Integer Transform of H.264/AVC
In previous standards, the DCT was defined as the ideal transform, with unlimited accuracy. This has the problem, that we have encoders with limited accuracy for the DCT, and decoders, with different accuracy for the inverse DCT. This then leads to reconstruction errors from the different implementations. Also, if we want to implement the DCT with low error, we need an arithmetic with long word length and floating point implementation, which makes the implementation complex. This is a problem for low end processors, it makes the coder hardware more expensive. Since Video decoding plays an increasing role in consumer hardware, where cost is very important, this problem was addressed in H.264. They solved the problem by specifying the DCT with integer arithmetic, such that the rounding errors we make are now known (since it is specified in the standard). In this way, we can still obtain perfect reconstruction (in the absence of quantization), because in the standard we can specify the inverse DCT with integer arithmetic, such that we obtain the exact inverse. In this way, the rounding errors „fit“ to each other, to obtain perfect reconstruction. In this way we get a different transform, which is not quite the
DCT, but only similar to it. The approach was to take the DCT matrix, multiply it with a certain factor, and round it to obtain a transform matrix with integer entries. A factor which turned out to be suitable is $\alpha = 2.5$ (see also: H. Malvar et. Al: „Low Complexity Transform and Quantization in H.264/AVC“, IEEE trans. on Circuits and Systems for Video Technology, July 2003). It still leads to a good coding gain, but allows a low wordlength processing (16 bit arithmetic). H.264 uses 8x8 and also 4x4 transform matrices. Here we look at the 4x4 version. We start with the 4x4 DCT type 2,

$$T = \begin{bmatrix}
0.5000 & 0.5000 & 0.5000 & 0.5000 \\
0.6533 & 0.2706 & -0.2706 & -0.6533 \\
0.5000 & -0.5000 & -0.5000 & 0.5000 \\
0.2706 & -0.6533 & 0.6533 & -0.2706
\end{bmatrix}$$

Here we assume that we multiply our signal $x$ from the right hand side, to obtain the transformed signal: $y^T = T \cdot x^T$, and for the inverse: $x^T = T^{-1} \cdot y^T$.

With the factor of $\alpha = 2.5$ we get

$$2.5 \cdot T = \begin{bmatrix}
1.2500 & 1.2500 & 1.2500 & 1.2500 \\
1.6332 & 0.6765 & -0.6765 & -1.6332 \\
1.2500 & -1.2500 & -1.2500 & 1.2500 \\
0.6765 & -1.6332 & 1.6332 & -0.6765
\end{bmatrix}$$

Now we can round it to obtain a useful transform matrix with integer entries,
\( H := \text{round}(2.5 \cdot T) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \)

This is the transform matrix used by H.264. We see that it is quite similar to the WHT, but gives better compression performance. Malvar writes that for a stationary Gauss-Markov input with correlation coefficient \( \rho = 0.9 \) the coding gain for the DCT is 5.39 dB, whereas for this integer version it is 5.38 dB.

(Generation of a stationary Gauss-Markov input \( y(n) \):

\[ x(n) \xrightarrow{\text{filter}} y(n) \]

\( y(n) \): filtered output, the stationary Gauss-Markov signal. In the z-domain we obtain a pole at \( z = 0.9 \) in the transfer function, which makes it a low pass. This fits to natural images, because they also have a low pass characteristic. But also observe that this is only a very crude and simple approximation of a natural image, but at least this gives as a simple model of a natural image.

The coding gain is defined as the arithmetic
average divided by the geometric average of the squares of the subband values $y_k^2$, to give us an estimate of the compression performance of a given subband decomposition (in dB). The arithmetic average is our usual average as the sum of values divided by the number of values, the geometric average is using the product and the Nth root instead of the sum and the division by N:

$$\text{coding gain: } \frac{1}{N} \cdot \frac{\sum_{k=0}^{N-1} y_k^2}{\sqrt[N]{\prod_{k=0}^{N-1} y_k^2}}$$

See also: Jayant, Noll: “Digital Coding of Waveforms”, Prentice Hall.
Observe: The coding gain is 1, or 0 dB, if all subband values $y_k$ are identical. The more different the $y_k$ become, the higher the coding gain, with the extreme of a $y_k$ being 0. In this case we have an infinite coding gain.

So at least for this artificial signal it is only a very small loss in coding gain of only 0.01 dB. For a WHT the loss in coding gain would be clearly higher. Observe that we can use the same approach (with the same factor) to obtain an integer 8x8 transform.
We still need an integer value inverse, for the decoder. Is it possible to get an integer valued exact inverse?

We can simply first compute the inverse of $H$,

$$H^{-1} = \begin{bmatrix} 0.2500 & 0.2000 & 0.2500 & 0.1000 \\ 0.2500 & 0.1000 & -0.2500 & -0.2000 \\ 0.2500 & -0.1000 & -0.2500 & 0.2000 \\ 0.2500 & -0.2000 & 0.2500 & -0.1000 \end{bmatrix}$$

Here we could use again a factor to obtain integer values for the inverse transform. In this way we would obtain the original, but scaled with the two factors. But the goal is to have factors as small as possible to have lower wordlengths for the arithmetic. The trick used here is to apply different factors to the columns, and allow factors which can be implemented with a shift operation (e.g. 0.5). These factors are 4,5,4,5. With these factors we get the matrix

$$\tilde{H}_{\text{inv}} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix}$$

This means we get the inverse as

$$H^{-1} = \tilde{H}_{\text{inv}} \cdot \text{diag}(1/4,1/5,1/4,1/5)$$

This shows that we just extracted the diagonal matrix from the inverse. The diagonal matrix does not need to be computed explicitly, because we can factor it into the inverse quantization of the decoder.

So, in this way we obtained the exact inverse.
with very simple numbers or fractions, easy to implement!

A fast implementation:

![Diagram of fast implementation](image)

Fig. 1. Fast implementation of the H.264 direct transform (top) and inverse transform (bottom). No multiplications are needed, only additions and shifts. (From [1]).

**Literature:**


Transactions on Circuits and Systems for Video Technology, July 2003


Video Technology,  
Volume: 20, Issue: 12  
Digital Object Identifier:  
10.1109/TCSVT.2010.2092613  
Publication Year: 2010, Page(s): 1688 - 1697

"Video Compression Using Nested Quadtree Structures, Leaf Merging, and Improved Techniques for Motion Representation and Entropy Coding"  
IEEE Transactions on Circuits and Systems for Video Technology,  
Volume: 20, Issue: 12  
Digital Object Identifier:  
10.1109/TCSVT.2010.2092615  
Publication Year: 2010, Page(s): 1676 - 1687

(All to be found at ieeexplore.ieee.org from inside the TU Ilmenau Network)