CS 116: Introduction to Digital Imaging **Spring 2001**

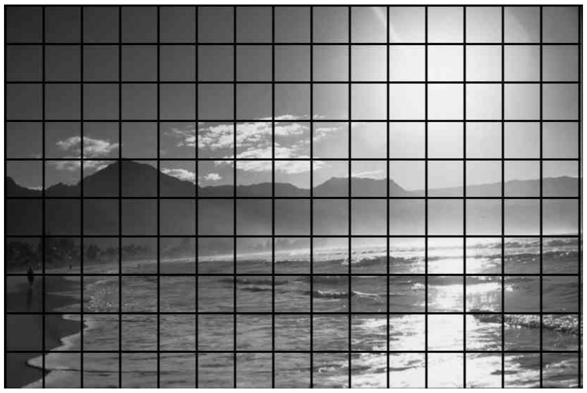
Digital Image Representation

Take an analog image, for instance, this 35mm slide image 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**. The number of pixels you should use depends on what your goal is for the image. To illustrate the process, I've chosen an array 15 pixels wide by 10 pixels high.

Note that the ratio of width to height in pixels (1.5) equals the ratio of the original image's width to height in inches (also known as the image's **aspect ratio**). This **doesn't** have to be the case. When it is, the pixels are **square**. In general they can be rectangular.



The number of pixels in width and height is the digital image's **spatial resolution**.

Because we could have chosen 1723 x 2222 or any other spatial resolution to suit our needs, 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"!

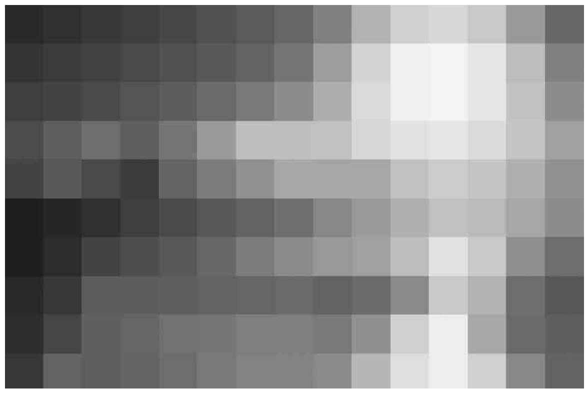
To connect size (in inches) to spatial resolution, you need another piece of data. We'll call it **pixel density**, or the number of pixels per inch, or simply **ppi**. Sometimes this is called **dots per inch** or **dpi**.

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

Look at the units. If you want **ppi**, just divide the number of pixels in one dimension by the number of inches in that same dimension. Pixels **per** inch.

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

We make this image digital by calculating a single intensity value for each pixel and storing this value as a number. Do this by averaging the values from the original image that lie within the pixel boundaries.



Assign the number 1.0 to pixels that are pure white, and assign 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.

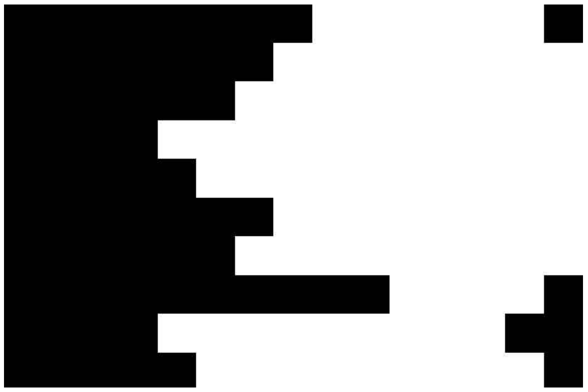
Clearly, 15 x 10 is not an adequate spatial resolution to accurately replicate our original beach image! For the time being, though, we'll stick with this spatial resolution to make a few points.

But we're not done yet. Now we want to store this digital image in a computer. That requires understanding how computers store numbers.

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
0 or 1

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. Any numbers between 0 and 1 would get assigned to whichever intensity is closer. This is what that image looks like.



Clearly, two intensity values isn't enough.

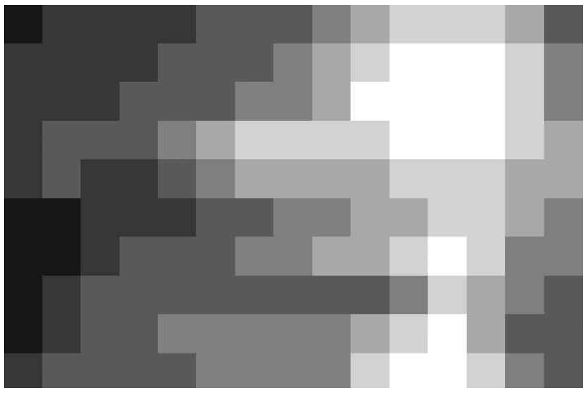
The trick to representing more values is to group bits together. For instance, **two bits** taken together can be in any one of **four** different states. If each state is assigned to a value, two bits can store four different values.

	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.

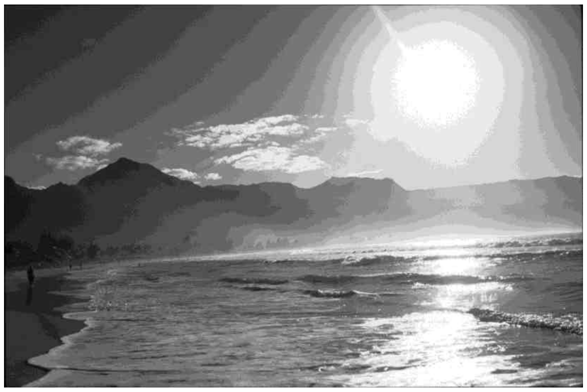
The **bit depth** of an image is the number of bits used to store the value in each pixel. The single-bit image we saw above has a bit depth of one bit per pixel. This image uses 3 bits (for 8 different values spread between black and white), and therefore has a bit depth of 3 bits per pixel.



Why 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**, or **256 unique values** between black and white, is enough to keep you from seeing what are known as **quantization artifacts.**



This image has a spatial resolution of 817 x 539 (much higher than 15 x 10) but a bit depth of only 4 bits per pixel. You can clearly see what are called **banding** artifacts as the intensity falloff from the sun is forced to take on one of 16 unique 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 817 x 539 pixels in spatial resolution and has a bit depth of 8 bits per pixel.

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.

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.

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.

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 bytes are hardly ever referred to as bytes. 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. In this case, the image is also approximately **440 kb** and **0.44 mb**. I think **440 kb** is the easier number.

440,363 bytes = 440.4 Kb = 0.44 Mb

Let's consider our version of the image that had the obvious banding artifacts. 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.

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. **There is always a trade-off between memory and quality**.

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

(Does it also make sense that the banded image has half the memory footprint of the original?)

Here's a thought experiment. Imagine that your computer has 64 Mb of RAM. If you choose a bit depth of 1 byte per pixel, how many image pixels could you store in your computer?

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

# of pixels x bytes per pixel = # of bytes

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

2) How many 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?

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?