TUM

# Discrete Cosine Transformation

Michael Eiler, Computer Science Student
Technical University of Munich

Pro Seminar ”Hackers Delight”, organized by:
Christoph Doblander
Department of Economic Computer Science
Technical University of Munich

All resources mentioned in the paper are available here: https://github.com/MichaelE1000/in0013

## 1. INTRODUCTION

Many different kinds of signal transformations are used in several cases of data manipulation. On the one hand, there are the classic methods of signal transformations like modulation, which for example is used by cable tv providers to transmit multiple tv channels through a single cable. On the other hand, it’s possible to manipulate digital information. This is often done in image and audio compression algorithms like MP3 and JPEG.

One of the most famous signal transformations is the Fourier transformation. It is very powerful, but uses the complex domain for calculations. Even so modern cpus have extensions to process complex numbers, the fact that there are many embedded systems without those and that most of the image and audio algorithms have been developed in the early 90s implies that there has to be a simpler, or let’s say nicer, method.

The Fourier transformation is based on the Fourier series. As seen later the Fourier series has a very interesting characteristic which allows us to deduce the discrete cosine transformation. The discrete cosine transformation converts signals without using the complex domain. This paper is about this transformation and a sample implementation of the jpeg decoding algorithm. The JPEG algorithm has been chosen because it is one of the best examples of how to use the (fast) discrete cosine transformation and how it is very effectively used to improve data compression.

## 2. BASIC SIGNAL TRANSFORMATIONS

This chapter provides the basic understanding of how signal transformations work. Most of the signal transformations are based on the Fourier series. It is used to interpolate the signals with cosine and sine components. This is done because transmitting or processing the raw signal would otherwise require many resources. The Fourier series has a few interesting properties which lead to an optimized behaviour if the previously mentioned operations are applied on the transformed signal.

Publication date: May 2013.

The Fourier series uses a trick which is very common in numerical mathematics. It interpolates the signal by using simpler sine and cosine components. The following formulas are used to describe and calculate the interpolated signal.

$$y(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k * cos(k * \omega_0 * t) + b_k * sin(k * \omega_k * t)) \tag{1}$$

$$a_k = \frac{2}{T_p} * \int_0^{T_p} x(t) * cos(k * \omega_0 * t) \mathrm{d}t \tag{2}$$

$$b_k = \frac{2}{T_p} * \int_0^{T_p} x(t) * sin(k * \omega_0 * t) \mathrm{d}t \tag{3}$$

$$T_p = \frac{2\pi}{\omega_0} \tag{4}$$

Now we apply those formulas to a simple signal and have a look at the result:

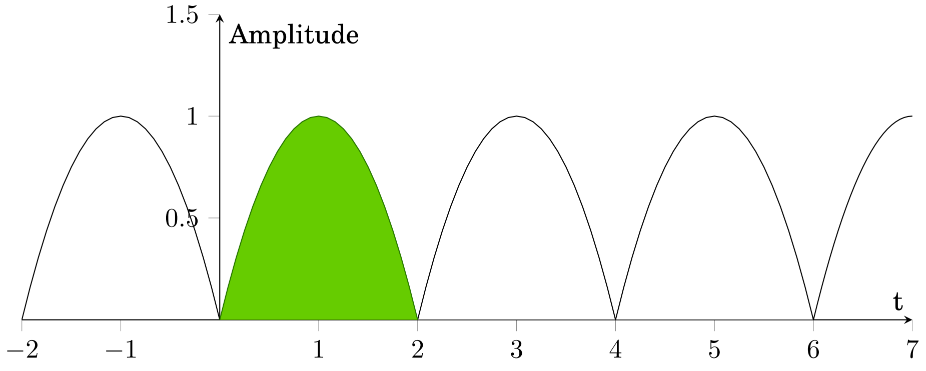

Fig. 1. A sample signal which we will interpolate by using the Fourier series

To calculate the Fourier series, we have to find an analytical expression for the repeating impulse because it is used in the formulas (2) and (3). In our case this is quite easy with a simple parabola.

$$x(t) = -t^2 + 1 \tag{5}$$

By inserting the impulse into those formulas, the coefficients of the Fourier series can be calculated:

$$\begin{aligned}\forall k \in \aleph \cup \{0\}, a_k = \int_0^2 (-t^2+1)cos(\pi kt)\mathrm{d}t = -\frac{2}{3} \\ \forall k \in \aleph, b_k = \int_0^2 (-t^2+1)sin(\pi kt)\mathrm{d}t = 0\end{aligned} \tag{6}$$

We benefit in two ways from the transformation. As seen we only have to store a single coefficient to exactly interpolate the original signal and we don't have to care about sine curves at all. But we aren't able to transform non-periodic signals. This is one of the advantages of the Fourier transformation compared to the Fourier series. It's based on the same ideas, but makes the the impulse interval more flexible, which allows us to expand it to $[-\infty, \infty]$. For further details on how the Fourier series and the Fourier transformation are related, read [Meyer 2011] page 29.

Fourier Transformation:

$$X(f) = \int_{-\infty}^{\infty} x(t) * \mathrm{e}^{-j2\pi t}\mathrm{d}t \tag{7}$$

Inverse Fourier Transformation:

$$x(t) = \int_{-\infty}^{\infty} X(f) * \mathrm{e}^{j2\pi t}\mathrm{d}t \tag{8}$$

But once again we face two problems. First, we only consider discrete singals, which means using a few data points instead of a continuous signal. Second, we are still working in the complex domain.

Solving the first problem is rather simple because we only have to use a sum instead of an integral for the discrete variant of the Fourier transformation.

$$X(f) = \sum_{n=-\infty}^{\infty} x(n) * \mathrm{e}^{-j2\pi fn} \tag{9}$$

The only problem left is that we still process the data using the complex domain. For this purpose the discrete cosine transformation has been invented. It describes a similar transformation as the Fourier transformation but uses only real numbers. This was achieved by using a property of the Fourier series: It only uses cosine components for the interpolation if the impulse is axially symmetric. The same thing applies to the Fourier transformation: If we transform the formula correctly, the complex part will vanish as shown in the deduction for the first variant of the DCT formula in chapter 3.1.

Usually the sampling points we want to transform aren't axially symmetric, therefore we have to flip them horizontally. This causes extra sampling points with useless data but thi sis unimportant as long as our original values are transformed/interpolated exactly. Unfortunately there are several possibilities of how to flip the signal which results in a few different variants of the discrete cosine transformation as seen and explained in the next chapter.

## 3. DISCRETE COSINE TRANSFORMATION

### 3.1. DCT-I

The first DCT-variant reflects the data points about exactly its first data point, which means the first data point isn't duplicated. The signal is also repeated exactly as it is and directly appended at the boundaries of the impulse. These conditions result in $2N - 2$ sampling points for every impulse.

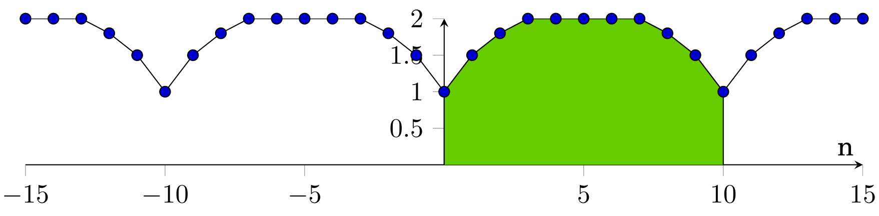

Fig. 2. Sample Image based on [Johnson 2013]

The following wrapper-function is used to mirror the signal.

$$x'(n) = \begin{cases} x(n), & \text{if } n \geq 1 \\ x(-n), & \text{otherwise} \end{cases} \tag{10}$$

#### *3.1.1. Transformation.*

Now we insert our wrapper-function and the number of sampling points into the formula of the discrete Fourier transformation and afterwords simplify it.

$$\begin{aligned} X(f) &= \sum_{n=-N+1}^{N-1} x'(n)\mathrm{e}^{-j2\pi\frac{f}{2N-2}n} \\ &= x(0) + x(N-1)\mathrm{e}^{-j2\pi\frac{f(N-1)}{2N-2}} + \sum_{n=1}^{N-2} x(n)(\mathrm{e}^{j2\pi\frac{f}{2N-2}n} + \mathrm{e}^{-j2\pi\frac{f}{2N-2}n}) \\ &= x(0) + x(N-1)\mathrm{e}^{-j\pi f} + 2\sum_{n=1}^{N-2} x(n)cos(\frac{\pi}{N-1}fn) \\ &= 2(\frac{1}{2}(x(0) + (-1)^f x(N-1)) + \sum_{n=1}^{N-2} x(n)cos(\frac{\pi}{N-1}fn)) \end{aligned} \tag{11}$$

In the first line we insert our previous values into the formula of the discrete Fourier transformation. Next, we replace $x'(n)$ with $x(n)$ and adjust the sum. In the third step we substitute the two exponential functions with the cosine function and simplify the rest. After another simplification we have the final formula.
Constant factors don't change the characteristics of a function and are usually omitted, therefore the formula found in literature is the following:

$$X(f) = (\frac{1}{2}(x(0) + (-1)^f x(N-1)) + \sum_{n=1}^{N-2} x(n)cos(\frac{\pi}{N-1}fn)) \quad (12)$$

*3.1.2. Inverse Transformation.*

To execute the inverse transformation, the DCT-I is used again with an additional factor, which makes this type of DCT simple to implement and use.

$$x(n) = \frac{2}{N-1} * \text{DCT-I} \quad (13)$$

### 3.2. DCT-II

The deduction of the formula is very similar for all discrete cosine transformations presented here, therefore we only discuss the final formula, beginning with this chapter. The second variant of the DCT is the most famous one, mainly due to the fact that it is used in many image and audio compression algorithms. The difference to the first variant is that the negative part of the impulse is moved one position to the left which also results in a copy of the sampling point at 0.

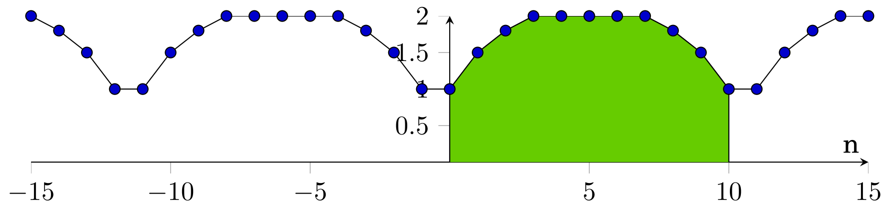

Fig. 3. Sample Image based on [Johnson 2013]

*3.2.1. Transformation.*

$$X(f) = \sum_{n=0}^{N-1} x(n)cos(\frac{\pi}{N}(n + \frac{1}{2})f) \quad (14)$$

*3.2.2. Inverse Transformation.*

The third variant of the DCT is used to calculate the inverse transformation.

$$x(n) = \frac{2}{N} * \text{DCT-III} \quad (15)$$

### 3.3. DCT-III

The third version of the DCT has the same mirrored base impulse as the first variant but the impulse is repeated differently. The impulse is not realined and every second impulse becomes multiplied with -1. This results in a zero-point in between the impulses.

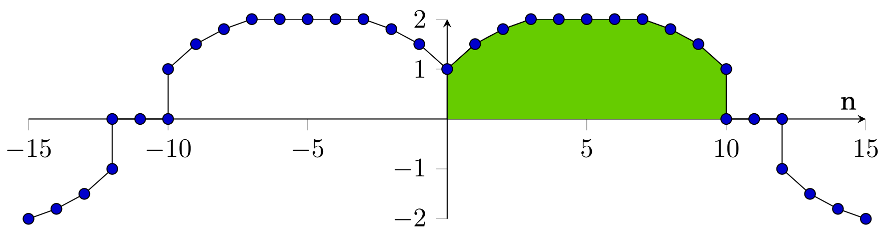

Fig. 4. Sample Image based on [Johnson 2013]

#### *3.3.1. Transformation.*

$$X(f) = \frac{1}{2}x(0) + \sum_{n=1}^{N-1} x(n)cos(\frac{\pi}{N}(f + \frac{1}{2})n) \tag{16}$$

#### *3.3.2. Inverse Transformation.*

$$x(n) = \frac{2}{N} * \text{DCT-II} \tag{17}$$

### 3.4. DCT-IV

The last variant is very similar to the third version of the DCT. The only difference is that the first sampling point is repeated in the centre like we also saw in the DCT-II.

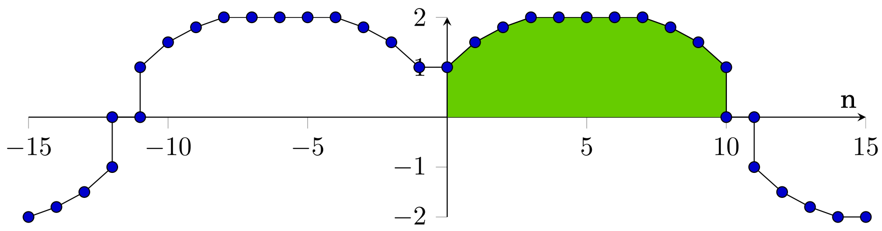

Fig. 5. Sample Image based on [Johnson 2013]

#### *3.4.1. Transformation.*

$$X(f) = \sum_{n=0}^{N-1} x(n)cos(\frac{\pi}{N}(f + \frac{1}{2})(n + \frac{1}{2})) \tag{18}$$

*3.4.2. Inverse Transformation.*

$$x(n) = \frac{2}{N} * \text{DCT-IV} \tag{19}$$

### 3.5. Two Dimensions

The following formulas calculate the two dimensional discrete cosine transformation for 8x8 matrices. These functions could be used to de- and encode jpeg image files. The only disadvantage is that the performance is extremely poor ($f(x, y), F(x, y) \in O(n^4)$ and an FPU is required).

*3.5.1. Transformation.*

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } k == 1 \\ 1, & \text{otherwise} \end{cases} \tag{20}$$

$$F(x, y) = C(x)C(y) \sum_{m=0}^{7} \sum_{n=0}^{7} f(m, n) cos(\frac{(2m + 1)x\pi}{16}) cos(\frac{(2n + 1)y\pi}{16}) \tag{21}$$

*3.5.2. Inverse Transformation.*

$$f(x, y) = \sum_{m=0}^{7} \sum_{n=0}^{7} C(m)C(n)F(m, n) cos(\frac{(2x + 1)m\pi}{16}) cos(\frac{(2y + 1)n\pi}{16}) \tag{22}$$

## 4. JPEG-ALOGRITHM

In this chapter we will have a look at compressing image data. This example will use raw input to demonstrate the JPEG codec. JPEG is a very common codec for lossy image compression. The specification contains several algorithms, but the algorithms only work well if the input is in a well suited representation for data compression. To achieve this, the whole data is splitted into 8x8 pixel blocks and transformed by a two dimensional discrete cosine transformation. These matrices are created for every colour component of the image. Usually the colour scheme is also converted from RGB to YCbCr. This allows us to further compress the image data because the human eye reacts more sensitive to luminance than chrominance.

Afterwards several other algorithms are used to compress the transformed data. Those algorithms are explained in the upcomming chapters and applied to some sample values, taken from [Wallace 1991], which are well suited for data compression and show the capabilities of the codec very well.

### 4.1. (Fast) Discrete Cosine Transformation

In the following table we see an 8x8 matrix, consisting of 64 pixels. They currently aren't very well suited for data compression, therefore we apply a discrete cosine transformation.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 139 | 144 | 149 | 153 | 155 | 155 | 155 | 155 |
| 144 | 151 | 153 | 156 | 159 | 156 | 156 | 156 |
| 150 | 155 | 160 | 163 | 158 | 156 | 156 | 156 |
| 159 | 161 | 162 | 160 | 160 | 159 | 159 | 159 |
| 159 | 160 | 161 | 162 | 162 | 155 | 155 | 155 |
| 161 | 161 | 161 | 161 | 160 | 157 | 157 | 157 |
| 162 | 162 | 161 | 163 | 162 | 157 | 157 | 157 |
| 162 | 162 | 161 | 161 | 163 | 158 | 158 | 158 |

*Source:* [Wallace 1991]
*Note:* Example values of an 8x8 pixel matrix

Like many other signal transformations the runtime of the classic transformation isn't really acceptable. Usually, implementations of the JPEG codec use a faster method, simply called ”Fast Discrete Cosine Transformation”. It is based on the well known fast Fourier transformation. Even so the second variant is a lot faster, the result is equivalent.
When looking at the resulting at the matrix it is striking, that the first value is the highest one. This value is called DC-value meaning the amplitude for the cosine curve with the lowest frequency and therefore has the highest impact on the interpolated signal. The other values are called AC-values and contain the amplitude values for the cosine curves with higher frequencies.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 256.6 | -1.0 | -12.1 | -5.2 | 2.1 | -1.7 | -2.7 | 1.3 |
| -22.6 | -17.5 | -6.2 | -3.2 | -2.9 | -0.1 | 0.4 | -1.2 |
| -10.9 | -9.3 | -1.6 | 1.5 | 0.2 | -0.9 | -0.6 | -0.1 |
| -7.1 | -1.9 | 0.2 | 1.5 | 0.9 | -0.1 | 0.0 | 0.3 |
| -0.6 | -0.8 | 1.5 | 1.6 | -0.1 | -0.7 | 0.6 | 1.3 |
| 1.8 | -0.2 | 1.6 | -0.3 | -0.8 | 1.5 | 1.0 | -1.0 |
| -1.3 | -0.4 | -0.3 | -1.5 | -0.5 | 1.7 | 1.1 | -0.8 |
| -2.6 | 1.6 | -3.8 | -1.8 | 1.9 | 1.2 | -0.6 | -0.4 |

*Source:* [Wallace 1991]
*Note:* Values after applying DCT

### 4.2. Quantization

The quantization process is used to produce many zero-values in the table because they are simpler to compress. The quantization is applied by dividing each value in the matrix with the matching one from the quantization table and rounding the result to an integer. The resulting table should now contain many zero-values which are well suited for Run-Length encoding.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 11 | 10 | 16 | 24 | 40 | 51 | 61 |
| 12 | 12 | 14 | 19 | 26 | 58 | 60 | 55 |
| 14 | 13 | 16 | 24 | 40 | 57 | 69 | 56 |
| 14 | 17 | 22 | 29 | 51 | 87 | 80 | 62 |
| 18 | 22 | 37 | 56 | 68 | 109 | 103 | 77 |
| 24 | 35 | 55 | 64 | 81 | 104 | 113 | 92 |
| 49 | 64 | 78 | 87 | 103 | 121 | 120 | 101 |
| 72 | 92 | 95 | 98 | 112 | 100 | 103 | 99 |

*Source:* [Wallace 1991]
*Note:* Quantization Table

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 15 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| -2 | -1 | 0 | 0 | 0 | 0 | 0 | 0 |
| -1 | -1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Source:* [Wallace 1991]
*Note:* Values after dividing with quantization values

### 4.3. Run-Length Encoding and Huffman Coding

Using four bytes for every integer in the final byte stream is wasteful, so here the next compression improvement is done. It saves another huge amount of memory by using Run-Length encoding. For this a new length value is created which stores the bit-length of the actual value in the lower 4 bits and the amount of previous zeros in the upper 4 bits. This value will even be further compressed by applying Huffman Coding.

Huffman Coding creates a binary tree containing all the values created by the Run-Length encoding algorithm but at a specific position inside of the tree. The path in the tree depends on how often the value is used in the image data. A shorter code indicates that the value is very often part of the compressed image data, a longer code indicates that the code isn't used very often.

The final bitstream contains the path (0 for the left child, 1 for the right child) generated by the Huffman tree and the actual value (without the leading zero bits) of the compressed value for every pixel in the 8x8 matrix. The Huffman tree itself is stored as part of the header information.

This way of creating the bit stream originally has been suggested in [ITU 1993] on page 125 and is also implemented in the sample jpeg decoder which I wrote for this seminar.

### 4.4. Decoding - Inverse DCT

The decoding process of course needs the same algorithms. The algorithms, or rather their counterparts, are just applied in the opposite order. The following table contains the values decoded from the bit stream after multiplication with the quantization values.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 240 | 0 | -10 | 0 | 0 | 0 | 0 | 0 |
| -24 | -12 | 0 | 0 | 0 | 0 | 0 | 0 |
| -14 | -13 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Source:* [Wallace 1991]
*Note:* After mutliplying with quantization values

The only thing left to do is to apply the (fast) inverse discrete cosine transformation, afterwards the values representing the image can be displayed to the user after applying the color scheme conversion.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 144 | 146 | 149 | 152 | 154 | 156 | 156 | 156 |
| 148 | 150 | 152 | 154 | 156 | 156 | 156 | 156 |
| 155 | 156 | 157 | 158 | 158 | 157 | 156 | 155 |
| 160 | 161 | 161 | 162 | 161 | 159 | 157 | 155 |
| 163 | 163 | 164 | 163 | 162 | 160 | 158 | 156 |
| 163 | 164 | 164 | 164 | 162 | 160 | 158 | 157 |
| 160 | 161 | 162 | 162 | 162 | 161 | 159 | 158 |
| 158 | 159 | 161 | 161 | 162 | 161 | 149 | 158 |

*Source:* [Wallace 1991]
*Note:* Result after applying Inverse DCT

The high loss of accuracy in the decoded values is caused by high quantization values which results in a very high compression, but also a high quality loss.

## REFERENCES

Ahmed Elgammal. 2012. Rutgers University (New Jersey), Introduction to Imaging and Multimedia, Chapter 9, Image Compression Standards. (2012). http://www.cs.rutgers.edu/~elgammal/classes/cs334/materials.htm

Prof. Bernd Girod. 2012. Stanford University, Lecture EE398A - Image and Video Compression. (2012). http://www.stanford.edu/class/ee398a/handouts/lectures/08-JPEG.pdf

Sebastion Hack. 2002. Universität Karlsruhe, Diskrete Cosinus Transformation. (2002). http://www.info.uni-karlsruhe.de/~rubino/ia64p/material/dct.pdf

International Telecommunication Union ITU. 1993. *Digital Compression and Coding of Continuous-Tone still Images - Requirements and Guidelines*. Technical Report T.81.

Steven G. Johnson. 2013. Visualization of the most common DCT variants. (2013). http://en.wikipedia.org/wiki/File:DCT-symmetries.svg License: GNU Free Documentation License, Version 1.2.

Martin Meyer. 2011. *Signalverarbeitung* (6th ed.). Vieweg+Teubner Verlag.

Tilo Strutz. 2009. *Bildkompression* (4th ed.). Viewer+Teubner Verlag.

Martin Vetterli. 1985. Fast 2-D discrete cosine transform. *IEEE International Conference on Acoustics, Speech, and Signal Processing* 1 (1985). http://infoscience.epfl.ch/record/34246/files/Vetterli85.pdf

Gergory K. Wallace. 1991. The JPEG still picture compression standard. *Commun. ACM* 34, 4 (April 1991), 30–44. DOI:http://dx.doi.org/10.1145/103085.103089