Representing and Compressing Images

In this lesson we continue the exploration of bits and binary numbers that we began in Unit 2. In this case we learn how to use bits, 1s and 0s, to represent images. The objectives of this lesson are:

* to understand that all digital data, including electronic documents, are composed of bits,
* to understand that representing an image digitally is another example of abstraction at work.

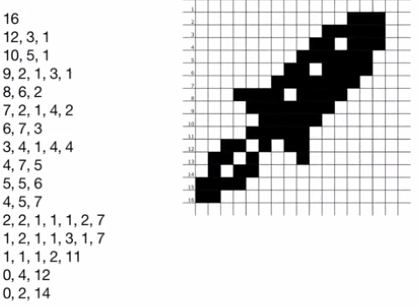
# CS Unplugged Video: Representing Images

Watch this [CS Unplugged video](https://www.youtube.com/watch?v=uaV2RuAJTjQ) on image compression for a quick look at how images are represented and compressed.

# Run-Length Encoding (RLE)

([Slide Presentation](https://docs.google.com/presentation/d/1MmKmh7fJKCCfwGzA238VPXZYLkCZvIIUQZ9L-RIDEYQ))

The image representation technique demonstrated in the video is known as run-length encoding (RLE) and it is an [image compression](http://en.wikipedia.org/wiki/Image_compression) technique.



RLE is especially useful fax machines and other applications where we typically have lots of  ***white space*** and it is used in certain image formats, such as for BMP (***bitmap***) files.

# Lossless vs. Lossy Compression





0,7

0,1,6

0,1,6

0,5,2

0,1,6

0,1,6

0,1,6

0,1,6



1,1,3,1,1

1,1,3,1,1

1,1,3,1,1

1,5,1

1,5,1

1,1,3,1,1

1,1,3,1,1

1,1,3,1,1

RLE is an example of a ***lossless*** compression technique, which means no data are lost when an image is compressed using RLE. When you *inflate* a compressed RLE image -- that is, when you uncompress it to get back the original image -- you get back the exact same image you started with. Lossless compression is good for important images such as medical and archive images.

Other compression techniques, such as JPEG (which stands for Joint Photographic Experts Group), are called ***lossy*** techniques because certain information is lost during compression. That is, you don’t necessarily get back the original image when you inflate the compressed file. Lossy techniques are suitable for images taken with digital cameras, where the loss of data can’t be perceived by the human eye.

# Compressing an Image with RLE

How good is the RLE compression in the case of our spaceship?

The spaceship was drawn on a 16 × 16 grid. That's 256 pixels of data.

When the image is represented using RLE -- i.e., by numbers and commas -- it requires 62 numbers. The largest number is 16 in this case, but for larger images, we might need bigger numbers. Let’s say we use 8 bit numbers to represent the runs. This will let us represent runs of up to 255 pixels. (NOTE: In 8 bits you can represent 28 = 256 numbers.) So our compressed spaceship file will require 62 × 8 = 496 bits.

# Compression

Is that good compression? Well, it depends on how many bits were used to represent each pixel in the the original image.

# Monochrome ImagesNeighborhood_watch_bw.png

Given that these are black and white pixels, we could use 1 bit to represent each pixel by letting binary 0 be a black pixel and binary 1 be a white pixel. In the “old days”, computer displays actually used 1-bit monochrome displays. Here’s an example of a 1-bit monochrome image. This used to be called a bitmap, although today that term is used more broadly to mean an image file that uses more than 1 bit per pixel. 

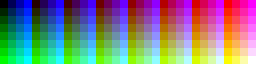
If we use the monochrome technique with our spaceship, then it would look like the image on the left and it would require only 256 bits of storage.

Run-length encoding wouldn’t be very useful in this case. In fact, it would increase the amount of memory needed to store the file to 496 bits. That’s almost twice as many bits. So RLE compression doesn’t help us in this case -- it doesn’t actually compress anything.

# 8-Bit Color Images

Of course, most pictures aren’t black and white. When colors are used we require more bits to represent the different colors. Suppose we use 8 bits to represent each pixel.

Using 8 bits for each pixel would let us represent 28 = 256 different colors. (NOTE: In 1 bit we can represent 21 = 2 coIors. In 2 bits, we could represent 22 = 4 colors. In 3 bits we could represent 23 = 8 colors.) For example, here’s an example of a color palette that is based on an [8-bit color representation](http://en.wikipedia.org/wiki/8-bit_color).



In this particular scheme, a color is made by mixing Red, Green, and Blue (RGB). Each pixel is represented by using 3 bits for red, 3 bits for green and 2 bits for blue in the following scheme:

Bit 0 1 2 3 4 5 6 7  
Data R R R G G G B B

In other words, going left-to-right, the first 3 bits are devoted to Red, the next three to Green, and the last 2 bits are for Blue. Given this scheme, black would be represented as 0000 0000 (that’s 0 in decimal) and white would be 1111 1111 (that’s 255 in decimal).

# Modified Run-Length Coding

256 colors is not very many colors by modern standards, but it’s okay for the sake of this example. Our spaceship image has only two colors, black and white, which we represent by 00000000 and 11111111, or 0 and 255 in decimal.

However, we have a problem: the above run-length coding scheme won’t work on color images -- it only works for two colors. For example, look at how the first two rows of the image are coded:

16

12, 3, 1

The assumption is that the first number always represents a run of white pixels, the second number a run of black pixels, the third is white pixels, and so on. This won’t work if have more than two colors:

**If we have more than two colors, our scheme has to include the color of**

**the pixels as well as the number of pixels in the run.**

A modified scheme could use **two numbers to represent each run** -- the number of the color and the number of consecutive pixels that have that color. For example, the first two rows could be coded as:

255,16 -- 16 pixels of white (255)

255,12, 0,3, 255,1 -- 12 pixels of white (255), 3 pixels of black (0), 1 pixel of white (255)

This new compression scheme needs twice as many 8-bit numbers to compress the 8-bit spaceship image. That’s 128 × 8 or 992 bits.

Is that good compression? Well, let’s see. How many bits are required to represent the spaceship now that we’re using 8-bits for each pixel? The answer is 256 × 8 or 2048. That’s a very good result -- in fact, around 56%. We’ve reduced the amount of data by more than half.

# 24-Bit Color Images

Using 8-bits for colored images is an old fashioned technique. Today’s computer screens use 24 bits to represent images, which gives 16,777,216 (224) different colors -- way more than our eyes can distinguish.

If we carry this example one step further, to use modern image representation schemes, the compression is more impressive. When 24 bits are used to represent each color, then 8 bits are used for Red, 8 bits for Green, and 8 bits for Blue. Black would be still be 0 (000000000000000000000000 represented as 24 bits) but white would be 1 (111111111111111111111111 represented as 24 bits). But 24-bit numbers are too big to write in binary, so let’s write them in decimal. Then 0 would be 0,0,0 and white would be 255,255,255.

Our representation in this model would be (Red, Blue, Green, Run Length) for every run of pixels, or (r,g,b,L). Given this representation, the first two rows of our spaceship would now be:

255,255,255,16

255,255,255,12, 0,0,0,3, 255,255,255,1

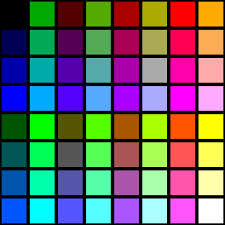
This requires a lot more characters. How many? Well now for each run we need 4 8-bit numbers, 3 to represent the color and 1 to represent the length of the run. That’s a total of 62 × 4 = 248 8-bit numbers for a total of 1984 bits.

Is it good compression? Well, how big would our uncompressed image be? Its 256 pixels times 24 bits per pixel. That’s 6,144 pixels. So in this case our compression is almost 70%. Quite good!

As these examples illustrate, the amount of compression can vary depending on how many bits are used to represent each pixel in the image.

But it depends even more on one other factor -- namely how many different colors are used in the actual image. For our black and white spaceship there were only 2 colors, so there were relatively few color changes and therefore lots of long runs. If this were a colored spaceship, there would be many color changes and therefore fewer long runs. So we would get much less compression.

To see this more clearly, consider the following two images:



In both of these cases, there are frequent transitions from one color to the next. So there would be lots of numbers in our compressed file. For example, in the B/W checkerboard, the rows would look something like:

0,0,0,1, 255,255,255,1, 0,0,0,1, 255,255,255,1, 0,0,0,1

255,255,255,1, 0,0,0,1, 255,255,255,1, 0,0,0,1, 255,255,255,1

With so many numbers, the compressed file would actually be larger than the original image.

# Interpreting Binary Sequences

As this example showed, bits (binary digits) can be used to represent the pixels in images. In a previous lesson we saw that they can represent numbers. Let’s leave this topic by looking at one more example of binary representation -- i.e., character data.

# Representing Characters (ASCII)

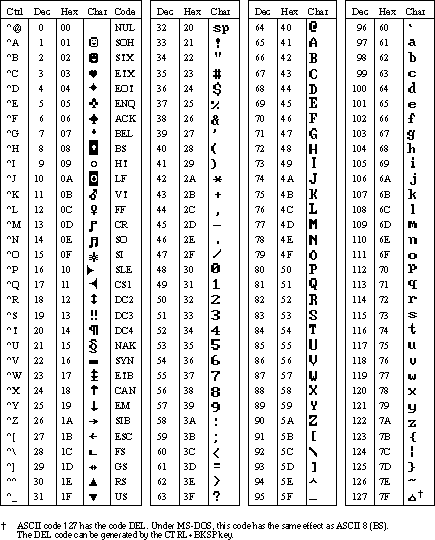
ASCII (American Standard Code for Information Interchange) is a 7-bit code for representing character data. It was introduced in the 1960s for use on some of the first computers.

As the following table shows, in the ASCII code, the letters ‘A’ through ‘Z’ are coded as decimal 65 through 90, and ‘a’ through ‘z’ are coded as decimal 97 to 122. Some of the low codes, 0 through 31 are for non-printing control characters. For example, the space character is ASCII 20.

What does this bit sequence represent: 01000001 ? Well it could represent the decimal number 65 or it could represent the letter ‘A’ or it could represent the color pink.png.

It depends on the what it’s being used for - the context it is being used in. Is it from an image file? A text message? A spreadsheet?

The point is: *The interpretation of a binary sequence depends on how it is used.*



# Activities

1. Write your own message in binary that someone else could decode. Trade messages in class and decode each others.
2. Web pages use the *HyperText Markup Language (HTML)* to encode web pages. Here’s a [color mixer app](http://www.w3schools.com/tags/ref_colormixer.asp) that lets you explore the different colors that are used in Web pages. How many bits are used to represent the colors? Figure out the Hex code for pure red? pure green? pure blue?
3. With a partner, research another image type (e.g. [GIF](http://en.wikipedia.org/wiki/Graphics_Interchange_Format), [PNG](http://en.wikipedia.org/wiki/Portable_Network_Graphics), TIFF, etc.) and compare and contrast the data needed to store information about the images. Include what type of compression is used. Share your findings with the class.
4. If you have a digital camera or a smartphone or tablet, find out what image representation scheme it uses? How come all images are not the same size?
5. ASCII is one type of character code, but 128 characters is not enough for today’s computers, which can represent Chinese, Hindi, and scripts from many other languages. Today’s computers use a system called [Unicode](http://en.wikipedia.org/wiki/Unicode), which has more than 100,000 different characters and covers more than 100 different scripts (languages). Use this [Unicode converter](http://pages.ucsd.edu/~dkjordan/resources/unicodemaker.html) to convert these Chinese characters to their hexadecimal Unicode values: 国话. Convert these Greek letters: οι. Convert these Russian letters: Я ю.

# Reflection: For Your Portfolio

1. Decode this message by converting it from binary to [ASCII](http://en.wikipedia.org/wiki/ASCII):  
   1000001 1110000 1110000 0100000 1001001 1101110 1110110 1100101 1101110 1110100 1101111 1110010 0100000 1010010 1001111 1000011 1001011 1010011 0100001
2. Describe what it means to say that [JPEG](http://en.wikipedia.org/wiki/JPEG) is a *lossy* compression technique and whether or not it affects the quality of camera pictures.
3. Give a specific example of a binary sequence that can represent more than one type of data -- e.g., a number, a color, a character -- and describe how to interpret its different values.