3. Image Formats

1. Introduction

With the growth in computer graphics and image applications the ability to store images for later manipulation became increasingly important. With no standards for image storage, application developers created their own formats. Hence, there are dozens of different formats including tif, pict, gif, jpeg, sun rasterfiles etc. These notes attempt to document the factors which must be considered when looking at image storage rather than to describe the details of any particular file format, which can be obtained when required.

2. Bitmap and Vector Representations

There are two principle and yet very different methods of representing graphics images: *bitmaps* (also called raster or pixel maps) and *vectors*. We will be concerned with *bitmap* representations of images. This is where the image has been broken up into a grid and the light value (lightness, darkness or colour) of each pixel is recorded individually. Typically, the position of a data point in a bitmap field determines what pixel the data point represents, *ie* the data points (bits) "map" to the image; hence the name, *bitmap*.

Vector representation means describing an image as a series of lines or shapes and the vector files look like programs when examined. This type of file is used for line art such as CAD drawings.

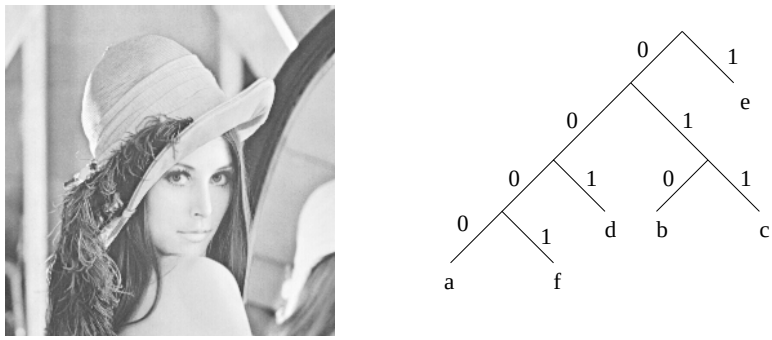

Figure 1: Example of bitmap and Vector representation images

Bitmap formats work well for images with complex variations in colours, or shapes, such as photographs, paintings, and digitised video frames. Some images, such as computer screen displays, originate in bitmap format and are therefore most easily recorded in the same way.

3. Monochrome and Colour Images

Electronically stored images, like photographs, may be either multi-colour or monochrome.

3.1 Monochrome

Monochrome images in their simplest form comprise pure "colour" regions, typically black, and pure white regions. In some monochrome images, such as photographs and shaded illustrations, shades of grey are needed, such images are often called *grey scale* images. Because the human eye can interpret a sufficiently fine pattern of black and white pixels as grey, a monochrome file format can be used for greyscale images; the data can be black and white. The process of using dots of

black or some single colour to get continuous shades of grey is called *halftoning* or *dithering*. This technique is used for most volume printing of images, although most image files are only converted to this form by the printing device.

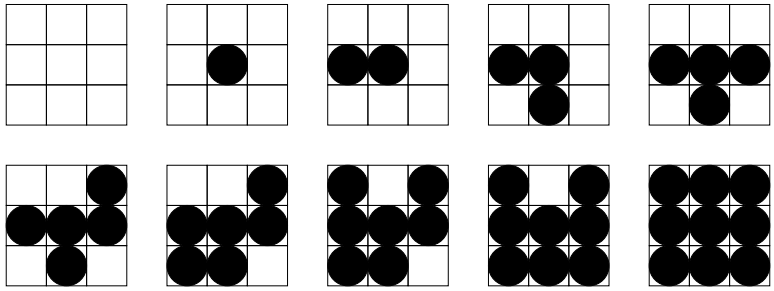

Figure 2: Dithering of a 3x3 grid to produce 10 grey levels

The use of halftoning or dithering has many disadvantages as it can make image editing very difficult and it reduces the effective resolution of the image.

3.2 True Greyscale Monochrome

Greyscale images can also be recorded as true shades of grey, rather than stored in dithered format. This is the preferred representation for images and is the format we have already considered.

3.3 Colour

When a wide range of colours is required, it is achieved by mixing a limited number of primaries. The two fundamental colour mixing schemes RGB and CMYK, depend on the intended output medium. RGB is the additive colour system used by devices which emit light, such as display screens. CMYK colour stands for the cyan, magenta, and yellow subtractive primaries, plus black for contrast and is used for printing materials. Because the black is also required for contrast CMYK is less efficient than RGB. Most output devices can handle files of either type, translating them as required. Another popular scheme is HSI, which stands for Hue, Saturation and Intensity. Luminance is often substituted for the word intensity and HSI is then called HSL.

Each pixel in a colour bitmap image requires a set of three or four values; one for each primary colour (RGB, CMYK) or independent variable (HSI), depending on the colour representation scheme used. These sets of values are typically organised on one of two ways in an image file.

The most common way of organising bitmap image data is by colour, into *colour planes*. This approach is most easily visualised as three or four monochrome images, one for each primary colour. In an RGB scheme for example, all the data for red, for every pixel in the image would be grouped together, then all the data for green, then all the data for blue.

As an alternative the image data can be organised into a single image plane in which the red, green and blue values for each pixel are grouped together.

The number of colours that an input or output device supports is quantified by the total number of bits. Thus a 24-bit colour supplies 2^{24} or more than 16 million colours. The disadvantage of using such high resolution colour images is their file size.

Some computer monitor graphics adaptors circumvent file size problems by adopting a *palette* or *colour map* approach to colour. The graphics file format then can have only so many different colours, even although each colour has a high resolution. For instance the image may be composed of colours from a 16 bit palette or 65,536 different colours. The pixel data for a palette image consists of $n-bit$ pixel codes, with each of the n codes pointing to a trio of RGB values held in a

palette table. The data file therefore must include not only the pixel data, but the table of palette data as well. This approach is also sometimes referred to as pseudo colour.

4. Common Methods of Encoding Data

Regardless of the variations in image file formats, data are encoded in certain common ways

4.1 Binary and Symbolic Coding

Graphics data may be encoded in either symbolic or binary form. Symbolic coding (almost always in ASCII) is used mainly for vector representation and for some graphics description languages. Binary coding is generally used for bitmap data.

4.2 Bit and Byte Order in Binary Data

Binary encoded data can use different byte orders, and in the case of bitmap images, can use different bit orders as well.

For instance, in a simple one-bit-deep bitmap image, the image can be scanned left to right, top to bottom, or in any other combination of directions. Furthermore, the first pixel scanned out of *n* could be mapped to either the high bit or the low bit of an *n – bit* long data word. In order to properly decode the image, you would require to know the scanning order, the word length and the bitmapping order.

Where there is more than one bit per pixel in a bitmap image, even more variations are possible. Each pixels multi-bit value can be recorded all at once, such as a 16 bit value being recorded as two consecutive bytes. This order can be thought of as creating a single 16 bit deep image plane. Alternatively, this image can be broken up into 16 one bit deep bit planes, in which the most significant bits of all pixels would be recorded together to form one plane, then all the next most significant bits to form the next plane, and so on.

The byte order can also vary and this will affect the images. The byte order usually depends on the processor used in the computer. One variation is the ordering of least and most significant bytes for numbers in excess of one byte. For example a 16 bit integer such as 51210 might appear as 00000001 00000000 (10_{16}) or 00000000 00000001 (01_{16}). The individual elements of floating point number (the sign, the fraction and the exponent) can be stored in various orders. Within the fraction and exponent, byte order can be either LSB or MSB first.

In the memory of PCs and other Intel CPU computers the LSB comes first; in Motorola systems, such as Apple Macs, the MSB comes first. UNIX workstations from Sun, IBM and HP usually store the MSB first; those from Intel and Dec store LSB data first.

The image file format must record some information about the bit and byte order if it is to be used on a variety of machines.

5. Image Compression

With the growing demands on image quality in computer applications, the amount of storage required has increased rapidly. This has resulted in the use of image compression techniques for the storage of data.

Compression is simply representing the data more efficiently. It takes advantage of the three common qualities of graphical data; they are often redundant, predictable or unnecessary.

5.1 Run Length Compression

A simple method of compressing data where a series of repeated values (eg. pixel values) is replaced by a single value and a count. For example using letters to represent values,

abbbbbbbccddddeebbb would be replaced by 1a7b2c4d2e3b. This approach works well with long strings of repeated values. Images with large areas of constant shade or hue are good candidates for this kind of compression.

5.2 Huffman Encoding

Huffman encoding works by substituting more efficient codes for data. The basic idea is to assign a binary code to each unique value, with the codes varying in length. Shorter codes are assigned to the more frequently occurring values. These assignments are stored in a conversion table, which is sent to the decoding software before the codes are sent.

For example, there are six unique values in *abbbcccddeeeeeeeeef*. The frequencies with which they appear are

$$a{:}\,1\ b{:}\,3\ c{:}\,3\ d{:}\,2\ e{:}\,9\ f{:}\,1$$

The minimum code is built using a binary tree as shown in figure 3. The basic algorithm pairs together the least frequently used elements; the pair is then treated as one element and their frequencies are combined. This continues until all pairs are combined.

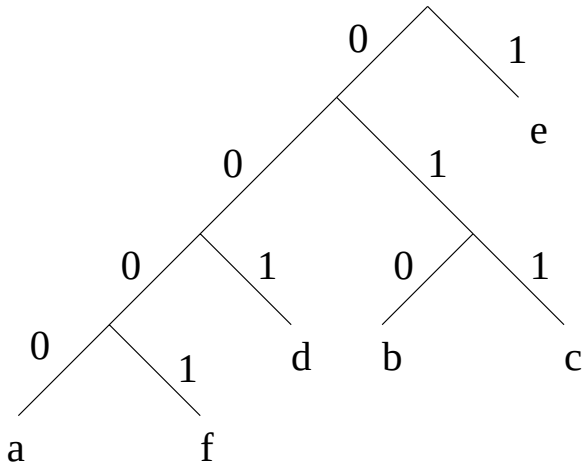

Figure 3: A binary tree for Huffman encoding

The efficiency of Huffman encoding varies with the precise algorithm and the type of image. It typically does not get compression rates above 8:1. It performs less well for files containing long runs of identical pixel values, which can be better compressed using run length or other encoding.

In addition, Huffman encoding requires accurate statistics on how frequently each value occurs. Therefore, the encoding is completed in two passes, the first to create the statistical model and the second to encode the data. Therefore, Huffman codes require a lot of processing to decode and the process is relatively slow.

A final problem with Huffman encoding is its sensitivity to dropped or added bits. Since all the bits are jammed together without regard to byte boundaries, the decoders only way of knowing when a code is finished is by reaching the end of the branch. If a bit is dropped or added, the decoder starts in the middle of a code and the rest of the data becomes nonsense.

5.3 LZW Compression

Developed by Welch building on the work of Lempel and Ziv (LWZ). LWZ is an *adaptive* code which builds an effective table of codes as it proceeds. It exploits redundancies in the patterns it finds.

LZW starts with a table with one code entry for each possible value in the data; 256 entries for an 8 bit image. It then adds entries to the table for each unique pattern of values it finds. It is necessary to fix the maximum length of the table so a code length can be established.

For example, consider the sequence *ababaaacaaaad* where each letter for simplicity represents 2 bit data values. The LZW encoder and decoder start with the same table and they track as the table expands. The initial code table would be:

For *ababaaacaaaad*

a:00 b:01 c:10 d:11

The LZW algorithm looks for the longest pattern it can recognise and find in its table. It finds the first value *a* and recognises it so it tries for the pattern *ab* and doesn't recognise it. So it transmits the code for the value it recognises (000) and makes a new table entry for the one it doesn't:-

a:000 b:001 c:010 d:011 ab: 100

The encode then takes the last value *b* and tries for a pattern with the next value *ba*, which is unrecognised so it sends the code for *b* (001).

The encoder can now recognise the next sequence *ab* and so can the decode. So a three bit code can be used to replace two 2 bit codes. Also in this example all subsequent multiple iterations of the value *a* will be entered into the table and will be represented by a single 3 bit code.

LZW encoding provides compression ratios between 1:1 and 3:1, although highly patterned images can sometimes be compressed as much as 10:1. Noisy images are hard to compress with LZW.

5.4 Arithmetic Compression

Arithmetic compression, like Huffman coding, uses shorter codes for frequently occurring things and longer codes for less common things. It is more efficient however, since (like LZW) it compresses sequences of values, not just values. Arithmetic compression is highly efficient.

There are many different forms of arithmetic coding. The basic principle is to map every different sequence of pixel values to a region on an imaginary number line between 0 and 1. That region is then represented as a binary fraction of variable precision (number of bits). Less common sequences require a higher precision (number of bits).

Arithmetic compression can reduce the file size dramatically, depending on the source and the accuracy of the statistical model used. To expect compressions of 100:1 is not unreasonable.

5.5 Lossy Compression

Lossy compression refers to techniques in which some aspect of the original data is discarded (or lost, hence, *lossy* compression), because it was deemed unnecessary to the final application. These sorts of technique are fine for commercial television and general photography but should not be used for critical applications such as medical imaging.

The things which can be discarded in lossy compression include things the eye does not need for perception in a given situation, such as unnecessary sharp edges or high spatial resolution of

colour.

The most commonly cited lossy compression scheme is the baseline algorithm for JPEG. JPEG (Joint Photographic Experts Group) defines several algorithms although the algorithm most commonly referred to as JPEG is the lossy, baseline algorithm. JPEG also defines a lossless algorithm based on differential pulse-code modulation. Both employ multiple compression algorithms in combination, including Huffman and arithmetic coding.

JPEG algorithms begin by separating the chroma (colour) information from the luminance (brightness) information, to take advantage of the eye's tolerance for lower colour resolution. The image is then broken up into smaller tiles. The lossy algorithm then applies a Discrete Cosine Transform to each of the tiles. After this transformation, the individual pixels nolonger exist but are represented by a series of patterns describing how rapidly the pixel values vary.

MPEG is an emerging standard related to JPEG and is intended for the storage of moving pictures. An MPEG file contains JPEG-like frames, along with information for interpolating other frames between the JPEG like frames.

5.6 JPEG Image Format

JPEG was designed for compressing either full-color or gray-scale images of natural, real-world scenes. It works well on photographs, naturalistic artwork, and similar material; not so well on lettering, simple cartoons, or line drawings. JPEG handles only still images, but as mentioned above there is a related standard called MPEG for motion pictures.

JPEG works on either full-colour or gray-scale images; it does not handle bilevel (black and white) images, at least not well. It doesn't handle colour-mapped images either; these must be pre-expanded into an unmapped full-colour representation. JPEG works best on "continuous tone" images. Images with many sudden jumps in colour values will not compress well.

JPEG is "lossy," meaning that the decompressed image isn't quite the same as the one you started with. JPEG is designed to exploit known limitations of the human eye, notably the fact that small color changes are perceived less accurately than small changes in brightness. Thus, JPEG is intended for compressing images that will be looked at by humans. If the images are to be machine-analyzed, the small errors introduced by JPEG may be a problem, even if they are invisible to the eye.

A useful property of JPEG is that the degree of lossiness can be varied by adjusting compression parameters. This means that the image maker can trade off file size against output image quality. You can make *extremely* small files if you don't mind poor quality; this is useful for applications such as indexing image archives. Conversely, if you aren't happy with the output quality at the default compression setting, you can increase the quality until you are satisfied, and accept lesser compression.

Another important aspect of JPEG is that decoders can trade off decoding speed against image quality, by using fast but inaccurate approximations to the required calculations. Some viewers obtain remarkable speedups in this way. (Encoders can also trade accuracy for speed, but there's usually less reason to make such a sacrifice when writing a file.)

There are a lot of parameters to the JPEG compression process. By adjusting the parameters, you can trade off compressed image size against reconstructed image quality over a very wide range. Usually the threshold of visible difference from the source image is somewhere around 10 to 20 times smaller than the original i.e., 1 to 2 bits per pixel for colour images. Grayscale images do not compress as much.

JPEG defines a "baseline" lossy algorithm, plus optional extensions for progressive and hierarchical coding. There is also a separate lossless compression mode; this typically gives about 2:1

compression, ie, about 12 bits per colour pixel. Most currently available JPEG hardware and software handles only the baseline mode.

5.6.1 Outline of the baseline compression algorithm:

The baseline algorithm consists of 5 basic steps.

1. **Colour Schemes and Sub Sampling**

   Transform the image into a luminance/chrominance colour space (YCbCr, YUV, etc). (Note: this is not required for greyscale images). The luminance component is grayscale and the other two axes are colour information. The reason for doing this is that you can afford to lose a lot more information in the chrominance components than you can in the luminance component: the human eye is not as sensitive to high-frequency chroma information as it is to high-frequency luminance. (See any TV system for precedents.)

   (Optional) Downsample each component by averaging together groups of pixels. The luminance component is left at full resolution, while the chroma components are often reduced 2:1 horizontally and either 2:1 or 1:1 (no change) vertically This step immediately reduces the data volume by one-half or one-third. In numerical terms it is highly lossy, but for most images it has almost no impact on perceived quality, because of the eye's poorer resolution for chroma information. Note that downsampling is not applicable to grayscale data; this is one reason why colour images are more compressible than grayscale.

2. **DCT Coding**

   The Discrete Cosine Transform (DCT) turns an array of intensity data into an array of frequency data that tells how fast the intensities vary. The result of the DCT is a set of spatial frequencies F(u,v). F(0,0) is the DC or average value. The rest of the coefficients are AC coefficients.

   To apply the DCT the pixels are grouped into 8x8 blocks. Each 8x 8 block is transformed by applying the DCT, giving a frequency map, with 8x8 components. Thus you now have numbers representing the average value in each block and successively higher-frequency changes within the block. The motivation for doing this is that you can now throw away high-frequency information without affecting low-frequency information, which the eye is more sensitive to.

3. **Quantization**

   The quantization step sets the precision to which each of the values resulting from the DCT are stored.

   In each 8x8 block, divide each of the 64 frequency components by a separate "quantization coefficient", and round the results to integers. This is the fundamental information-losing step. The larger the quantization coefficients, the more data is discarded. Note that even the minimum possible quantization coefficient, 1, loses some information, because the exact DCT outputs are typically not integers. Higher frequencies are always quantized less accurately (given larger coefficients) than lower, since they are less visible to the eye. Also, the luminance data is typically quantized more accurately than the chroma data, by using separate 64-element quantization tables. Tuning the quantization tables for best results is something of a black art, and is an active research area. Most existing encoders use simple linear scaling, using a single user specified "quality" setting to determine the scaling multiplier. This works fairly well for midrange qualities (but is quite nonoptimal at very high or low quality settings).

   The DC component is then treated specially: since adjacent DC components tend to be similar, each DC component is stored as the difference from the DC component of the preceding block. The other frequency components are logically placed in zigzag order (see figure 4), which puts

the lowest frequency components first. The quantised high frequency components will often be zero and the final compression stage works well on groups of zero values.

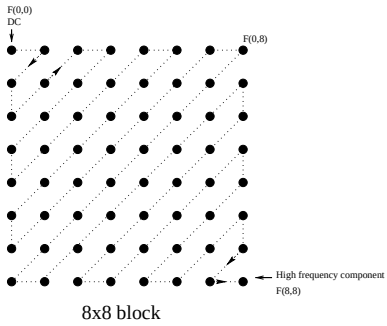

Figure 4: Zigzag sequence

4. Final Compression

The results of the quantisation are compressed (entropy coded) using either Huffman or arithmetic coding. Notice that this step is lossless, so it doesn't affect image quality. The arithmetic coding option uses a patented system. Most existing implementations support only the Huffman mode, so as to avoid license fees. The arithmetic mode offers maybe 5 or 10% better compression.

5. Form Image

Add on appropriate image headers, etc, and output the result. In a normal JPEG file, all of the compression parameters are included in the headers so that the decompressor can reverse the process. These parameters include the quantization tables and the Huffman coding tables. For specialized applications, the specification permits those tables to be omitted from the file; this reduces the file size, but it means that the decompressor must know a-priori what tables the compressor used. Omitting the tables is safe only in closed systems.

The decompression algorithm reverses this process. The decompressor multiplies the reduced coefficients by the quantization table entries to produce approximate DCT coefficients. Since these are only approximate, the reconstructed pixel values are also approximate, but if the design has done what it's supposed to do, the errors won't be highly visible. A high-quality decompressor will typically add some smoothing steps to reduce pixel-to-pixel discontinuities.