posited in this paper draws on this fact. The binary data of any file can be encoded into image pixels and the original file may be recovered from the image by analysing the pixels, extracting the bytes and appropriately concatenating them to reconstruct the original file. Moreover, this technique can be improved upon to incorporate other lossless image file processing techniques.

## 2. Methodology

### *2.1. Encoding*

Described in the succeeding text is the methodology to convert an input file into a bitmap image file. This process is called *encoding* of the file into an image. The output of this process is a bitmap image file with the extension *.bmp*.

#### 2.1.1. Equations and Variables

Let the number of bytes in the input file be '$n$'. Since every bitmapped image is either a square or a rectangle, therefore there should exist positive integers '$a$' & '$b$' such that –

$$n = k(a \times b) \quad (1)$$

Where $k$ is the number of bytes encoded per pixel. In case of a bitmap image, $k = 4$.

So then, '$a$' & '$b$' would become the length and breadth of the bitmap image in number of pixels respectively. But, this is not possible for every value of '$n$'. Therefore, it is imperative that some redundant bits be appended to the end of the file so as to meet this criterion. We could try to compute the optimal values for $a$ and $b$ so as to minimize the number of redundant bytes as much as possible, but that would be unnecessarily computationally expensive. Therefore, we maintain the shape of the bitmap image outputted to be a square. By incorporating the notion of redundant bytes, Eq. 1 changes to Eq. 2.

$$n + R = k(a \times b) \quad (2)$$

Here, $R$ is the number of redundant bytes to be appended at the end of the file.

Now since the output image will be a square, $\therefore a = b$ which changes Eq. 2 to –

$$n + R = ka^2 \quad (3)$$

Therefore, $a$ is the length of the side of the square image in number of pixels. In a completely ideal scenario, $a = \sqrt{\frac{n}{k}}$. By *ideal* scenario, it is meant that $n$ is a multiple of $k$ and is also a perfect square, hence eliciting zero redundant bytes. But, in practical application, the formula to calculate $a$ is –

$$a = 1 + \left\lfloor \sqrt{1 + \frac{n}{k}} \right\rfloor \quad (4)$$

Here, $\lfloor \ \rfloor$ refers to floor function. Therefore, calculating $a$ from here, the value of $R$ can be computed by the following equation-

$$R = ka^2 - n \quad \text{[From (3)]} \ (5)$$

Therefore, knowing $n$ and $k$ and using Eq. 4 and Eq. 5, $R$ can be calculated. Once $R$ has been calculated, '$R$' number of redundant bytes can be appended to the rear end of the file. A redundant byte is assumed a value of '0'. After this, the file byte array can be read byte by byte and encoded as an image file.

2.1.2. Encoding Algorithm

Here, a function Encode (file [ ], k) will be defined with the following parameters-

Input –

- file [ ]: The byte array pertaining to the input file. (file. Length denotes the number of bytes in the file[] byte array)
- k: Number of bytes that can be encoded into one pixel. For bitmap images, k = 4.

Output (Returned Parameters) –

- bmp: Output Bitmap type object [4] which will hold the final converted bitmap file.

Bitmap objects are special variables which hold all information pertaining to a bitmap image file and have methods to read and write each component of each pixel of the bitmap image file. A Bitmap object can be directly saved as a *.bmp* file on a computer. The most prominent methods pertaining to a Bitmap object are namely SetPixel & GetPixel. They are defined as follows –

- SetPixel(Row, Col, AVal, RVal, GVal, BVal) –

This method sets the ARGB (Alpha, Red, Green, Blue) components of a pixel at a given index with the specified input parameters. The input parameters with their definitions are given as follows –

- Row: The row index of the target pixel whose ARGB components are to be set.
- Col: The column index of the target pixel whose ARGB component are to be set.
- AVal: Value of Alpha component of target pixel to be set to.
- RVal: Value of Red component of target pixel to be set to.
- GVal: Value of Green component of target pixel to be set to.
- BVal: Value of Blue component of target pixel to be set to.

- GetPixel(Row, Col) –

This method returns a Pixel type object which contains information about the magnitudes of the ARGB components of the pixel at a given index specified by the Row & Col input parameters which denote the row and column indices respectively, of the target pixel whose details are to be returned. If there is a variable 'p' which is a Pixel object, then the individual properties (magnitude of ARGB components) of 'p' can be accessed by –

- p.Alpha: Value of Alpha component of target pixel 'p'.
- p.Red: Value of Red component of target pixel 'p'.
- p.Green: Value of Green component of target pixel 'p'.
- p.Blue: Value of Blue component of target pixel 'p'.

**Algorithm-**

*Encode (file [ ], k)*

1. n ← file. Length
2. a ← 1 + sqrt(1 + n/k)
3. R ← $ka^2 - n$
4. Initialize new byte array "file2[]" with length = n + R
5. Copy contents of file[] to 1st 'n' cells of file2[] array.
6. Initialize remaining cells of file2[] from index 'n + 1' to index 'n + R' to '0'.
7. Initialize a Bitmap variable "bmp".
8. byteArrayPos ← 1
9. For i from 1 to a
   9.1. For j from 1 to a
      9.1.1. bmp.SetPixel(i, j, file2[byteArrayPos], file2[byteArrayPos + 1], file2[byteArrayPos + 2], file2[byteArrayPos + 3])
      9.1.2. byteArrayPos ← byteArrayPos + k
   9.2. End Loop
10. End Loop
11. Return bmp

Here, the sqrt function returns the square root of the input number as the greatest integer lesser than or equal to the actual square root of the input number.

*2.2. Decoding*

Elucidated in the succeeding text is the methodology to retrieve the original file from a given bitmap image. The input is a Bitmap object and the output is an array of bytes representing the original file. The byte array can be directly saved as a computer file. This process is called *decoding* the image to the original file.

2.2.1. Equation, Variables and methodology outline

For decoding the image, let the side length in pixels of the bitmap image be $a$, and the length of the file to be recovered be $l$. Then the relation between $l$ and $a$ will be –

$l = a^2$ (4)

The process involves traversing the image pixel by pixel left to right per row. The rows are traversed from top to bottom. When a pixel is encountered, the ARGB values are extracted and are serially appended to a byte array. When all the pixels have been traversed, the byte array so obtained is the recovered file from the image. This array can be directly saved as a computer file. The algorithm for this process is enumerated in Sec. 2.2.2.

2.2.2. Decoding Algorithm

Here, a function Decode (bmp, k, size) will be defined with the following parameters-

Input –

- bmp: the Bitmap object variable pertaining to the input bitmap image to be decoded.
- k: number of bytes that has been encoded per pixel. For bitmap images, k = 4.
- size: The size of the square input bitmap image measured as number of pixels along each side of the image.

Output (Returned Parameters) –

- file [ ]: Byte array pertaining to the decoded file recovered from the input image.

**Algorithm-**

*Decode (bmp, k, size)*

1. n ← size * size
2. Initialize new byte array "file[]" with length = n
3. Initialize new Pixel Object "pix"
4. byteArrayPos ← 1
5. For i from 1 to a
   5.1. For j from 1 to a
      5.1.1. pix ← bmp.GetPixel(i, j)
      5.1.2. file[byteArrayPos] ← pix.Alpha
      5.1.3. file[byteArrayPos + 1] ← pix.Red
      5.1.4. file[byteArrayPos + 2] ← pix.Green
      5.1.5. file[byteArrayPos + 3] ← pix.Blue
      5.1.6. byteArrayPos ← byteArrayPos + 1

5.2. End Loop
6. End Loop
7. Return file [ ]

Here, initially the size of the file encoded in the image is initially calculated and stored in variable n, including the redundant bytes. The number of redundant bytes is negligibly small; hence, they don't add much to the file size. Next, a byte array variable file[] is initialized which will store the file recovered from the input image. The variable byteArrayPos is used to keep a track as to how many bytes of the file have been recovered from the image. Next, the image is traversed left to right, row by row, top to bottom and bytes are extracted from each pixel and stored in the file[] byte array hence reconstructing the original file which was encoded into the image.

**3. Implementation and Observations**

A Windows Presentation Foundation application [6] was developed using C# programming language [5] to implement this methodology. The results were found to be commendable. The user interface of the application is illustrated in Figure 1.


*Figure 1: File-Image Switching Application User Interface*

To convert a file to image, it is loaded in the upper section titled "Convert File to Image" and the "Convert To Image" button is clicked. Subsequently the Encoding algorithm is implemented on the loaded file and an image is generated. A typical image so generated is illustrated in Figure 2.

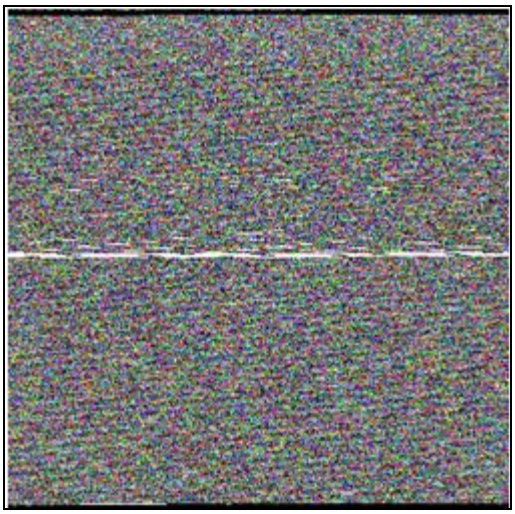

*Figure 2: Typical Image Generated When a File is Encoded to an Image*

The image shown above was generated when an MP3 file of size 380 KB was encoded to an image. The resolution of the image so generated was 312 × 312. Also, when this image was loaded in the application (lower section of the UI) and decoded, the original file was successfully retrieved with no loss.

**4. Conclusion**

In this paper, a novel way to represent a computer file as an image and to retrieve the file from the image has been presented. Images are highly pervasive in the digital world. They can be presented anywhere, from documents to social networking sites to internet based messaging services etc. The amount of versatility of image files is far greater than that of ordinary computer files. Moreover, techniques like cryptography and file compression find a huge scope in image files. Therefore, the methodology posited in this paper seeks to extend the privileges enjoyed by image files to ordinary computer files. For example, one might encode a given executable file to an image and share it on a social networking site like images are shared. Interested people may download the image and retrieve the executable from it. Considering the fact that most social networking sites do not allow sharing of computer files with as much flexibility as image files, this method surely has the potential to spark a new trend in file sharing.

**5. References**


1. AD Miller, WK Edwards, “Give and take: a study of consumer photo-sharing culture and practice”, Proceeding
2. CHI '07 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Pages 347-356, ACM New York, NY, USA ©2007, ISBN: 978-1-59593-593-9, doi>10.1145/1240624.1240682
3. Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble, “Measurement study of peer-to-peer file sharing systems”, Proc. SPIE 4673, Multimedia Computing and Networking 2002, 156 (December 10, 2001); doi:10.1117/12.449977
4. Ziedan, I.E., Fouad, M.M., Salem, D.H., “Application of data encryption standard to bitmap and JPEG images”, Radio Science Conference, 2003. NRSC 2003. Proceedings of the Twentieth National, March 2003
5. Ben Smith, Chapter “Object Oriented Programming”, Pages 1-25, AdvancED ActionScript 3.0: Design Patterns, Apress Publishing, ©2011, ISBN(Online): 978-1-4302-3615-3, doi: 10.1007/978-1-4302-3615-3_1
6. Anders Hejlsberg, Scott Wiltamuth, Peter Golde, “C# Language Specification”, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©2003, ISBN:0321154916
7. Jones, Allen, and Adam Freeman. “Windows Presentation Foundation.” Visual C# 2010 Recipes. Apress, 2010. 789-904