

Wavelets and Multiresolution Modeling

Project 2: An Image Compression Tool

due on Monday 02/12/2001

Martin Bertram, bertram@cs.utah.edu

1 Fundamentals

1.1 Image Compression

Most image compression algorithms consist of the following three steps:

- A linear transform providing a sparse set of coefficients.
- Quantization of the coefficients.
- Encoding the quantized coefficients.

The first step transforms an image into a different basis providing a set of coefficients that have most likely very small absolute values. This transform removes *correlation*, i.e. similarity of neighboring pixel values, from an image, resulting in a less-redundant representation. Possible choices for this transform are the discrete cosine transform (DCT) and a wavelet transform.

The second step, quantization of coefficients, is only required for lossy compression schemes where the transform is computed in floating-point arithmetic. The coefficients are scaled into a certain target range and then rounded to integers. The error introduced by quantization of coefficients is typically less visible in a reconstructed image than a similar quantization applied directly to the pixel values. The target range used for quantization controls compression rates and reconstruction errors. For lossless compression, however, a bijective transform mapping integers to integers is used, such that no quantization is necessary.

Finally, the sparse set of integer coefficients is encoded into a bit sequence. This step actually performs the compression, since the number of coefficients is the same as the number of pixels. Coding schemes, like arithmetic coding and Huffman coding, exploit that certain (small) coefficient values appear much more frequently than most others. The complete set of coefficients can be reconstructed without loss from the typically much shorter bit sequence.



Figure 1: An image and its DCT applied to frames of 16×16 pixels. A grey level was added to the DCT, since its coefficients can be negative.

1.2 Discrete Cosine Transform (DCT)

Before the fast wavelet transform was discovered in 1989, image-compression schemes like JPEG had to rely on Fourier methods. The Fourier transform analyses the different frequencies that a function is composed of (*spectral analysis*). The DCT has the advantage that it produces real coefficients rather than complex ones. For an image composed of $m \times n$ pixels $f_{ij}, i = 0, \dots, m - 1; j = 0, \dots, n - 1$, the DCT is defined as

$$\begin{aligned}
 C f_{kl} &= \frac{2}{\sqrt{mn}} c(k)c(l) \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} f_{ij} \cos\left(\frac{(2i+1)k\pi}{2m}\right) \cos\left(\frac{(2j+1)l\pi}{2n}\right), \\
 c(k) &= \begin{cases} \sqrt{\frac{1}{2}}, & \text{if } k = 0, \\ 1, & \text{otherwise.} \end{cases}
 \end{aligned}
 \tag{1.1}$$

The inverse DCT is defined as

$$f_{ij} = \frac{2}{\sqrt{mn}} \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} C f_{kl} c(k)c(l) \cos\left(\frac{(2i+1)k\pi}{2m}\right) \cos\left(\frac{(2j+1)l\pi}{2n}\right).
 \tag{1.2}$$

Task1: write a program that reads a 256×256 -greyscale image (pgm-ascii format: "P2" m n max $f[m][n]$, examples are provided on the web at <http://www.cs.utah.edu/classes/cs6941>). Store the image as a matrix of normalized (divided by max) floating-point numbers. Write a function that subdivides the image into 16×16 -blocks (necessary for efficiency and stability) and computes the DCT for every block, see Figure 1. Write a function for the inverse DCT on these blocks and verify that it correctly reconstructs the image.

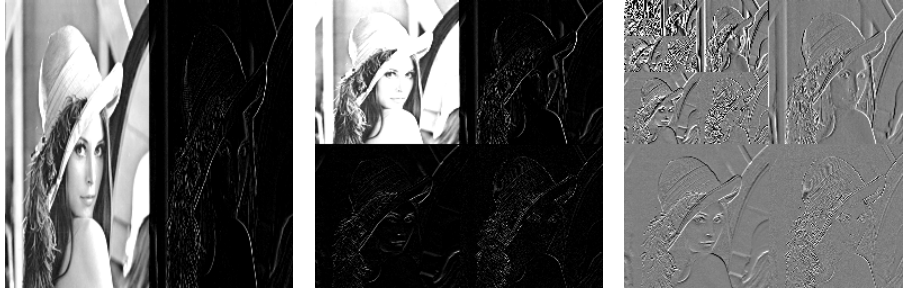


Figure 2: The one-dimensional Haar-decomposition is applied to every row (left) and then to every column (middle) of an image. This transform is recursively repeated with the upper-left quarter of an image, until one pixel is left. The result is the Haar-wavelet transform, shown with a grey level added (right).

1.3 Haar-Wavelet Transform

The Haar-wavelet transform is the simplest wavelet transform using piecewise constant basis functions. For a one-dimensional array f_i ($i = 0, \dots, 2n - 1$), the Haar-*decomposition* computes a low-resolution approximation s_i ($i = 0, \dots, n - 1$) and a difference vector w_i ($i = 0, \dots, n - 1$). The s_i are called *scaling-function coefficients* and the w_i are *wavelet coefficients*. Haar-decomposition is computed by re-labeling the coefficients

$$\begin{aligned} s_i &:= f_{2i} \\ w_i &:= f_{2i+1} \end{aligned} \quad (i = 0, \dots, n - 1), \quad (1.3)$$

followed by some local modifications (*lifting operations*),

$$\begin{aligned} w'_i &:= s_i - w_i \\ s'_i &:= s_i - \frac{1}{2} w'_i \end{aligned} \quad (i = 0, \dots, n - 1). \quad (1.4)$$

After computing these lifting operations, the coefficients are scaled (to normalize the corresponding basis functions), and written back into the array f_i such that the first half is filled with scaling-function coefficients and the second half contains wavelet coefficients,

$$\begin{aligned} f'_i &:= \sqrt{2} s'_i \\ f'_{i+n} &:= \sqrt{2} w'_i \end{aligned} \quad (i = 0, \dots, n - 1). \quad (1.5)$$

The equations (1.3–1.5) define the one-dimensional decomposition for the Haar-wavelet transform.

The Haar-wavelet transform for images computes a one-dimensional decomposition for every row of pixels and then for every column. After this step, the upper-left quarter of the image (provided that the indices run from left to right and from top to bottom) contains a low-resolution approximation of the image. (The intensity of this smaller image is scaled by two.) The upper-left quarter of

the image is recursively transformed by applying the one-dimensional decomposition to its rows and columns, see Figure 2. This leads to smaller and smaller approximating images in the upper-left corner until only one pixel remains as the coarsest level of resolution. All other coefficients represent differences between the individual levels of detail.

The inverse of a one-dimensional decomposition step is called *reconstruction*. The reconstruction formula is simply obtained by inverting equations (1.3–1.5) in reverse order. In Particular, the inverse of equation (1.4) is implemented as

$$\begin{aligned} s_i &:= s'_i + \frac{1}{2} w'_i \\ w_i &:= s_i - w'_i. \end{aligned} \quad (i = 0, \dots, n-1). \quad (1.6)$$

The inverse Haar-wavelet transform computes for every decomposition step a reconstruction step. We note that the order of all operations needs to be reversed to provide the correct reconstruction result.

Task2: implement the Haar-decomposition and reconstruction formulae for one-dimensional arrays f_i ($i = 1, \dots, 2^j$), where j is an integer. Verify the correctness of these operations. Then, implement the Haar-wavelet transform and its inverse for 256×256 -images.

1.4 Linear B-Spline Wavelet

Analogously to the Haar-wavelet transform, we can construct wavelet transforms using different basis functions by applying different lifting operations. The overall computation of the transform is the same as for the Haar wavelet. We obtain a linear B-spline-wavelet transform when replacing equation (1.4) by

$$\begin{aligned} w'_i &= w_i - \frac{1}{2}(s_i + s_{i+1}) \quad (i = 0, \dots, n-2), \\ w'_{n-1} &= w_{n-1} - s_{n-1}, \\ s'_0 &= s_0 + \frac{1}{2}w'_0, \quad \text{and} \\ s'_i &= s_i + \frac{1}{4}(w'_i + w'_{i+1}) \quad (i = 1, \dots, n-1). \end{aligned} \quad (1.7)$$

Equations (1.3) and (1.5) remain unchanged.

Task3: Copy your code from the Haar-wavelet transform and replace the lifting operations to implement a linear B-spline-wavelet transform. Find the inverse of equation (1.7) and plug it into your reconstruction routine. Again, test your implementation of the transform and its inverse using a 256×256 -image.

1.5 A simple Coding Scheme

In order to get compression, we need to quantize the floating-point coefficients and then encode the resulting integer numbers. for quantization, the user may specify a scaling parameter t which controls the compression rate. The coefficients are then scaled and rounded to integers,

$$f'_{ij} := [tf_{ij}], \quad (1.8)$$

where $[\cdot]$ returns the integer that is closest to the argument.

If the parameter t is small enough, then the array of quantized coefficients will have long sequences of zeros, which can be exploited to obtain high compression rates. We traverse the whole set of coefficients and use the following two-bit symbols for encoding:

- 00, indicating that the current coefficient is the first one that is zero in a sequence of length $1 \leq n \leq 63$. The subsequent six bits are used to store n . Encoding continues with the next non-zero coefficient.
- 01, indicating that the current coefficient leads a sequence of zeros of length $n > 63$. the subsequent 30 bits encode n . Encoding continues with the next non-zero coefficient.
- 10, indicating that the absolute value of the current coefficient is less than 32. Its value is encoded in the subsequent six bits (one bit is needed for the sign).
- 11, indicating that the current coefficient needs more space than 6 bits. Its value is stored in the subsequent 30 bits.

Task 4: Implement a function that encodes a transformed image into a binary file, using the coding scheme defined above. Write another function to decode a file and restore the coefficients (except for the quantization error). You can use `fputc` and `fgetc` to write/read bytes to/from binary files.

2 Project Specification

Implement a simple image compression and reconstruction tool for 256×256 -greyscale images (this project does not require a GUI). The user may specify a file containing an image (pgm format), a transform (DCT, Haar, or B-spline wavelet), and a scaling parameter t for quantization. The compression tool then generates a binary file containing a symbol specifying which transform is used, the parameter t , and the encoded set of coefficients. The reconstruction tool should recover an image from such a binary file, without using any other input. Your program should be able to display the original image, the transformed one (add a grey level or use absolute values), the reconstructed image, and a difference image subtracting the original from the reconstructed version (use absolute values). Compute the root-mean-square error,

$$rmse = \sqrt{\frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (f_{ij} - g_{ij})^2}, \quad (2.1)$$

where f_{ij} is the original and g_{ij} the reconstructed image. Which transform provides the lowest quantization errors at (nearly) same compression rates?

Which transform produces the smallest visible artifacts? What compression rates can be reached while maintaining reasonable image quality (compared to 65536 Bytes, uncompressed)?

ATTENTION!!! You must develop your own code Sharing code with fellow students is not permitted. Do not erase your code—you may use it for the following projects.