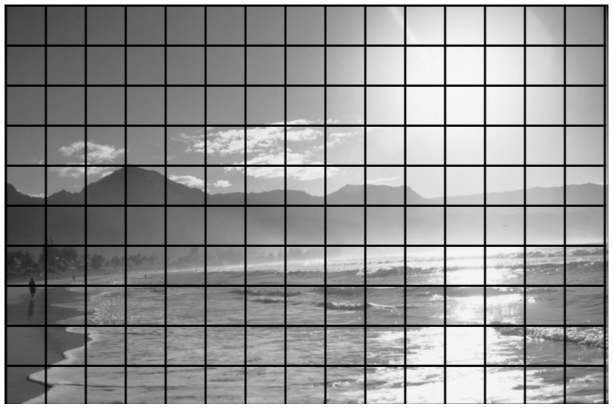
Begin with an analog image, for instance, this 35mm slide is roughly 1.5" by 1" in actual size.

Our goal is to make a digital version of it. In other words, we want to use numbers to represent this image.

Begin by dicing it into small rectangles, known as **pixels**. For each small rectangle we choose a **single numerical value** that best represents the range of intensities covered by that rectangular area. Because of this, the number of pixels you should use depends on what you want to eventually do with the digital image. For instance, if you want to create a small icon to be used on a web page, a small number of pixels is probably ok. If, on the other hand, you want to print the image and frame it on your wall, you would want to have a larger number of pixels.



To illustrate the process, let's use a rectangular array that is 15 pixels wide by 10 pixels high.

Note that the ratio of width to height in pixels (15:10 or 1.5:1) equals the ratio of the original image's width to height in inches (1.5":1.0"). This ratio is referred to as an image's **aspect ratio**.

In general, the ratio of width to height in pixels **does not** have to equal the image's physical aspect ratio. When they are equal, the pixels will be square. When they are not, the pixels are rectangular.

The number of pixels you choose to represent your digital image is called the digital image's **spatial resolution**. Spatial resolutions are typically written as "width x height" or 15 x10 in our example.

Because we could have chosen 1723 x 2222 or any other spatial resolution we wanted, you cannot (without further information) claim that an image with a particular spatial resolution has a particular size in the real world.

15 x 10 = 150 pixels
1723 x 2222 = 3,828,506 pixels

… but the original image is **still** 1.5" x 1.0"!

That's important enough to repeat. The same 1.5" x 1" slide could be digitized into tens, hundreds, thousands, millions or more pixels! **Spatial resolution and physical size are not inherently related**.

To connect physical size to spatial resolution, you need another piece of data. We'll call it **pixel density**, or the number of pixels per inch, or simply **ppi**. Sometimes this is called **dots per inch** or **dpi**, though the safest term to use for now is ppi.

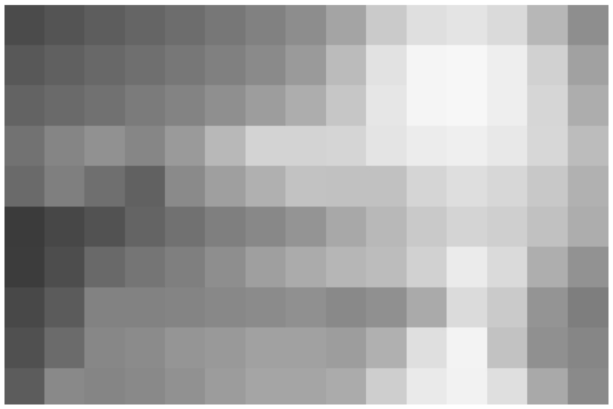
The pixel density is easy to compute if you have the physical size in inches and the spatial resolution. Just look at the units: If you want **pixels per inch**, just divide the number of pixels in one dimension by the number of inches in that same dimension. Pixels **per** inch.

15 pixels / 1.5 inches = 10 ppi (width)
10 pixels / 1.0 inches = 10 ppi (height)

1723 pixels / 1.5 inches = 1148.7 ppi (width)
2222 pixels / 1.0 inches = 2222 ppi (height)

A digital image's pixel density often hints at its reason for existence. Images with low ppi's are generally destined for the web or for display on computer monitors. High ppi's are needed for film images and hardcopy output where the digital image should look just like the original.

Back to our 15 x 10 digital image example. Once we have the grid we must calculate a single intensity value for each pixel and store this value as a number. Here's one way to imagine doing this. Look back to the 15 x 10 grid we constructed over our source image and average the intensity values that are visible within each grid square. A way to approximate this in your head is to look at the original image and squint until the values blur together. Determine appropriate pixel values from the blurry image.



So far we've only said that we're going to use numbers to represent the image. What numbers should we use? An easy choice is something called **normalized** color units. To use normalized units we assign the number 1.0 to pixels that are pure white and the number 0.0 to pixels that are pure black. For pixels in between white and black, use a number between 0 and 1 that accurately reflects the intensity. There. We have a digital image! The image on the right shows our beautiful 15 x 10 version of the original slide.

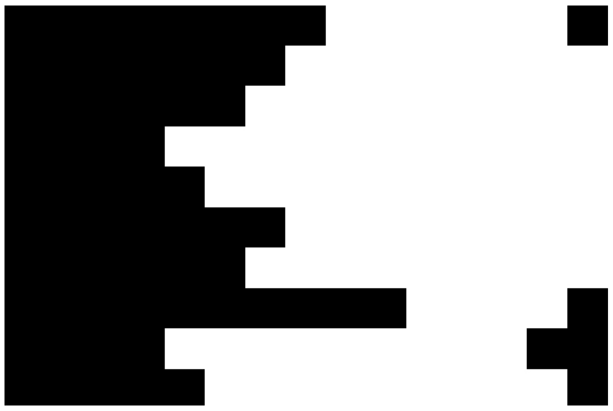
Clearly, 15 x 10 is not an adequate spatial resolution to accurately replicate our original beach image! We'll have to use more pixels if we want to capture more of the original detail. For the time being, though, we'll stick with this spatial resolution to make a few points.

Has anything we've done so far required a computer? No! We can make digital images with a pen and paper if we like. But all the fun happens when we start playing with digital images on computers. So we have to go one step further and learn how computers store numbers. Sadly, it's more complicated than you might think.

At their core, computers count using **bits**. A bit is like a light switch in that it can have only two states: **on** and **off**. It is standard practice to represent **on** with the number 1 and **off** with 0.

on or off
1 or 0

One bit only has two states and can therefore only represent two unique values. If we wanted to, we could assign the "on" state to the color white and the "off" state to the color black and store our beach image that way. Recall our use of normalized color units where we defined 1 as white and 0 as black. Great! When a bit has value 0 we can think of this as color value 0 (black) and when our bit is1 we can think of it as color value 1 (white). Any color values between 0 and 1 would get assigned to whichever intensity is closer since we're only allowing ourselves to store a 0 or a 1. This is what that image looks like.



Clearly, two intensity values aren't enough to capture the dynamic range of light present in the original slide. Consider an extreme case where the original image is a smooth gradient from black to white. Using one bit to store each number would force all gray values to either pure black or pure white, thereby throwing away a lot of the original information.

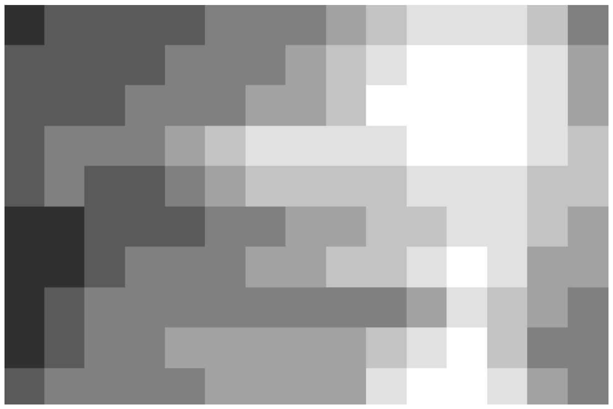
The trick to representing more values on a computer is to group bits together. For instance, **two bits** taken together can be in any one of **four** different states (see right). If each state is assigned to a value, two bits can store four different values. Four different values means four different intensities on the black-to-white intensity ramp. That's certainly better than two when it comes to replicating our original image.

	bit A	bit B
state 1:	0	0
state 2:	0	1
state 3:	1	0
state 4:	1	1

This is basic **base two** or **binary** counting. The number of unique settings of *n* bits taken as a group is 2^n .

number of unique values = 2^n
1 bit = 2^1 = 2 values
2 bits = 2^2 = 4 values
3 bits = 2^3 = 8 values
… and so on.

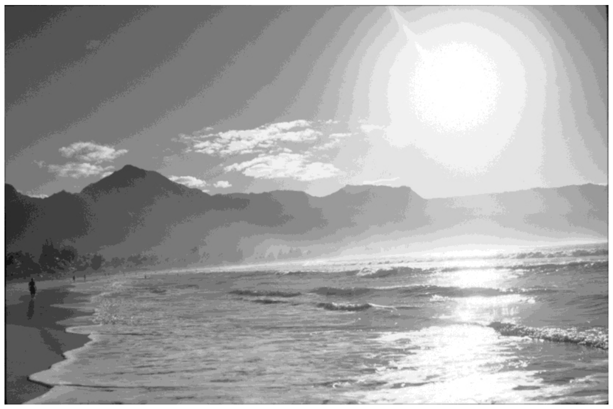
The **bit depth** of an image is defined to be the number of bits used to store the numerical value in each pixel. The single-bit image we saw above has a bit depth of one bit per pixel. This new image on the right **uses 3 bits per pixel**. The formula above ($2^3 = 8$) tells us that there are 8 unique states possible when we use 3 bits. This image was made by mapping those 8 states to 8 different intensity values spread between black and white.



Why the term bit **depth**? Spatial resolution determines the number of atomic units (pixels) that define an image's **width** and **height**. Bits are the atomic units that define the number of colors each pixel can possibly take on. Picture the digital image as a 3D array of width, height, and number of bits instead of just a 2D array of pixels. Then, **depth** is a sensible choice.

You can generally use as many bits per pixel as you would like. How many should you use?

For most purposes **8 bits per pixel** (2^8 = **256 unique values**) is enough to keep you from seeing what are known as **quantization artifacts.**



The image on the right has a spatial resolution of 817 x 539 (much higher than 15 x 10) but a bit depth of only 4 bits per pixel. 4 bits means that each pixel can take on only one of 16 different values between black and white. You can clearly see what are called **banding** artifacts as the bright areas around the sun are forced to take on one of these 16 values.

It is probably clear by now that digital images look more like their analog counterparts when the **spatial resolution and the bit depth are high**. The first image in the handout, for instance, is actually not a slide - it is a digital image (ooh, I'm so tricky). It has a spatial resolution of 817 x 539 pixels and has a bit depth of 8 bits per pixel. If you were fooled into thinking of it as a non-digital image then the choices of spatial resolution and bit depth were good ones.

817 x 539 = 440,363 pixels
8 bits per pixel = 2^8 = 256 different values

The reason that we can't simply use the biggest spatial resolution and bit depth possible when we make digital images is because **computer memory is finite**.

Each bit of information that is used for our image needs to be stored somewhere in the computer. Luckily, computer memory is measured in units that are closely related to bits.

A **byte** is a group of eight bits and can therefore represent 256 unique values. The bit depth of the first image of the handout is 1 byte per pixel.

1 byte = 8 bits

The memory required to store a particular image is equal to that image's size in bytes. The bit depth of the image determines how much memory each pixel requires, and the spatial resolution of the image determines the total number of pixels. To compute the total memory required to store an image you simply multiply the spatial resolution by the bit depth.

bit depth = memory per pixel
spatial resolution = number of pixels

bit depth x number of pixels = **total memory**

The first image on the handout has a spatial resolution of 817 x 539 and a bit depth of 8 bits per pixel. Determining the memory **footprint** of this image is straightforward. See the computation to the right to discover that the memory required to store that image is over 440,000 bytes! (As you'll learn, that's actually not so much)

817 x 539 = 440,363 pixels
8 bits per pixel = 1 byte per pixel

440,363 pixels x 1 byte/pixel = 440,363 bytes

Computer memory is so bountiful these days that it is hardly ever counted in bytes. Instead, larger units are used. A thousand bytes is called a **kilobyte** (**kb**), and a million bytes is called a **megabyte** (**Mb**).

1,000 bytes = 1 kilobyte
1,000,000 bytes = 1 megabyte
1,000 kilobytes = 1 megabyte

One generally converts the memory footprint to whichever byte units (kb or Mb) make for the easiest number to say/write/understand. Rounding up or down is just fine as long as you stay in the ballpark. As we've seen, the first image on the handout has a memory footprint of 440,363 bytes. Changing units, this value is also approximately **440 kb** or **0.44 Mb**. I think **440 kb** is the easiest in this case since you don't need to use a decimal.

440,363 bytes = 440.4 Kb = 0.44 Mb

Let's compute the memory required to store the version of the image that had the obvious banding artifacts. Recall, that image was the same spatial resolution but only used 4 bits per pixel.

440,363 pixels x 4 bits/pixel = 1,761,452 bits

Converting bits to bytes is easy since eight bits equals one byte. I encourage use of a calculator for conversions like this.

1,761,452 bits / 8 (bits/byte) = 220,181.5 bytes
220,181.5 bytes = 220 kb

Note that the memory footprint of the banded image is smaller. That should not be surprising. **There is always a trade-off between memory and image quality**.

220 kb for the banded image
440 kb for the good-looking image

(Do you understand why the banded image can be stored in exactly half the memory required to store the original image?)

Here's a thought experiment. Imagine that your computer has 64 Mb of RAM. RAM is Random Access Memory, the memory that your computer uses to store an image while it is running. If you choose a bit depth of 1 byte per pixel, how many image pixels could you store in your computer's RAM?

64 Mb / 1 (byte/pixel) = 64 Megapixels
or
64,000,000 byte / (1 byte/pix) = 64,000,000 pix

Just use the equation to compute memory in reverse. Be careful that you balance your units (note how I used "megapixels" to represent millions of pixels).

You should consider trying these conversions in your head as long as you are working with the same units. 64 million bytes of memory and one byte per pixel leads to 64 million pixels without too much effort, right?

You need to become fluent with the conversion between spatial resolution/bit depth and memory!

# of pixels x bytes per pixel = # of bytes
or
spatial resolution x bit depth = memory

Try to answer the following questions.

1) How many unique values can be represented with 5 bits?

2) How many unique values can be represented with 24 bits?

3) How many bits are in 20 Kb?

4) You have an image that is 250 x 400 pixels in spatial resolution. The image has a bit depth of 8 bits per pixel. What is the memory footprint of the image, in bytes?

5) You have a digital image that takes up 240 Kb. You know that the spatial resolution is 600 x 200. What is the bit depth (in bits or bytes)?

6) Your computer has 36 Mb of RAM. What is the spatial resolution of the largest square image you can store at 1 byte per pixel?