About the DCT matrix

Without going into deep academic

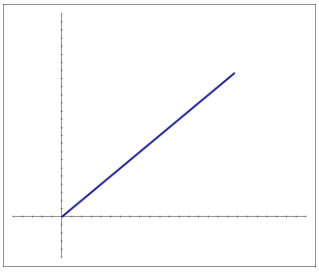
Some investigations how to understand the DCT- and inverse DCT, as it is used in different image- and video Decoder Standards. Because I've written some decoders for JPEG, MPEG, HEVC (AVC defines its own transform, there is nothing else to do there), I had to wrap my head around this and computing the transform efficiently. Here we look at different order-2^n DCT matrices with some spreadsheet help.

Basics: the DCT matrix

The discrete cosine transform (1-D DCT type-II) on a series of data points x_i is defined as $y_i = \sum_{j=0}^{N-1} x_i \cos\left(\frac{2\pi}{4N}(2j+1)i\right)$. This equation can be written in matrix form, a matrix-vector multiplication of y = Mx, where x and y are vectors, and the M matrix takes its values from $\cos\left(\frac{2\pi}{4N}(2j+i)\right)$. For example the 4-point transform is equivalent to and can be written as:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\pi}{8}0\right) & \cos\left(\frac{\pi}{8}0\right) & \cos\left(\frac{\pi}{8}0\right) & \cos\left(\frac{\pi}{8}0\right) \\ \cos\left(\frac{\pi}{8}1\right) & \cos\left(\frac{\pi}{8}3\right) & \cos\left(\frac{\pi}{8}5\right) & \cos\left(\frac{\pi}{8}7\right) \\ \cos\left(\frac{\pi}{8}2\right) & \cos\left(\frac{\pi}{8}6\right) & \cos\left(\frac{\pi}{8}10\right) & \cos\left(\frac{\pi}{8}14\right) \\ \cos\left(\frac{\pi}{8}3\right) & \cos\left(\frac{\pi}{8}9\right) & \cos\left(\frac{\pi}{8}15\right) & \cos\left(\frac{\pi}{8}21\right) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \Gamma x = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 3 & -3 & -1 \\ 2 & -2 & -2 & 2 \\ 3 & -1 & 1 & -3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

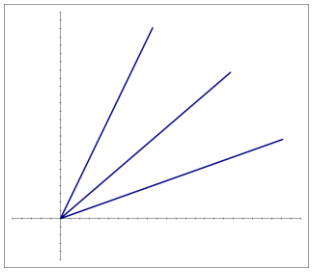
The matrix values are cosine of $\frac{\pi}{8}$ rotated around by some integer *k*. On the right side the $\Gamma_{i,j} = \pm\gamma(k_{i,j})$ matrix has values of $\gamma(k) = \cos\left(\frac{\pi}{8}k\right)$ based on cosine symmetry, resulting in only cosines of $\gamma(0,1,2,3)$ with alternating signs. It can be seen that the 2-point DCT rotates $\frac{\pi}{4}$, the 4-point rotates $\frac{\pi}{8}$ around, the 8-point DCT rotates $\frac{\pi}{16}$ and so forth. All the final cosine terms fall into the first quarter, symmetric to $\frac{\pi}{4}$ with alternating signs (0 not drawn):



N=2

$\frac{\pi}{4}$

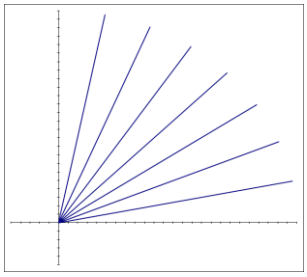
0	0
1	-1



N=4

$\frac{\pi}{8}$

0	0	0	0
1	3	-3	-1
2	-2	-2	2
3	-1	1	-3



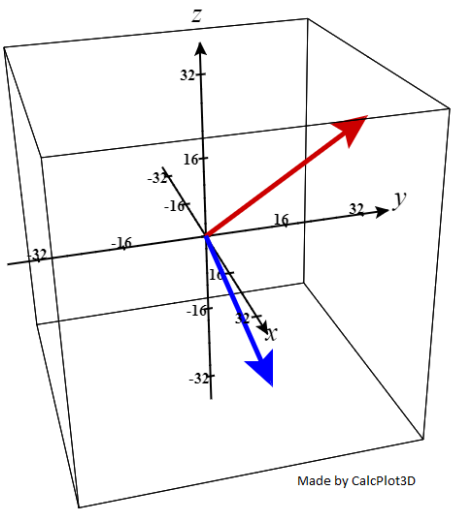
N=8

$\frac{\pi}{16}$

0	0	0	0	0	0	0	0
1	3	5	7	-7	-5	-3	-1
2	6	-6	-2	-2	-6	6	2
3	-7	-1	-5	5	1	7	-3
4	-4	-4	4	4	-4	-4	4
5	-1	7	3	-3	-7	1	-5
6	-2	2	-6	-6	2	-2	6
7	-5	3	-1	1	-3	5	-7

We can observe the symmetries of values in the rows of these matrices. Also, in every even row, the same cosine values appear from the lower order DCT matrix. For example where *k* is even in the 4-point DCT matrix, there $\cos\left(\frac{\pi}{8}k\right)=\cos\left(\frac{\pi}{4}\frac{k}{2}\right)$, the same as in the odd rows of the 2-point DCT matrix. This is the basis for many fast recursive DCT computation algorithms (aka. recursive kernel factorization).

The DCT as rotation



One way to look at the DCT is an N-dimensional vector transform, where rows of the DCT matrix are the basis vectors. Applying the DCT matrix on a vector rotates it in the N-dimensional vector space in such a way, that most of the *length* (energy) will concentrate in one direction (x), while the other vector coordinates (y, z …) will be small. This is only true, when all the vector coordinates are close to each other – as it is on neighboring pixel values of smooth images, like photographs. Some small coordinate values can be omitted and still *recover* the original vector closely by the inverse transform – thus achieving compression, like in JPEG. Only for illustration in 3D, something similar happens with a vector (red) after applying the DCT transform (blue). The transformed vector mainly points in the x-direction.

Basis vectors of the DCT matrix

Because of the special angles chosen the (row) vectors of the DCT matrix are mutually orthogonal. N orthogonal vectors in the N-dimensional vector space form an orthogonal basis. Orthogonality can be verified by computing all dot products of every vector combinations, for example compute the MM^T matrix product in a spreadsheet (poor-man's method). Dot products are commutative, a·b=b·a, and the result is a symmetric matrix with squares of magnitudes ("length") of each vector along the main diagonal. For N = 4:

Dot products

M^T

1	0.924	0.707	0.383
1	0.383	-0.707	-0.924
1	-0.383	-0.707	0.924
1	-0.924	0.707	-0.383

M

	m_0	m_1	m_2	m_3
m_0	1	1	1	1
m_1	0.924	0.383	-0.383	-0.924
m_2	0.707	-0.707	-0.707	0.707
m_3	0.383	-0.924	0.924	-0.383

MM^T

	m_0	m_1	m_2	m_3
m_0	$\|\|\mathbf{m_0}\|\|^2$	$\mathbf{m_0 \cdot m_1}$	$\mathbf{m_0 \cdot m_2}$	$\mathbf{m_0 \cdot m_3}$
m_1	$\mathbf{m_1 \cdot m_0}$	$\|\|\mathbf{m_1}\|\|^2$	$\mathbf{m_1 \cdot m_2}$	$\mathbf{m_1 \cdot m_3}$
m_2	$\mathbf{m_2 \cdot m_0}$	$\mathbf{m_2 \cdot m_1}$	$\|\|\mathbf{m_2}\|\|^2$	$\mathbf{m_2 \cdot m_3}$
m_3	$\mathbf{m_3 \cdot m_0}$	$\mathbf{m_3 \cdot m_1}$	$\mathbf{m_3 \cdot m_2}$	$\|\|\mathbf{m_3}\|\|^2$

=

MM^T

4	0	0	0
0	**2**	0	0
0	0	**2**	0
0	0	0	**2**

All dot products are indeed zero, which is due to sign-symmetry of the vector coordinates, these 4 orthogonal vectors form a basis in 4-dimension. When it comes to the magnitude of the four vectors, the length of the first basis vector is $\|m_0\| = 2$, while for all the others $\|m_i\| = \sqrt{2}$.

It can be seen that for any order-2^n of these kinds of DCT matrices

$$\|m_0\|^2 = N$$

and

$$\|m_i\|^2 = \frac{1}{2}N .$$

In other words length of $m_0 = \sqrt{N}$, and length of the other basis vectors are $m_i = \sqrt{\frac{N}{2}}$. The latter can be proven from that all coordinates are symmetric cosine terms with alternating signs; all angles are symmetric to $\frac{\pi}{4}$, and based on the trigonometric identities of $\cos^2(\gamma) = \frac{1}{2}(1+\cos(2\gamma))$ and of $\sin^2(\gamma) = \frac{1}{2}(1-\cos(2\gamma))$. All $\cos(2\gamma)$ falls out of the equation and the sum (the square of magnitude) will be $N\frac{1}{2}$.

Normalization of the DCT matrix

We could use the previous matrix equations for the DCT and the inverse DCT (the matrix is invertible), but in praxis the DCT uses a normalized form of these vectors, making the matrix and the transform orthonormal. An orthonormal basis is orthogonal, but with orthogonal unit vectors. Normalization is just scaling the row vectors uniformly in all *directions* that the length will be 1. This preserves orthogonality. Normalization of the N=4 matrix can be described as follows:

1. First normalize m_0 by a factor f_0 to the length of the other m_i basis vectors ("equalized DCT matrix") by solving $f_0\sqrt{N} = \sqrt{\frac{N}{2}}$. We get $f_0 = \frac{1}{\sqrt{2}}$. Scaling m_0 by $\frac{1}{\sqrt{2}}$ obtains the following matrix, where every vector has a magnitude of $\sqrt{2}$:

0.707	0.707	0.707	0.707
0.924	0.383	-0.383	-0.924
0.707	-0.707	-0.707	0.707
0.383	-0.924	0.924	-0.383

M

0.707	0.924	0.707	0.383
0.707	0.383	-0.707	-0.924
0.707	-0.383	-0.707	0.924
0.707	-0.924	0.707	-0.383

M^T

=

2	0	0	0
0	**2**	0	0
0	0	**2**	0
0	0	0	**2**

$M M^T$

2. Then scale all vectors by a factor of f so that all magnitude is one, becoming unit vectors. By solving $f\sqrt{\frac{N}{2}} = 1$ we get $f = \sqrt{\frac{2}{N}}$. Finishing the normalization by scaling every m_i above by $f = \frac{1}{\sqrt{2}}$ obtains the orthonormal order-4 DCT matrix with 4 orthonormal unit vectors:

0.5	0.5	0.5	0.5
0.653	0.271	-0.271	-0.653
0.5	-0.5	-0.5	0.5
0.271	-0.653	0.653	-0.271

M

0.5	0.653	0.5	0.271
0.5	0.271	-0.5	-0.653
0.5	-0.271	-0.5	0.653
0.5	-0.653	0.5	-0.271

M^T

=

1	0	0	0
0	**1**	0	0
0	0	**1**	0
0	0	0	**1**

$M M^T$

Looking at other order-2^n matrices, there is an interesting relationship. The scaling factor for the first row vector, $f_0 = \frac{1}{\sqrt{2}}$ is independent of N and will be the same for all "equalized" matrices (that is all vector coordinates will be $\frac{1}{\sqrt{2}}$). After step 1 all row vectors have the same magnitude: $\|m_i\| = \sqrt{\frac{N}{2}}$. The matrix is a linearly scaled version of the orthonormal N-point DCT matrix. Computing MM^T after step 1 reveals the scaling factor for step 2 in the squares of magnitudes along the main diagonal:

N	"Equalized DCT" After Step 1	$\|m_i\|$ length	MM^T	f^2	f
2	0.707 0.707 0.707 -0.707	1	**1** 0 0 **1**	1	1
4	0.707 0.707 0.707 0.707 0.924 0.383 -0.383 -0.924 0.707 -0.707 -0.707 0.707 0.383 -0.924 0.924 -0.383	$\sqrt{2}$	**2** 0 0 0 0 **2** 0 0 0 0 **2** 0 0 0 0 **2**	$\frac{1}{2}$	$f=\sqrt{\frac{2}{N}}=\sqrt{\frac{2}{4}}=\frac{1}{\sqrt{2}}$
8	0.707 0.707 0.707 0.707 0.707 0.707 0.707 0.707 0.981 0.831 0.556 0.195 -0.195 -0.556 -0.831 -0.981 0.924 0.383 -0.383 -0.924 -0.924 -0.383 0.383 0.924 0.831 -0.195 -0.981 -0.556 0.556 0.981 0.195 -0.831 0.707 -0.707 -0.707 0.707 0.707 -0.707 -0.707 0.707 0.556 -0.981 0.195 0.831 -0.831 -0.195 0.981 -0.556 0.383 -0.924 0.924 -0.383 -0.383 0.924 -0.924 0.383 0.195 -0.556 0.831 -0.981 0.981 -0.831 0.556 -0.195	2	**4** 0 0 0 0 0 0 0 0 **4** 0 0 0 0 0 0 0 0 **4** 0 0 0 0 0 0 0 0 **4** 0 0 0 0 0 0 0 0 **4** 0 0 0 0 0 0 0 0 **4** 0 0 0 0 0 0 0 0 **4** 0 0 0 0 0 0 0 0 **4**	$\frac{1}{4}$	$f=\sqrt{\frac{2}{N}}=\sqrt{\frac{2}{8}}=\frac{1}{2}$
16		$\sqrt{8}$	8 $\mathbf{I_{16}}$	$\frac{1}{8}$	$f=\sqrt{\frac{2}{N}}=\sqrt{\frac{2}{16}}=\frac{1}{\sqrt{8}}$
32		4	16 $\mathbf{I_{32}}$	$\frac{1}{16}$	$f=\sqrt{\frac{2}{N}}=\sqrt{\frac{2}{32}}=\frac{1}{4}$
2^n		$\sqrt{2}^{n-1}$	2^{n-1} $\mathbf{I_N}$	$2^{-(n-1)}$	$f=\sqrt{2}^{-(n-1)}$

This factor "2" relationship will have importance in HEVC, where the same matrix is used for all transform sizes of N = 4, 8, 16 and 32 (see below).

It also gives possibilities to calculate the transform with any scalar scaled version of the DCT matrix, possibly giving some simplifications for the calculation, and then simply divide the results at the end. For example the N=8 matrix after step 1 in the table above can also be used for the DCT, then divide the results by *2*. In formula *Mx = (fM)x = f(Mx).* This is how the LLM works (see below).

The 2-D DCT

$$y_{u,v} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} x_{i,j} \cos\left(\frac{2\pi}{4N}(2i+1)u\right)\cos\left(\frac{2\pi}{4N}(2j+1)v\right)$$

According to this equation computing the N=8 (8×8) transform is a series of 8×8×8×8 = 4096 multiplications by irrational numbers. For a fast image/video codec this is unacceptable. Most fast algorithms use the separability of the 2-D transform to successive 1-D row- and column transforms. By doing that the matrix form for the 2-D DCT transform becomes:

$Y = M\ X\ M^T$

- Column then row transform: $Y = (\ M\ X\ )\ M^T$
- Row then column transform: $Y = M\ (\ X\ M^T\ )$

For the inverse transform, when M is orthonormal:

$X = M^{-1}\ Y\ M = M^T\ Y\ M$

- Column then row transform: $X = (\ M^T\ Y\ )\ M$
- Row then column tra4nsform: $X = M^T\ (\ Y\ M\ )$

This shows that the order of row/column or column/row leads to the same result – based on matrix product associativity. Here demonstrated by a live example during JPEG image decoding, 2-D inverse DCT of an 8×8 decoded coefficient block. We get the same results:

904	-89	55	31	26	24	14	4
-183	-109	67	8	27	34	11	0
-61	-37	-16	14	16	25	5	10
22	-36	23	1	2	1	-16	-2
2	32	-12	-5	-3	-6	0	3
25	25	-12	-9	4	10	-6	-3
14	1	8	8	0	9	3	0
3	6	16	8	-15	0	3	0

ImageProxy.jpg first block decoded coefficients

transform coefficients

221.5	-97.94	49.99	19.81	29.24	39.5	5.202	5.429
209.4	-106.1	43.98	17.13	26.55	20.37	12.76	3.933
279.1	-42.93	43.26	17.92	7.865	19.91	16.94	-0.872
330.8	3.567	8.374	-5.968	11.84	0.289	3.618	-2.836
360.9	-10.07	33.75	8.357	-10.36	-14.1	0.514	-1.453
395	-27.55	17.63	9.565	6.518	0.051	-6.185	-2.248
392.1	5.408	-10.12	6.292	0.08	2.099	-3.722	0.601
368.2	23.84	-31.3	14.57	1.803	-0.232	10.46	8.759

col transform

84	**12**	**30**	**63**	**66**	**92**	**139**	**141**
67	**10**	**28**	**52**	**69**	**96**	**124**	**145**
117	**67**	**76**	**78**	**79**	**115**	**125**	**133**
125	**115**	**116**	**120**	**113**	**110**	**110**	**127**
134	**140**	**116**	**100**	**117**	**134**	**135**	**145**
139	**132**	**118**	**131**	**139**	**144**	**155**	**159**
139	**139**	**138**	**143**	**145**	**140**	**138**	**127**
137	**125**	**143**	**137**	**149**	**137**	**113**	**100**

row transform

904	-89	55	31	26	24	14	4
-183	-109	67	8	27	34	11	0
-61	-37	-16	14	16	25	5	10
22	-36	23	1	2	1	-16	-2
2	32	-12	-5	-3	-6	0	3
25	25	-12	-9	4	10	-6	-3
14	1	8	8	0	9	3	0
3	6	16	8	-15	0	3	0

transform coefficients

333.2	261.6	270.4	291.4	310	342.3	367.4	380.6
-62.78	-129.3	-112.9	-86.93	-89.49	-51.1	-3.739	18.58
-26.75	-64.38	-32.4	-11.48	-7.467	-11.3	-0.809	-17.94
-1.107	3.864	-15.95	-1.471	3.315	6.504	33.86	33.2
6.343	15.37	16.07	5.735	4.645	-7.939	-16.43	-18.14
14.57	15.1	18.04	27.51	6.377	-4.137	0.699	-7.442
15.54	0.317	2.037	2.297	-0.937	7.572	9.873	2.903
9.991	9.754	2.432	-13.84	-10.57	6.945	6.326	-2.546

row transform

84	**12**	**30**	**63**	**66**	**92**	**139**	**141**
67	**10**	**28**	**52**	**69**	**96**	**124**	**145**
117	**67**	**76**	**78**	**79**	**115**	**125**	**133**
125	**115**	**116**	**120**	**113**	**110**	**110**	**127**
134	**140**	**116**	**100**	**117**	**134**	**135**	**145**
139	**132**	**118**	**131**	**139**	**144**	**155**	**159**
139	**139**	**138**	**143**	**145**	**140**	**138**	**127**
137	**125**	**143**	**137**	**149**	**137**	**113**	**100**

col transform

Computing the N=8 (8×8) transform by this method requires only (8+8)×64 = 1024 multiplications – compared to the original formula with 4096 multiplications.

Scaled DCT: LLM

When computing the 2-D transform with the row/column method the same matrix is applied twice on the input block. This gives possibilities to use some scaled version of the orthonormal DCT matrix, in which it is simpler to calculate the vector-matrix product. The results are then *down-scaled* at the end. This is how the LLM method works: the orthogonal 8-point DCT matrix is scaled by $2\sqrt{2}$ (or any *equalized* order-2^n matrix by $\sqrt{2}$), which yields:

1	1	1	1	1	1	1	1
1.387	1.176	0.786	0.276	-0.276	-0.786	-1.176	-1.387
1.307	0.541	-0.541	-1.307	-1.307	-0.541	0.541	1.307
1.176	-0.276	-1.387	-0.786	0.786	1.387	0.276	-1.176
1	-1	-1	1	1	-1	-1	1
0.786	-1.387	0.276	1.176	-1.176	-0.276	1.387	-0.786
0.541	-1.307	1.307	-0.541	-0.541	1.307	-1.307	0.541
0.276	-0.786	1.176	-1.387	1.387	-1.176	0.786	-0.276

MM^T

8	0	0	0	0	0	0	0
0	**8**	0	0	0	0	0	0
0	0	**8**	0	0	0	0	0
0	0	0	**8**	0	0	0	0
0	0	0	0	**8**	0	0	0
0	0	0	0	0	**8**	0	0
0	0	0	0	0	0	**8**	0
0	0	0	0	0	0	0	**8**

Here y_0 and y_4 can be computed without multiplications – and among other simplifications reaches the LLM method computing the 8-point transform with only 11 multiplications. Because this matrix is applied twice on the input block, after successive row- and column transformation the results are *over scaled* by $(2\sqrt{2})^2 = 8$. Dividing by 8 at the end is just right shift – a very fast operation for the CPU ALU. Computing the magnitudes of these (row) vectors all give $\sqrt{8}$.

Scaled integer DCT: HEVC

HEVC defines an integer transform process together with one 32×32 integer transform matrix used for various transform sizes. The integer matrix is based on the orthogonal 4-point DCT-II formula, which yields:

K =

0,5	0,5	0,5	0,5
0,653	0,271	-0,271	-0,653
0,5	-0,5	-0,5	0,5
0,271	-0,653	0,653	-0,271

K multiplied by 128 and rounded to nearest integer is very similar to M_4, the 4×4 integer transform matrix defined by HEVC:

$$[128K] = \begin{pmatrix} 64 & 64 & 64 & 64 \\ 84 & 35 & -35 & -84 \\ 64 & -64 & -64 & 64 \\ 35 & -84 & 84 & -35 \end{pmatrix} \qquad M_4 = \begin{pmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{pmatrix}$$

The Standard defines the inverse integer 2-D transform (decoder) by the row/column method, with column transform first, and after each transform the results are right-shifted (with

rounding). The 1-D inverse integer transform of vector y (decoded coefficients) can be computed using integer arithmetic as:

$$x = \frac{1}{128}(My) = (My) >> 7 .$$

The 2-D inverse integer transform with the row/column method applies this matrix twice on the input matrix Y according to:

$$X = (M^T Y)M .$$

It can be computed as:

$X = (((M^T Y) >> 7)M) >> 7$, obtaining the reconstructed image sample values.

For higher order-N integer transforms the original scaling of the order-4 integer matrix is kept. For the 8-point integer transform the following matrix is used:

$M_8 =$

64	64	64	64	64	64	64	64
89	75	50	18	-18	-50	-75	-89
84	35	-35	-84	-84	-35	35	84
75	-18	-89	-50	50	89	18	-75
64	-64	-64	64	64	-64	-64	64
50	-89	18	75	-75	-18	89	-50
35	-84	84	-35	-35	84	-84	35
18	-50	75	-89	89	-75	50	-18

We've seen the relationships earlier before different order-2^n DCT matrices, M_8 in this form is *overscaled* by $\sqrt{2}$ compared to M_4. Applying this matrix twice overscales the results by 2. Division by 2 can be easily incorporated in the previous integer equation as:

$$X = (((M^T Y) >> 7)M) >> 8$$

For the 16-point transform by >> 9, and for the 32-point transform by >> 10. This is how the same matrix – as defined by the Standard – can be used for all transform sizes.

End