

In order to manipulate digital media, like pictures, we need an understanding of

- The analog, real-world, physics of images and how we, as humans, perceive them.
- The digital representation, or encoding, of those images

The focus in this unit will be only on **two-dimensional** images. We leave three-dimensional image representation and manipulation for a later time!

Real world images and how we perceive them

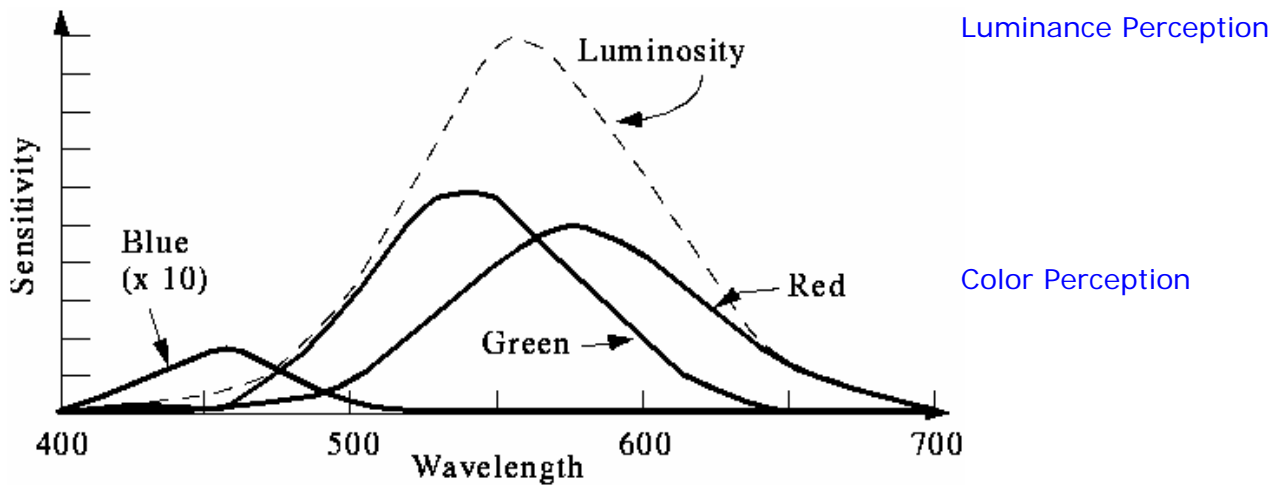
- Our perception of images is based on two complementary mechanisms in our brain.

*Our color perception through color sensors*

*Our luminance perception, which is our perception of the amount of light*

## ***The Real World***

- Color is **continuous**
  - Visible light is wavelengths between 370 and 730 nm
    - That's 0.00000037 and 0.00000073 meters
- But we **perceive** light with three color sensors that peak around 425 nm (blue), 550 nm (green), and 560 nm (red).
  - Our brain figures out which color is which by figuring out how much of each kind of sensor is responding
  - Dogs and other simpler animals have only two kinds of sensors
    - They *do* see color. Just *less* color.



Luminance perception allows us to perceive

- Borders of objects
- Motion
- Depth

Note that luminance is not the actual amount of light (which we can measure as the number of photons reflected off a particular color) but our perception of it.

In addition, humans have low visual acuity, which is the visual detail that we can perceive and distinguish. Other animals (such as the eagle) have much greater visual acuity and can perform more detailed differentiation in the objects they see.

Take-away: We perceive light different from how it actually is, and our perception is limited compared to the reality of visible light wavelengths

## *The Digital World*

- We digitize pictures into lots of little dots
  - Each dot has a **single color** associated with it
- Enough dots and it looks like a continuous whole to our eye. This **works** because of the real world aspects:
  - Our eye has limited resolution
  - Our background/depth *acuity* is particularly low

- Our background/depth *acuity* is particularly low
- Each picture element is referred to as a *pixel*

Picture element

We know that real-world pictures are two-dimensional. So we need a representation or encoding of a picture that takes this into account. Each individual dot, or pixel, that makes up a digital picture should have a location within this two dimensional space (i.e. an x coordinate and a y coordinate), along with the color associated with that pixel.

In the context of defining classes and having objects representing instances of those classes, we have already established the following requirements:

- At the highest level of abstraction, we need a class (that we will call **Picture**) that represents a two-dimensional coordinate system of Pixels.
  - It must have a width, and a height, and it has a relationship with some file (jpeg) from which it originates.
  - It must also encode this two-dimensional coordinate system of Pixels.
- Implied by the above, we need a class (that we will call **Pixel**) that represents a single picture element. An instance (or object) of class Pixel should:
  - Encode its associated Color.
  - Relate back to the Picture object that it is a part of, including being able to find out where the pixel is located in the picture. (i.e. What are the x and y coordinates of this pixel.)
- Implied by the above, we need a class (that is called **Color**) that represents and encodes a single visual color.

visual color.

So we have two things outstanding before we are able to complete our understanding of how to represent a digital picture

- How do we represent/encode a Color?
  - This is primarily concerned with how colors are encoded and how the devices that display the color (like our display) are best able to do so
- How do we represent a two dimensional coordinate system?
  - This is primarily concerned with facilities of the Java language to represent a large set of these Pixel objects (or any type) without having to create a variable name for each individual element in the coordinate system.

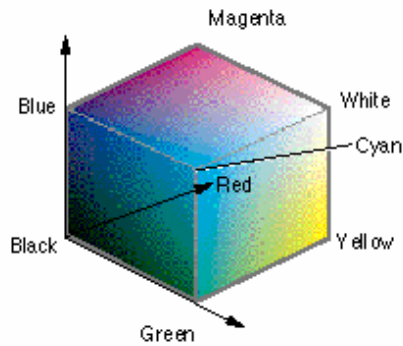
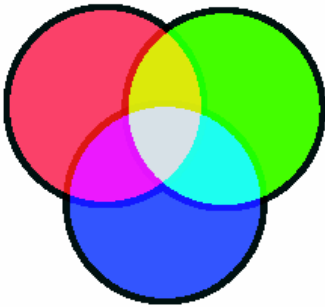
## RGB Model for Encoding Color

We know from our perception of color that humans have sensors for the amount of red, green, and blue that are the constituent parts of any color. We can make up any human-visible color by combining amounts of red, green, and blue light. This leads to the RGB method of encoding color:

- In RGB, each color has three component colors:
  - Amount of redness
  - Amount of greenness
  - Amount of blueness
- Each does appear as a separate dot on most devices, but our eye blends them.
- In most computer-based models of RGB, a single *byte*

(8 bits) is used for each

- So a complete RGB color is 24 bits, 8 bits of each



Q: If the amount of red, green, or blue is each represented by a byte, how many different values of each are there?

- Each component color (red, green, and blue) is encoded as a single byte
- Colors go from (0,0,0) to (255,255,255)
  - If all three components are the same, the color is in greyscale
    - (50,50,50) at (1, 1)
  - (0,0,0) (at position (0, 1) in example) is black
  - (255,255,255) is white

A Matrix of Colors

	0	1	2
0	100, 10, 5	5, 10, 100	255, 0, 0
1	0, 0, 0	50, 50, 50	0, 100, 0

To pick up the final topic: How do we represent a two-dimensional coordinate system.

A two dimensional coordinate system can be well represented by a table or matrix, just as we did above to represent a set of colors.

Just as when we were using our Shapes classes on a Canvas

object, the x dimension starts at 0 and **increases to the right**; while the y dimension starts at 0 and **increases down**. (note that this is different from Cartesian coordinate system in geometry where y **increases up**)

In the Java programming language, a matrix is known as a **two-dimensional array** (because there is also a programming language representation for a one-dimensional array)

In a two-dimensional array, we group together a collection of elements arranged in both a horizontal (column) and a vertical (row) sequence. For each dimension, we have an index that begins at 0. To refer to an element at column index i, and row index j, and whose two-dimensional array is call myArray, we use the syntax:

```
myArray[i][j]
```

Let the two-dimensional array of colors above be named colorExample. To refer to the green color, I would use

```
colorExample[2][1]
```

Note that this is an **expression**. The **type** of the expression is the same as the base type of the two-dimensional array ... in this example, the each element represents a **single Color**. The **value** is given in the example as (0, 100, 0) and would be a reference to an object that had been created to have that color value.

For now, the details of the encoding of a picture as a two-dimensional array of Pixel objects will be hidden by the **Picture class**. The Picture class is our blueprint for creating new objects or instances of pictures. When we create a new instance, we need to specify a source file, giving a filename from the filesystem to create a Picture object from a jpeg digital image.

Per many examples in the book:

```
String myFilename = "/Volumes/Student Personal Workspace/cs171/20_Tom.jpg";  
Picture myPicture = new Picture(myFilename);
```

where the particular filename may vary, and can be retrieved with a FileChooser.pickAFile() invocation (which is a method that returns a String) instead of a hard-coded String constant.

At this point, we understand the representation of digital pictures and can turn our focus to their manipulation.