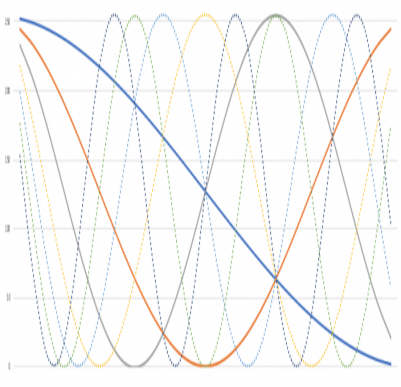


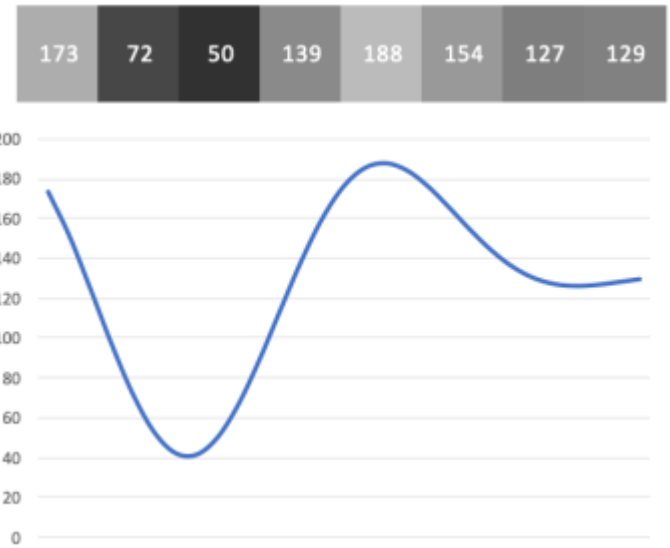
Beyond averages: DCT graphics

Description

default watermark

The visual field is full of smooth curves. Shadows donâ??t have sharp edges but fade out. Colours blend, sometimes imperceptibly. At some level of detail, most things transition smoothly, but to varying degrees. The variation across such transitions is noticeable in a high quality digital photograph, i.e. a pixel image.

Here is a random row of 8 adjacent pixels taken from a much larger grey scale picture. A value of 0 is black, 255 is white, with numbers between indicating shades of grey. Beneath that is a graph showing these values plotted and joined by a curve. The darker pixels are presented as a dip in the graph. The higher part of the graph corresponds to the brighter pixels.



When averaged, wavy lines produce other wavy lines. The line above looks in part a bit like a regular cosine curve, but not quite. Cosine curves repeat as a regular up and down curve with the high and low parts of the curve repeating at regular intervals.

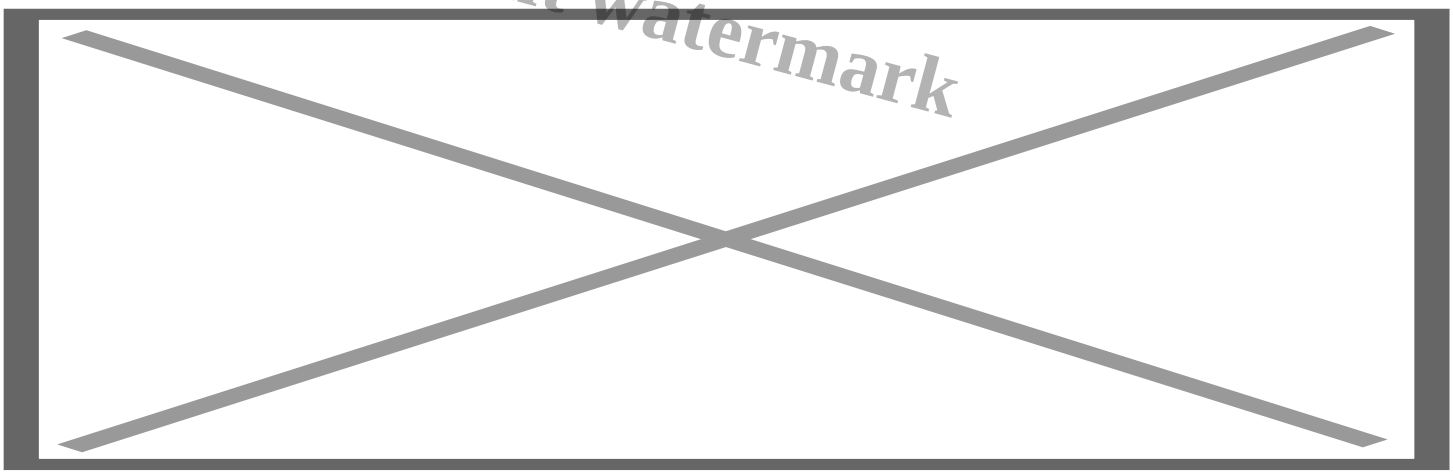
My curve can in fact be reconstructed from a set of cosine curves of varying frequencies across the span of pixels, e.g. zero cycles across 8 pixels; half a cycle across 8 pixels; a full cycle across 8 pixels, etc increasing in units of half a cycle. Our original blue curve above can be redrawn as the weighted average of a series of cosine curves. Those cosine curves have amplitudes (values on the vertical axis) represented by the formula

$$= \cos\left[\pi \frac{i}{N}\left(x + \frac{1}{2}\right)\right]$$

where ***x*** is the value on the horizontal axis, and ***i*** is the frequency of the wave (actually the number of half frequencies), i.e. the number of repeated oscillations across ***N*** increments along the horizontal axis. In this case, ***N*** is the number of pixels (8).

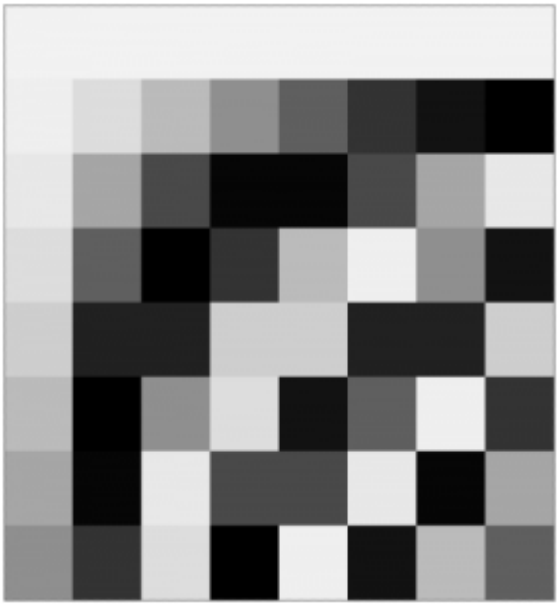
Here are some curves generated by this method. Iâ??ve shifted and multiplied the amplitude to correspond to the 0-255 grey-scale range to bring the process into 8-bit pixel space. To keep the formula simple, Iâ??ve not shown that adjustment in the formula. Here are the values for the curves plotted as a graph.

default watermark



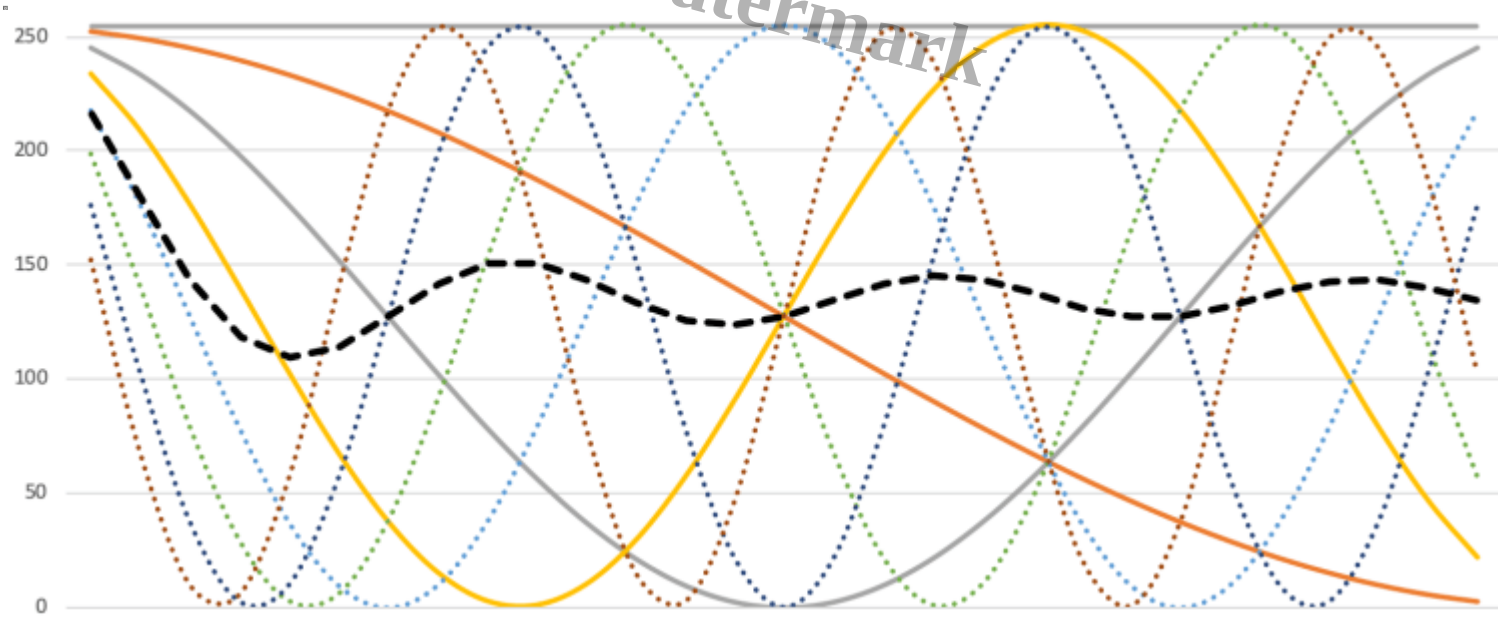
Iâ??ve only shown 7 frequency curves. An ***i*** of 0 gives a horizontal straight line at 255. I left that out so I could enjoy the symmetry of the curves.

Here are the grey-scale equivalents across 8 horizontal increments. The rows have 8 values sitting on the curves of each of the frequencies. They have grey-scale values 0-255.



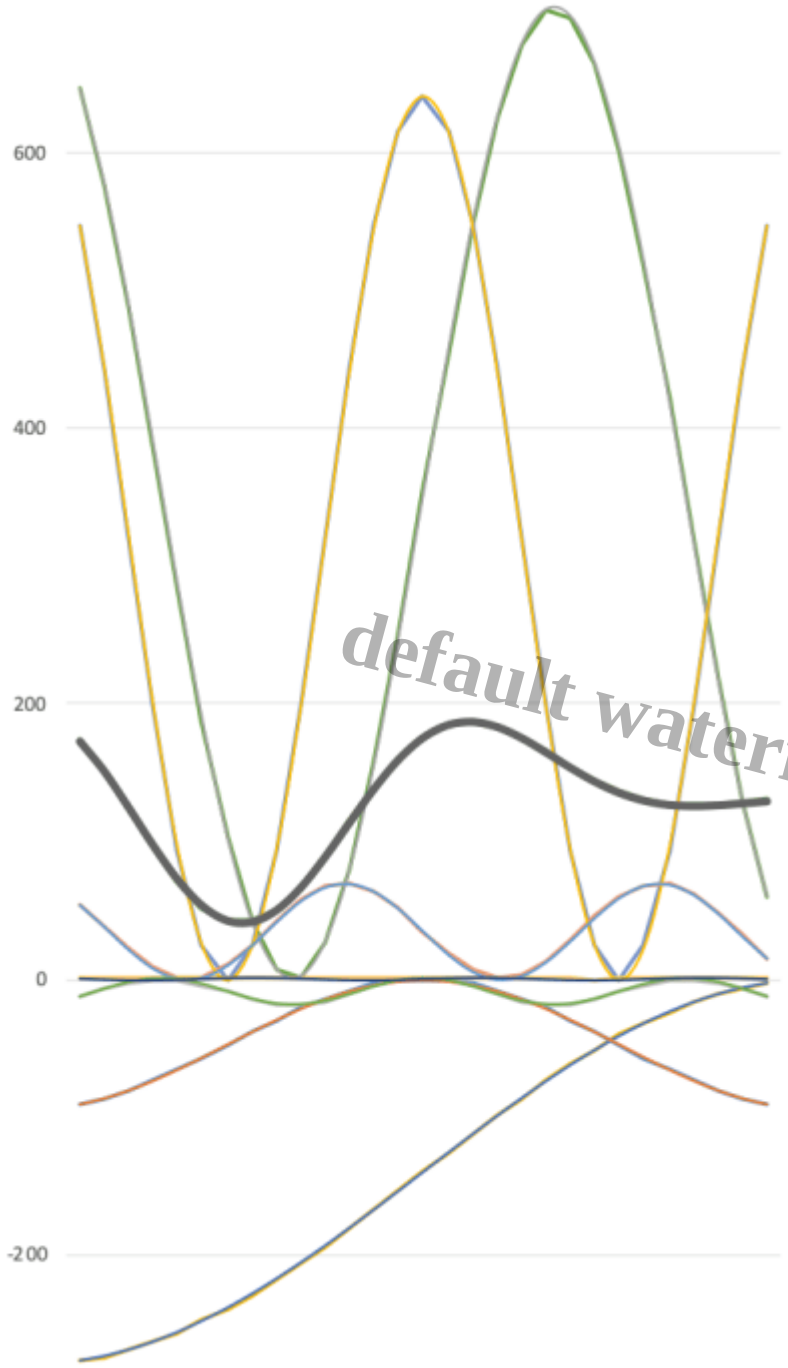
Here are all the curves again, this time with the *i*=0 value drawn in and the average of all the curves drawn as a dashed black line.

default watermark



Adjusting the amplitudes of the constituent waves results in a very close approximation of any wavy (sinusoidal) line. Each constituent curve can be multiplied by a coefficient (positive or negative) to construct a new curve.

Here I have plotted how each of the constituent waves would have to be adjusted, so that when averaged, they produce the original curve. If you are interested, the coefficients that I used to multiply each of the constituent curves are 2.0380, -1.1004, -0.3748, 2.7690, 2.5144, 0.2742, -0.0724, and 0.0031.



In preparing this example I worked backwards from the standard curves and made up some coefficients to produce the series of 8 pixel values that I claimed above as my starting point. It turns out that the algorithm that derives the multiplication coefficients for those standard constituent cosine curves is similar to the cos formula shown above. The formulas for deriving the coefficients and for reconstructing the original curve are known as the Discrete Cosine Transformation (DCT) function and the inverse DCT function.

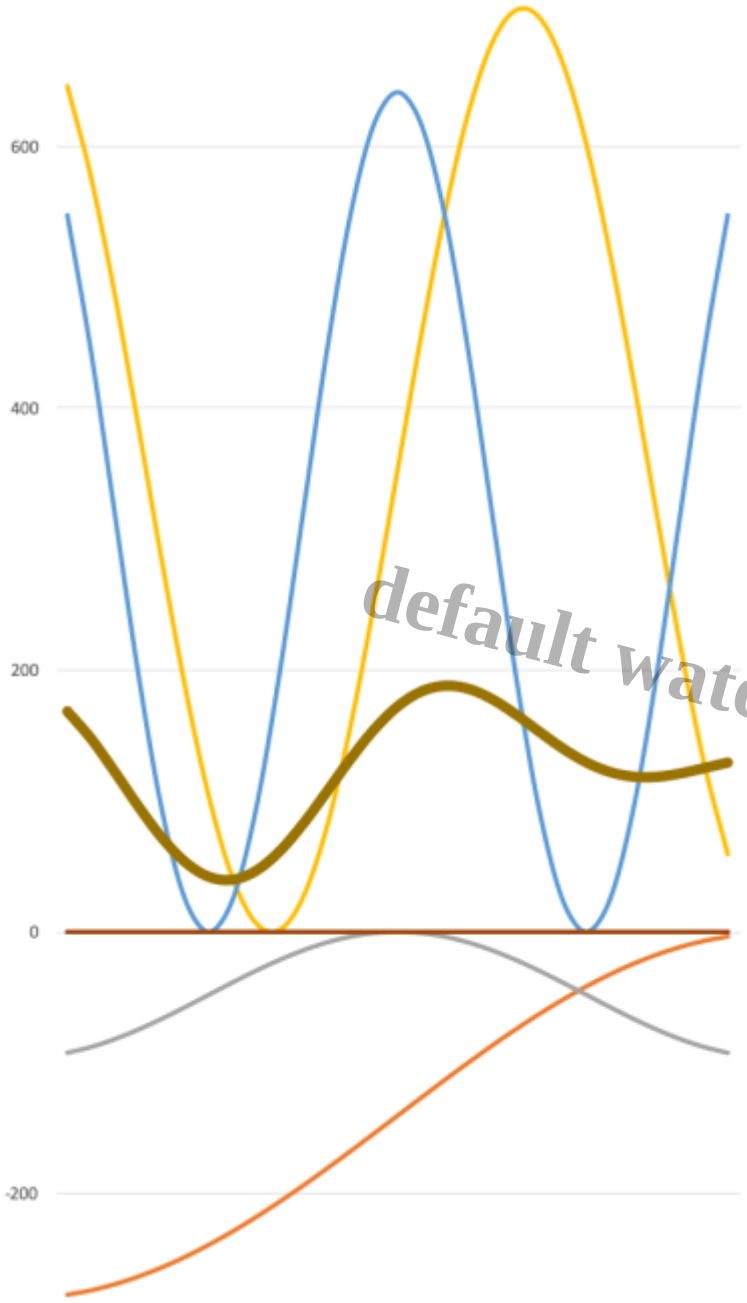
In this example, a series of 8 pixel values is converted to 8 coefficient values, which in turn can be converted back to the original pixel values, as long as thereâ??s an algorithm that takes into consideration those 8 standard frequency curves.

Why break a curve into sinusoidal components?

One advantage of breaking a wavy line into discrete cosine constituents is that it effectively teases out high and low frequency components. The human visual system is not so adept at picking up the high frequency components in an image, i.e. we are not good at noticing small differences in the luminance values between adjacent pixels or within small clusters of pixels. So a compression algorithm can generally discard the high frequency component of an image. The JPEG image compression algorithm does this and on a varying scale that can be adjusted by the user of the imaging software.

Here I show what happens to the curve if we set the coefficients of the 3 highest frequency curves to 0. They were low anyway in this case. But their removal doesnâ??t appreciably affect the shape of the curve, and hence the values along the row of 8 pixels with which I began.

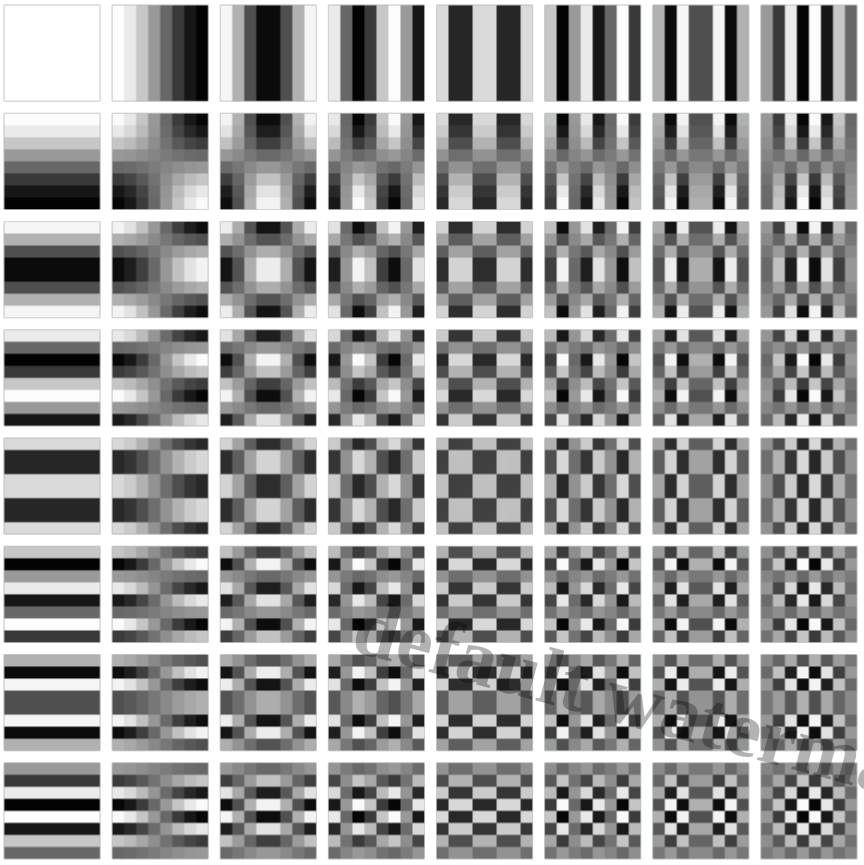
default watermark



Nor do the coefficients need to be to four decimal places. So the 8 pixels that I started with could be stored as a series of coefficients: 2.04, -1.10, -0.37, 2.77, 2.51, 0, 0, and 0.

Curves in 2D

Iâ??ve only shown the application of the DCT technique in one dimension. An image is a 2-dimensional array of pixels. Hereâ??s the standard 2D grey-scale chart used by graphics and signal processing experts to explain the cosine space. This is sourced from: By Devcore â?? Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=10414002



My 8Ã?8 grey scale matrix above corresponds to the values in the first row of this diagram, though I had arranged the spectra vertically instead of horizontally.

Another way of looking at the DCT algorithm in 2D is to imagine that any 8Ã?8 pix array from a digital image could be represented as combinations of some of these 8Ã?8 frequency spectra laid on top of each other, i.e. combined by weighted averaging. Thereâ??s a nice video of Mike Pound explaining this process. See the bibliography section below.

There are a lot of â??8 by 8â?³s in these attempts to explain DCT functions. I have to keep reminding myself that the coefficients donâ??t correspond to pixels, but to the amplitudes of a standard set of cosine curves that go to make up the values on an 8Ã?8 array of pixels.

Hiding things in the coefficients

As Iâ??m interested in steganography, a further advantage of the DCT compression of images is that information can be concealed in the coefficients, especially around the 0 coefficients, i.e. the high frequency components of an image, with relatively little deterioration in image quality.

This is my understanding of DCTs at present, and Iâ??m happy to be corrected on details. Iâ??m also conducting this exploration with standard functions on an Excel spreadsheet, and without resorting to routines in Visual Basic. I was keen to see the cosine curves, and so I used a pixel image array of more than 8 pixels to generate the curves using the Excel chart feature. Visualisations of DCT calculations are usually much more â??chunky.â?•

Bibliography

- Anon. 2019. Testing different image hash functions. *The Content Blockchain Project*. Available online: https://content-blockchain.org/research/testing-different-image-hash-functions/ (accessed 19 July 2020).
- Berry, Nick. 2013. Discrete cosine transformations. *Data Genetics*. Available online: http://datagenetics.com/blog/november32012/index.html (accessed 21 July 2020).
- Pound, Mike. 2015. JPEG DCT, Discrete Cosine Transform (JPEG Pt2). *Computerphile*, 22 May 2015. Available online: https://www.youtube.com/watch?v=Q2aEzeMDHMA&feature=youtu.be (accessed 24 July 2020).
- Rabie, Tamer, and Ibrahim Kamel. 2017. High-capacity steganography: A global-adaptive-region discrete cosine transform approach. *Multimedia Tools Applications*, (76)6473â??6493.

Note

- Compression algorithms donâ??t waste CPU time by drawing curves. In fact, the cos calculations in the formula shown above are pre-calculated and stored in tables in the software doing the compression.

default watermark

Category

1. Architecture

Tags

1. cryptography
2. DCT
3. discrete cosine function
4. steganography

Date Created
August 8, 2020
Author
rcoyne99