An introduction to JPEG2000-like compression using MATLAB

Arno Swart

October 30, 2003

1 Introduction

The JPEG-2000 format is the official successor to the popular JPEG image compression format. The JPEG-2000 standard is more complex and has more functionality than JPEG. Some features of JPEG-2000 are

- State of the art in lossless and lossy compression using wavelets.
- Random access to the bit stream and 'region of interest' coding.
- Processing of compressed data (rotating, cropping, etc.)
- Inclusion of meta-data (dpi for printing, copyright information, etc.)

As yet a total of 12 parts have been submitted for standardisation, we will only focus on part 1: the core coder. Of this part we will only discuss a subset since we are primarily interested in the compression performance (and the gain in) using wavelets.

This document comes with a MATLAB implementation of the subset of JPEG-2000 that is discussed here. Throughout the text there are several experiments you can try for yourself.

2 Tiling, YUV transform and the DWT

The first step is the same as in ordinary JPEG: if the image consists of three color components a transform to the YUV color space can be made. Actually, JPEG-2000 defines a more general *Reversible Component Transform* for use in lossless compression. A component does not need to be a color channel, it can also be an opacity channel or separate foreground and backgrounds. But we will not discuss these more general aspects. Next, the Y, U and V components are subdivided in rectangular tiles, all of the same size except maybe at the boundary. The same subdivision for each component. These tiles are not needed for good performance of the wavelet transform (wavelets enjoy a nice localisation property), they are included to allow for fast access to different regions of the image. One could also use just one tile.

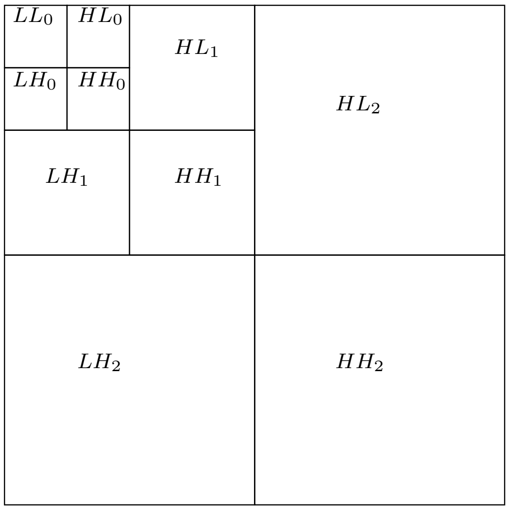

Figure 1: The distribution of the sub-bands over the matrix of wavelet coefficients. In this example a three level DWT is used.

Question 1 *Try this for yourself, use different tile sizes and see how it affects the compression ratio.*

Next an L-level dyadic discrete wavelet transform is applied to each tile. For lossy compression the standard proposes the bi-orthogonal $(9,7)$ wavelet. Wavelets come in different scales, the lowest level with least detail is at level zero, while the highest level is L. A scale is commonly referred to as a *sub-band*. A sub-band is assigned the name LL, LH, HL, HH with a subscript indicating the level. The H and L stand for high-pass and low-pass in the horizontal and vertical direction. See picture (1) for the location of the sub-bands in the coefficient matrix, for a particular component.

Question 2 *Do you understand why the pictures in the sub-bands look like scaled versions of the original picture? What do the different intensities mean in the LH, HL and HH bands?*

Question 3 *Do discrete wavelet transforms on the example pictures. Use different levels, try do do backward wavelet transforms with less levels. What do you notice?*

3 Boundary effects

The figure (1) is very suggestive. One might think that we can fit all wavelet coefficients in one picture of the width and height of the original untransformed

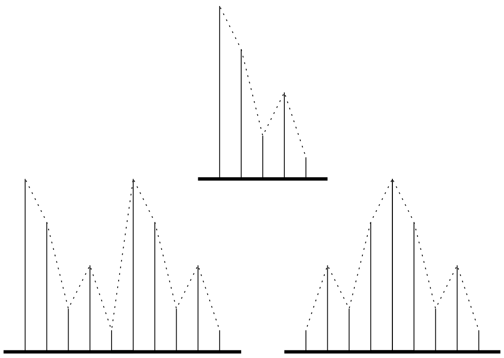

Figure 2: The signal (top), with periodic extension (left) and symmetric extension (right)

picture. This is not true because of boundary effects. To make the exposition simpler we will look at a 1D signal. If the filter length is M and the signal length is N then the number of coefficients after one transform is $N + M - 1$. More wavelet levels makes the situation worse. We could use *periodic extension*, which means we extend the signal with it's own copy at the boundary. In view of figure(2) we see that we can get discontinuities, and these are not very well compressible.

A better alternative is to use symmetric extension, as in figure(2). This does not give discontinuities. A new problem arises: adding a symmetric copy to a signal doubles the length. This doesn't really matter for the signal itself (we only store half of it) but it also doubles the number of wavelet coefficients: in general, the wavelet coefficients do *not* form a *symmetric signal*. Symmetry is only guaranteed if the analysing wavelet is symmetric or antisymmetric. The only orthogonal candidate (which is real valued and compactly supported) is the Haar wavelet. Giving up orhogonality offers more possibilities.

In JPEG-2000 the wavelet of choice has become the Daubechies (9/7) biorthogonal wavelet. This wavelet is symmetric and almost orthogonal. Of course there are more wavelets that could have been used, yet this wavelet gave best results in compression tests.

Question 4 *Try some other symmetric wavelets. Do you notice anything special at the boundary? Verify that for the Haar wavelet the number of coefficients is indeed exactly the number of pixels.*

4 Quantisation

Now every sub-band of every transformed tile of every component is quantised. Quantisation is the main source of compression. The quantisation is somewhat simpler than the JPEG quantisation tables: every sub-band is uniformly quantised. One fixed quantisation step (the number by which you divide the wavelet coefficients) is allowed per sub-band, the result is rounded down. The MATLAB code allows you to give every sub-band and the LL_0 band it's own quantisation step.

5 Lossless compression

The subsequent lossless compression that we will use is not JPEG-2000 standard. The reason for this is simplicity that allows use to use the 'lossless part' of our JPEG code. A brief outline of the official subdivision and compression scheme is given in the next section. Readers who wish to explore these options in more detail are referred to the links on the course's internetpages.

We will simply divide each sub-band into code blocks of size 8×8, if needed we append zeros to create multiples of eight. Each code block will be zero run length and Huffman coded, as in the original JPEG scheme. The use of the zigzag scheme is not useful.

Question 5 *Make some graphs of compression ratio's against mean squared errors. Is it better than the 'old' JPEG? Note that mean squared error is not the only possible measure. The error's made in JPEG-2000 are different than those in JPEG. Try also making some visual comparisons between JPEG-2000 and JPEG for comparable error rates.*

6 Brief description of further JPEG2000 compression [optional]

The tiling of the image allows for (coarse grained) localisation in image space. JPEG-2000 defines a further subdivision in rectangular *packet partitions*, this gives a localisation in wavelet-coefficient space. Every packet partition is then divided in rectangular *code blocks*. These code blocks are the basic units that will be entropy coded with a clever context-based coder called the MQ-coder.

7 Further features of JPEG-2000

A nice feature of JPEG-2000 is the possibility to store parts of the code blocks in *layers*. Each layer contains some bits of every code block. The more layers you send, the better image quality you get. You could for example have a 50 layer high quality file. If you want to publish it in low quality on the internet you send only 10 layers. Another possibility is storing a foreground and a background in separate layer sets, so that they may be split later. This is very useful for scientific visualisation purposes. Finally the layers can be decoded in parallel, should you have a huge image file (and a supercomputer).