Implementation of the DCT transform with application to the JPEG transform of test images

François Chaplais

April 16, 2012

1 DCT transform in 2-D

1.1 Theoretical result

We recall this first result

Proposition 1. *The two dimensional DCT of $m \times n$ matrix A is the product*

$$\widehat{A} = C_m A C_n^T \tag{1}$$

where the matrix C_N has elements

$$C_N(k,r) = u_k \cos\left(\frac{\pi}{N} k \left(r + \frac{1}{2}\right)\right) \tag{2}$$

with $u_0 = \sqrt{\frac{1}{N}}$ and $u_k = \sqrt{\frac{2}{N}}$ for $k > 0$. The inverse DCT is computed by

$$A = C_m^T \widehat{A} C_n \tag{3}$$

1.2 Matlab exercise

The JPEG transform uses the DCT on blocks of 8×8 pixels, so it is profitable to precompute C_N for $N = 8$. In Matlab, write a script that compute C_8 using (2) and store the result as a variable named C8 in a file with the same name. Be careful that Matlab starts its indices at 1.

Check that the transpose of C_8 is indeed the inverse of C_8 (if not, check your script).

2 Getting images on the internet

An interesting set of images can be found at
http://sipi.usc.edu/database/ Of particular interest ar the
Lena image

and the
Mandrill image
because both have been used as benchmarks in image processing. These are TIFF, uncompressed images. Download the two images. Each image is 791 KB in size.

2.1 Importing into Matlab

Using the "Import Data..." item of the "File" menu, import the TIFF images. Each one is imported as $256 \times 256 \times 3$ array of 8 bit unsigned integers, i.e. each pixel is coded on three values between 0 and 255. It is highly recommended that you rename the variable to a more descriptive name. To display an image , use the "figure" command and type

```
image(imageName);
axis image; % for correct aspect ratio
```

to display the image stored in the variable `imageName`.

Figure 1 shows the Lena image, and figure 2 shows tha Mandrill image.

Figure 1: The original Lena image, scanned in the 70's. This probably explains the lack of saturation in the image

Figure 2: The Mandrill image. The hairy face makes it a challenge for accurate compression

2.2 Color conversion

JPEG uses the color space YCBCr, as the RGB space is very redundant. The Y channel is the luminance channel and provides the grayscale version of a color image. The Cb and Cr are chrominance channels.

The conversion from RGB to YCBCr is (up to truncation errors) given by

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} \begin{pmatrix} R \\ V \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix} \tag{4}$$

The provided Matlab routine

```
[Y,Cb,Cr]=convertYCbCr(x)
```

performs the conversion.

The inverse transform is given by

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.000 & 0 & 1.402 \\ 1.000 & -0.34414 & -0.71414 \\ 1.000 & 1.772 & 0 \end{pmatrix} \begin{pmatrix} Y \\ Cb - 128 \\ Cr - 128 \end{pmatrix} \tag{5}$$

The provided Matlab routine

```
z=convertYC2RGB(Y,Cb,Cr)
```

performs this conversion.

2.2.1 Matlab exercise

You will convert the two image to grayscale images using the formula (4) and keeping only the Y channel. The output will be a 256×256 matrix of unsigned 8 bit integers. Name the new images "GrayLena" and "GrayMandrill".

To preview the images (the GrayLena image for instance), type

```
figure; colormap(gray(256));image(GrayLena);axis image;
```

3 JPEG encoding

3.1 Block divide

The first task is to divide the grayscale images into blocks of 8×8 pixels. Fortunately, the dimension of the images are multiple of 8, so we will not have to pad the image with zeros. This means we will have an array of size $8 \times 8 \times 32 \times 32$.

3.1.1 Matlab exercise

Write a function that will split an matrix of size 256×256 into an array of size $8 \times 8 \times 64 \times 64$. Name the function "splitImage".

If the split version of the Lena Grayscale image is called `splitLena`, then accessing the block of coordinates `i,j` is done by

```
myblock = splitLena(:,:,i,j)
```

3.2 Matlab exercise: Block DCT

Now use the `C8` matrix to perform a block DCT on each of the split images. Script this into a function called "blockDCT". Remember: to address a block in a split image, put colons into the 2 first coordinates.

3.3 Quantization

A quantization matrix for the JPEG encoding is

$$\texttt{DCTQ} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 194 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 121 & 100 & 103 & 99 \end{bmatrix} \tag{6}$$

This quantization matrix can be multipied by a scaling factor r that will affect the compression performance and the quality of the restored image after decompression. Let $T = r \times$ `DCTQ`

The quantization step consists in replacing each element $x_{i,j}$ of an image block by

$$y_{i,j} = \text{round}\left(\frac{x_{i,j}}{T_{i,j}}\right) \tag{7}$$

3.3.1 Matlab exercise

Write a function named "JPEGquant" that takes as argument a scaling factor r, the qantization matrix DCTQ and a block split DCT of an image to produce the quantized block split image.

Note: the DCTQ matrix is stored in the matlab file "DCTQ.mat", so you can retrieve it by executing

```
load DCTQ;
```

3.4 Compression

It seems that Matlab does on the fly compression when saving .mat files. Let us consider the original "GrayLena" array and the quantized DCT of it that we have called "quantLena" (with $r = 1$). Both arrays take the same space in Matlab's workspace, i.e. `262144 uint8`. However the "GrayLena.mat" file takes 217 Kb on disk, its zipped version takes the same space, while the file "quantLena.mat" takes 61 Kb of disk space, as its zipped version does.

In short, Matlab already compresses the .mat files. We can use another approach with sparse matrices to store the JPEG encoding.

3.4.1 Matlab exercise

Write a function "sparseSave" with arguments the previous quantization and a filename, that does the following:

- After the block DCT has been quantized, convert back with the "mergeImage" as defined in section 4.3
- convert the result to double format
- use the sparse function to convert the matrix where zero entries are ignored
- save the sparse matrix to the disk. This will be useful for the tests.

4 JPEG decoding

4.1 Rescaling the data blocks

Remember that we have quantized the DCT by

$$y_{i,j} = \text{round}\left(\frac{x_{i,j}}{T_{i,j}}\right)$$

To recover the DCT (up to the quantization), we have to modify each block according to the formula

$$x_{i,j} = y_{i,j} T_{i,j} \tag{8}$$

where T is the quantization matrix DCTQ multiplied by the scaling factor r.

4.1.1 Matlab exercise

Write a function "rescaleQuant" which takes the same arguments as "JPEGquant" and does the rescaling Do not forget to convert the input signal to double, as r is double. The output is double.

4.2 Inverse DCT

The inverse DCT is computed with the same matrix C_N.

Proposition 2.

Let C_N as in proposition 1. Then the inverse DCT is given by

$$A = C_N^T \widehat{A} C_N \tag{9}$$

4.2.1 Matlab exercise

Write a line of code that computes the inverse block DCT using the routine "blockDCT".

4.3 Merging the blocks

We need now to merge the reconstructed split image into a conventional image. To do so, write a function "mergeImage" modeled after "splitImage" that recombines the blocks. The output must be uint8.

5 Collecting the work

5.1 Grayscale JPEG

Write 2 functions, "JPEGencode" and "JPEGdecode" that performs the coding and decoding with a scaling r.

Write a script that encodes an graysale image, decodes it and displays the resulting image. Play with r to test the quality.

See figures 3, 4 and 5 for comparisions. Try the Mandrill image.

Figure 3: The original grayscale Lena image. The file is 215490 bytes on disk

5.2 ColorJPEG

The compression scheme in the color space does not rely on the quantization of the DCT, but rather on the similarity (or not) of the chrominance of neighboring pixels. In that, it is very close in spirit to the Haar transform that we shall see in wavelets. We will implement a Haar variant to compress the chrominance signal.

After looking at the implementation of the wavelet transform, you will approximate (and later compress) color images the following way

- convert the image from RGB to YCbCr
- JPEG compress the Y channel by quantizing its DCT as explained before
- for each of the two Cb and Cr channel, perform a Haar decomposition of the channels at 2 scales. The Haar filters are obtained by loading the `Daub2.mat` file, and using two scales means that you are working on blocks of 4×4 pixels.
- depending on a number n that you decide, you only keep the n largest

Figure 4: The compressed Lena grayscale image with $r = 1$. The sparse matrix file is 34405 bytes on disk.

wavelet coefficients. In the Haar basis, this means you keep only the largest variations in chrominance, either from pixel to pixel, or from one 2×2 block of pixels to another.

- reconstruct the Cb and Cr channels from their wavelet transform
- combine the three channels to obtain an RGB image

This is *very* similar to the scheme in figure 6 that you will find on Wikipedia. In figure 7 you will fing a color JPEG approximation of the Mandrill image. Figure 8 shows the approximation of the Lena image with $r = 2$ and the others parameters being the same as for the Mandrill image.

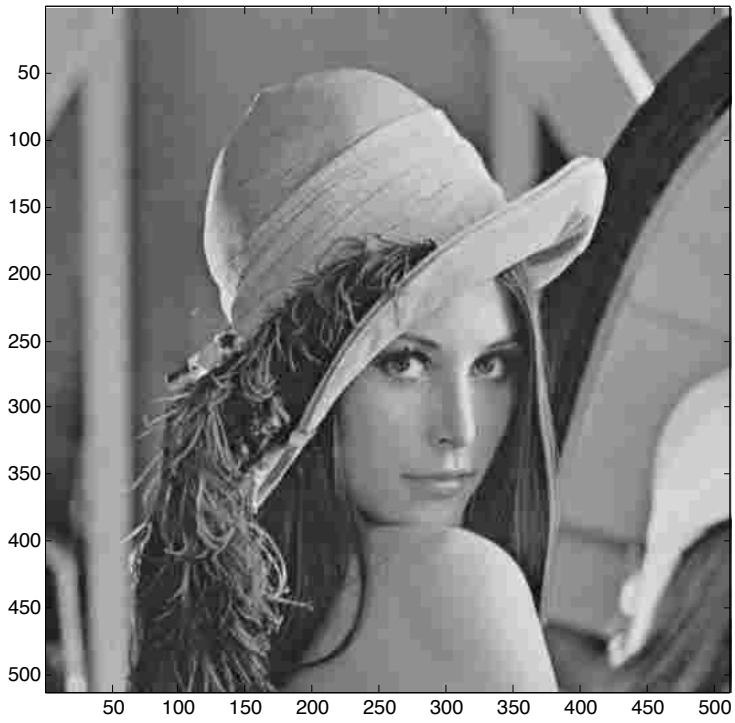

Figure 5: The compressed Lena grayscale image with $r = 4$. The sparse matrix file is 12855 bytes on disk.

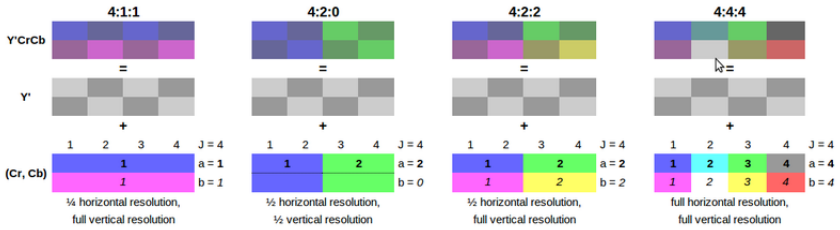

Figure 6: The JPEG algorithm averages the chrominance of neighboring pixels if the deviation is not too large. This is similar to a nonlinear approximation in the Haar basis.

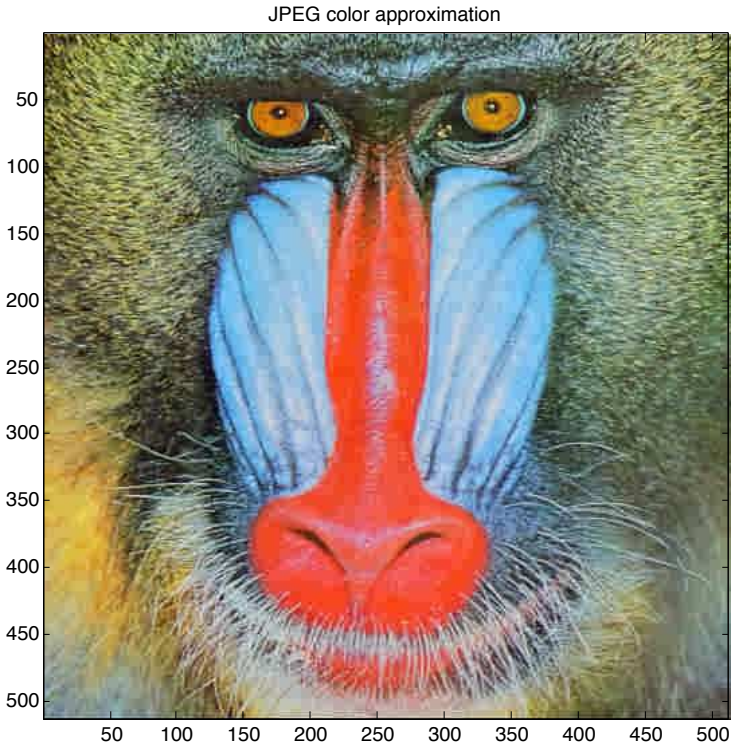

Figure 7: A color approximation of the Mandrill image. The grayscale r parameter has been set to 4. On the color channels, only 1 out of 20 Haar wavelet coefficients have been kept (meaning the other have been set to zero). Setting a wavelet coefficient to zero means that you neglect the variations between neighboring blocks of pixels. In practice, this means that the size of the color channel images has (potentially) been divided by 20.

Figure 8: A color approximation of the Lena image. The grayscale r parameter has been set to 2. On the color channels, only 1 out of 20 Haar wavelet coefficients have been kept (meaning the other have been set to zero).