COMP120: Creative Computing

1: Tinkering Graphics I

Learning Outcomes

- **Explain how** pictures are digitised into raster images by a computer system
- **Apply** pair programming practices to solve media computation problems

FALMOUTH
UNIVERSITY

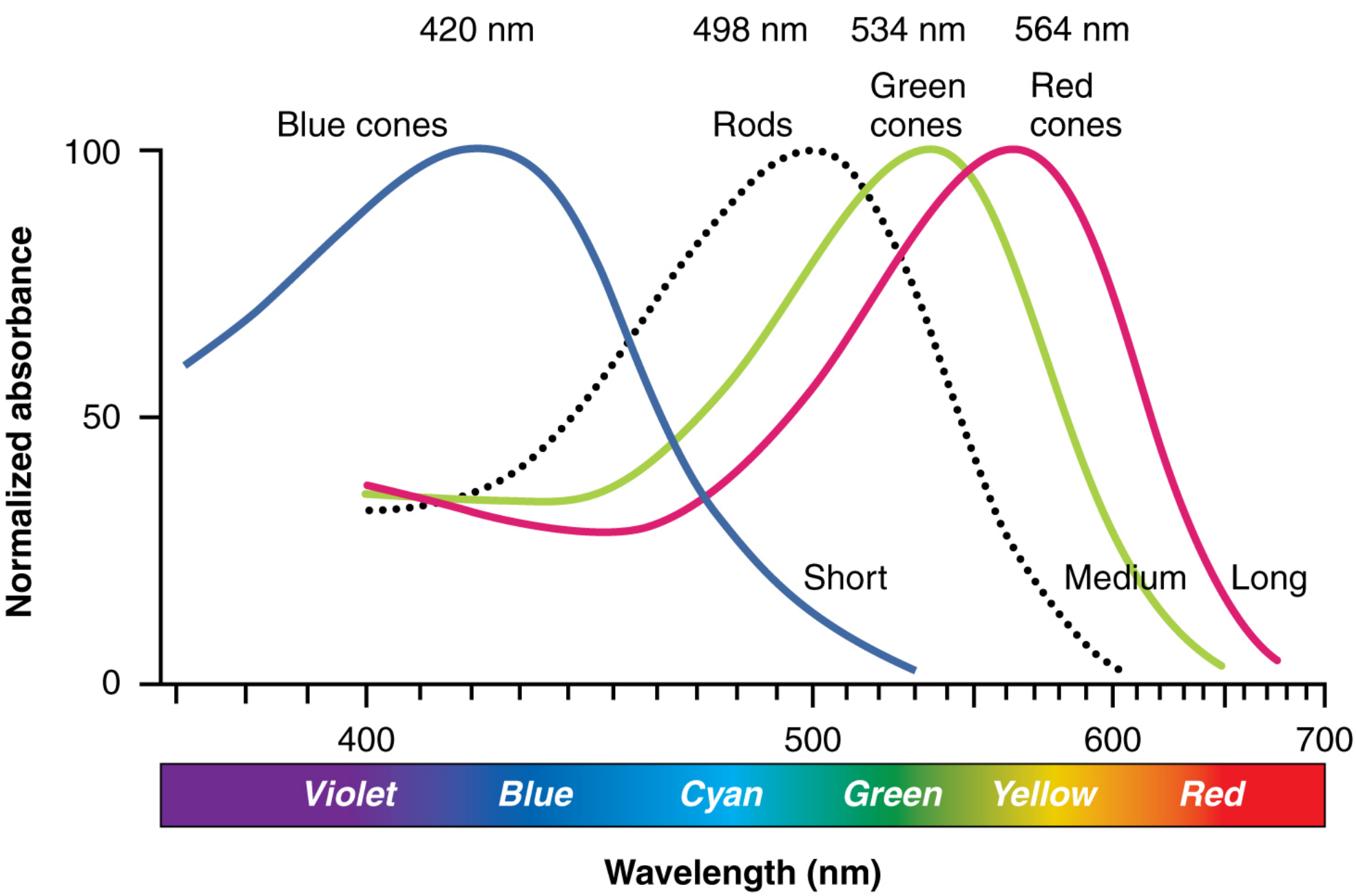
Tinkering Graphics

Light Perception

- Colour is continuous:
  - Visible light is in the wavelengths between 370nm and 730nm
  - i.e., 0.00000037 — 0.00000073 meters
- However, we *perceive* light around three particular peaks:
  - Blue peaks around 425nm
  - Green peaks around 550nm
  - Red peaks around 560nm

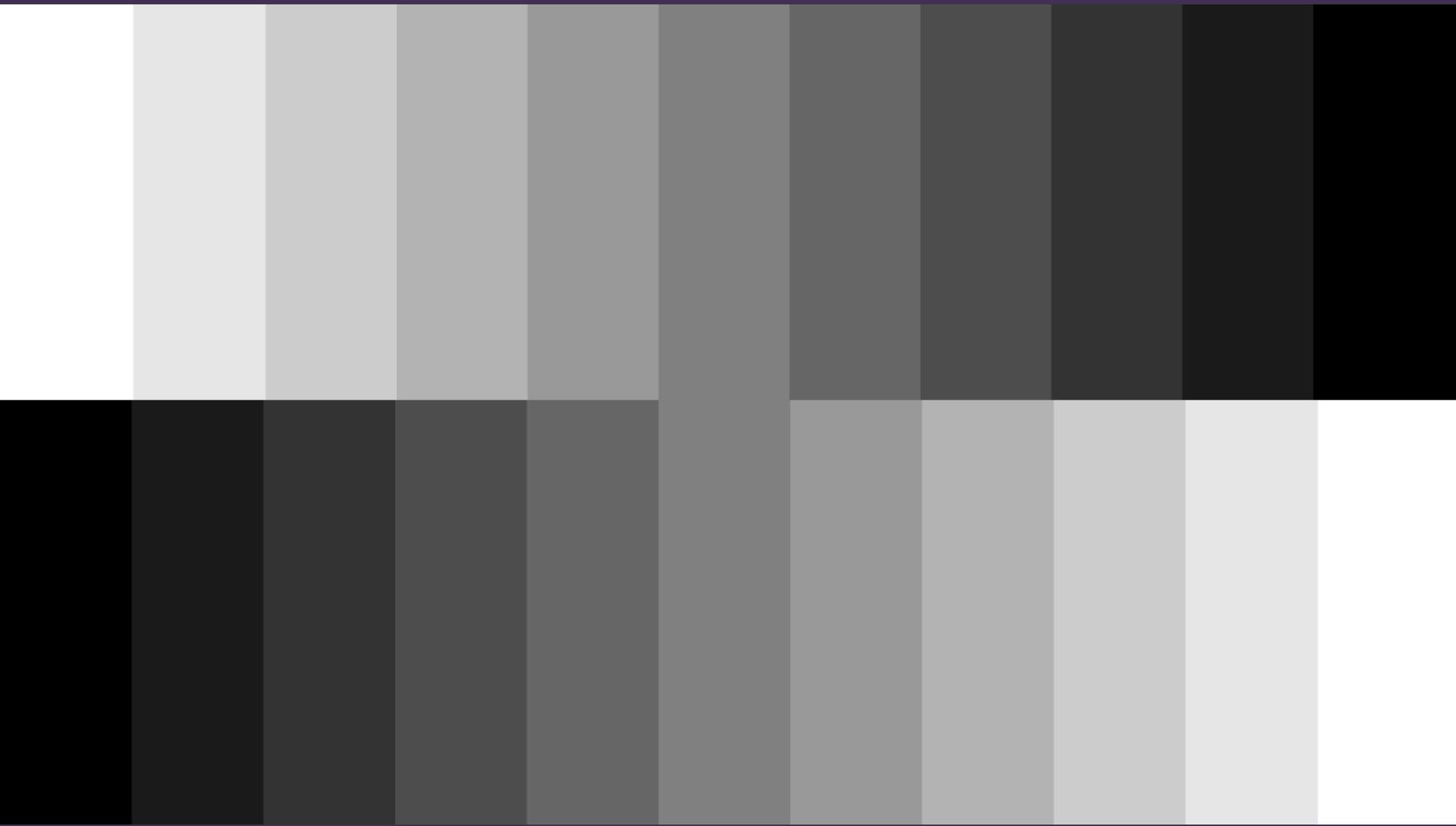
Light Perception

- Our eyes have three types of colour-sensitive photoreceptor cells called `cones' that respond to light wavelengths
- Our perception of colour is based on how much of each kind of sensor is responding
- An implication of this is perception overlap: we see two kinds of `orange' — one that's spectral and one that's combinatorial


420 nm
498 nm
534 nm
564 nm
Blue cones
Rods
Green cones
Red cones
100
50
0
Normalized absorbance
Short
Medium
Long
400
500
600
700
Violet
Blue
Cyan
Green
Yellow
Red
Wavelength (nm)

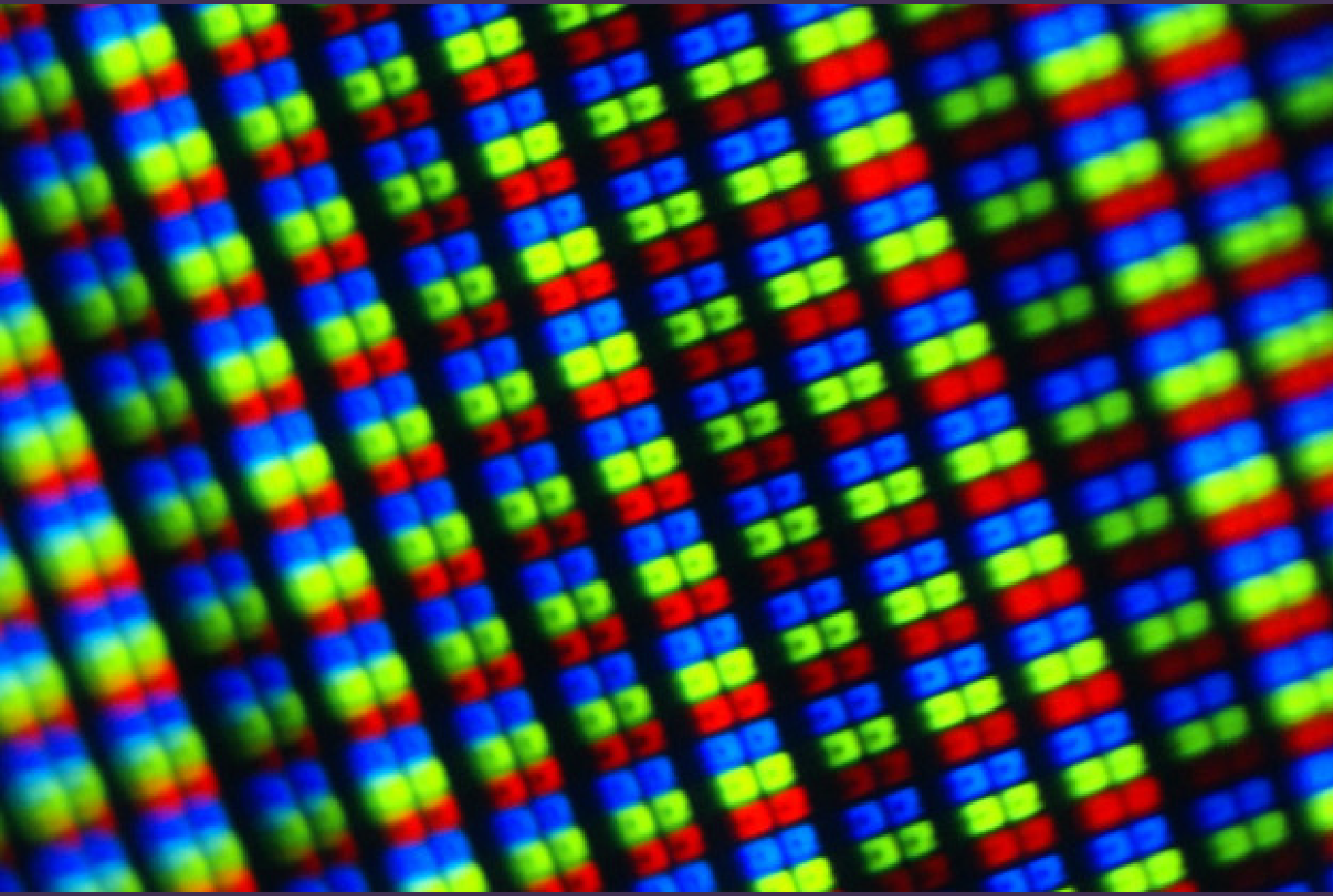
Luminance vs Colour

- Our eyes have another type of photoreceptor cells called 'rods' that respond to light intensity
- Our perception, however, is actually luminance: a relativistic contrast of *borders* of things (i.e., motion)
  - Luminance is *not* the amount of light, but our perception of the amount of light
  - Much of our luminance perception is based on comparison to background, not raw values
- An implication of this is perception overlap: we see blue as 'darker' than red when the intensity is actually the same



Resolution

- We have a limited number of rods and cones in our eyes
- This means humans perceive vision in a limited resolution — yet, we perceive vision as continuous
- We take advantage of this human characteristic in computer monitors



Pixels

- We digitize pictures into many little dots
- Enough dots and it looks like a continuous whole to our eye
- Each element is referred to as a *pixel*

Pixels

Pixels must have:

- a color
- a position

Pictures and Surfaces

In PyGame, a `Surface` is a *matrix* of pixels

- It is not a continuous line of elements, that is, a one-dimensional *array*
- A picture has two dimensions: width and height
- It's a two-dimensional *array*

Pictures and Surfaces

- (x, y) —or— (horizontal, vertical)
- The origin (0,0) is top-left
- (1,0) = 12
- (0, 2) = 6

Encoding Colour

- Each element in the matrix is a pixel, with the matrix defining its position and the value defining its colour
- Computer memory stores numbers, so colour must be encoded into a number:
  - CMYK = cyan, magenta, yellow, black
  - HSB = hue, saturation, brightness
  - RGBA = red, green, blue, alpha (transparency)
- By default, Visual Studio and C# uses RGBA

Encoding RGB

- Each component color (red, green, and blue) is encoded as a single byte
- Colors go from
  - If all three components are the same, the colour is in grey-scale
  - `(0,0,0)` is black
  - `(255, 255, 255)` is white

Encoding Bits

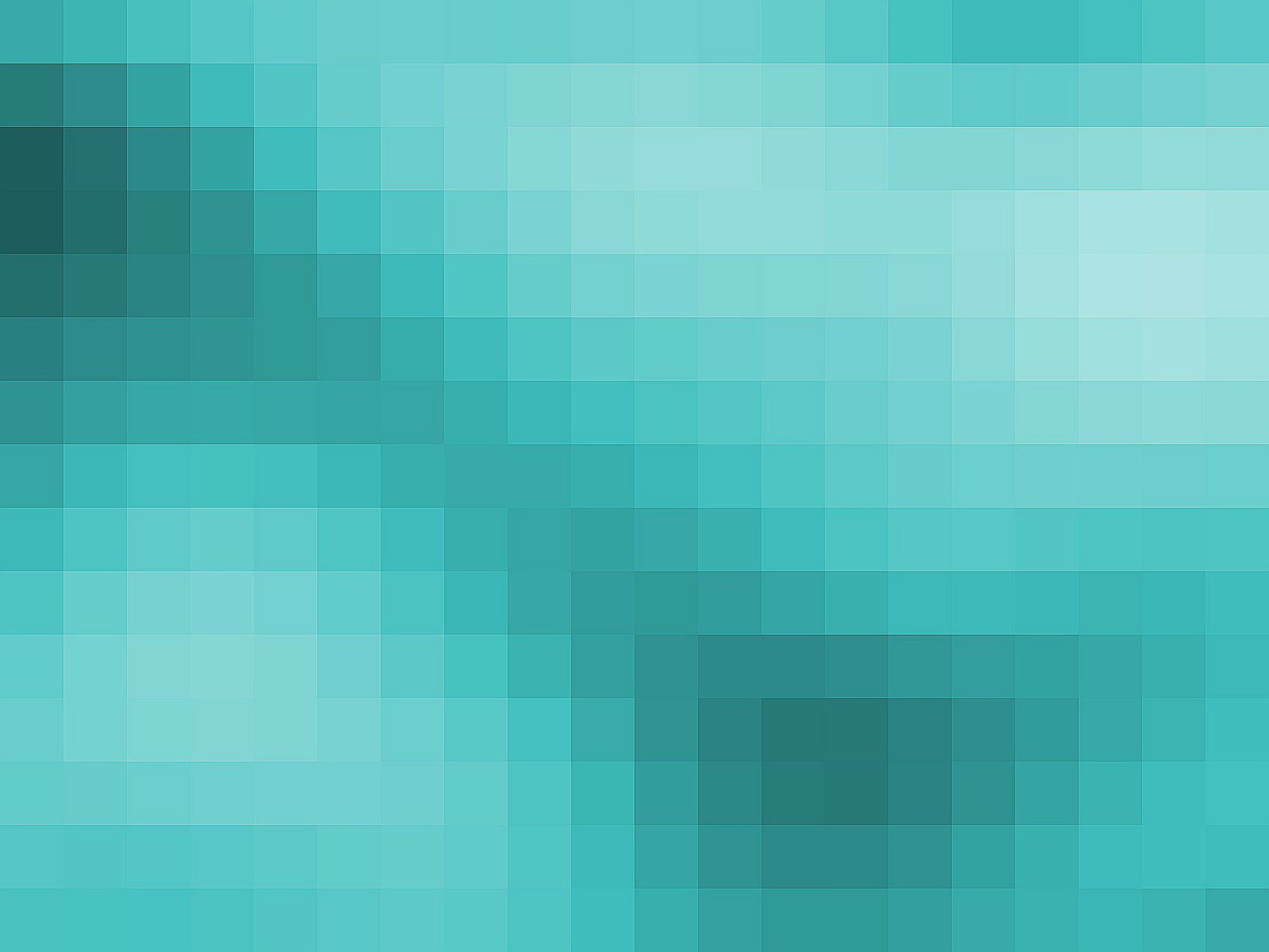
Why `255`?

- If we have one bit, we can represent **TWO** patterns:
  - 0
  - 1
- If we have two bits, we can represent **FOUR** patterns:
  - 00
  - 01
  - 10
  - 11
- With n bits, we can have 2^n patterns
- With 8 bits, there will be 256 patterns
- One of these patterns will be 0, so the highest value we can represent with 8 bits is: $2^8 - 1$, or 255

R: 255 G: 0 B: 0 A: 165	R: 255 G: 0 B: 0 A: 255	R: 255 G: 0 B: 0 A: 255	R: 255 G: 0 B: 0 A: 255	R: 222 G: 33 B: 0 A: 255
R: 0 G: 255 B: 0 A: 59	R: 126 G: 128 B: 0 A: 243	R: 253 G: 2 B: 0 A: 255	R: 255 G: 0 B: 0 A: 255	R: 255 G: 0 B: 0 A: 255
R: 0 G: 255 B: 0 A: 249	R: 0 G: 255 B: 0 A: 255	R: 77 G: 178 B: 0 A: 255	R: 242 G: 12 B: 0 A: 254	R: 255 G: 0 B: 0 A: 255
R: 0 G: 255 B: 0 A: 255	R: 0 G: 255 B: 0 A: 255	R: 0 G: 255 B: 0 A: 233	R: 119 G: 135 B: 0 A: 92	R: 255 G: 0 B: 0 A: 221
R: 0 G: 255 B: 0 A: 255	R: 0 G: 255 B: 0 A: 207	R: 0 G: 255 B: 0 A: 30	R: 0 G: 0 B: 0 A: 0	R: 255 G: 0 B: 0 A: 19

Encoding Bits

- RGB uses 24-bit color (i.e., $3 * 8 = 24$)
  - That's 16,777,216 possible colours
  - Our eyes cannot discern many colours beyond this
  - A challenge is display technology: monitors and projectors can`t reliably reproduce 16 million colours
- RGBA uses 32-bit colour
  - No additional colour, but offers support for transparency
  - This transparency channel is called `alpha`
  - The alpha channel also requires 8 bits
- Assuming `1 byte` == `8 bits`
- We can use this information to estimate the size of a bitmap:
  - 320x240x24 = 230,400 bytes
  - 640x480x32 = 1,228,800 bytes
  - 1024x768x32 = 3,145,728 bytes



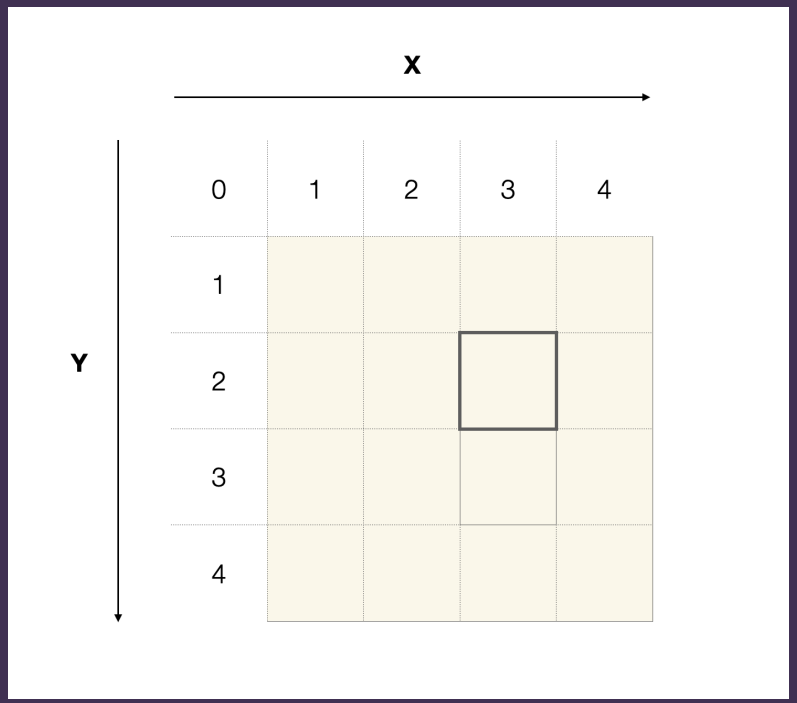
Manipulating Bitmap Pixels

- Images are controlled and manipulated using the `Bitmap` class in C#
- We can use `GetPixel` and `SetPixel` methods to both find and change pixels.

```
myImage.GetPixel(x, y);
myImage.SetPixel(x, y, newColor);
```

- Both methods use cartesian coordinates (x and y) to define the position of a specific pixel

Manipulating Bitmap Pixels




```
myImage.GetPixel(3, 2);
```

We can use the method to discover the ARGB values at the above position