asg03: images

- I/O, UI
- transformations
  - g: color → greyscale
  - n: negative image
    (e.g., color = 1 − color)
  - r: rotate 90°
  - s: shrink (reduce by 2)
- 3 input images
  (autograder: $\frac{1}{12}$ * correctness)

milestones:

1. write simple_image
   (e.g., blue, red, etc.)

2. read & write image
   (i.e. copy image)

3. tackle each of 4 transforms
   (rotation hardest)
   (scaling next hardest)
   (negative fairly easy)
   (greyscale: several ways to do it)

So what is an image?

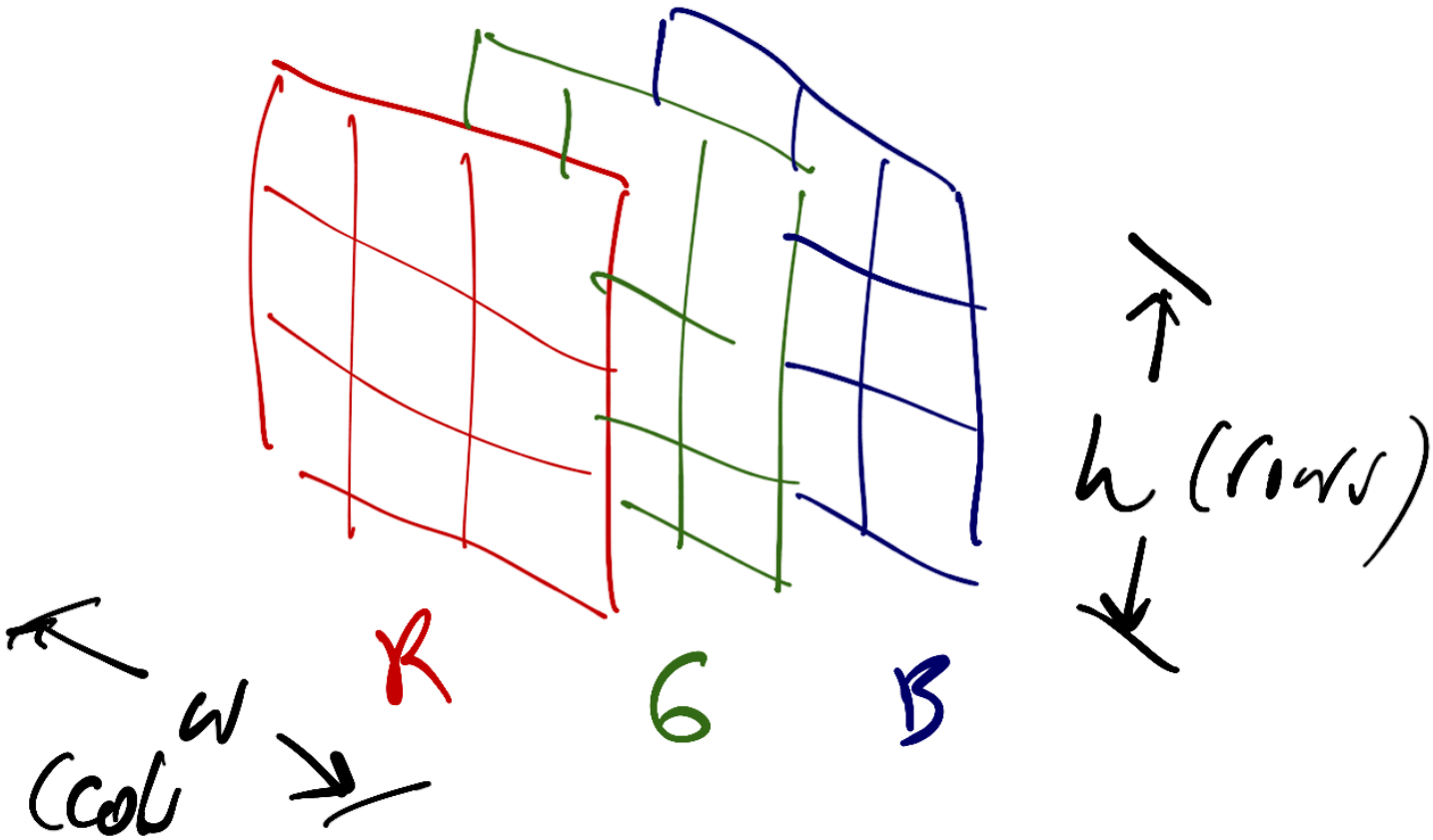

image file

```
P6\n
# comment... \n
# comment... \n
w h\n
255\n
rgb rgb ...
```

example of comment lines

out Tiger.ppm

P6 \n

CREATOR: GIMP PNM. - \n

561 375 \n

255 \n

rgb rgb ...

each is an unsigned char

8 bits

00000000	0x00
00000001	0x01
⋮	⋮
11111111	0xFF

11111111

1

128+64+32+16+8+4+2+1 = 255

each pixel : 255 × 255 × 255 = 16M

- so that's the data: how to store in memory?
(data structure)

- need:

```
int   h;   // height
int   w;   // width
int   max; // max color per channel (255)

float *rpix;
float *gpix;
float *bpix;
```

} 3 1D arrays, one for each channel

- How large is each channel array?

width * height

(cols) * (rows)

w * h

- How to allocate memory?

```
float *rpix = NULL;
rpix = (float *) malloc(w*h*
                        sizeof(float));
```

- How to access each pixel?

each pixel is at row i, col j.

```
for (int i = 0; i < h; i++) {
    for (int j = 0; j < w; j++) {
        r[i*w + j] = 1.0;
        g[i*w + j] = 0.0;
        b[i*w + j] = 0.0;
    }
}
```

— using float arrays:
remember to scale
it by 255.0 before
writing out the
unsigned char

scaled
to 255

normalized
pixel values
[0, 1]