# Comparison of Wavelet and Discrete Cosine Transform compression methods

ASM 420GU Project in Image Processing, Project 2

Matz Johansson Bergström
matz.johansson@chalmers.se

October 14, 2013 (corr)

## Summary

The purpose of this project is to compare methods of compressing an image using two wavelet packet transforms (WPT) *bior 4.4* and *rbio 6.8*, and the Discrete Cosine transform (DCT). Also, the compression artifacts and the reason for why they appear will also be discussed.

The wavelet packet transform using *rbio 6.8* provides a smaller $e_{rms}$ error after compression than *bior 4.4* and DCT. WPT are very good at masking edge artifacts and higher frequency errors because of the adaptive way it analyses the signal it transforms.

By comparison, DCT is a very poor algorithm to compress images, because it is static. WPT can be compressed by a factor of 20 and get the same quality result as DCT using a factor 8.

## 1 Introduction

Compressing an image could be accomplished using either a lossless or a lossy compression algorithm. Lossless compression is usually associated with zip files, the complete data is retained after compression. Compressing images is very different because the eye (and brain) is forgiving of small errors in different situations. Depending on the compression technique, an image can be compressed by a large factor without any noticable difference in image quality.

In this project the purpose is to compare how well two different lossy compression methods can compress three gray scale images (see Figure 1.1).

The images were carefully chosen to push the compression algorithms to the limit with high frequency detail, sharp edges and large secions with uniform colors. The images also look very different and thus represent a wide range of types of photographs. The leftmost photo is of the pianist Glenn Gould[1]. The image was taken in may 1957 as part of his Russian tour. The image consists of a back-lit Glenn Gould with very sharp edges and the silhouette itself is uniform and will reveal if any artifacts are produced in low frequency parts of a compressed image.

The rightmost photo is of a small pig[2] featuring hair with very high frequency and a background with out-of-focus blur. The high frequency and low frequency in the same image.

[1] Source: Library and Archives Canada/Glenn Gould fonds/MUS 109-230
[2] Source: http://eofdreams.com/photo/pig/03/

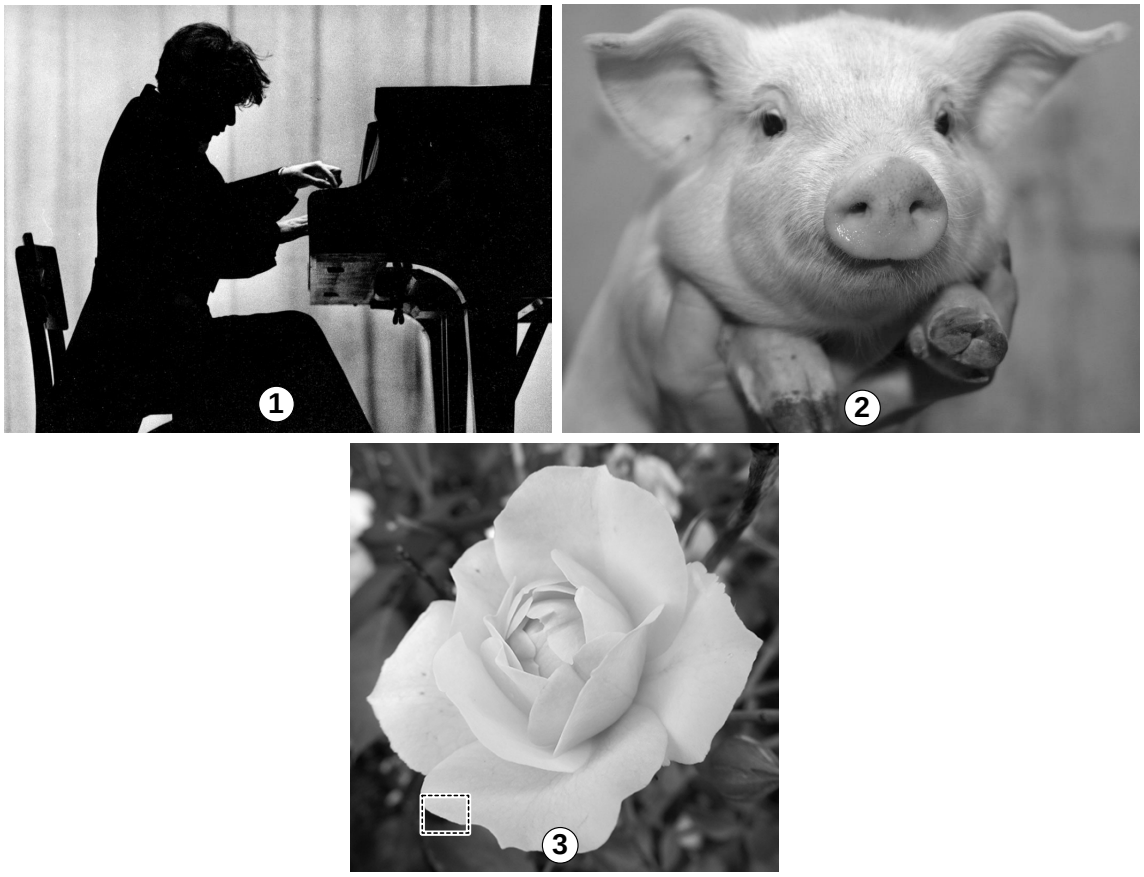


**Figure 1.1:** ① - Pianist Glenn Gould shown in a silhouette by the piano. ② - A photograph of a pig. Please note the detail of the hair and the smooth section of the image that is out of focus. ③ - A photograph of a rose, taken by the author. In the photo, a rectangle is indicated and will be zoomed in to in later figures.

### 1.1 Evaluation methods

The error measurement used in this report is the Root mean squared measurement. To compare between original and compressed image, the root mean squared error is defined as

$$e_{rms} = \sqrt{\frac{1}{MN}\sum_{x}^{M-1}\sum_{y}^{N-1}[I(x,y) - \widetilde{I}(x,y)]^2}. \tag{1}$$

The *compression factor* used in this report is defined as the total number of coefficients (for DCT) divided by the number of coefficients we keep, so

$$\text{compression factor} = \frac{100}{100 - \text{percent of zeroes}}. \tag{2}$$

## 2 Results

This section presents the results of the examination of the compressability of DCT and then wavelet packet transforms separately. Each section has a conclusion and discussion and the report ends with a conclusion section to wrap up all the conclusions that were mentioned in the previous sections.

The codes for most of the figures and results can be seen in the Appendix section on page 12.

## 2.1 Compression using the Discrete Cosine Transform

The Discrete Cosine transform (DCT) is closely related to the Fourier transform. As with the Fourier transform, the DCT assumes the input image is periodic. Since photos are not periodic and often discontinuous, the periodicity and continuous properties of the DCT will begin to surface as image artifacts. To combat this issue the image is divided into non-overlapping blocks, of a specific size ($8 \times 8$). Each block is transformed using DCT and then compressed. In this report the compression is simply setting a fixed number of the smallest (in absolute value) coefficients to zero. This is done for each block over the image. This operation is basically removing high frequency detail.

Worth mentioning is that in the JPEG standard [Wal91], which uses DCT for compression, the compression step use a quantizer matrix, Q, to modify the value of the blocks, truncation step and then an encoding step, to store the non-zero coefficients in an efficient way. Please note: To reproduce the results in this section, the code provided in the course, **dctdemo**, calculates the $e_{rms}$, error (see Equation 1) in the wrong way. The authors (Mathworks) takes the mean of the difference of the squares, instead of the mean of the square of the differences.

### 2.1.1 Choosing block size

The block size in the JPEG standard is $8 \times 8$. To understand why this size was chosen, we will look at the quality of the compression and the error $e_{rms}$ for blocksizes $4 \times 4$, $8 \times 8$, $16 \times 16$ and $32 \times 32$. By dividing the image into blocks, the result is that the error is instead localized to inside the boundaries of the block, thus supressing errors that would otherwise propagate over the image. In Figure 2.1 the rectangle in the flower photo of Figure 1.1 is zoomed in and 25 percent of the zeros are kept. In the difference images, which is the difference between the original uncompressed photo and the compressed images, we can see how the ripples are introduced. On each difference image, the contrast is adjusted using histeq. The important part is to see the tendency of the introduced noise for each image and not to compare them side-by-side.

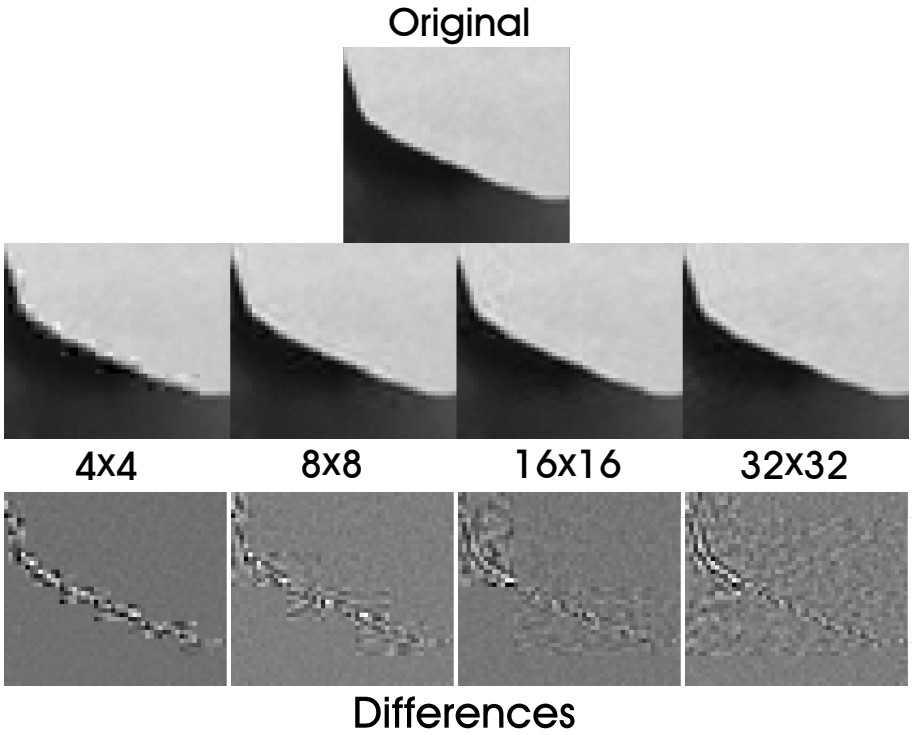


**Figure 2.1:** One of the petals from the flower photo is zoomed in and displayed on the top row. The first row shows the DCT compression. The bottom row shows the difference.

For smaller block sizes (of size $4 \times 4$), a strong blocking artifact will appear. The blocking artifact is much stronger than the ringing artifact, so it will give a larger $e_{rms}$ error. An example of this can be seen in figure 2.2. For block sizes of $2 \times 2$ (not shown), keeping 25 percent of the coefficients, the only coefficient that is seen is the DC coefficient, which is the mean of the block. In this case the result is a halving of the resolution and not interesting from an image compression point of view.

The error seems to be smeared over the size of the block. We believe this is due to the resolution of the block. Much like with the FFT, we larger the input to the DCT, the smoother the output. We are removing the same percentage of zeros, but the fact that we have more coefficients that represents the edge, the smaller the error of this edge will become.

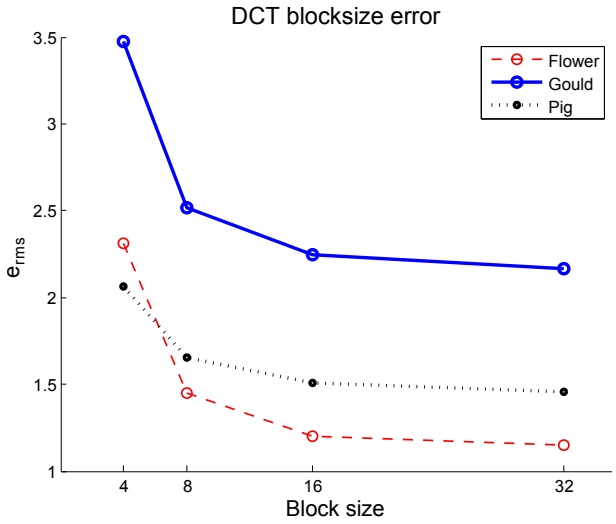


**Figure 2.2:** The error of the different block sizes.

Interesting to note is that the Gould photo contains larger error than the others. The reason for this is what is seen in the petal of the flower in Figure 2.1. By attempting to approximate a sharp edge, the DCT encounter the same problems as with Fourier synthesis. If an image has a discontinuity, a sharp edge, the simple discrete summation of cosines will never produce an accurate result. As a contrived example, the Figure 2.3 shows a $512 \times 512$ image of a black disc and the difference between the original and the compressed image to the right. The DCT blocksize is $128 \times 128$ and the compression factor (see Equation 2) used was 8, which means removing 87.5 % of the coefficients. In this simple example it is clear how these ringing artifacts are small, but they are introduced over the block size. So the block size needs to be small enough to contain this error but large enough to be able to reconstruct edges in a good way.

In Fourier analysis, the effect of ringing is called *Gibbs phenomenon*. The problem is that sharp edges require an infinite number of terms in the Fourier series to be represented exactly.

Please note that we could not have used FFT directly to do the same compression. By doing that, the block edges would introduce another discontinuity, which adds another ringing artifact. DCT handles block edges well by simply mirror the signal [GW08, page 573]. Since the computation used on the computer is not infinite, the summation of the cosine terms will always contain these ringing artifacts.

An interesting thing to note is the performance of the methods. The DCT of block size $4 \times 4$ took 4 times longer to finish than the $8 \times 8$. For the subsequent increasing blocksizes the time improvement was insignificant, if any. The reason for the performace impact with respect to block size is that the strength of FFT and DCT is to transform large vectors and matrices. The small overhead of calling DCT thousands of times becomes larger for smaller blocks. The overhead

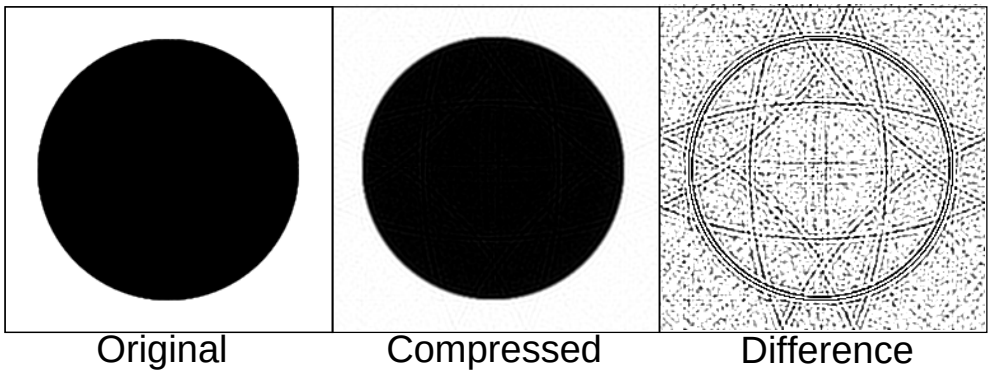


**Figure 2.3:** Testing DCT with large blocksize of a sharp edge image of a disc. Please note that the edges are repeating much like the propagating ripples in a water tank.

| Image | Method | Factor | $e_{rms}$ |
|---|---|---|---|
| flower | DCT | 8 | 2.8 |
| Gould | DCT | 8 | 4.4 |
| pig | DCT | 8 | 2.6 |

**Table 2.1:** Table showing $e_{rms}$ of DCT for compression factor 8.

may be is small, but scaled thousands of times. Another, related reason, is that DCT computation of large matrices use vectorization (SSE, SSE2 instructions) and these instructions are fast for large vectors.

### 2.1.2 Conclusion and discussion

The reason to use the standard $8 \times 8$ block size is to minimize both blocking and also minimize the ringing artifacts which are contaminating the rest of the block. A block size of $8 \times 8$ is a nice "middle ground" between the two artifacts. Additionally, the processing speed for compressing an image is also important. Image compression algorithms in digital cameras cannot take too long to finish or they will use up unnecessary amount of energy. An explanation to the poor performance for small block sizes is that the DCT computation has to be repeated more times than for larger blocks. DCT is a vectorized code, hence faster for larger vectors.

The error $e_{rms}$ seems to give a good indication of the overall error in the compressed image for DCT compressed images, but it is still important to check visually how good of quality the image has (we will see more of this in the next section).

Worth mentioning is that Matlab provides (with the image processing toolbox) the **blockproc** function which processes an image in blocks. By using the parallel version of **blockproc**, it is possible to speed up the blocking processing. In this report the uniprocessor version was used.

In Figure 2.4 the images are compressed by a factor of 8 and the (enhanced) difference image is shown. The blocking artifacts of the rim of the flower and at the edge of Gould is very high. The pig image has lots of errors surrounding the snout. This is because the snout is in focus, hence very high frequency detail is contained, which is difficult to compress.

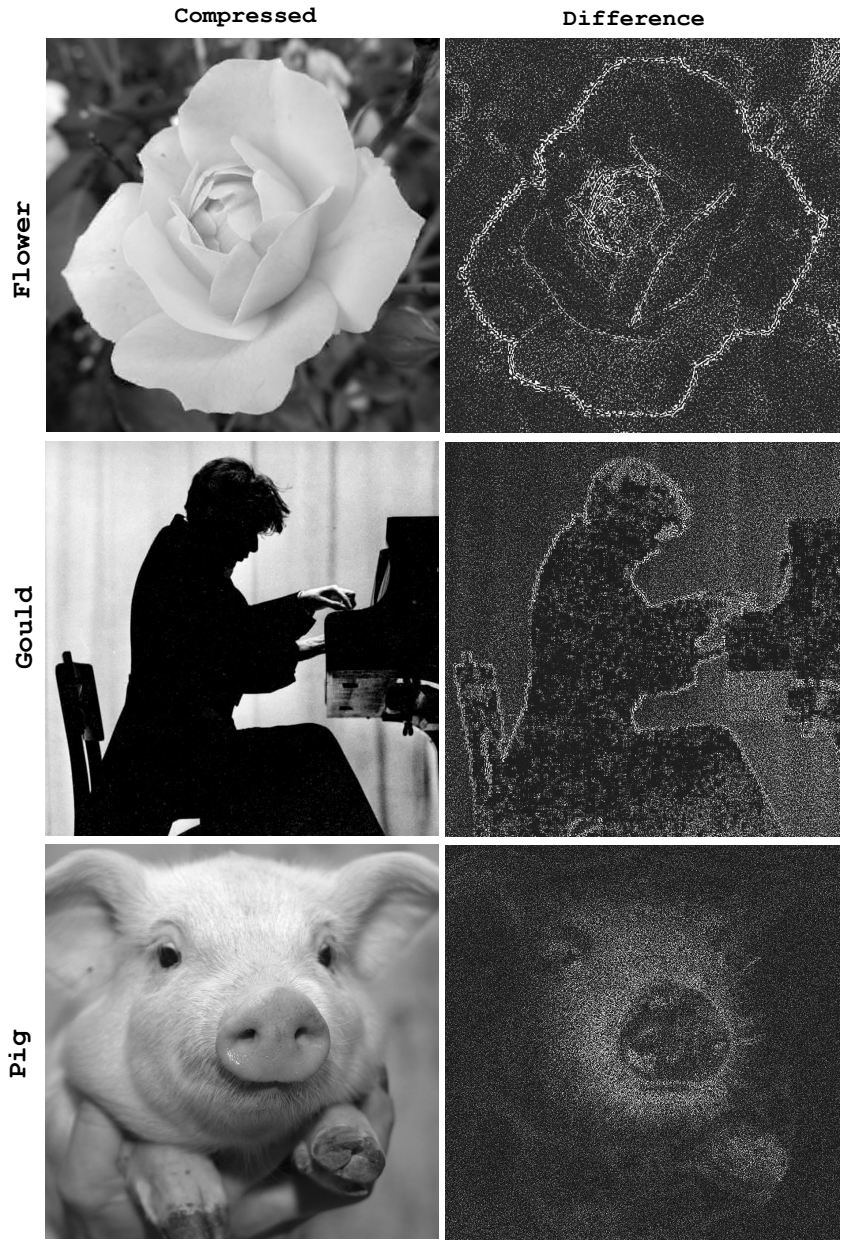


**Figure 2.4:** The stress test of the DCT compression algorithm. Notice how the edges are difficult to compress and that the high frequency details around the pigs snout is difficult to reconstruct.

## 2.2 Compression using the Wavelet Packet Transform

The wavelet packet transform (WPT) is a version of the ordinary wavelet transform. The main difference between the two methods are that the packet transform does not filter the data by halving the frequencies for each level, such as with the wavelet transform, instead it stores the filtered images (subbands) in a tree. The splitting is finished either when the cost criterion is met or when the depth is reached. In this way it adaptively finds the most efficient representation of the image [Lin04, page 20-21].

For the following experiments the maximum depth of the tree is 5. The comparisons are made between the DCT and two WPTs for compression factor 8 and 64. The wavelets used are *bior 4.4*, the "FBI wavelet" and the *rbio 6.8*.

In this project, the Matlab image analysis user interface **wavemenu**, was used to compress with the *wavelet packet transform*[3]. The compressed images were saved in lossless **.mat** files. The $e_{rms}$ error was then calculated in Matlab. Wavemenu returns 8-bit images of values $\in [0, 255]$, so normalization and conversion was performed. To reproduce the results seen below, the reader is kindly referred to the Appendix in Section 4.

Compression factor of 64 is the same as removing 98.44% of the coefficients. The first thing to notice is that the way the compression is performed in the previous section on DCT, the compression factor 64 will only keep the DC component of each $8 \times 8$ block. This means that the DCT compressed images of factor 64 will simply be a pixelation of the original image, so these images are omitted.

The first table (Table 2.2) shows the DCT and wavelet packet methods for compressing the flower image. The *rbio 6.8* is consistently better than *bior 4.4*. The DCT is orders of magnitude worse than both Wavelet packet techniques.

For the compression factor 8, $e_{rms}$ is very low for the wavelet packet methods. This is also evident as there are no discernable differences between these images[4].

The next tables (Table 2.3 and Table 2.4) shows the errors of the Gould photo and pig photo respectively.

| Image | Method | Factor | $e_{rms}$ |
|---|---|---|---|
| flower | bior 4.4 | 8 | 0.0057 |
| flower | rbio 6.8 | 8 | 0.0049 |
| flower | bior 4.4 | 64 | 0.016 |
| flower | rbio 6.8 | 64 | 0.012 |

**Table 2.2:** Table showing $e_{rms}$ of *bior 4.4*, *rbio 6.8* and DCT at compression factor of 8 and 64.

| Image | Method | factor | $e_{rms}$ |
|---|---|---|---|
| Gould | bior 4.4 | 8 | 0.01 |
| Gould | rbio 6.8 | 8 | 0.0076 |
| Gould | bior 4.4 | 64 | 0.024 |
| Gould | rbio 6.8 | 64 | 0.021 |

**Table 2.3:** Table with the error for the Gould image.

[3] Call the dwtmode('zpd') after you open wavemenu to add zero padding of the image

[4] These images were omitted.

| Image | Method | factor | $e_{rms}$ |
|---|---|---|---|
| pig | bior 4.4 | 8 | 0.0071 |
| pig | rbio 6.8 | 8 | 0.006 |
| pig | bior 4.4 | 64 | 0.015 |
| pig | rbio 6.8 | 64 | 0.013 |

**Table 2.4:** Table with the error for the pig image.

As mentioned above, the compression using the wavelet methods are very efficient and does not create any discernable artifacts for compression factor 8. The factor 64 is more interesting, the compression artifacts starts to reveal themselves in an interesting way. In the next set of figures we will see the three photos compressed using the wavelet methods and the (modified for visual quality, see code in appendix) difference between the original images and the compressed image for compression factor 64.

The flower in Figure 2.6 shows the largest leap in $e_{rms}$ in Table 2.2. This is possibly due to the soft gradients and relatively smooth edges, we believe.

We can also notice a ringing of the edge of the image. We do not know why this ringing artifact appears. It almost look like when you forget to pad the image when using FFT, but we did pad the image, changing the dwtmode to zero pad after we opened wavemenu.

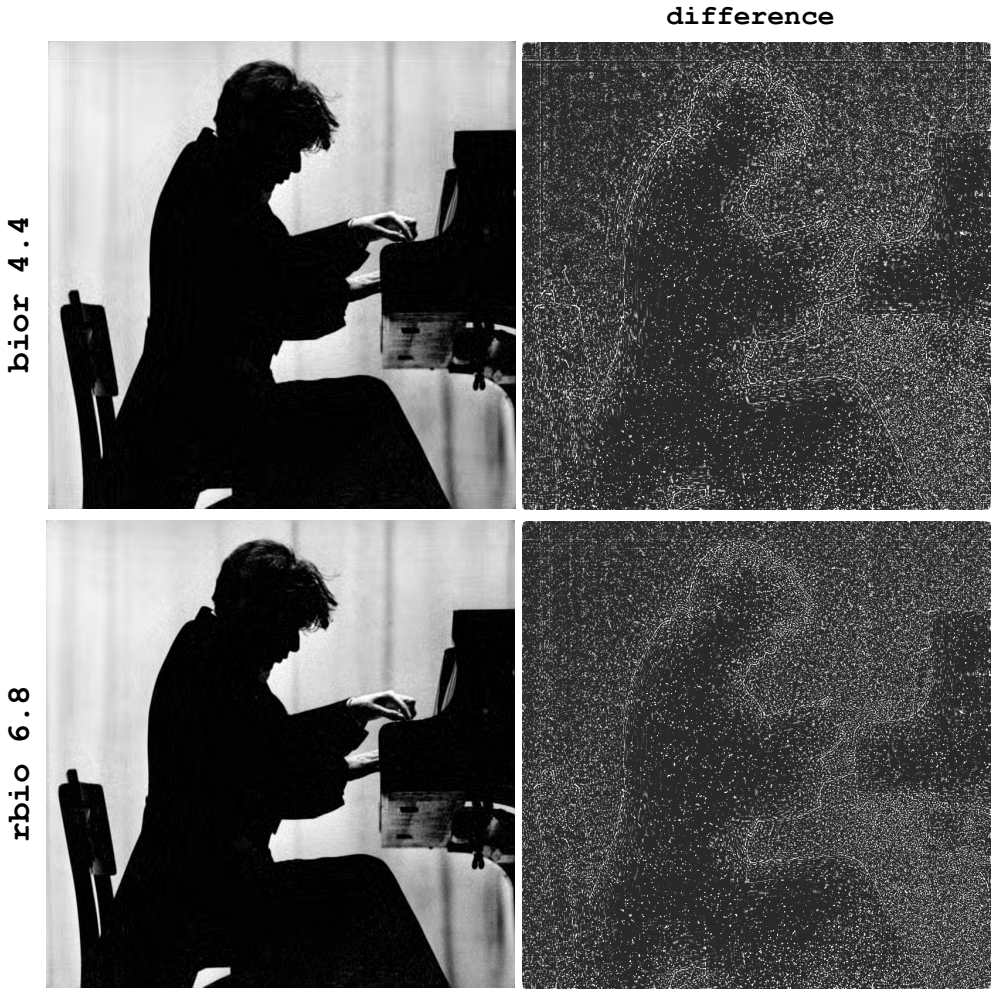


**Figure 2.5:** Compression (factor 64) of the photo of Glenn Gould. Notice the almost uniform noise over the difference image.

The Gould photo, see Figure 2.5, shows the difference error as a more uniform noise over the image, mainly focused at the edges and decreasing further in the black areas.

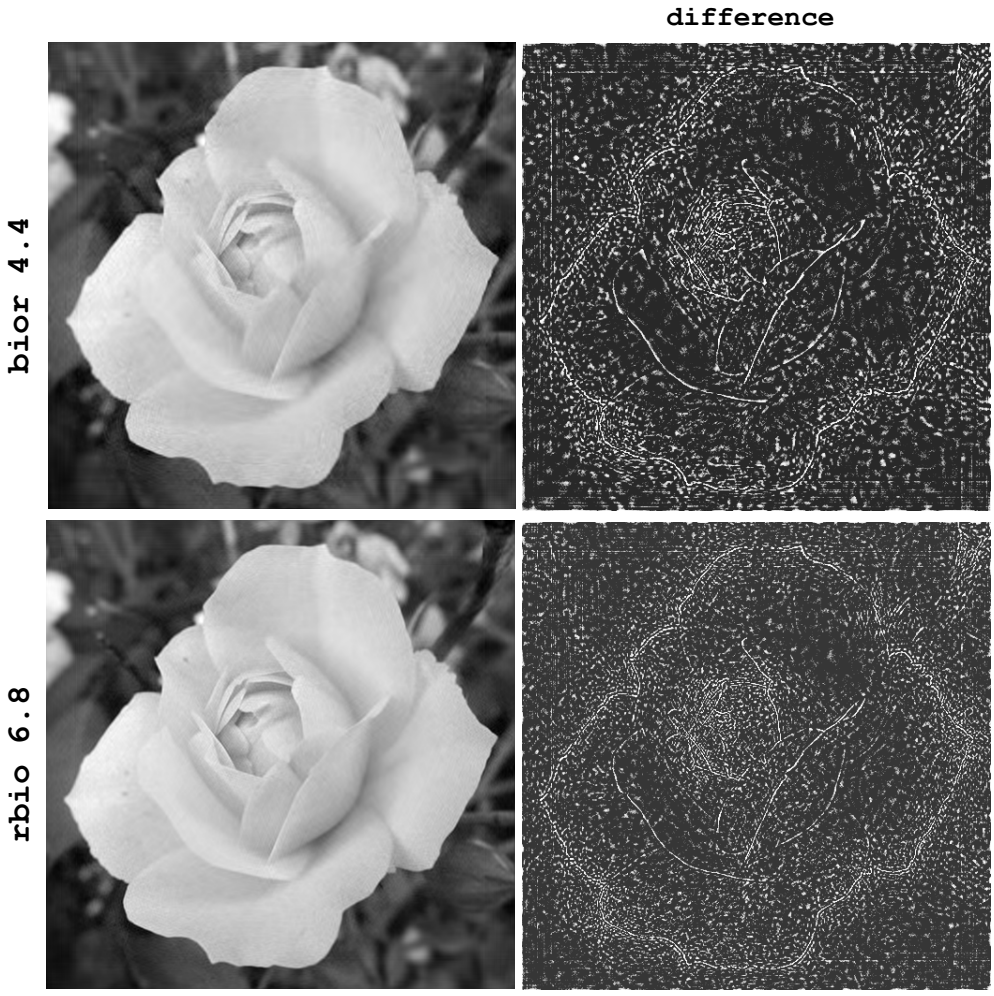


**Figure 2.6:** Compressing (factor 64) the flower using the two wavelet methods shows the different details in the difference image.

The last photo, of the pig, Figure 2.7, shows the high frequency area as in Figure 2.4, it is difficult to reproduce high frequency data removing 98.44% of the signal. Although the image is very complicated, the compression artifacts are almost invisible.

We believe that the artifacts are "buried" in the details, which is difficult for the human eye to discern from noise. So, in this case the pig image is best suited for compression with high frequency parts and very low frequency parts. The camera depth of field introduce the smooth low frequency background, the hair of the pig is in the camera focus, i.e. containing high frequency details.

### 2.2.1 Conclusion and discussion

The wavelet packet transform provide a high compression ratio and very small errors. There are differences between the two wavelet transforms we examined. The *bior 4.4* has 4 vanishing moments, compared to *rbio 6.8* which has 8 vanishing moments (for decomposition). It is this main difference that we believe is the reason why *rbio 6.8* is a better choice. The WPT is also good at hiding the noise, especially seen in the pig image above.

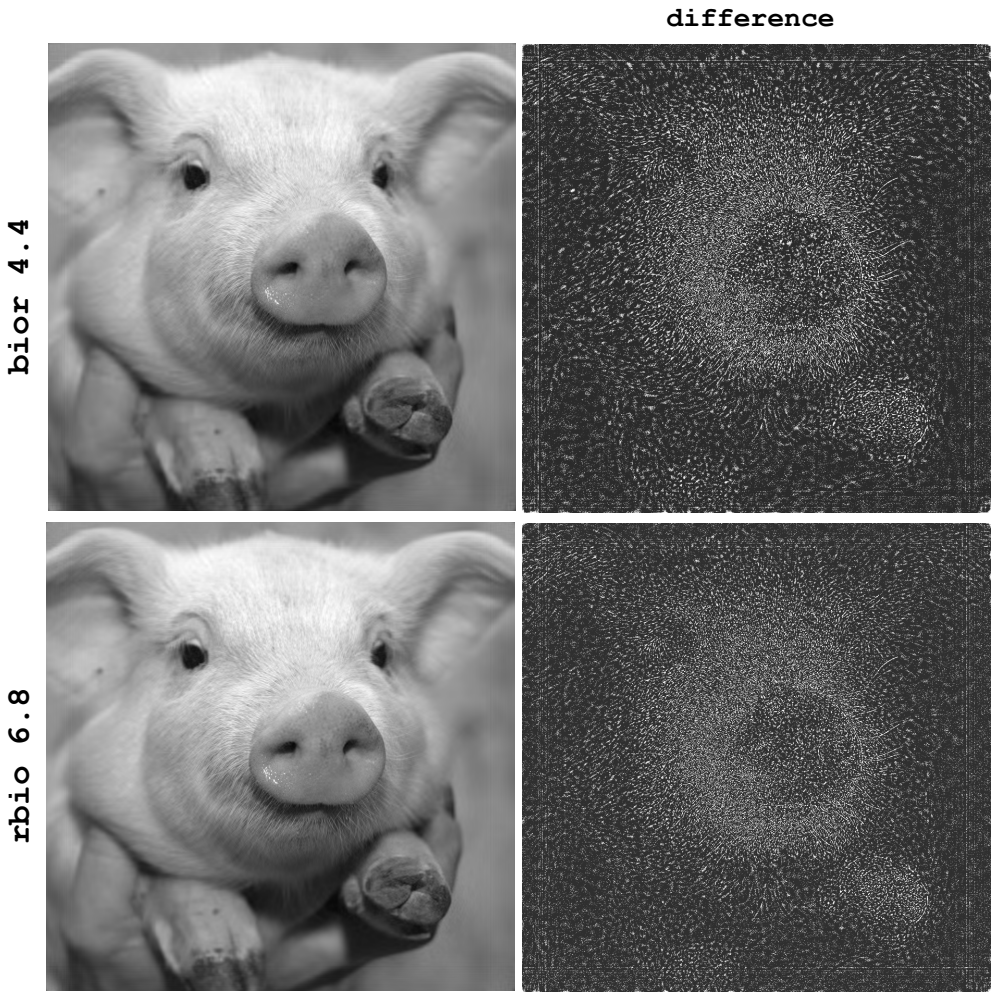


**Figure 2.7:** The compression (factor 64) of the pig has the least visible compression artifacts.

## 2.3 Comparison between DCT and Wavelet packet

We have seen compression using DCT and then the much better *wavelet packet transform*. The wavelet packet transform and its variants are better, they give a higher accuracy of the image, they supress and almost paint the artifacts around edges, making them less visible.

Even though we already know DCT is a poor compression algorithm, we will compare just how much it is possible to compress using WPT given a known $e_{rms}$ error from a DCT compressed image. The two images will then be compared side-by-side. The example we will use is the DCT image of the pig using a compression factor of 8. The $e_{rms}$ error was 2.6. By compressing the images using wavelet packet transform with *rbio 6.8*, the level of compression for which the error is the same as the DCT compression was found. The percentage of zeros was 97% for piglet, 93% for Gould and 98% for the flower. The lower the percentage, the harder it is to compress the image. This implicates that the Gould photo is harder to compress than the other two.

It is also of interest to compare how much an image can be compressed judging by visual quality. The results can be seen in Figure 2.8. By increasing the number of zeros, ringing from the edges of the image will resonate inside the image. By lowering the number of zeros to 95% the image quality is almost exactly the same as the DCT for the piglet.

The same is true for *bior 4.4*, although minor differences between the quality was observed.

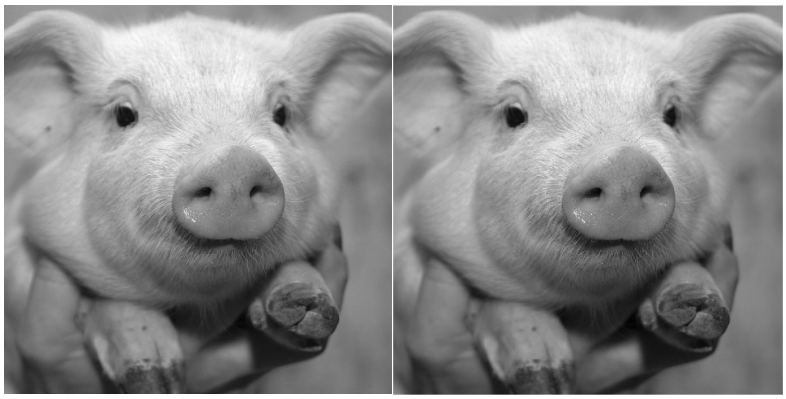


**Figure 2.8:** Compressing the image of the piglet to match the visual quality of the DCT 8.

# 3 Conclusions and Discussion

We conclude that wavelet packet transforms are the superior method of compression compared to DCT. The DCT compression introduce visible artifacts very quickly and has difficulty to reproduce sharp features in an image when a relatively small number of coefficients are removed. The DCT also introduce ringing artifacts within the boundaries of a block, which is apparent for high compression factors.

The wavelet packet methods are adaptive, giving a more compact (compressed) representation of the image. This allows us to remove more data without much loss of image quality. The artifacts are heavily revealed when compressing by a factor of 64. At that point it was evident that the better method was *rbio 6.8* as it clearly could approximate the image in finer detail than *bior 4.4*, although the differences were minor.

A reason for *rbio 6.8* being the better choice has (most probably) to do with that *rbio 6.8* has 8 vanishing moments, as opposed to the 4 vanishing moments of *bior 4.4*. This means that the *rbio* has the ability to more closely approximate the signal at each discontinuity. This is most clearly seen in the Figure 2.6. The errors are seen as spots on the flower, the *rbios* spots are smaller in size than *bior*. Further, visually the *rbio 6.8* gives a slightly sharper compressed image, with less disturbing compression artifacts.

The reason the wavlet packet transform is working so well in these conditions is that the high frequency parts of the images are not given much importance, which is good, because the human eye does not appreciate high frequency details. This can clearly be seen in the pig image Figure 2.7, where the high frequency details have lots of errors, but are thus not seen by the eye.

## References


[GW08] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson internation edition, 2008.

[Lin04] Anna Linderhed. "Adaptive Image Compression with wavelet packets and empirical mode decomposition". PhD thesis. http://www.icg.isy.liu.se/publications/Thesis_Linderhed.pdf: Linköpings University, 2004, p. 20.

[Wal91] Gregory K. Wallace. *The JPEG Still Picture Compression Standard*. Tech. rep. Digital Equipment Corporation, Maynard Masachusetts, 1991.


# 4 Appendix

In this section all the Matlab code can be seen in separate sections with some commentary. Since we believe it is important to be able to reproduce the results exactly, the code for most of the experiments in this report has been added to this appendix.

## 4.1 RMS error

**Listing 1: code/rms.m**

```
function k = rms(A, Aprime)
k = sqrt( mean((A(:) - Aprime(:)).^2) );
end
```

## 4.2 Zeroing the coefficients

The zeroing function has a global variable *keep_percentage*, which is defined outside. The function is called by blockproc and sets the smallest coefficients of each block of the DCT transformed image to zero.

**Listing 2: code/zeroblock.m**

```
%Sort the elements of the matrix and zero the 100-keep_percentage zeroes
function B  = zeroblock(A)
global keep_percentage

data        = A.data;
[u, v, ~]   = size(data);

nr_zeros    = round(keep_percentage/100*u*v);
[val, keep] = sort(abs(data(:)), 'descend');
keep        = keep(1:nr_zeros);
B           = zeros(size(data));
B(keep)     = data(keep);
```

## 4.3 DCT Compression

Very simple algorithm to compress an image using DCT. If the parallel computing toolbox is present, it is possible to speed up the blockproc function call by passing an extra argument, see code for details.

**Listing 3: code/DCT_compression.m**

```
global keep_percentage %global passed to zeroblock

%matlabpool 4 %uncomment to use parallel blockproc

%function handles to DCT and its inverse
f      = @(block) dct2(block.data);
finv   = @(block) idct2(block.data);

%the percentage of coefficient to keep
keep_percentage = 100-100*(factor-1)/factor;

if block_test
    keep_percentage = 25 %override for blocksize test
end

%ADD: 'UseParallel', 1); to use parallel blockproc
A = blockproc(I, [n_block, n_block], f);
C = blockproc(A, [n_block n_block], @zeroblock);
E = blockproc(C, [n_block n_block], finv);

tmp_rms = rms(255*I, 255*E);
%Note: The referenced book "Digital Image processing 3rd edition" use 8-bit...
     images range [0,255]
% to make rms comparable, we decided to scale the data to this intervall.
```

## 4.4 DCT Block test

**Listing 4: code/DCT_block_test.m**

```
global keep_percentage %used by function <zeroblock>

keep_percentage = 25;
block_test      = 1;
factor          = 8;

debug           = 0
block_sizes     = [4, 8, 16, 32];
rmss            = zeros(length(block_sizes), 3);

names      = {'flower', 'Gould', 'pig'};

for img = 1:3 %for each of the three images

    %OBS: image must be stored in 'RGB mode' in Photoshop
    fprintf(1, 'loading %s...\n', names{img});
    I = im2double(imread([names{img},'.tif']));

    I = rgb2gray(I);

    for i = 1:length(rmss) %run through the block sizes
        tic
        n_block = block_sizes(i);
        DCT_compression;
        rmss(i, img)  = tmp_rms;
        times(i, img) = toc; %save the timing

        figure
        imshow(E, [])
        title(['blocksize:', num2str(n_block)])
```

```
        if img==1 %flower

            figure
            E_cropped = E(411:455, 84:135); %zoomed in segment of each ...
                block test
            imshow(E_cropped, [])
            title(['blocksize:', num2str(n_block)])
            imwrite(E_cropped, ['Flower_', num2str(n_block), '.bmp'], 'bmp...
                ')

            if i==1 %create zoomed in segment of original image
                figure
                I_cropped = I(411:455, 84:135);
                imshow(I_cropped, [])
                title(['Original image'])
                imwrite(I_cropped, 'Flower_0.bmp', 'bmp')
            end
        end
    end
end


figure
title('DCT blocksize error', 'fontsize', 14)
hold on
h1 = plot(block_sizes, rmss(:, 1), 'rO--', 'linewidth', 1);
h2 = plot(block_sizes, rmss(:, 2), 'bo-', 'linewidth', 2);
h3 = plot(block_sizes, rmss(:, 3), 'ko:', 'linewidth', 2);

set(h1, 'markersize', 7)
set(h2, 'markersize', 7)
set(h3, 'markersize', 4)
set(gca, 'XTick', block_sizes)

xlabel('Block size', 'fontsize', 13)
ylabel('$e_{rms}$', 'fontsize', 13, 'interpreter', 'latex')

legend('Flower', 'Gould', 'Pig')

figure
title('Timing of blocksize performance', 'fontsize', 14)
hold on
plot(block_sizes, times(:, 1), 'rO--', 'linewidth', 1);
plot(block_sizes, times(:, 2), 'bO-', 'linewidth', 2);
plot(block_sizes, times(:, 3), 'ko:', 'linewidth', 2);

set(h1, 'markersize', 7)
set(h2, 'markersize', 7)
set(h3, 'markersize', 4)
set(gca, 'XTick', block_sizes)

xlabel('Block size', 'fontsize', 13)
ylabel('t (sec)', 'fontsize', 13)

legend('Flower', 'Gould', 'Pig')
```

## 4.5 Wavelet packet and DCT

This code consist of both the DCT and the Wavelet runs. The code produces a LaTeXtable output of the results. The mat files are the result of the compression tests from wavemenu.

**Listing 5: code/Wavelet_packet_runs.m**

```
%A bunch of tests
```

```
%Reads mat files from wavemenu, prints latex table output in the Matlab ...
    prompt
%and saves the images for all methods (DCT and wavelet) in png format.

clc

addpath('C:\Users\Matz\Dropbox\Dropbox\Projects\Kurser\Image Processing\...
    Project\figures')
cd('C:\Users\Matz\Dropbox\Dropbox\Projects\Kurser\Image Processing\Project...
    \code\results')


warning('off', 'images:initSize:adjustingMag')

factor = 8;
fprintf(1, 'factor 8: ')
remove_zeros = 100*(factor-1)/factor;
fprintf(1,'Remove %s%%\n', num2str(remove_zeros, 4));


fprintf(1,'factor 64: ')
factor = 64;
remove_zeros =  100*(factor-1)/factor;
fprintf(1, 'Remove %s%%\n\n', num2str(remove_zeros, 4));


names      = {'flower', 'Gould', 'pig'};
n_block    = 8; %the dimension of the blocks
block_test = 0; %are we performing the block test?
exp=5; %exponent for the diff images, changing the how strong the features ...
    will be in the difference image
decpoints=2; %decimal points used in ouput of rms

for i=1:3


    img_orig = im2double(imread([names{i},'.tif']));
    img_orig = rgb2gray(img_orig);
    appendname = '_bior_8';

    load([names{i}, appendname, '.mat'])
    tmp_img = X/255;
    tmp_rms = rms(img_orig, tmp_img);
    fprintf(1, '%s & %s & %d & %s\\\\ \n', names{i}, 'bior 4.4', 8, ...
        num2str(tmp_rms, decpoints));
    imwrite(tmp_img, [names{i}, appendname, '.png'])

    tmp = histeq(img_orig-tmp_img, 256).^exp;%was 20* before, fixed
    imwrite(tmp, [names{i}, appendname, '_diff.png'])

    appendname = '_rbio_8';
    load([names{i}, appendname, '.mat'])
    tmp_img = X/255;
    tmp_rms = rms(img_orig, tmp_img);
    fprintf(1, '%s & %s & %d & %s\\\\ \n', names{i}, 'rbio 6.8', 8, ...
        num2str(tmp_rms, decpoints));
    imwrite(tmp_img, [names{i}, appendname, '.png'])

    tmp = histeq(img_orig - tmp_img, 256).^exp;
    imwrite(tmp, [names{i}, appendname, '_diff.png'])
    I = img_orig;

    factor = 8;
    appendname = '_DCT_8';
    DCT_compression
    imwrite(E, [names{i}, appendname, '.png'])
```

```
    tmp = histeq(img_orig - E, 256).^exp;
    imwrite(tmp, [names{i}, appendname, '_diff.png'])


    fprintf(1, '%s & %s & %d & %s\\\\ \n', names{i}, 'DCT', 8, num2str(...
        tmp_rms, decpoints));
    fprintf(1, '\\hline\n');

    appendname = '_bior_64';
    load([names{i}, appendname, '.mat'])

    tmp_img = X/255;
    tmp_rms = rms(img_orig, tmp_img);
    fprintf(1, '%s & %s & %d & %s\\\\ \n', names{i}, 'bior 4.4', 64, ...
        num2str(tmp_rms, decpoints));
    imwrite(tmp_img, [names{i}, appendname, '.png'])

    tmp = histeq(img_orig - tmp_img, 256).^exp;
    imwrite(tmp, [names{i}, appendname, '_diff.png'])

    if i == 1
        img_cropped = tmp_img(411:455, 84:135);
        imwrite(img_cropped, [names{i}, appendname, '_cropped.png'])
    end

    appendname = '_rbio_64';
    load([names{i}, appendname, '.mat'])

    tmp_img = X/255;
    tmp_rms = rms(img_orig, tmp_img);
    fprintf(1, '%s & %s & %d & %s\\\\ \n', names{i}, 'rbio 6.8', 64, ...
        num2str(tmp_rms, decpoints));
    imwrite(tmp_img, [names{i}, appendname, '.png'])

    tmp=histeq(img_orig - tmp_img, 256).^exp;
    imwrite(tmp, [names{i}, appendname, '_diff.png'])

    if i == 1
        img_cropped = tmp_img(411:455, 84:135);
        imwrite(img_cropped, [names{i}, appendname, '_cropped.png'])
    end

    I          = img_orig;
    factor     = 64;
    appendname = '_DCT_64';
    DCT_compression
    imwrite(E, [names{i}, appendname, '.png'])

    tmp = histeq(img_orig - E, 256).^exp;
    imwrite(tmp, [names{i}, appendname, '_diff.png'])


    fprintf(1, '%s & %s & %d & %s\\\\ \n', names{i}, 'DCT', 64, num2str(...
        tmp_rms, decpoints));
    fprintf(1, '\\hline\n');
end
```