Math in the Real World: Digital Imaging

CEMC

The Connection

We're used to uploading our photos on services like Facebook, Twitter, and Instagram.

The Connection

We're used to uploading our photos on services like Facebook, Twitter, and Instagram.

Every photo that goes up on these platforms takes up space on a server somewhere, and it's usually not necessary to have your photos be displayed at the highest resolution that your camera outputs.

The Connection

We're used to uploading our photos on services like Facebook, Twitter, and Instagram.

Every photo that goes up on these platforms takes up space on a server somewhere, and it's usually not necessary to have your photos be displayed at the highest resolution that your camera outputs.

Your photos, which are stored as numerical information, are *compressed* when they are uploaded.

The Connection

We're used to uploading our photos on services like Facebook, Twitter, and Instagram.

Every photo that goes up on these platforms takes up space on a server somewhere, and it's usually not necessary to have your photos be displayed at the highest resolution that your camera outputs.

Your photos, which are stored as numerical information, are *compressed* when they are uploaded.

We will start by discussing how these images are stored on a computer and then we will move on to how this data is transformed for different purposes.

The Binary Number System

The number system that we use on a daily basis is known as *base 10* or *decimal*. However, computers use the *binary* or *base 2* system.

The Binary Number System

The number system that we use on a daily basis is known as *base 10* or *decimal*. However, computers use the *binary* or *base 2* system.
Let's take for example the number 473. Here's how we would break this number down in both systems:

Decimal

$$473 = 4(100) + 7(10) + 3(1)$$
$$473 = 4(10^2) + 7(10^1) + 3(10^0)$$
$$473 = (473)_{10}$$

The Binary Number System

The number system that we use on a daily basis is known as *base 10* or *decimal*. However, computers use the *binary* or *base 2* system.
Let's take for example the number 473. Here's how we would break this number down in both systems:

Decimal

$$473 = 4(100) + 7(10) + 3(1)$$
$$473 = 4(10^2) + 7(10^1) + 3(10^0)$$
$$473 = (473)_{10}$$

Binary

$$473 = 1(256) + 1(128) + 1(64) + 0(32) + 1(16) + 1(8) + 0(4) + 0(2) + 1(1)$$
$$473 = 1(2^8) + 1(2^7) + 1(2^6) + 0(2^5) + 1(2^4) + 1(2^3) + 0(2^2) + 0(2^1) + 1(2^0)$$
$$473 = (111011001)_2$$

The Binary Number System

The number system that we use on a daily basis is known as *base 10* or *decimal*. However, computers use the *binary* or *base 2* system.
Let's take for example the number 473. Here's how we would break this number down in both systems:

Decimal

$$473 = 4(100) + 7(10) + 3(1)$$
$$473 = 4(10^2) + 7(10^1) + 3(10^0)$$
$$473 = (473)_{10}$$

Binary

$$473 = 1(256) + 1(128) + 1(64) + 0(32) + 1(16) + 1(8) + 0(4) + 0(2) + 1(1)$$
$$473 = 1(2^8) + 1(2^7) + 1(2^6) + 0(2^5) + 1(2^4) + 1(2^3) + 0(2^2) + 0(2^1) + 1(2^0)$$
$$473 = (111011001)_2$$

As you can see, binary gives us a string of 1's and 0's.

The Binary Number System

The number system that we use on a daily basis is known as *base 10* or *decimal*. However, computers use the *binary* or *base 2* system.
Let's take for example the number 473. Here's how we would break this number down in both systems:

Decimal

$$473 = 4(100) + 7(10) + 3(1)$$
$$473 = 4(10^2) + 7(10^1) + 3(10^0)$$
$$473 = (473)_{10}$$

Binary

$$473 = 1(256) + 1(128) + 1(64) + 0(32) + 1(16) + 1(8) + 0(4) + 0(2) + 1(1)$$
$$473 = 1(2^8) + 1(2^7) + 1(2^6) + 0(2^5) + 1(2^4) + 1(2^3) + 0(2^2) + 0(2^1) + 1(2^0)$$
$$473 = (111011001)_2$$

As you can see, binary gives us a string of 1's and 0's.
Try converting 255 to binary! Make sure you start by finding the largest power of 2 that is less than 255.

The Colour Scales

Every colour that you see on your screen has a distinct keycode.

The Colour Scales

Every colour that you see on your screen has a distinct keycode.

The simplest colour scale is *greyscale*. Technically, this is a shade scale that ranges from pure black to pure white. Black is 0 and white is 255.

The Colour Scales

Every colour that you see on your screen has a distinct keycode.

The simplest colour scale is *greyscale*. Technically, this is a shade scale that ranges from pure black to pure white. Black is 0 and white is 255.

Now you can see why it was important to find out 255 in binary (it was 11111111)! Using this largest colour value, we consider every keycode to be eight digits long. This means that black is 00000000.

The Colour Scales

Every colour that you see on your screen has a distinct keycode.

The simplest colour scale is *greyscale*. Technically, this is a shade scale that ranges from pure black to pure white. Black is 0 and white is 255.

Now you can see why it was important to find out 255 in binary (it was 11111111)! Using this largest colour value, we consider every keycode to be eight digits long. This means that black is 00000000.

Image source: http://www.whydomath.org/node/wavlets/images/grayrange.gif

The Colour Scales

When we rewatch our favourite *Harry Potter* movie, we watch it in full colour!

The Colour Scales

When we rewatch our favourite *Harry Potter* movie, we watch it in full colour! Whereas greyscale is based on one coordinate (a single number), the full colour spectrum is based on three coordinates: red, green, and blue.

The Colour Scales

When we rewatch our favourite *Harry Potter* movie, we watch it in full colour! Whereas greyscale is based on one coordinate (a single number), the full colour spectrum is based on three coordinates: red, green, and blue.

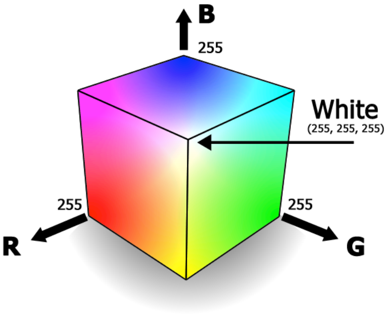


Image source: https://www.medialooks.com/mformats/docs/images/CK_color_cube.png

The Colour Scales

For example, the University of Waterloo's official gold colour is (255, 213, 79):

The Colour Scales

For example, the University of Waterloo's official gold colour is (255, 213, 79):

Try it out:

1. What is the binary coordinate representation of Waterloo Gold?
2. What would black (on the RGB scale) be in decimal?
3. What would black (on the RGB scale) be in binary?

Storing Information: Pixels and Bits

All digital images are made up of tiny pixels. Your screen is lit up with these tiny squares, each with different colour values.

Storing Information: Pixels and Bits

All digital images are made up of tiny pixels. Your screen is lit up with these tiny squares, each with different colour values.

The more pixels per inch (PPI) there are (for the same image size in different resolutions), the higher quality your image is. Your image looks less grainy, as you are able to encode more detailed colours with your smaller pixels in the same area.

Storing Information: Pixels and Bits

All digital images are made up of tiny pixels. Your screen is lit up with these tiny squares, each with different colour values.

The more pixels per inch (PPI) there are (for the same image size in different resolutions), the higher quality your image is. Your image looks less grainy, as you are able to encode more detailed colours with your smaller pixels in the same area.

Take a look at two examples of PPI below:

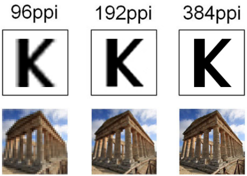


Image source: http://s01.shiftdelete.net/img/content/16-10/04/ppi_karsilastirmasi.jpg

Storing Information: Pixels and Bits

We now have crisp colours in High Definition (HD) 1080p format, and the more recently introduced Ultra HD (UHD) 4K 2160p format.

Storing Information: Pixels and Bits

We now have crisp colours in High Definition (HD) 1080p format, and the more recently introduced Ultra HD (UHD) 4K 2160p format.

These displays pack 1080 pixels and 2160 pixels respectively in the vertical direction of the display.

Storing Information: Pixels and Bits

We now have crisp colours in High Definition (HD) 1080p format, and the more recently introduced Ultra HD (UHD) 4K 2160p format.

These displays pack 1080 pixels and 2160 pixels respectively in the vertical direction of the display.

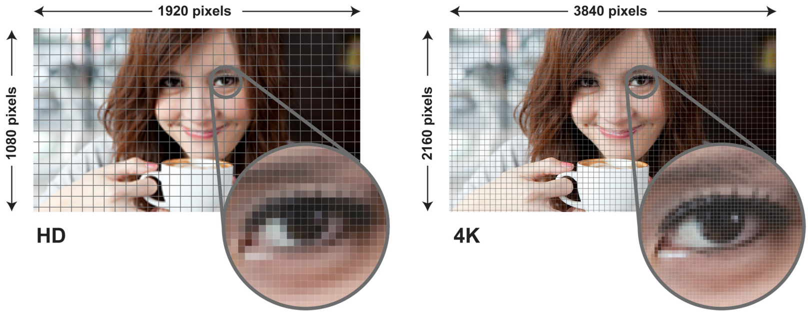


Image source: http://graphics.secondipity.com/gr/images/nw/2013X900A1.jpg

Storing Information: Pixels and Bits

High quality images do come at a cost: large file size.

Storing Information: Pixels and Bits

High quality images do come at a cost: large file size.

Every pixel stores one colour using the RGB scale discussed in the previous section.

Storing Information: Pixels and Bits

High quality images do come at a cost: large file size.

Every pixel stores one colour using the RGB scale discussed in the previous section.

These colour keycodes are stored in binary format, meaning each coordinate is eight digits long.

Storing Information: Pixels and Bits

High quality images do come at a cost: large file size.

Every pixel stores one colour using the RGB scale discussed in the previous section.

These colour keycodes are stored in binary format, meaning each coordinate is eight digits long.

Each digit in a binary sequence is considered to be one *bit*. One bit is the base unit of storage on a computer.

Storing Information: Pixels and Bits

High quality images do come at a cost: large file size.

Every pixel stores one colour using the RGB scale discussed in the previous section.

These colour keycodes are stored in binary format, meaning each coordinate is eight digits long.

Each digit in a binary sequence is considered to be one *bit*. One bit is the base unit of storage on a computer.

Eight bits make up one *byte*, and $2^{10} = 1024$ bytes make up one kilobyte (KB).

Storing Information: Pixels and Bits

High quality images do come at a cost: large file size.

Every pixel stores one colour using the RGB scale discussed in the previous section.

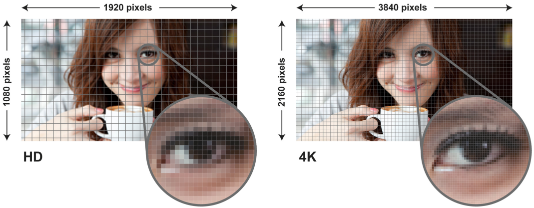
These colour keycodes are stored in binary format, meaning each coordinate is eight digits long.

Each digit in a binary sequence is considered to be one *bit*. One bit is the base unit of storage on a computer.

Eight bits make up one *byte*, and $2^{10} = 1024$ bytes make up one kilobyte (KB).

Furthermore, $(2^{10})^2 = 1024^2 = 1048576$ bytes make up one megabyte (MB). The megabyte, along with the gigabyte (GB) are probably the most familiar storage units.

Storing Information: Pixels and Bits



Try it out:

1. How many bits are in one black and white (greyscale) pixel?
2. How many bits are in one colour (RGB) pixel?
3. How many bits are in the HD *colour* image pictured above?
4. How many bits are in the 4K *colour* image pictured above?
5. What is the file size of the HD image in MB?
6. What is the file size of the 4K image in MB?
7. What percentage of the size of the HD file is the size of the 4K file?
8. How many bytes do you think there are in 1 GB?

A Component of Compression: Huffman Coding

When we upload our pictures onto social media, they get compressed so they take up less space on the social media company's servers.

A Component of Compression: Huffman Coding

When we upload our pictures onto social media, they get compressed so they take up less space on the social media company's servers.

This is why the quality of your photos decreases when you upload them!

A Component of Compression: Huffman Coding

When we upload our pictures onto social media, they get compressed so they take up less space on the social media company's servers.

This is why the quality of your photos decreases when you upload them!

One simplistic method of data compression that is occasionally utilized: Huffman Coding.

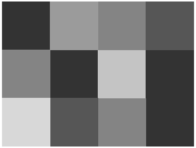
A Component of Compression: Huffman Coding

This method can be computerized to deal with large amounts of data, but for workability, we will go back to greyscale images.

A Component of Compression: Huffman Coding

This method can be computerized to deal with large amounts of data, but for workability, we will go back to greyscale images.

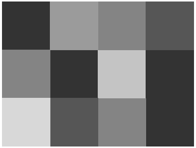
Examine the section of pixels captured below, along with its numeric greyscale values:



51	155	132	86
132	51	196	51
216	86	132	51

A Component of Compression: Huffman Coding

This method can be computerized to deal with large amounts of data, but for workability, we will go back to greyscale images.
Examine the section of pixels captured below, along with its numeric greyscale values:

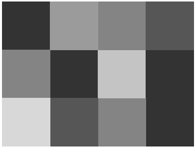


51	155	132	86
132	51	196	51
216	86	132	51

To start our Huffman Coding, we need to create a frequency table:

A Component of Compression: Huffman Coding

This method can be computerized to deal with large amounts of data, but for workability, we will go back to greyscale images.
Examine the section of pixels captured below, along with its numeric greyscale values:



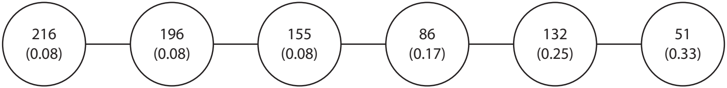
51	155	132	86
132	51	196	51
216	86	132	51

To start our Huffman Coding, we need to create a frequency table:

Value	Frequency	Relative Frequency
51	4	$4/12 = 0.33$
132	3	$3/12 = 0.25$
86	2	$2/12 = 0.17$
155	1	$1/12 = 0.08$
196	1	$1/12 = 0.08$
216	1	$1/12 = 0.08$

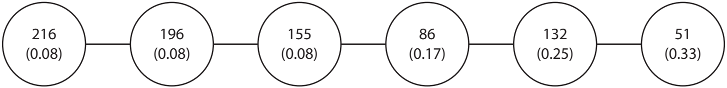
A Component of Compression: Huffman Coding

Next, we sort the values by relative frequency (smallest to largest):

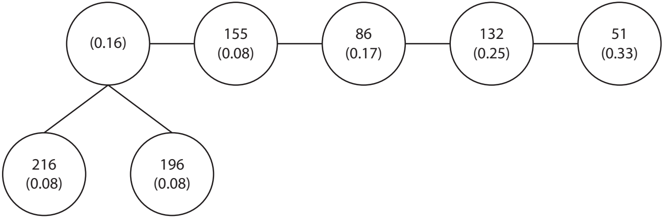


A Component of Compression: Huffman Coding

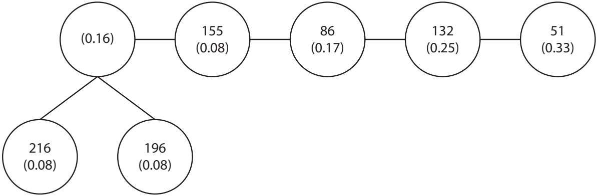
Next, we sort the values by relative frequency (smallest to largest):



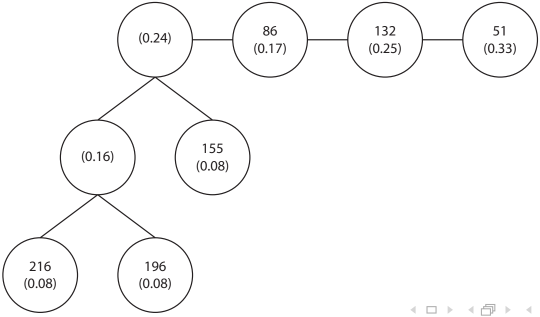
Now, add the relative frequences of the two leftmost nodes and create a new node with two children:



A Component of Compression: Huffman Coding

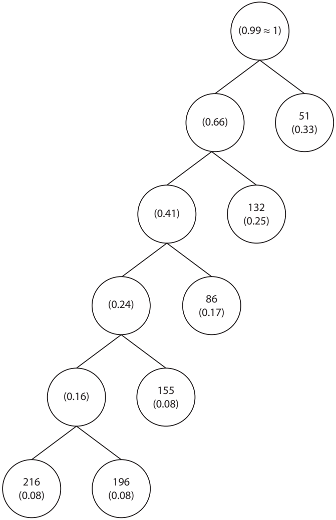


Repeat the previous step:



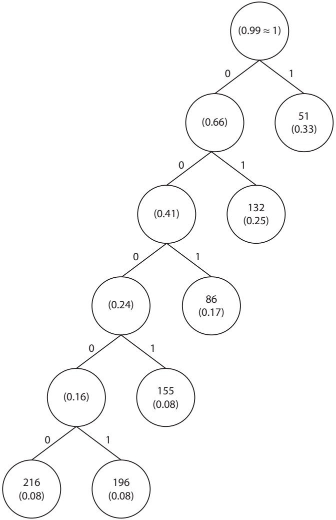
A Component of Compression: Huffman Coding

Keep repeating until there is a single node of frequency 1 at the top:



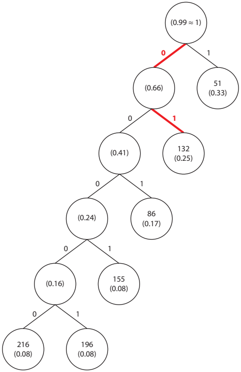
A Component of Compression: Huffman Coding

Lastly, label the branches connecting nodes. Left branches get a 0, right branches get a 1.



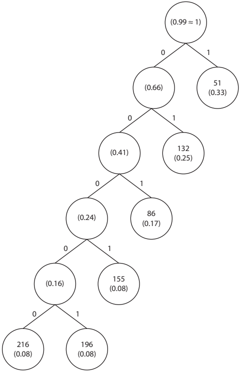
A Component of Compression: Huffman Coding

Originally, the value of 132 was 10000100. From our Huffman Coding, the new value of 132 is 01 (see the path highlighted in red below). We have saved 6 bits for every occurence of 132.



A Component of Compression: Huffman Coding

51	155	132	86
132	51	196	51
216	86	132	51



Try it out:

1. How many bits were originally used to store the section of greyscale pixels?
2. Create a table listing the original pixel values (in decimal) and their new Huffman values.
3. How many bits are used to store the Huffman coded section of pixels?
4. What percentage of the size of the original file is the size of the Huffman file?
5. What made Huffman Coding so effective with this specific section of pixels? In what situation would Huffman Coding not be as effective?
6. Create the Huffman tree for the frequency table below and list the new Huffman values.

Value	Frequency	Relative Frequency
115	5	$5/10 = 0.5$
97	3	$3/10 = 0.3$
102	1	$1/10 = 0.1$
114	1	$1/10 = 0.1$

Transforming Images with Functions

We manipulate digital images in different ways. Sometimes we want to flip or rotate our image, other times we want to brighten it up.

Transforming Images with Functions

We manipulate digital images in different ways. Sometimes we want to flip or rotate our image, other times we want to brighten it up.

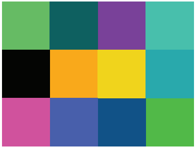
We're going to take a look at a basic transformation method involving function notation.

Transforming Images with Functions

We manipulate digital images in different ways. Sometimes we want to flip or rotate our image, other times we want to brighten it up.

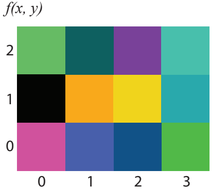
We're going to take a look at a basic transformation method involving function notation.

Once again, let's examine a section of pixels:



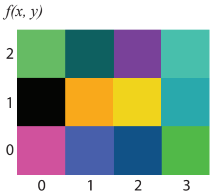
Transforming Images with Functions

Now, we're going to label the pixels with coordinates as follows:



Transforming Images with Functions

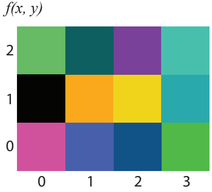
Now, we're going to label the pixels with coordinates as follows:



We can use function notation to represent our grid of pixels. The function $f(x, y)$ calls on the pixel located at (x, y) and gives us the RGB colour code for that pixel.

Transforming Images with Functions

Now, we're going to label the pixels with coordinates as follows:

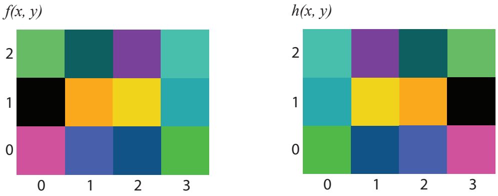


We can use function notation to represent our grid of pixels. The function $f(x, y)$ calls on the pixel located at (x, y) and gives us the RGB colour code for that pixel.

For example, $f(0, 1) = (0, 0, 0)$ which we know is black. We can use this notation to manipulate the pixels.

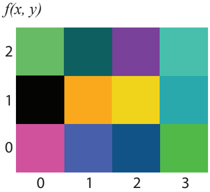
Transforming Images with Functions

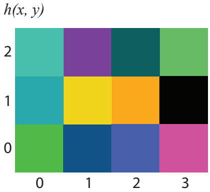
Let's flip/mirror the section of pixels horizontally. We will call the function that represents our new section of pixels $h(x, y)$.



Transforming Images with Functions

Let's flip/mirror the section of pixels horizontally. We will call the function that represents our new section of pixels $h(x, y)$.

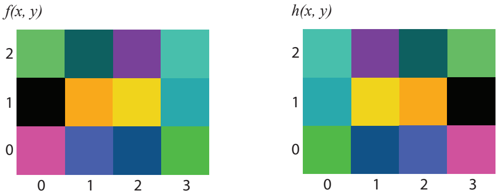




Choosing a few points as examples, we see that $h(0, 0) = f(3, 0)$, $h(1, 2) = f(2, 2)$, and $h(3, 2) = f(0, 2)$.

Transforming Images with Functions

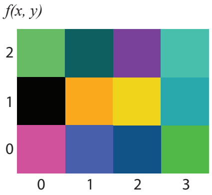
Let's flip/mirror the section of pixels horizontally. We will call the function that represents our new section of pixels $h(x, y)$.

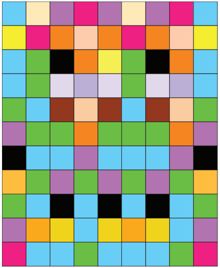


Choosing a few points as examples, we see that $h(0, 0) = f(3, 0)$, $h(1, 2) = f(2, 2)$, and $h(3, 2) = f(0, 2)$.

From these sample points, we can see that the horizontally flipped image is $h(x, y) = f(3 - x, y)$.

Transforming Images with Functions





Try it out:

1. What function $v(x, y)$ flips/mirrors the original section of pixels $f(x, y)$ vertically?
2. What function $r(x, y)$ rotates $f(x, y)$ 180°? (*Hint: Don't think of it as a rotation.*)
3. The 11×9 grid of pixels $a(x, y)$ is shown to the left. Draw $b(x, y) = a(2x, 2y)$ with the restrictions $x \leq 4, y \leq 5$. (*Hint 1: $b(x, y)$ should be a 6×5 grid. Hint 2: You can immediately eliminate the 69 squares that don't show up in $b(x, y)$ if you think about what is happening with the function.*)

Thank You!

Visit **cemc.uwaterloo.ca** for great mathematics resources!