Abstract. This paper describes implementation of the Discrete Cosine Transform (DCT) algorithm to MATLAB. This approach is used in JPEG or MPEG standards for instance. The substance of these specifications is to remove the considerable correlation between adjacent picture elements.

The objective of this paper is not to improve the DCT algorithm itself, but to re-write it to the preferable version for MATLAB thus allows the enumeration with insignificant delay. The method proposed in this paper allows image compression calculation almost two hundred times faster compared with the DCT definition.

Keywords
MATLAB, algorithms, image compression, 2D DCT, NRMSE, calculation time.

1. Introduction

The typical still images contain areas where neighboring pixels have almost the same values therefore the spatial redundancy is considerable. To remove the redundancy, different methods (for lossy and lossless compression) have been developed.

One of the lossy methods is known as JPEG, the Joint Photographic Experts Group. Officially, JPEG corresponds to the IS1/IEC international standard 10928-1 and has been established by members from the International Telecommunication Union (ITU) and the International Organization for Standardization (ISO).

In this contribution we attempt to optimize JPEG arithmetic operations for MATLAB. To examine computational requirements of the proposed algorithm, we used seven test images with different dimensions.

2. JPEG Standard Based Image Compression

In technique used in JPEG, the source image is divided in 8×8 blocks and each block is transformed using the DCT. Each of the 64 DCT coefficients is achieved via quantization followed by variable length coding.

By Wallace [1] the 2D DCT (Forward Two-Dimensional Discrete Cosine Transform) is defined in the following way:

\[ P_{u,v} = \frac{C_u \cdot C_v}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} p_{x,y} \cdot D_{x,u} \cdot D_{y,v} \]  

(1)

where \( P_{u,v} \) is the 2D DCT coefficient, \( u,v=0,\ldots,7 \), \( p_{x,y} \) is the intensity of picture element and constants \( C_0=2^{-\frac{3}{2}} \) and \( C_k=1 \) for \( k=1,\ldots,7 \). We construct a DCT transform base as follows:

\[ D_{a,b} = \frac{\cos \left( \frac{(2a+1)b\pi}{16} \right)}{16} \]  

(2)

where parameters \( a \in \{x,y\} \) and \( b \in \{u,v\} \).

In a similar way we define the Inverse Two-Dimensional Discrete Cosine Transform (2D iDCT) of the form:

\[ p_{x,y} = \sum_{a=0}^{7} \sum_{b=0}^{7} \frac{C_x \cdot C_y}{4} \cdot p_{x,y} \cdot D_{a,x} \cdot D_{b,y} \]  

(3)

It is well-known rule to use as few cycles as possible in environment of MATLAB. Consequently we can not program (1-3) by series of two nested loops for one DCT coefficient, like we can probably do in the language C. In the next text we present source codes suitable for image compression in MATLAB.

2.1 2D DCT MATLAB Implementation

The essence of the image coding implementation consists in a conversion to the domain convenient for MATLAB, i.e. matrix operations [2]. Both of sums from (1, 3) can be overwritten to the scalar or matrix multiplication [2, 3]. Next paragraphs show source code of the encoder main function when test image is divided into 8×8 blocks and each block is multiplied by DCT base \( mUX, mYV \) defined in (2) and by scaling matrix \( mNORM \) which contains constants \( C_k \). Subsequently, the DCT coefficients are quantized by a quantize matrix \( Qtab \) and save to a logical file \( fout \) via function \( store \).

\[
\text{for } i = 0 : (\text{nbx} - 1) / 8, \\
\text{for } j = 0 : (\text{nby} - 1) / 8, \\
\quad \text{block} = \text{img} (i*8+1 : (i+1)*8, \\
\qquad j*8+1 : (j+1)*8) - 128; \\
\text{end} \\
\text{end}
\]
coeff = mNORM .* (mUX * block * mYV) ;
coeff = round (coeff ./ Qtab) ;
store (coeff, fout) ;
end
end

All used constants are represented by the global variables defined in function initCts. Variable QtabJPEG specifies the quantization step size for each of the 64 DCT coefficients and it is the original quantize matrix for the luminance signals in JPEG standard [5].

function initCts

global mUX, mYV, mNORM, QtabJPEG, zig
mUX = cos ([0:7]' * (2*[0:7]+1) * pi/16) ;
mYV = cos ((2*[0:7]+1) * [0:7] * pi/16) ;
mNORM = [1/2 ones(1,7) / sqrt(2); ones(7,1) / sqrt(2) ones(7,7)] / 4 ;
QtabJPEG = [16 11 10 16 24 40 51 61; 12 12 14 19 26 58 60 55; 14 13 16 24 57 69 103 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];
zig = [ 1 9 2 3 10 17 25 18 ... 11 4 5 12 19 26 33 41 ... 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];

Last variable is a vector named zig with a size of 1×64 which guarantees the scanning order for all DCT coefficients. This order is shown in Fig.1 and is so-called zig-zag scanning when the coefficients which represent the lower frequencies are coded in advance and higher frequencies subsequently.

\[
Qtab\_new = \frac{QtabJPEG \cdot scal + 50}{100}
\]

where \( scal=5000/qf \) for \( qf<50 \), \( scal=200-2\cdot qf \) for \( qf\geq 50 \), quality factor \( qf \in [1,100] \) and \( \lfloor \cdot \rfloor \) represents rounding down.

function [Qtab] = QuantizeTab(qf)
if (qf < 50)
scal = 5000 / qf ;
else
scal = 200 - qf * 2 ;
end
Qtab_new = floor (((QtabJPEG .* scal) + 50) ./ 100) ;
idz = find (Qtab_new <= 0) ;
Qtab_new (idz) = zeros (size(idz)) ;
idz = find (Qtab_new > 255) ;
Qtab_new (idz) = 255*ones (size(idz)) ;
Qtab = Qtab_new ;

The DCT based compression system is limited by a block based segmentation of the source image. The higher compress ratio the higher is image degradation.

Each coefficient is coded by variable length coding into the form of [number of zeros before non-null coefficient, value of non-null coefficient] and stored to a file.

function store(coeff, fout)
[Qtab, zig] = initCts ;
coeff_vector = coeff(:) ;
vector = coeff_vector (zig) ;
n = 0 ;
frame = [vector(1)] ;
for j = 2 : 64,
if vector(j) ~= 0
frame = [frame vector(j)] ;
n = 0 ;
else
n = n + 1 ;
end
end
frame = [frame 0 0]' ;
fwrite (fout, frame, 'integer*1') ;

Last part of the encoding/decoding chain is the Inverse DCT transform. By analogy with the Forward DCT we can write down the source code as follows.

for i = 0 : (nbx - 1) / 8,
for j = 0 : (nby - 1) / 8,
coeff = loadDCT (fin) ;
_block = (mYV*(mNORM.*_coeff)*mUX) ;
_img (i*8+1 : (i+1)*8, j*8+1 : (j+1)*8) = _block + 128 ;
end
end

In JPEG the degree of compression is determined by a quality factor. Decreasing the quality factor leads to coarser quantization, which gives higher compress ratio and lower decoded image quality. In this way the quantize matrix can be re-counted by (4) or by function QuantizeTab.
2.2 Compression Quality Appreciation

The fundamental aspect in testing image compression methods is a final quality examination as well. We used the Normalized Root Mean Square Error (NRMSE) defined by (5).

\[
NRMSE = \sqrt{\frac{\sum_{i=0}^{n_{bx}-1} \sum_{j=0}^{n_{by}-1} (p_{i,j} - \hat{p}_{i,j})^2}{\sum_{i=0}^{n_{bx}-1} \sum_{j=0}^{n_{by}-1} (p_{i,j})^2}}
\]

where \( p_{i,j} \) is the intensity of picture element and \( \hat{p}_{i,j} \) is the intensity of picture element of the reconstructed image when MATLAB source code is suggested below.

```matlab
NUM = img - _img ;
NUM = NUM .^ 2 ;
num = sum (NUM(:)) ;
NRMSE = sqrt (num / sumsqr (img)) ;
```

3. Results

For testing a computation speed of the image compression system we used seven test images (368×192, 512×512, 512×768, 576×720, 8 bits/pixel) with different spatial and frequency characteristics: Barbara, Boat, Goldhill, Lena, Monarch, Sail and Spout.

<table>
<thead>
<tr>
<th>File name (* .bmp)</th>
<th>Dim.</th>
<th>Number of 8x8 blocks</th>
<th>Original algo. time [s]</th>
<th>Proposed algo. time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>barbara</td>
<td>576x720</td>
<td>6,480</td>
<td>159.875</td>
<td>0.890</td>
</tr>
<tr>
<td>boat</td>
<td>512x512</td>
<td>4,096</td>
<td>100.750</td>
<td>0.531</td>
</tr>
<tr>
<td>goldhill</td>
<td>576x720</td>
<td>6,480</td>
<td>159.547</td>
<td>0.891</td>
</tr>
<tr>
<td>lena</td>
<td>512x512</td>
<td>4,096</td>
<td>101.234</td>
<td>0.515</td>
</tr>
<tr>
<td>monarch</td>
<td>512x768</td>
<td>6,144</td>
<td>151.188</td>
<td>0.781</td>
</tr>
<tr>
<td>sail</td>
<td>512x768</td>
<td>6,144</td>
<td>151.187</td>
<td>0.781</td>
</tr>
<tr>
<td>spout</td>
<td>368x192</td>
<td>1,104</td>
<td>27.125</td>
<td>0.109</td>
</tr>
</tbody>
</table>

Tab. 1. Computation speed comparison (IDCT).

Algorithm proposed in Section 2.1 is compared with the Forward DCT transform definition (1, 2). Computation times for both approaches for different images are shown in Tab. 1 and were tested in PC (AMD Athlon XP 2700, 512 MB of RAM) operating with WindowsXP and MATLAB 6.1.

As shown above the original algorithm in untreated version is very demanding in terms of time while for all test cases the proposed algorithm is almost two hundred times faster and allows image compression with irrelevant latency.

Acknowledgements

The research described in the paper was financially supported by the Czech Grant Agency under grant No. 102/03/H109 and by the research program MSM 262200011.

References


About Authors...

Tomáš FRÝZA was born in 1977 in Nový Jičín (Czech Republic). He received the Ing. (M.Sc.) degree in electrical engineering from the Faculty of Electrical Engineering and Communication, Brno University of Technology (FEEC BUT) in 2002. At present he is a Ph.D. student at the Dept. of Radio Electronics, FEEC BUT.

Stanislav HANUS was born in Brno, Czechoslovakia, in 1950. He received the Ing. (M.Sc.) and CSc. (Ph.D.) degrees from the Brno University of Technology. He is Associate Professor at the Dept. of Radio Electronics, Faculty of Electrical Engineering and Communication in Brno. His research is concentrated on Mobile Communications, Television Technology and Circuit Theory.