

What is Image Processing?

Image Processing is the science of image manipulation

Image Processing is **NOT** Computer Graphics

Computer Graphics is concerned with generating or creating images. With image processing, we already have an image that we wish to manipulate. Computer graphics often involves 3D objects, image processing usually does not.

Image processing algorithms therefore alter images.

Where is Image Processing Used?

- Scientific Imaging – reconstruction, filtering, enhancement
- Industrial control – product inspection, quality control
- Movies – Special Effects, composition, morphing etc.
- Medical – CAT (or CT computer aided tomography) PET (positron emission tomography) - shows chemistry, MSI (magnetic source imaging) - shows brain activity (electrical), MRI (magnetic resonance imaging) – shows soft tissues, X-rays.
- Law enforcement – fingerprint comparison, image enhancement, ageing mugshots.

What is a pixel?

A Pixel is a **Pix. Element** or Picture Element. It refers to the representation of a single point within an image. A pixel may be simply a bit (for black & white images), or a much larger data structure.

What is a Bitmap?

Technically, a bitmap should refer to a 1 bit per pixel black & white image. Most people use it as a more general term.

A bitmap is a binary representation of a picture or image.

Generally, we represent images either in bitmap (pixel based) formats (e.g. GIF) or vector based formats (e.g. Corel Draw, or AutoCad). Typically a colour bitmap will consist of three bytes per pixel (giving the red, green & blue components of that pixel's colour –so called truecolour). These pixels are logically arranged in a 2 dimensional array. Remember though, that a two dimensional array is simply a way we view a "chunk" of memory. The image will be stored in RAM or on disk as simply a sequence of pixel values.

The amount of memory needed to hold a bitmap depends on the size of the image, the size of the variables used to hold each pixel's value, and the colour depth of the image.

Consider a 1024 x 768 pixel image.

Bit Depth	Number of Colours	Image size
1	2	96 kilobytes
8	255	768 kilobytes
24	16,581,375 (2^{24})	2,304 kilobytes

The Portable Bitmap (PBM) file formats

The portable bit map family of file formats are very simple and portable.

Most people (and software) talks about PBM files. More correctly we have:

- PBM (Portable BitMap)
- PGM (Portable GrayMap)
- PPM (Portable PixelMap)

Most operating systems have utilities to process these file types.

Free source packaged as *pbmplus* or *netpbm* is available for all unix based operating systems (and ms-dos). These are normally included in linux distributions. (Redhat 7.0 ships with netpbm-9.5-5).

Worksheet 3 introduces the pbm utilities available here on Solaris

These files are relatively unsophisticated and **LARGE!**

No form of compression is supported, (if you want to compress these files use the unix commands *compress* or *gzip*).

In order to do anything useful, (and for the first part of your assignment!) we will need to write functions that can read and write files in these formats.

We can find the precise details for each of these formats by using the unix manual and looking in section 5.

Assuming you have aliased the macro from worksheet 3

pbman -s5 ppm will give you the format of the pixmap type on Solaris

or

man 5 ppm will do the same on most linux systems

We'll look at this manual page later. While you are free to implement the full modern standard (post 2000) if you wish, **this is not required!**

We do not need all of these features and therefore like much commercial software, we'll choose to ignore them!

All files have a header followed by image data and come in two flavours, ASCII & RAW.

First part of the header is a magic number:

P1	ASCII bitmap	P4	RAW bitmap
P2	ASCII grayscale	P5	RAW grayscale
P3	ASCII colour	P6	RAW colour

This in ASCII must be the first two characters of the file.
This must be followed by whitespace.

Any line containing a # is a comment from that point to the end-of-line and is ignored.

Next part is the width in pixels, written in ASCII.
This must be followed by whitespace.

Next part is the height in pixels, written in ASCII.
This must be followed by whitespace.

If not binary:

Next is the max {colour component, gray} value, written in ASCII. PBM supports 16 bits per colour, you do not need to! (up to 255 is fine)
This must be followed by whitespace.

Then follows the image data. If the image is ASCII then values are written in ASCII separated by whitespace, with no line longer than 70 characters.

If the image is RAW, then the final header field must be followed by a single character of whitespace, followed by the binary data.

Lets look at what the manual says:

The PPM Manual Page

NAME

ppm portable pixmap file format

DESCRIPTION

The portable pixmap format is a lowest common denominator color image file format.

It should be noted that this format is egregiously inefficient. It is highly redundant, while containing a lot of information that the human eye can't even discern. Furthermore, the format allows very little information about the image besides basic color, which means you may have to couple a file in this format with other independent information to get any decent use out of it. However, it is very easy to write and analyze programs to process this format, and that is the point.

It should also be noted that files often conform to this format in every respect except the precise semantics of the sample values. These files are useful because of the way PPM is used as an intermediary format. They are informally called PPM files, but to be absolutely precise, you should indicate the variation from true PPM. For example, "PPM using the red, green, and blue colors that the scanner in question uses."

The format definition is as follows.

A PPM file consists of a sequence of one or more PPM images. There are no data, delimiters, or padding before, after, or between images. A common

subformat, and the only one defined before July 2000, has exactly one image. Most tools to process PPM files ignore any data after the first image.

Each PPM image consists of the following:

- A "magic number" for identifying the file type. A pgm file's magic number is the two characters "P6".
- Whitespace (blanks, TABs, CRs, LFs).
- A width, formatted as ASCII characters in decimal.
- Whitespace.
- A height, again in ASCII decimal.
- Whitespace.
- The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536.
- Newline or other single whitespace character.
- A raster of Width * Height pixels, proceeding through the image in normal English reading order. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.
- In the raster, the sample values are proportional to the intensity of the CIE Rec. 709 red, green, and blue in the pixel. A value of Maxval for all three samples represents CIE D65 white and the most intense color in the color universe of which the image is part (the color universe is all the colors in all images to which this image might be compared).
- Characters from a "#" to the next endofline, before the maxval line, are comments and are ignored.

Note that you can use **pnmddepth** To convert between a the format with 1 byte per sample and the one with 2 bytes per sample.

There is actually another version of the PPM format that is fairly rare: "plain" PPM format. The format above, which is generally considered the normal one, is known as the "raw" PPM format. See `pbm(5)` for some commentary on how plain and raw formats relate to one another.

The difference in the plain format is:

- There is exactly one image in a file.
- The magic number is P3 instead of P6.
- Each sample in the raster is represented as an ASCII decimal number (of arbitrary size).
- Each sample in the raster has white space before and after it. There must be at least one character of white space between any two samples, but there is no maximum. There is no particular separation of one pixel from another just the required separation between the blue sample of one pixel from the red sample of the next pixel.
- No line should be longer than 70 characters.

Here is an example of a small pixmap in this format:

```
P3
# feep.ppm
4 4
15
0 0 0 0 0 0 0 0 0 15 0 15 0 0 0
0 15 7 0 0 0 0 0 0 0 0 0 15 0 15
0 0 0 0 0 0
0 15 7 0 0 0
0 0 0 0 0 0
```

Programs that read this format should be as lenient as possible, accepting anything that looks remotely like a pixmap.

COMPATIBILITY

Before April 2000, a raw format PBM file could not have a maxval greater than 255. Hence, it could not have more than one byte per sample. Old programs may depend on this.

SEE ALSO

giftopnm(1), gouldtoppm(1), ilbmtoppm(1), imgtoppm(1), pgmtoppm(1), pi1toppm(1), picttoppm(1), pjtoppm(1), rgb3toppm(1), sldtoppm(1), spctoppm(1), sputoppm(1), xpmtoppm(1), yuvtoppm(1), ppmttoacad(1), ppmtogif(1), ppmtopcx(1), ppmtopgm(1), ppmtopi1(1), ppmtopict(1), mtvttoppm(1), pcxtoppm(1), qrttoppm(1), rawtoppm(1), tgatoppm(1), ximtoppm(1), ppmtoicr(1), ppmtoilbm(1), ppmtopj(1), ppmtopuzz(1), ppmtorgb3(1), ppmtosixel(1), ppmtotga(1), ppmtouil(1), ppmtoxpm(1), ppmtoyuv(1), ppmtdither(1), ppmforge(1), ppmhist(1), ppmmake(1), ppmpat(1), ppmquant(1), ppmquantall(1), ppmrelief(1), pnm(5), pgm(5), pbm(5)

AUTHOR

Copyright (C) 1989, 1991 by Jef Poskanzer.

What do we need to do?

The manual page is more thorough but basically repeats what the previous slides said.

We need to think about how we will code our file reading function.

- What do we want to return to the calling function?
- What do we want to return?
- How do we handle errors?
- What problems are we likely to face?

BREAK IT DOWN!

Lets just consider P6 type files.

We can create a memory block for this data with either *malloc()* or *calloc()* e.g. something like:

```
#include <stdlib.h>

main()
{
    unsigned char *data;

    /* process header & get width height & bpp */

    data = (unsigned char ) malloc(width*height*bpp);
    fread(data, 1, width*height*bpp, FilePointer);

    /* more code */
}
```

We can describe using a flowchart or "JSP like" diagram, what our file will look like.

We need to read, collect together the characters for, and convert to a numeric type the data from the header (filetype, width, height, depth).

We need to read and throw away the whitespace separating these, and any comments.

We can return a value to identify causes of failure, part of a possible scheme being:

- 0 OK
- 1 Couldn't open file
- 2 Not a raw ppm file
- 3 Header is corrupt
- 4 Couldn't allocate memory
- 5 Data is truncated
- 6 Undefined error

Classification of Algorithms

Image processing algorithms can be classified into four classes according to what they operate on.

1. Point algorithms

These modify a pixels value depending on that pixels position or value. In Adobe Photoshop, the "Image/Adjust/Levels" command belongs to this class.

2. Area algorithms

These modify a pixels value depending on the value of that pixel and its neighbours. In Adobe Photoshop the "Filters/Artistic/Palette Knife" command belongs to this class.

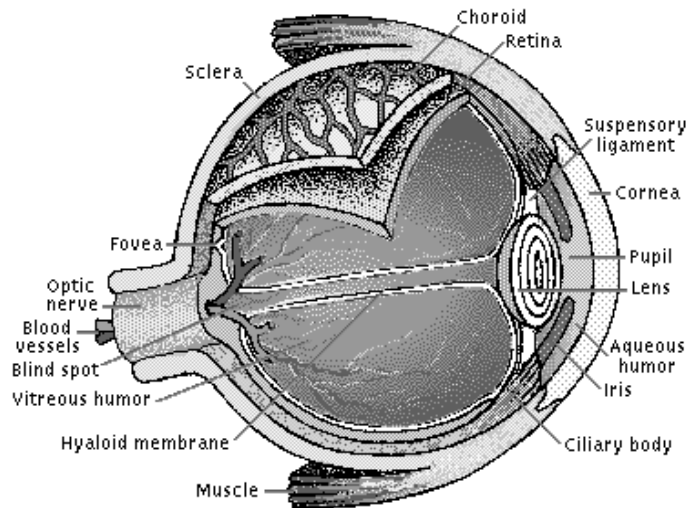
3. Geometric algorithms

These modify the position or arrangement of pixels. In Adobe Photoshop, the "Filters/Blur/Gaussian Blur" command falls into this class.

4. Frame algorithms

These generate new pixel values depending on two or more images. Combining images, even operations as simple as cut & paste fall into this class.

How do we perceive images?

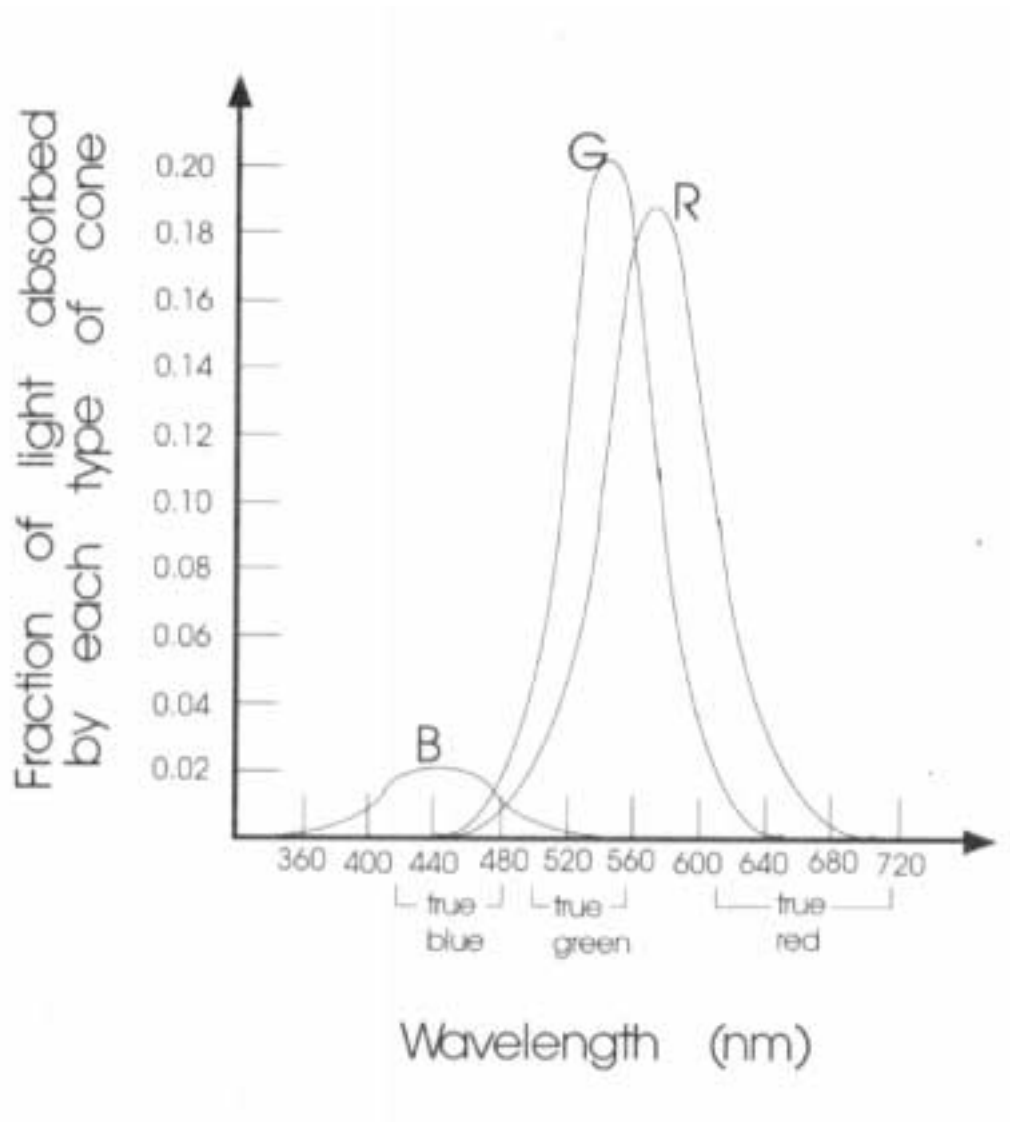


Human Eye - courtesy of encarta

When we see, light reflected by an image enter the eye through the **cornea** (the fixed outer lens of the eye). The **ciliary** muscles flex, altering the shape of the **lens**, to focus the image on the **retina**. The retina contains photoreceptors, called rods & cones, having around 125×10^6 rods and 7×10^6 cones.

- Rods are sensitive to **luminance** or **intensity**. They work in very low light but play no part in colour vision.
- Cones are concentrated in the **fovea**, at the centre of the retina. There are three types of cone, sometimes referred to as red, green and blue. From these we get our sense of **hue**. Calling the cones red, green and blue is misleading. For most colours, more than one type of cone will respond, giving us the ability to distinguish colour.
- Vision is sharpest (i.e we can see small detail clearly) in the fovea, decreasing as we move out. Light sensitivity is greatest in the peripheral vision however due to a higher percentage of rods.

Cone Response (graph from Crane)

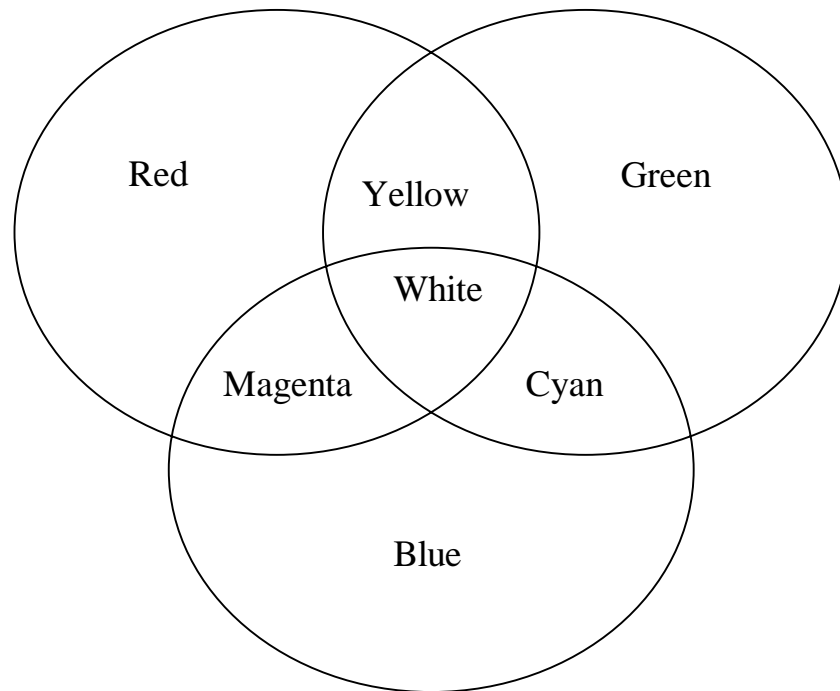


Colour Models

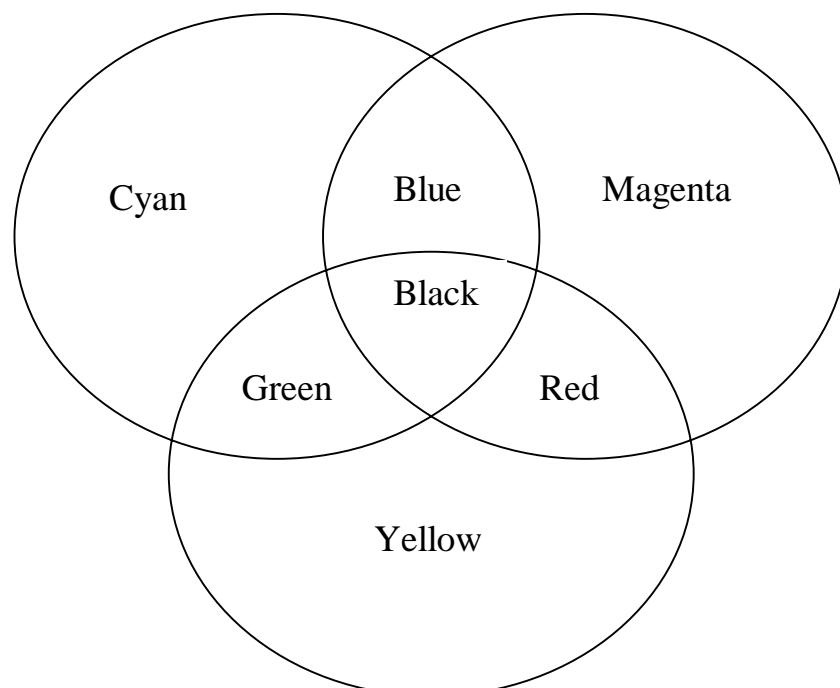
Since the human eye responds to three colours, many colour models are based on three values, often referred to as *tristimulus* values. Many many different models exist, a new very common model being *sRGB*.

Additive v Subtractive

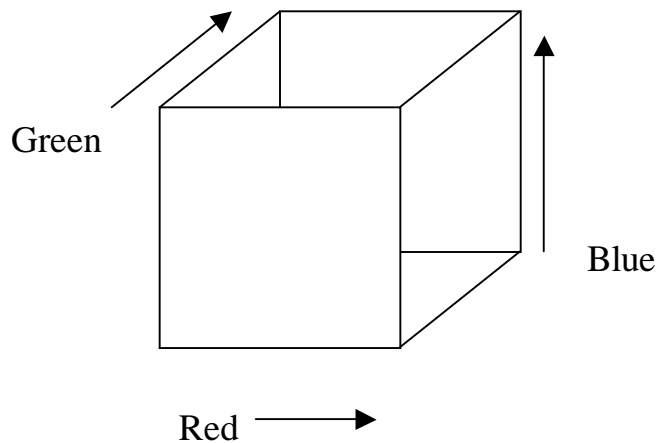
Generally, when we discuss RGB models we are talking about an additive model, whilst CMY generally means subtractive. The additive (RGB) model is show below.



The subtractive model:



In general, we can regard these colour spaces as cubic. That is each colour can be represented by a box within a three dimensional array or cube, having an axis for (with a subtractive model) R, G, & B



We will need to look at a totally different model shortly, but for now we can consider our next programming task.

Gray scale conversion.

Our first programming problem!

Gray scale conversion can be performed as a simple average $\frac{1}{3} R + \frac{1}{3} G + \frac{1}{3} B$.

The NTSC standard suggests $0.299 R + 0.587 G + 0.114 B$, since our perception in intensity terms is different for each colour.

There are some alternative methods, based on the HIS colour model which we will meet shortly.

Remember $255+255+255$ will not fit in a byte!