

PPM Images

There are many formats to store images digitally. The two basic approaches are *pixel/bitmaps* and *vectors*. Vectors provide a series of geometric/mathematical commands for generating the image (e.g. put a line there, shade that part green, etc.). The advantage is that the image can scale easily; the potential disadvantage is speed of use. Bitmaps represent the picture as a grid of pixels and specify the color of each pixel. A potential disadvantage is the size of the file. It should be noted that in order for us to see a vector image, it needs to be interpreted/resolved as pixels.

There are also several ways to specify a *color*. One version is to consider the primary components of light as red, green, and blue; and any color is specified as the intensity of each. In 8-bit graphics, each color-intensity is specified as a number in the range 0 to 255.

In this course, we consider one of the many formats for storing a bitmap image, called *ppm*. It is one of a sextet of formats in the “Portable pixmap” family. On our UNIX system, a ppm file can be viewed with the program *xv* and several others. There are a few free previewers on Windows.

All these formats have a header portion, giving facts about the image such as its size, followed by the actual data. The header for a ppm file is:

```
P6
Width Height Max
```

where the Width is the number of pixels in a row, given in decimal; the Height is the number of pixels in a column, given in decimal; and the Max is the largest possible number for a color-intensity, which for us will always be 255. For example, the header could be

```
P6
100 100 255
```

There must be *no spaces* after the Max, just a single line-return. After that, comes the actual data. This is an **uninterrupted** stream of 8-bit numbers. This is often called binary data, since the data does not correspond to standard characters, and does not make sense when opened with a wordprocessor.

We will write C programs that create and manipulate ppm files. These will read and write the 8-bit numbers as **unsigned char**.

Here is code to create a checkered pattern and the pattern:

```
// checkered.c
// prints out image that is BW for B=Black
//                               WB      and W=White
#include <stdio.h>

int main()
{
    const int MAXVAL = 255;
    const int BLOCK_SIZE = 50;
    int r,c;

    printf("P6\n");
    printf("%d %d %d\n", 2*BLOCK_SIZE, 2*BLOCK_SIZE, MAXVAL);

    for(r=0; r<BLOCK_SIZE; r++) {
        for(c=0; c<BLOCK_SIZE; c++)
            printf("%c%c%c", 0, 0, 0);
        for(c=0; c<BLOCK_SIZE; c++)
            printf("%c%c%c", 0, MAXVAL, MAXVAL);
    }

    for(r=0; r<BLOCK_SIZE; r++) {
        for(c=0; c<BLOCK_SIZE; c++)
            printf("%c%c%c", 0, MAXVAL, MAXVAL);
        for(c=0; c<BLOCK_SIZE; c++)
            printf("%c%c%c", 0, 0, 0);
    }

    return 0;
}
```

