Image Compression Using Wavelets

Jens Gerlach Christensen

January 6, 2003

1 Introduction

The goal of this project is to investigate the use of wavelets for image compression and to compare it to JPEG image compression. The main part of the project is to describe and implement how to use wavelets for image compression. To bring the technique into perspective I will compare wavelet compressed pictures with JPEG compressed pictures. The latter may seem somewhat unscientific, but I think that it really shows (very roughly) the advantages and disadvantages of wavelets for picture compression.

I have just finished a course in Wavelets at the Department of Mathematics at University of Copenhagen. This was a very theoretical course and I felt the need to see some applications of wavelets.

I study computer science too, and I hope it will be inspiring to use those two parts of my education at the same time.

2 Representation of Images

A picture can be represented in many ways. A normal two-dimensional picture can be regarded as a matrix where each entry in the matrix corresponds to a point (a dot of some width and height). This entry has a value that describes the exact colour of the point. The value can for example be an integer (specifying a certain grey tone) or a vector consisting of red,green and blue tones. These are not the only way to represent a picture. Therefore I had to consider which format I wanted to use.

First of all I had to get the pictures in a format that I could read and understand quite easily. I settled on a format called portable greymap (short PGM) since this is just a sequence of positive integers. There is a maximum number in the file which corresponds to white and 0 is black. The other numbers are shades of grey on a linear scale between white and black (if 100 is white, then 90 is light grey and 10 is a dark grey). Since the data is arranged that way it should be the easiest format to apply wavelet transformation to.

There is a colour picture format PPM that is very similar to PGM. Each point in the picture consists of three numbers that are shades of red,green and blue. Wavelet transformation splits these colours and performs the same transformation on each of them as for the PGM picture.

Since the same technique applies to greyscale and colour pictures I have only considered greyscale pictures. For a specification of the PGM format see appendix B. Part of the program I have written is a parser of that format (means I can read the format and draw out the useful information). There is a binary and a clear text format of PGM and I use the clear text one. If I continue to develop this compression program I will probably start using an image library to do the conversion.

3 JPEG Picture Compression

JPEG is a picture format invented for the internet. To save bandwidth (and thus money) when transfering a picture it is desirable to compress the picture (to a smaller file size but still trying to maintain picture quality). There are different ways to do this, but JPEG compression is widely used. JPEG removes information from the picture, but tries to do it such that the eye will notice little difference from the original. JPEG does this compression on blocks of size 8×8. At high compression levels JPEG will not be able to "cheat" the eye, because the result gets very blocky (the picture will consist of squares where each square is filled with one colour). At lower compression rates the results are visually very good and the file sizes are small.

The understanding of JPEG compression is not important in this project, since the comparison of pictures compressed with wavelets and JPEG will only be visual. The only important property of the JPEG pictures is their filesize. The reason for this is, that I will use file sizes to choose equivalent wavelet and JPEG compressed images. I will then compare the visual quality of these equivalent pictures.

4 Theory of Filters

This section contains a summary of filter theory. It is an extract of the filter theory presented in the notes [4] from the wavelet course I followed in the winter of 2001. After that I describe the application of filters to still pictures.

4.1 1-dimensional filter

I will only describe the use of 1-dimensional filters since this is the one we use for pictures as we will see later.

Let $x \in \ell^2(\mathbb{Z})$ and define its Fourier transform $\hat{x}$ by

$$\hat{x}(t) = \sum_{n \in \mathbb{Z}} x_n e^{-int}. \tag{1}$$

x can be found from its Fourier transform by

$$x_n = \frac{1}{2\pi} \int_0^{2\pi} \hat{x}(t) e^{int} dt. \tag{2}$$

The following defines a filter

Definition 4.1 *A filter is an element h of $\ell^2(\mathbb{Z})$ such that its Fourier transform $\hat{h}(t)$ is bounded. The elements h_n are called the filter coefficients.*

h is called a FIR (Finite Impulse Response) filter when only finitely many coefficients are nonzero.

We shall only use FIR filters in this text.

Definition 4.2 *The downsampling operator $(2\downarrow)$ and the upsampling operator $(2\uparrow)$ are defined as follows:*

$$(2\downarrow) : x \mapsto (\dots, x_{-4}, x_{-2}, x_0, x_2, x_4, \dots) \tag{3}$$

$$(2\uparrow) : x \mapsto (\dots, x_{-1}, 0, x_0, 0, x_1, \dots) \tag{4}$$

A filter can be applied to a signal x, which is regarded as a periodic sequence. We now look at the special function H that applies the filter h to a signal x. H is defined by

$$H : x \mapsto y, y_k = \sum_{n\in\mathbb{Z}} \overline{h}_{n-2k} x_n. \tag{5}$$

This is well defined if H is FIR. If H is not a FIR then we would have to define our signal to be a finite sequence in order to assure the sum in (5) to converge.

With $\tilde{h}$ defined by $\tilde{h}_n = \overline{h}_{-n}$ and the convolution map C_h

$$C_h x = h * x = (\sum_{n\in\mathbb{Z}} \overline{h}_{n-k} x_n)_{k\in\mathbb{Z}} \tag{6}$$

this corresponds to

$$H = (2\downarrow) C_{\tilde{h}}. \tag{7}$$

If the signal x has period $2m$ then so does $C_h x$ and thus Hx is m-periodic. Therefore we only need m numbers to fully describe Hx. If we represent the signal x by only one of its periods starting at x_0, then the matrix representing H is given by:

$$\begin{pmatrix}
h_0 & h_1 & h_2 & \dots & h_j & & & & & & & & & & \\
 & & h_0 & h_1 & h_2 & \dots & h_j & & & & & & & & \\
 & & & & & & & \ddots & & & & & & & \\
 & & & & & & & & h_0 & h_1 & h_2 & \dots & h_j & & \\
h_{j-1} & h_j & & & & & & & & & h_0 & h_1 & h_2 & \dots & h_{j-2} \\
 & \vdots & & & & & & & & & & & & \vdots & \\
h_2 & h_3 & \dots & h_j & & & & & & & & & & h_0 & h_1
\end{pmatrix} \tag{8}$$

Now the wavelets come into play. Let φ be a scaling function for an MRA (MultiResolution Analysis) and let $\varphi_{j,k}$ be defined by

$$\varphi_{j,k}(x) = 2^{j/2}\varphi(2^j x - k). \tag{9}$$

Let ψ be the wavelet associated with φ. Define the filters h and g by

$$h_n = <\varphi, \varphi_{1,n}> = \int \varphi(x)\sqrt{2}\overline{\varphi}(2x-n)dx \tag{10}$$

$$g_n = <\psi, \varphi_{1,n}> = \int \psi(x)\sqrt{2}\overline{\varphi}(2x-n)dx \tag{11}$$

From these filters we define H and G according to (5) using h and g respectively.

Definition 4.3 *Let h and g be filters. The filters are called quadrature mirror filters (QMF) if the filter coefficients satisfy the following*

$$\sum_{k\in\mathbb{Z}} h_k \overline{h}_{k-2n} = \delta_{0n} \tag{12}$$

$$\sum_{k\in\mathbb{Z}} h_k \overline{h}_{k-n} = (-1)^n \sum_{k\in\mathbb{Z}} g_k \overline{g}_{k-n} = \tag{13}$$

$$\sum_{k\in\mathbb{Z}} h_k \overline{g}_{k-2n} = 0 \tag{14}$$

for all $n \in \mathbb{Z}$.

The filters in this text will be QMF, most of them with coefficients found in Daubechies' [1, p. 195]. Beware that the coefficients in that text are normalised, such that the sum gives $\sqrt{2}$.

Equation (2.46) from [4] states that g can be chosen such that if h has its coefficients at $0, \ldots, 2N-1$ so does g. The filter g is then given by

$$g_n = (-1)^n \overline{h}_{2N-n-1} \tag{15}$$

These coefficient can be arranged to form a matrix like in (8). Assume that the signal x has period $2m$, then H and G are $m \times 2m$ matrices. If we put them on top of each other to form a $2m \times 2m$ matrix the resulting matrix is invertible (according to the first two properties in definition 4.3):

$$\begin{pmatrix}
h_0 & h_1 & h_2 & \ldots & h_j & & & & & & & & & & \\
 & & h_0 & h_1 & h_2 & \ldots & h_j & & & & & & & & \\
 & & & & & & & \ddots & & & & & & & \\
 & & & & & & & & h_0 & h_1 & h_2 & \ldots & h_j & & \\
h_{j-1} & h_j & & & & & & & & & h_0 & h_1 & h_2 & \ldots & h_{j-2} \\
 & \vdots & & & & & & & & & & & & \vdots & \\
h_2 & h_3 & \ldots & h_j & & & & & & & & & & h_0 & h_1 \\
g_0 & g_1 & g_2 & \ldots & g_j & & & & & & & & & & \\
 & & g_0 & g_1 & g_2 & \ldots & g_j & & & & & & & & \\
 & & & & & & & \ddots & & & & & & & \\
 & & & & & & & & g_0 & g_1 & g_2 & \ldots & g_j & & \\
g_{j-1} & g_j & & & & & & & & & g_0 & g_1 & g_2 & \ldots & g_{j-2} \\
 & \vdots & & & & & & & & & & & & \vdots & \\
g_2 & g_3 & \ldots & g_j & & & & & & & & & & g_0 & h_1
\end{pmatrix} \tag{16}$$

The inverse is simply the transposed matrix.

In appendix D and E respectively, I have written the matrices for the Haar wavelet and Daubechiess-4 wavelet.

h is called a highpass filter and g is a lowpass filter. The intuitive explanation for this is that every other entry in g has negative value according to (15). If these entries are assumed to be numerically close to each other the sum will be close to zero (when applied

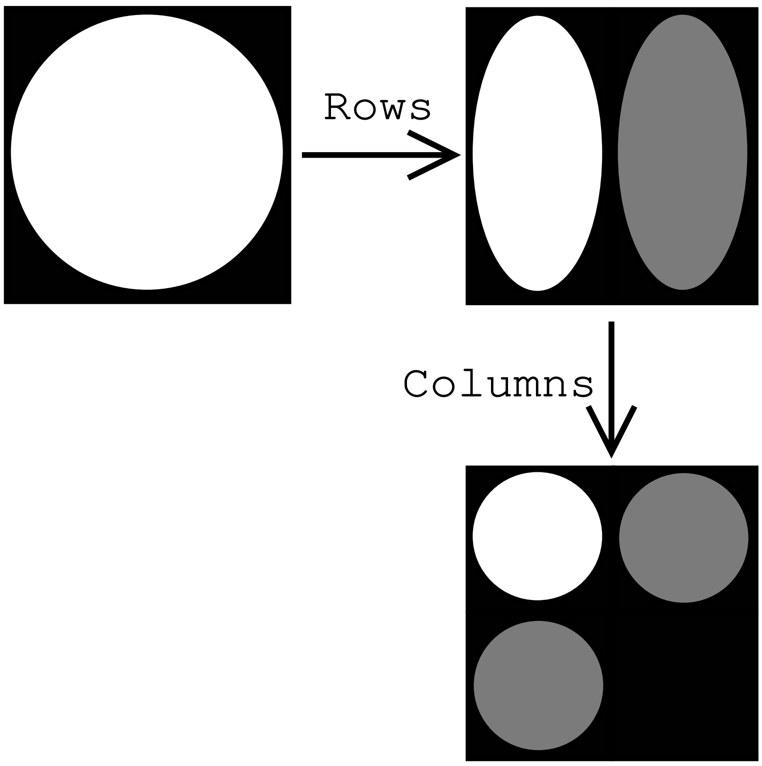

Figure 1: One transformation of a white circle on black background

to values of the same size). When we apply g the values will be close to zero, but for h we nearly double the values. Therefore it looks like h lets the high values through and g lets the low values through. This is only an attempt to explain the naming of high- and lowpass filters.

4.2 Application to image transformation

Now we can transform a 1-dimensional vector of even length by wavelet filters and obtain the original vector by the inverse transform (only QMF). Here we will see how this applies to the transformation of pictures.

An image is represented by a number of columns and a number rows and values for each point. It therefore looks like a matrix. We apply the filters by first transforming each row in the image and then transforming each column in the resulting matrix. I will write H_r when H has been applied to the rows and H_c when H has been applied to the columns. The notation is used for G. H lets the high values through (high pass filter) and G lets the small values through (low pass filter). Remember that black is 0 and white is the highest value specified in the picture (see section 2 on page 1). Thus if we have a white circle on a black background its transformation will look like figure 1. The gray areas in the figure show that the values in the areas are small (compared to the white area). Most of the information in the picture will be found in the top left quarter of the transformed matrix, and we will only apply the next transformation to that quarter. Two transformations of the white circle on black will look like illustrated on figure 2

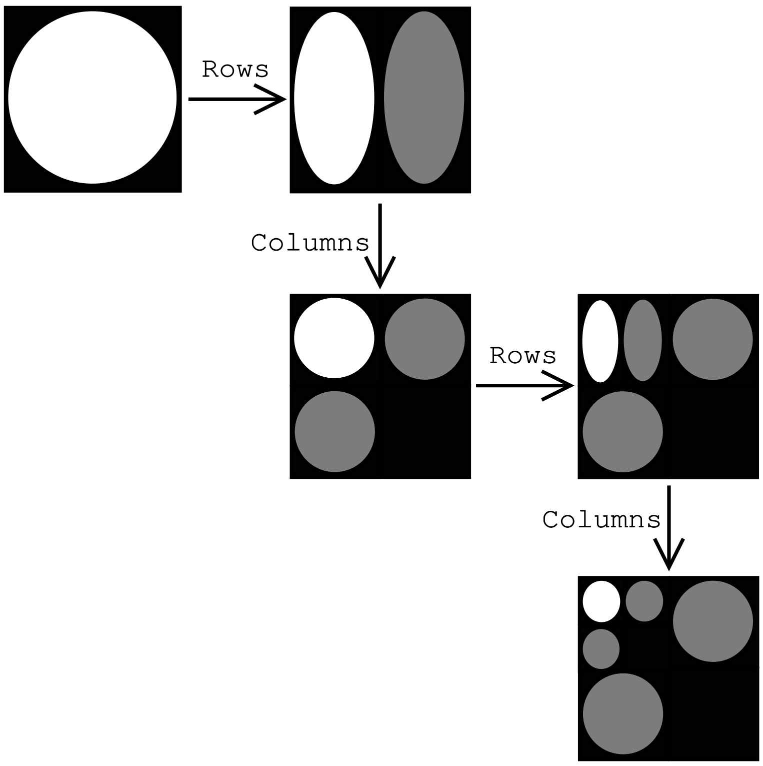

Figure 2: Two transformations of a white circle on black background

Each time we do a transformation it will be required that the part of the image we transform has an even number of both rows and columns. Thus if we do n decompositions the number of rows and columns must be divisible by 2^n. I ensure this by cutting the picture to the biggest size such that the property holds. We lose part of the picture that way.

The inverse transform of the picture will use the 1-dimensional inverse on the columns first and then the rows. Since the matrix (16) is invertible this can be done without loss of picture quality. We will need to know how many times the picture has been transformed to be able to restore the picture to its original form. An illustration on the inverse transform can be seen on figure 3.

We can now transform pictures with the use of wavelets and restore the picture without loss of picture quality.

5 Compression

By using wavelet filters we are able to decompose a picture such that most of the information will be concentrated in arbitrarily few (not less than 1) values. Now we will cut the rest of the picture away to do lossy compression of the picture.

When we transform the picture n times most of the information will be in the top left part (of size $(1/4)^n$) in the resulting matrix. If we cut the rest of the matrix by setting the entries equal to 0 we will not lose much information. This is a lossy compression scheme

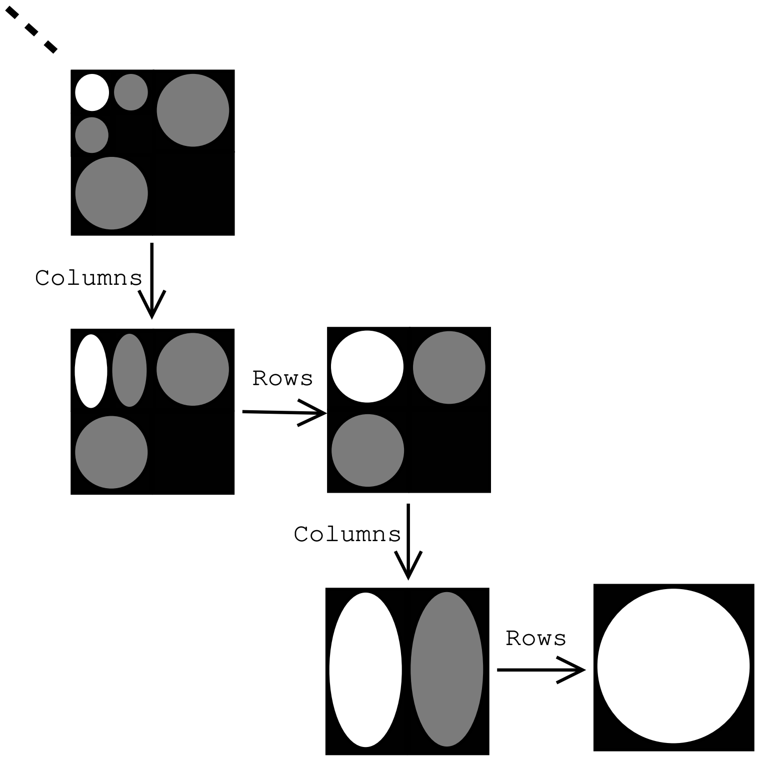

Figure 3: The last two inverse transformations of a white circle on black background

since we lose information (even though I just explained why we do not lose much). If we cut more than $1 - (1/4)^n$ we will lose part of the most important information and the picture will get blury when we try to restore it.

I want to save the compressed picture in a file on my computer. To save space I use the lossles compresion scheme I explain here. We only need to save the entries that are different from 0. But at the same time we need information about the position of each non-zero value in the matrix. The matrix can be represented by an array where the rows are just placed after each other. The total number of entries E is the number of columns times the number of rows. Each entry has a unique position between 0 and $E - 1$. I can save this number and the value the entry holds (only doing so for non-zero values to save space) in a file. But I can save more space by only writing the difference in entrynumber between a non-zero value and the preceding non-zero value. This difference is smaller than the entrynumber an we then save more space.

This information can in principle be written to a plain text file (values separated by spaces or newline characters). To save extra space on the computer disk I use file compression. It is lossles and will be explained in section 6.

6 The Implementation

The algorithms have been implemented in C++ on Linux but should be usable on other Unix flavours. The source code is found in appendix A starting on page 12.

My program consists of two classes: `matrix` and `dwt` (Discrete Wavelet Transform).

6.1 The `matrix` class

The class `matrix` is an array A that holds values of decimal numbers (of type double) and two variables `columns` and `rows`. The array is aligned after the columns such that the second column starts at offset `columns` from the beginning of A. An instance of matrix can be declared with two integers setting `columns` and `rows` in that order. If no parameters are given then a 1×1 matrix is created. The methods in the matrix class are

- `get(i,j)` gets the value of entry (i,j) in the matrix
- `set(i,j)` sets the value of entry (i,j) in the matrix
- `mult()` multiplies two matrices. Since the filters are generally of small size compared to the picture using `mult` would result in many multiplications by zero (to save calculation time I don't use it).
- `readpgm()` reads a PGM format file and stores it in a matrix
- `writepgm()` writes the matrix to a file in the PGM format

6.2 The `dwt` class

The `dwt` class inherits the `matrix` class. It contains the following methods:

- `readfilter()` reads a filter from a file
- `transform()` transforms the picture a number of times
- `itrans()` makes the inverse transform the same number of times
- `compress()` takes away a given ratio of the values the ratio is a decimal number between 0 and 1
- `bzwritetrans()` writes the transformed picture to a compressed text file (bzip2 format)
- `bzreadtrans()` reads the same format to get values of transformed picture

The methods are called in the programs `enc` and `dec` that encode and decode the picture respectively.

I have used the bzip2 compressed file format, to be able to compare the sizes of my wavelet encoded files and the corresponding JPEG files. For information on bzip2 look at `http://sourceware.cygnus.com/bzip2`. On the basis of the sizes I will make a qualitative comparison of the wavelet transform and the JPEG transform. The reason for this is that if you save a 75% quality JPEG picture it doesn't mean that 75% of the information in the picture has been cut out. Therefore I had to make my own scale to be able to compare different formats, but it is a very rough scale.

6.3 How to use the programs

To translate the programs go to the src directory and type make. When compilation has finished there are two programs to use:enc and dec. To use the programs you must define some filter files. Three examples are already defined: haar,daub4 and daub6. They simply hold information about the filter to use. Remember that the coefficients in [1, p. 195] are normalised such that the sum gives $\sqrt{2}$. This means that we have to divide each coefficient by the squareroot of the sum of the squares.

6.3.1 enc

This program encodes a picture of the PGM format and writes the result to a compressed file. The program takes five arguments.

1. the type, which is the same as the name of the filter file you want to use
2. the number of transformations you want
3. the ratio of the transformed picture you want to discard (number between 0 and 1 0.98).
4. the input filename, the name of the PGM formatted file
5. the output filename, the name of the file to write compressed picture to

6.3.2 dec

Since no graphic programs understand my picture format we have to decode the transformed picture into a format (I've chosen PGM format since I'm already working with it). The program only takes two arguments (input and output filenames) since most information is saved in the file.

6.3.3 Examples

To build the programs type `make`. The programs are run by typing `./enc` or `./dec` followed by the arguments.

```
./enc daub4 5 0.98 test.pgm test.daub4
./dec test.daub4 testout.pgm
```

The resulting picture will be `testout.pgm` which can be viewed with any graphics program. The file test.daub4 is only used for comparison of size to JPEG files.

7 Comparison and Analysis of Results

Now it is time to do a very rough investigation of the use of wavelet compression. I will do this by comparing it with JPEG compression. It is very hard to do a scientific comparison, perhaps it is impossible to do so (just try to define a fair way to do this after reading this section). Therefore this section only contains my ideas of what would be a good indication of the usability of wavelet compression.

7.1 Generation of Pictures for Comparison

I found some testpictures at `ftp://nic.funet.fi/pub/graphics/misc/test-images/`. I picked lena and baboon since lena is a normal face, so it has many surfaces with same color and the baboon because it does not.

I converted both to PGM format with the program `xv` which can be downloaded from `http://www.trilon.com/xv/` to be able to use my programs.

I then made wavelet compressed versions of each picture using three different wavelet filters (`haar,daub4,daub6`) using my program `enc`. I generated pictures at compression ratios of 97,98 and 99%. To be able to compress at ratios up to 99% I had to make 5 transforms since $1 - (1/4)^5 < 0.99$ (see section 5 about compression). The file sizes of pictures at the same compression ratio where almost the same. I then used the program `cjpeg` to make JPEG compressed images of various qualities of both pictures. `cjpeg` is part of the `jpeg6b` sources found at `http://www.ijg.org` I then picked JPEG pictures of nearly the same filesize as my wavelet compressed pictures. To be able to compare the pictures I had to convert my wavelet compressed pictures to a format readable by for example `xv`. I did this with my program `dec`. The results can be seen in appendix C from page 30 onwards. For each picture I have registered the size of the compressed file in number of bytes.

7.2 Comparison of Results

I found that at 99% compression the wavelet transform gave a better picture. At 98% compression it was pretty much the same (except for the Haar wavelet) and at 97% the JPEG pictures were the best.

The test pictures are very nice in the sence that they are of size $2^9 \times 2^9$. They are divisible by 2^5 and therefore they will not be cut to be transformable. With other pictures we cannot cut them too much since part af a persons face might be missing. JPEG cuts the picture by at most 8 points in each dimension since it splits the picture into blocks of size 8×8 before it does any compression. With $2^5 = 32$ we might have to cut up to 32 points in each dimension. This is still not too bad with big pictures, but cutting more than that could be fatal to the picture. All the compressions have been done with 5 transformations.

The pictures transformed with the Haar wavelet are very blocky and not very pleasing to look at. I would not use the Haar wavelet for picture compression at all. The Daubechies wavelets are much better suited for pictures compression, but I cannot see which of Daubechies-4 or Daubechies-6 is better. Since Daubechies-4 requires less calculations it would be faster to use and therefore be my choice of wavelet.

If the reader wants to play with wavelets feel free to download the source files at `http://home.tiscali.dk/vepjan/wavelets.tgz` and unpack. See the section 6 for instructions on how to use the sources.

8 Conclusion

The goal of this project was to find out more about the use of wavelets for image compression. Before I started I thought that I would have to use two-dimensional wavelets.

It turned out to be easier than that since only one-dimensional wavelet transforms are used. I also wanted to compare it to JPEG image compression. I found that wavelets were better suited for high compression ratios that JPEG. For lower ratios JPEG was the best. If a picture contains big even surfaces of same colour then the wavelets were good but for very messy pictures the result was very blurry. By using other wavelets we might be able to do better. Different techniques like adaptive wavelets transformation (see [6]) may also make wavelets compression better. In the new JPEG2000 standard wavelets are actually used.

Implementing the algorithms was the biggest task in this project. I hope others with an interest in wavelets will find this project and the sourcecode useable for further investigations.

References

[1] Ingrid Daubechies, *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics

[2] Los Alamos National Laboratory Research Library, *Numerical Recipes in C*, `http://lib-www.lanl.gov/numerical/bookc.html`

[3] E. J. Stollnitz, T. D. DeRose and D. H. Salesin, *Wavelets for Computer Graphics - Theory and Applications*, Morgan Kaufmann Publishers (1996)

[4] Gerd Grubb, *Notes on Wavelets 2001*, Matematisk Institut, Københavns Universitet

[5] Lars Hörmander, *Four Lectures on Wavelets*, Department of Mathematics, University of Lund

[6] Tilo Strutz and Erika Müller, *Adaptive Wavelet Transformation for Image Coding Using a Forecast Decomposition Selection*, Proc. of Picture Coding Symposium (PCS'97), Berlin, Germany, 10.-12. September, 1997, pp.67-72

A Source Code

Here follows the source code for the wavelet transformation. Both Makefile, header files and C++ main files are listed.

A.1 Makefile

```
#configure only the next five lines
RATIOS = 97 98 99
FILTERS = haar daub4 daub6
TRANSFORMS = 5
PICTURES = lena  baboon
JPGRATIOS = 4 6 8 10 12 14 16 18 20 22 24 26 28 30
#don't configure beyond this unless you know what you're doing

CC = /usr/bin/g++
CFLAGS = -O

LIBS = -L/usr/local/lib -lm -lz -lbz2

IFLAGS  =

OBJECTS = matrix.o dwt.o

PROGS   = enc dec

.c.o:
$(CC) $(CFLAGS) $(IFLAGS) -c $*.c

.cc.o:
$(CC) $(CFLAGS) $(IFLAGS) -c $*.cc

.o:
$(CC) $(CFLAGS) -o $* $*.o $(OBJECTS) $(LIBS)

.c:
$(CC) $(CFLAGS) $(IFLAGS) -c $*.c
$(CC) $(CFLAGS) -o $* $*.o $(OBJECTS) $(LIBS)

.cc:
$(CC) $(CFLAGS) $(IFLAGS) -c $*.cc
$(CC) $(CFLAGS) -o $* $*.o $(OBJECTS) $(LIBS)

all: $(PROGS)

enc: enc.o $(OBJECTS)
$(CC) $(CFLAGS) -o enc enc.o $(OBJECTS) $(LIBS)

dec: dec.o $(OBJECTS)
```

```
$(CC) $(CFLAGS) -o dec dec.o $(OBJECTS) $(LIBS)


clean:
rm -f *.o $(PROGS)

pics:
for l in $(PICTURES) ; do \
for k in $(TRANSFORMS) ; do \
for j in $(FILTERS) ; do \
for i in $(RATIOS) ; do \
  ./enc $$j $$k 0.$$i pic/$$l.pgm $$l.$$k.$$i.$$j ; \
  ./dec $$l.$$k.$$i.$$j $$l.$$j.$$k.$$i.pgm ; \
   done ; done ; done ; \
mkdir -p pic/$$l ; \
mv $$l* pic/$$l ; done

jpeg:
for l in $(PICTURES) ; do \
for j in pic/$$l/*.pgm ; do \
  cjpeg -q 100 $$j > pic/$$l/`basename $$j .pgm`.jpg ; \
done ; \
for j in $(JPGRATIOS) ; do \
  cjpeg -q $$j pic/$$l.pgm > pic/$$l/$$l.$$j.jpg ;\
done ; done

epspics:
for l in $(PICTURES) ; do \
for k in pic/$$l/*.jpg ; do \
jpeg2ps $$k > pic/$$l/`basename $$k .jpg`.eps ; \
done ; done

test: all pics jpeg epspics
```

A.2 matrix.h

```
#ifndef MY_MATRIX_H
#define MY_MATRIX_H

#include<iostream>
#include<vector>
#include<string>
#include<fstream>
#include<cmath>

class matrix {

 public:

  double* A;
  int columns,rows;
  matrix(int c,int r);
  void readpgm(char*);
  void writepgm(char*);
  void crop(int);
  matrix(void);
  matrix* mult(matrix,matrix);
  double get(int,int);
  void set(int,int,double);

 private:

  double max();
  double min();
  double partzero(void);
};

#endif
```

A.3 matrix.cc

```
#include "matrix.h"

matrix::matrix(int c,int r) {
  columns = c;
  rows = r;
  A = new double[c*r];
  for (int i =0 ; i< c*r ; i++) { A[i]=0; }
}

matrix::matrix() {
  columns = 1;
  rows = 1;
  A = new double[1];
  A[0]=0;
}

double matrix::get(int a,int b ) {
  return A[a*columns + b];
}

void matrix::set(int a, int b,double c) {
  A[a*columns + b] = c;
}

matrix* matrix::mult(matrix X,matrix Y) {
  if ( X.columns != Y.rows || X.rows != (*this).rows
        || (*this).columns != Y.columns ) {
    cout << "incompatible dimensions" << endl;
    return NULL;
  }
  for (int i = 0 ; i < X.rows ; i++) {
    for (int j = 0; j < Y.columns ; j++) {
      (*this).set(i,j,0);
      for ( int k = 0 ; k < X.columns ; k++) {
        (*this).set(i,j,(*this).get(i,j) + X.get(i,k)*Y.get(k,j));
      }
    }
  }
  return this;
}

double matrix::max() {
  double biggest=A[0];
  for (int i=1;i<columns*rows;i++) {
    if(A[i] > biggest) {
      biggest = A[i];
    }
```

```
  }
  return biggest;
}

double matrix::min() {
  double smallest=A[0];
  for (int i=1;i<columns*rows;i++) {
    if(A[i] < smallest) {
      smallest = A[i];
    }
  }
  return smallest;
}

void matrix::writepgm(char* name) {
  int j=1;
  int k;
  ofstream file(name);
  file << "P2" << endl;
  file << columns << " " << rows << endl;
  file << rint(max()) << endl;
  for(int i=0;i<columns*rows;i++) {
    if(j == 18) { //hver optager hoejst 4 tegn, der maa vaere 70 tegn pr linie
      j = 1;
      file << endl;
    }
    k = abs((int) rint(A[i]));
    if(k%10==k) {
      file << "  " << k << " ";
    }
    else {
      if(k%100==k) {
        file << " " << k << " ";

      }
      else {
        file <<  k << " ";
      }
    }
    j++;
  }
  file.close();
}

void matrix::readpgm(char* name) {
  char buf[80];
  char *p,*q;
  int i = 0;
  int mod = 1;
```

```
  delete A;
  ifstream file(name);
  file.getline(buf,80);
  if (*buf == 'P'); //ignore
  file.getline(buf,80);
  while(*buf == '#') {
    file.getline(buf,80);
  }

  // find columns
  q=buf;
  for(p=q;!isdigit(*p);p++);
  for(q=p;isdigit(*q);q++);
  *q='\0';
  q++;
  columns = atoi(p);

  // find rows
  for(p=q;!isdigit(*p);p++);
  for(q=p;isdigit(*q);q++);
  *q='\0';
  q++;
  rows =  atoi(p);

  // now make the matrix A from the file
  A = new double[columns*rows];
  file.getline(buf,80); //ignore max value
  while(file.getline(buf,80)) {
    if(*buf == '#') {
      continue;
    }
    q=buf;
    while(*q!='\0' && q<buf+80 && i < columns*rows) {
     for(p=q;!isdigit(*p);p++);
     for(q=p;isdigit(*q);q++);
     *q='\0';
     q++;
     A[i]=atoi(p);
     i++;
    }
  }
  file.close();
}

void matrix::crop(int antal) {
  int mod = 1;
  for(int j = 0;j<antal;j++) {
    mod = mod*2;
  }
```

```
  int newrows = rows - rows%mod;
  int newcols = columns - columns%mod;

  double *newA =  new double[newcols*newrows];

  for(int k=0;k<newcols;k++) {
    for(int j=0;j<newrows;j++) {
      newA[k + j*newcols] = get(j,k);
    }
  }

  delete A;

  columns = newcols;
  rows = newrows;

  A = newA;
}

double matrix::partzero() {
  int j=0;
  for(int i=0;i<rows*columns;i++) {
    if(A[i]==0) {
      j++;
    }
  }
  return (double) j/(rows*columns);
}
```

A.4 dwt.h

```
#ifndef DWT_H
#define DWT_H

#include<iostream>
#include<stdio.h>
#include<vector>
#include<string>
#include<fstream>
#include<cmath>
#include "matrix.h"
#include<zlib.h>
#include<bzlib.h>

class dwt : public matrix {

  int no_of_trans;
  int filterlength;
  double filterh[20];
  double filterg[20];
  double filterhinv[20];
  double filterginv[20];

 public:

  void readfilter(char*);
  void transform(int);
  void itrans();
  void compress(double);
  void bzwritetrans(char*,char*);
  void bzreadtrans(char*,char*);

 private:
  void transrows(int);
  void transcolumns(int);
  void itransrows(int);
  void itranscolumns(int);
  void trans1dim(double*,int);
  void itrans1dim(double*,int);

};

#endif
```

A.5 dwt.cc

```
#include "dwt.h"

void dwt::readfilter(char* name) {
  //hvis filter.name eksisterer
  char buf[80];
  ifstream file(name);
  if(file==NULL) { cout << "file not existing" << endl; }
  file.getline(buf,80);
  while(*buf == '#') {
    file.getline(buf,80);
  }
  filterlength=atoi(buf);
  //lav det ene filter
  for(int i=0;i<filterlength;i++) {
    file.getline(buf,80);
    filterh[i]=strtod(buf,NULL);
  }
  file.close();

  //lav de andre filtre ud fra det foerste
  for(int i=0;i<filterlength;i++) {
    if(i%2==0) {
       filterg[i]=filterh[filterlength-i-1];
    }
    else{
      filterg[i]=-filterh[filterlength-i-1];
    }
  }
  for(int i=0;i<filterlength/2;i++) {
    filterhinv[i]=filterh[filterlength-2-2*i];
    filterhinv[i+filterlength/2]=filterg[filterlength-2-2*i];
    filterginv[i]=filterh[filterlength-1-2*i];
    filterginv[i+filterlength/2]=filterg[filterlength-1-2*i];
  }
}

void dwt::transform(int antal) {
  (*this).crop(antal);
  no_of_trans = antal;
  int j = rows;
  for(int i=0;i<antal;i++) {
    (*this).transcolumns(j);
    j = j/2;
  }
  j = columns;
  for(int i=0;i<antal;i++) {
    (*this).transrows(j);
```

```
    j = j/2;
  }
}

void dwt::transrows(int c) {
  double TMP1[c];
  for(int i=0;i<rows;i++) {
    for(int j=0;j<c;j++) {
      TMP1[j]=A[i*columns+j];
    }
    trans1dim(TMP1,c);
    for(int j=0;j<c;j++) {
      A[i*columns+j]=TMP1[j];
    }
  }
}

void dwt::transcolumns(int r) {
  double TMP1[r];
  for(int i=0;i<columns;i++) {
    for(int j=0;j<r;j++) {
      TMP1[j]=A[j*columns+i];
    }
    trans1dim(TMP1,r);
    for(int j=0;j<r;j++) {
      A[j*columns+i]=TMP1[j];
    }
  }
}

void dwt::itrans() {
  int antal = no_of_trans;
  int j = columns;
  for(int i=0;i<antal;i++) {
    j = j/2;
  }

  for(int i=0;i<antal;i++) {
    j = 2*j;
    (*this).itransrows(j);
  }
  j = rows;
  for(int i=0;i<antal;i++) {
    j = j/2;
  }
  for(int i=0;i<antal;i++) {
    j = 2*j;
    (*this).itranscolumns(j);
  }
```

```
}

void dwt::itransrows(int c) {
  double TMP1[c];
  for(int i=0;i<rows;i++) {
    for(int j=0;j<c;j++) {
      TMP1[j]=A[i*columns+j];
    }
    itrans1dim(TMP1,c);
    for(int j=0;j<c;j++) {
      A[i*columns+j]=TMP1[j];
    }
  }
}

void dwt::itranscolumns(int r) {
  double TMP1[r];
  for(int i=0;i<columns;i++) {
    for(int j=0;j<r;j++) {
      TMP1[j]=A[j*columns+i];
    }
    itrans1dim(TMP1,r);
    for(int j=0;j<r;j++) {
      A[j*columns+i]=TMP1[j];
    }
  }
}

// compare is the same as <
int compare(const void *a,const void *b) {
  double c;
  c = *((double*)a) - *((double*)b);
  if(c<0) {
    return -1;
  }
  if(c>0) {
    return 1;
  }
  return 0;
}

//e can be between 0 and 1
void dwt::compress(double e) {
  double smallest;
  double TMP[columns*rows];
  for(int i=0;i<columns*rows;i++) {
    TMP[i]=fabs(A[i]);
  }
  qsort(TMP,columns*rows,sizeof(double),compare);
```

```
  smallest = TMP[(int) rint(e*columns*rows)];
  for(int i=0;i<rows*columns;i++) {
    if(fabs(A[i])<=smallest) {
      A[i]=0;
    }
    else {
      A[i]=rint(A[i]);
    }
  }
}

void dwt::bzwritetrans(char* name,char* format) {
  BZFILE* bzfile;
  int     bzerror;
  int     nWritten;
  int     nBuf;
  char    buf[80];
  int entry=0;
  bzfile = BZ2_bzopen (name,"w" );
  sprintf(buf,"%s\n",format);
  BZ2_bzWrite ( &bzerror, bzfile, buf, strlen(buf) );
  sprintf(buf,"%d %d\n",columns,rows);
  BZ2_bzWrite ( &bzerror, bzfile, buf, strlen(buf) );
  sprintf(buf,"%d\n",no_of_trans);
  BZ2_bzWrite ( &bzerror, bzfile, buf, strlen(buf) );
  for(int i=0;i<columns*rows;i++) {
    if(A[i]!=0) {
      sprintf(buf,"%d %d\n",i-entry,(int) A[i]);
      BZ2_bzWrite ( &bzerror, bzfile, buf, strlen(buf) );
      entry=i;
    }
  }
  BZ2_bzclose(bzfile);
}

char* bzgets(BZFILE* f,char* b,int l) {
  char* q;
  q=b;
  int bzerror;
  for(int i=0;i<l;i++) {
    BZ2_bzRead ( &bzerror, f, q, 1 );
    if(bzerror==BZ_STREAM_END) {
      return NULL;
    }
    if(*q=='\n') {
      q++;
      *q='\0';
      return b;
```

```
    }
    else {
      q++;
    }
  }
  return b;
}

void dwt::bzreadtrans(char* name,char* format) {
  int bzerror;
  BZFILE* bzfile = BZ2_bzopen(name,"r");
  char buf[80];
  char *p,*q;
  int i = 0;
  int mod = 1;
  int entry=0;
  delete A;
  bzgets(bzfile,buf,80);
  if (*buf == *format); //ignore
  bzgets(bzfile,buf,80);
  // find columns
  q=buf;
  for(p=q;!isdigit(*p);p++);
  for(q=p;isdigit(*q);q++);
  *q='\0';
  q++;
  columns = atoi(p);
  // find rows
  for(p=q;!isdigit(*p);p++);
  for(q=p;isdigit(*q);q++);
  *q='\0';
  q++;
  rows =  atoi(p);
  // now make the trans A from the file
  A = new double[columns*rows];
  bzgets(bzfile,buf,80);
  no_of_trans=atoi(buf);
  while(bzgets(bzfile,buf,80)!=Z_NULL) {
    //lav matricen
    q=buf;
    for(p=q;!isdigit(*p);p++);
    for(q=p;isdigit(*q);q++);
    *q='\0';
    q++;
    entry = entry+atoi(p);

    for(p=q;!isdigit(*p) && *p!='-';p++);
    for(q=p;isdigit(*q)||*q=='.'||*q=='-'||*q=='e';q++);
    *q='\0';
```

```
    q++;
    A[entry]=strtod(p,NULL);
  }
  BZ2_bzclose(bzfile);
}

void dwt::trans1dim(double* X,int length) {
  double TMP[length];
  for(int i=0;i<length;i++) {
    TMP[i]=0;
  }
  int j= length/2;
  int j1[filterlength];
  for(int i=0;i<j;i++) {
    //udtrk de entries der skal bruges
    j1[0]=2*i;
    for(int k=1;k<filterlength;k++) {
      j1[k]=(j1[k-1]+1)%length;
    }
    //udregn selve transformationen
    for(int k=0;k<filterlength;k++) {
      TMP[i]=TMP[i]+filterh[k]*X[j1[k]];
      TMP[i+j]=TMP[i+j]+filterg[k]*X[j1[k]];
    }
  }
  for(int i=0;i<length;i++) {
    X[i] = TMP[i];
  }
}


void dwt::itrans1dim(double* X,int length) {
  double TMP[length];
  for(int i=0;i<length;i++) {
    TMP[i]=0;
  }
  int j= length/2;
  int j1[filterlength];
  for(int i=0;i<filterlength;i++) {
    j1[i]=1;
  }
  for(int i=0;i<j;i++) {
    j1[0]=(j+i-filterlength/2+1)%j; //filteret skal have mindst laengde 2
    j1[filterlength/2]=j1[0]+j;
    for(int k=1;k<filterlength/2;k++) { //filterlength skal vaere lige
      j1[k]=(j1[k-1]+1)%j;
      j1[k+filterlength/2]=j1[k]+j;
    }
    //udregn selve transformationen
```

```
    for(int k=0;k<filterlength;k++) {
      TMP[2*i]=TMP[2*i]+filterhinv[k]*X[j1[k]];
      TMP[2*i+1]=TMP[2*i+1]+filterginv[k]*X[j1[k]];
    }
  }
  for(int i=0;i<length;i++) {
    X[i] = TMP[i];
  }
}
```

A.6 enc.cc

```
#include <stdio.h>
#include <iostream.h>
#include "dwt.h"

int main(int argc,char** argv) {
  if(argc!=6) {
    cout << "Usage: enc type #transforms rate infile outfile" << endl;
    return -1;
  }
  char* type = argv[1];
  int no=atoi(argv[2]);
  char* infile = argv[4];
  char* outfile = argv[5];
  double rate=strtod(argv[3],NULL);

  dwt d;
  d.readfilter(type);
  d.readpgm(infile);
  d.transform(no);
  d.compress(rate);
  d.bzwritetrans(outfile,type);
  return 0;
}
```

A.7 dec.cc

```
#include <stdio.h>
#include <iostream.h>
#include "dwt.h"

int main(int argc,char** argv) {
  int bzerror;
  char buf[80];
  char *q;
  if(argc!=3) {
    cout << "Usage: dec infile outfile" << endl;
    return -1;
  }
  char* infile = argv[1];
  BZFILE* file = BZ2_bzopen(infile,"rb9");
  BZ2_bzRead(&bzerror,file,buf,80);
  BZ2_bzclose(file);
  char* outfile = argv[2];

  dwt d;
  //hvis filter eksisterer ???
  for(q=buf;!isspace(*q);q++);
  *q='\0';
  d.readfilter(buf);
  d.bzreadtrans(infile,buf);
  d.itrans();
  d.writepgm(outfile);
  return 0;
}
```

B PGM format specification

C Sample pictures

C.1 Original pictures

Figure 4: Lena, original picture

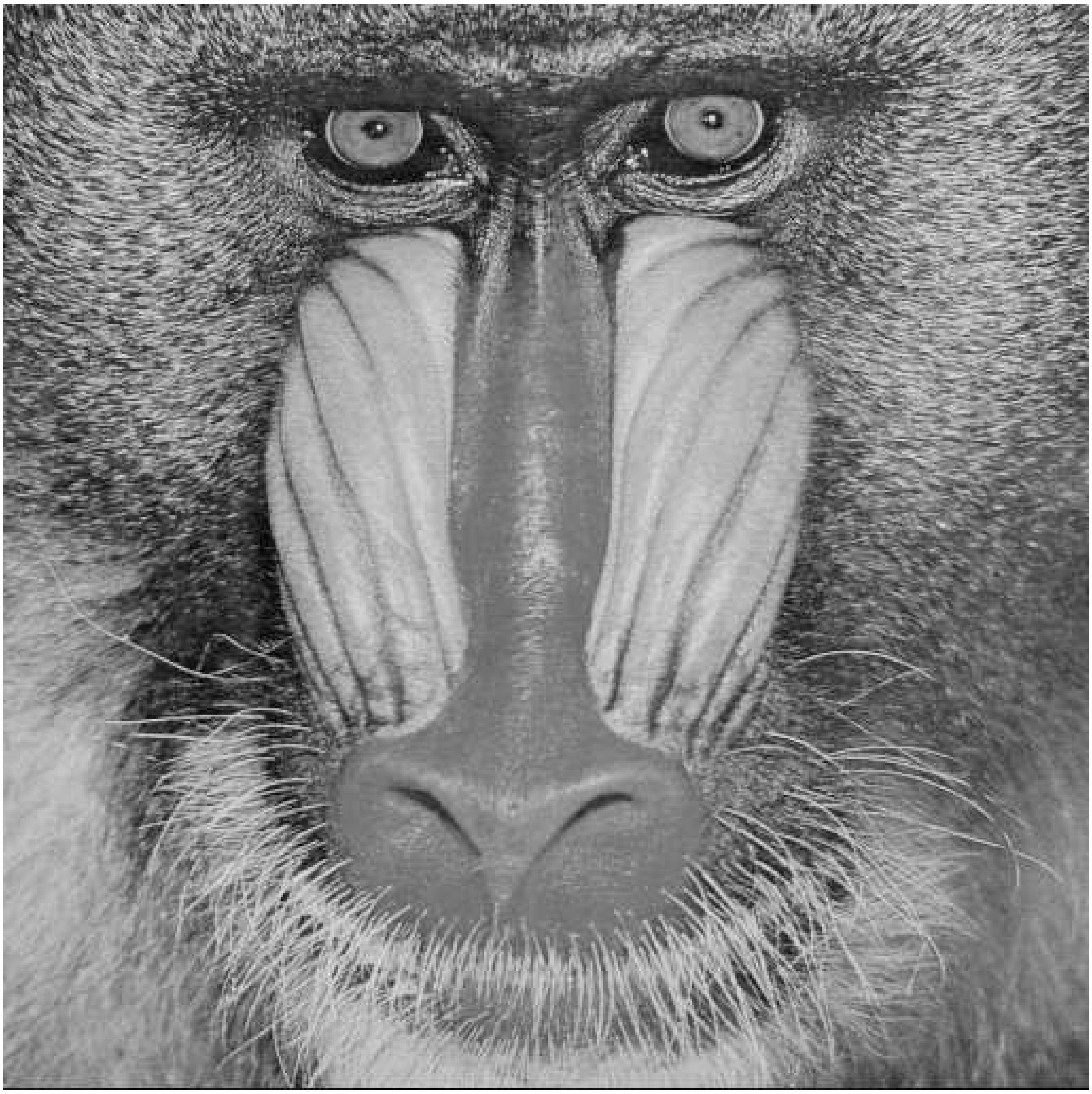

Figure 5: Baboon, original picture

C.2 Pictures with 99% compression

Figure 6: Lena,Haar, 5 transforms, 99% removed,size=5970 bytes

Figure 7: Lena,Daubechies-4, 5 transforms, 99% removed,size=5895 bytes

Figure 8: Lena,Daubechies-6, 5 transforms, 99% removed,size=5872 bytes

Figure 9: Lena, JPEG corresponding to: 5 transforms, 99% removed,size=5449 bytes

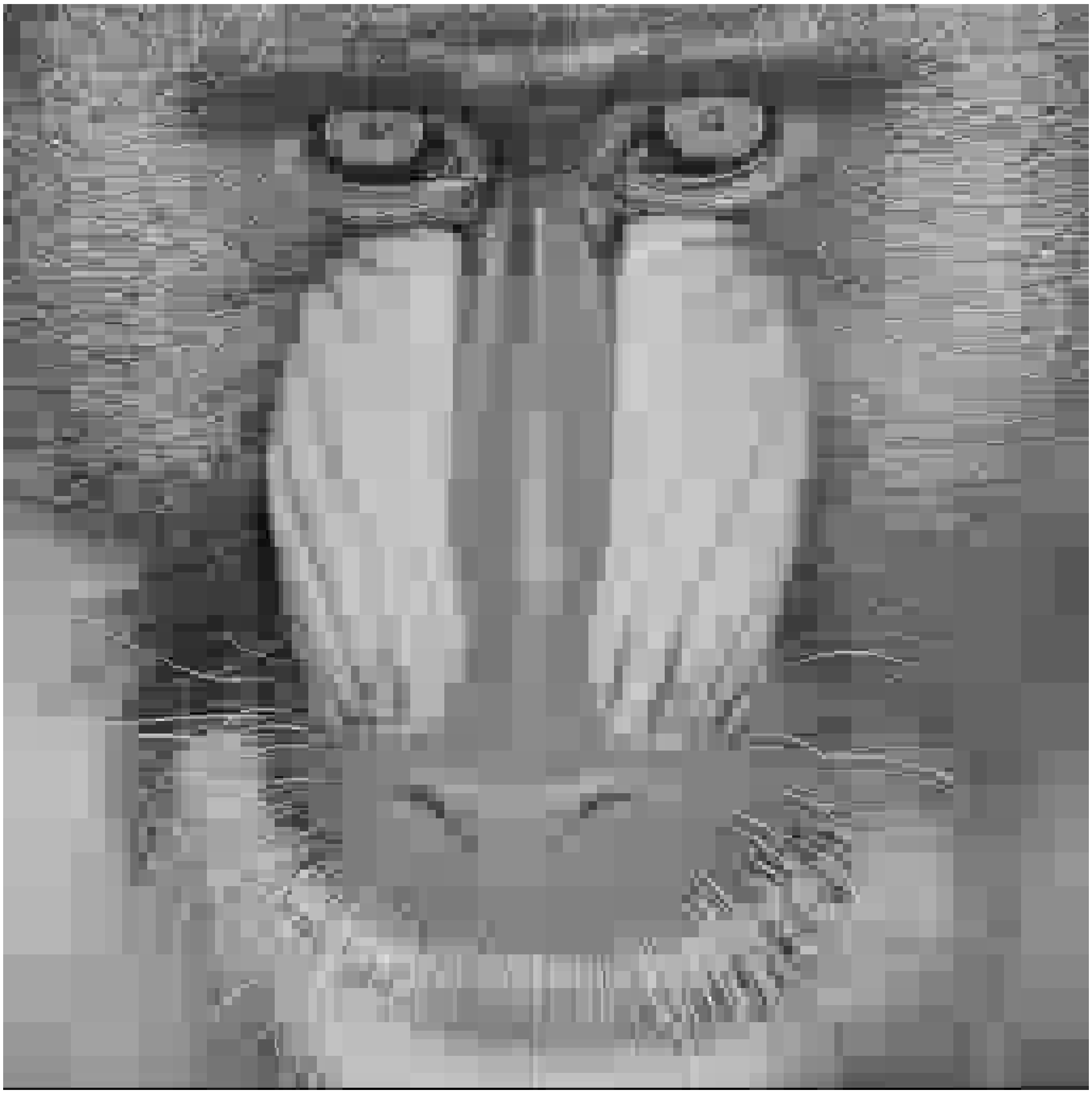

Figure 10: Baboon,Haar, 5 transforms, 99% removed,size=5981 bytes

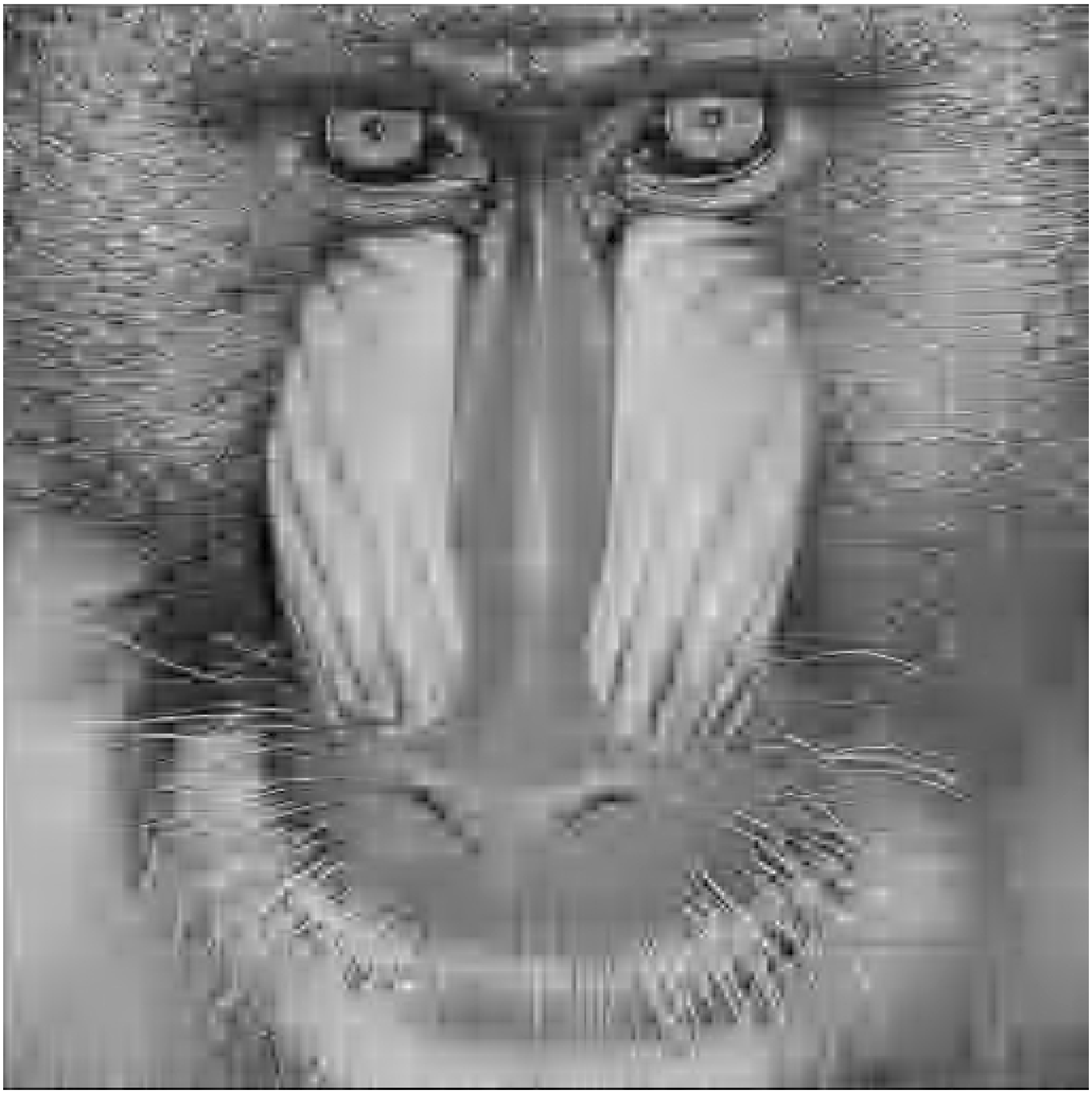

Figure 11: Baboon,Daubechies-4, 5 transforms, 99% removed,size=5920 bytes

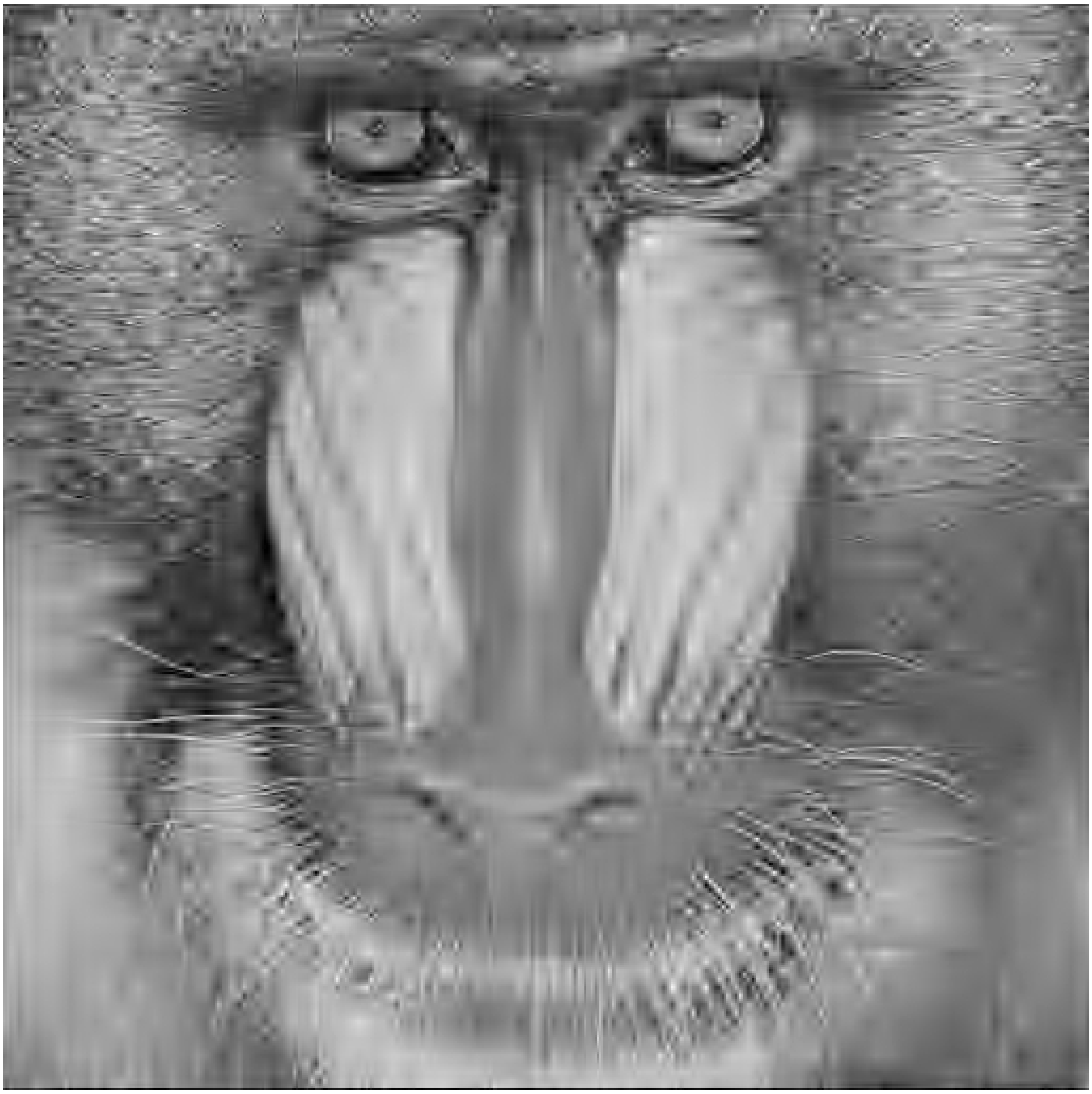

Figure 12: Baboon,Daubechies-6, 5 transforms, 99% removed,size=5938 bytes

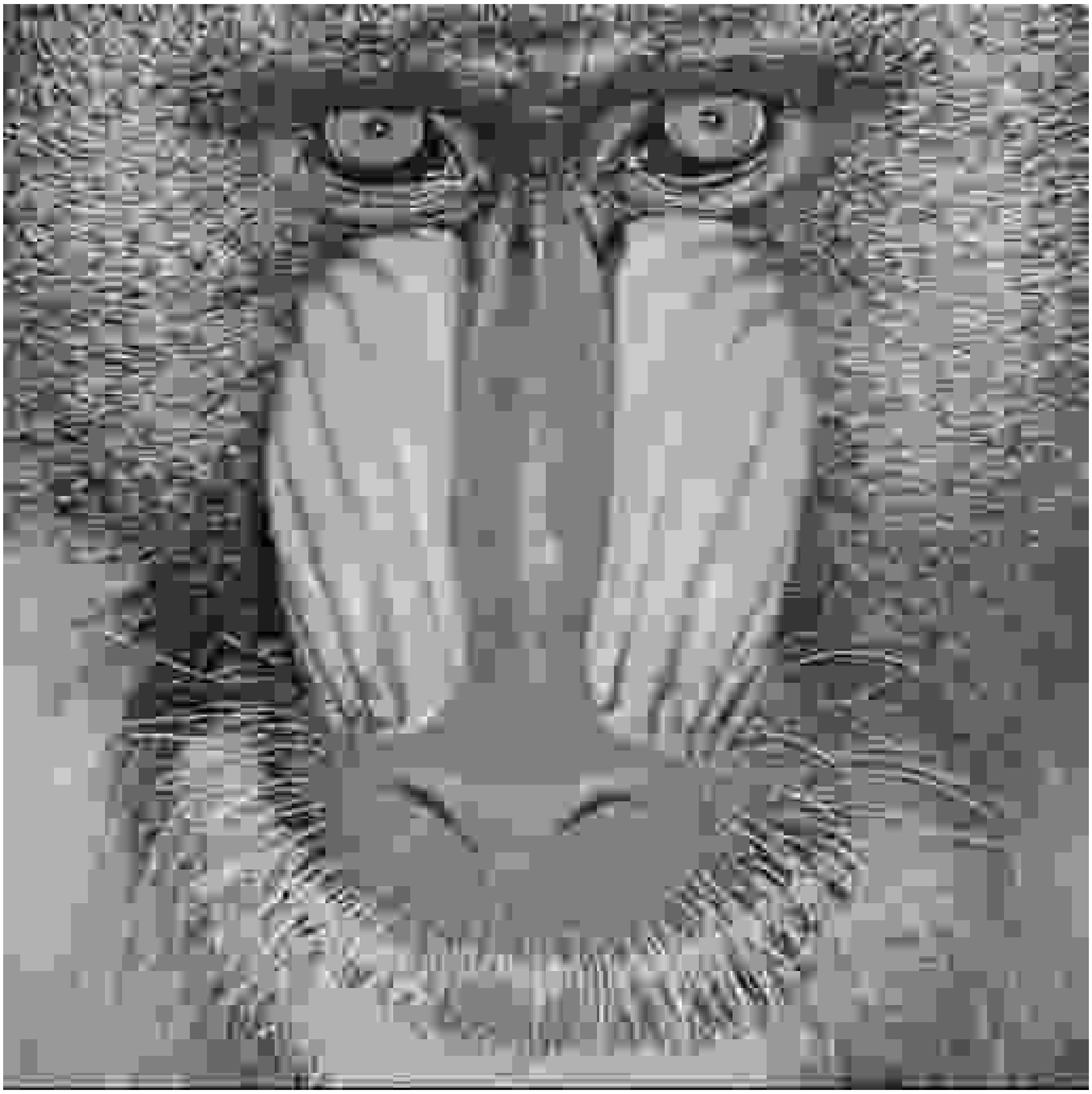

Figure 13: Baboon, JPEG corresponding to: 5 transforms, 99% removed, size=6772 bytes

C.3 Pictures with 98% compression

Figure 14: Lena,Haar, 5 transforms, 98% removed,size=10792 bytes

Figure 15: Lena,Daubechies-4, 5 transforms, 98% removed,size=10647 bytes

Figure 16: Lena,Daubechies-6, 5 transforms, 98% removed,size=10486 bytes

Figure 17: Lena, JPEG corresponding to: 5 transforms, 98% removed, size=10455 bytes

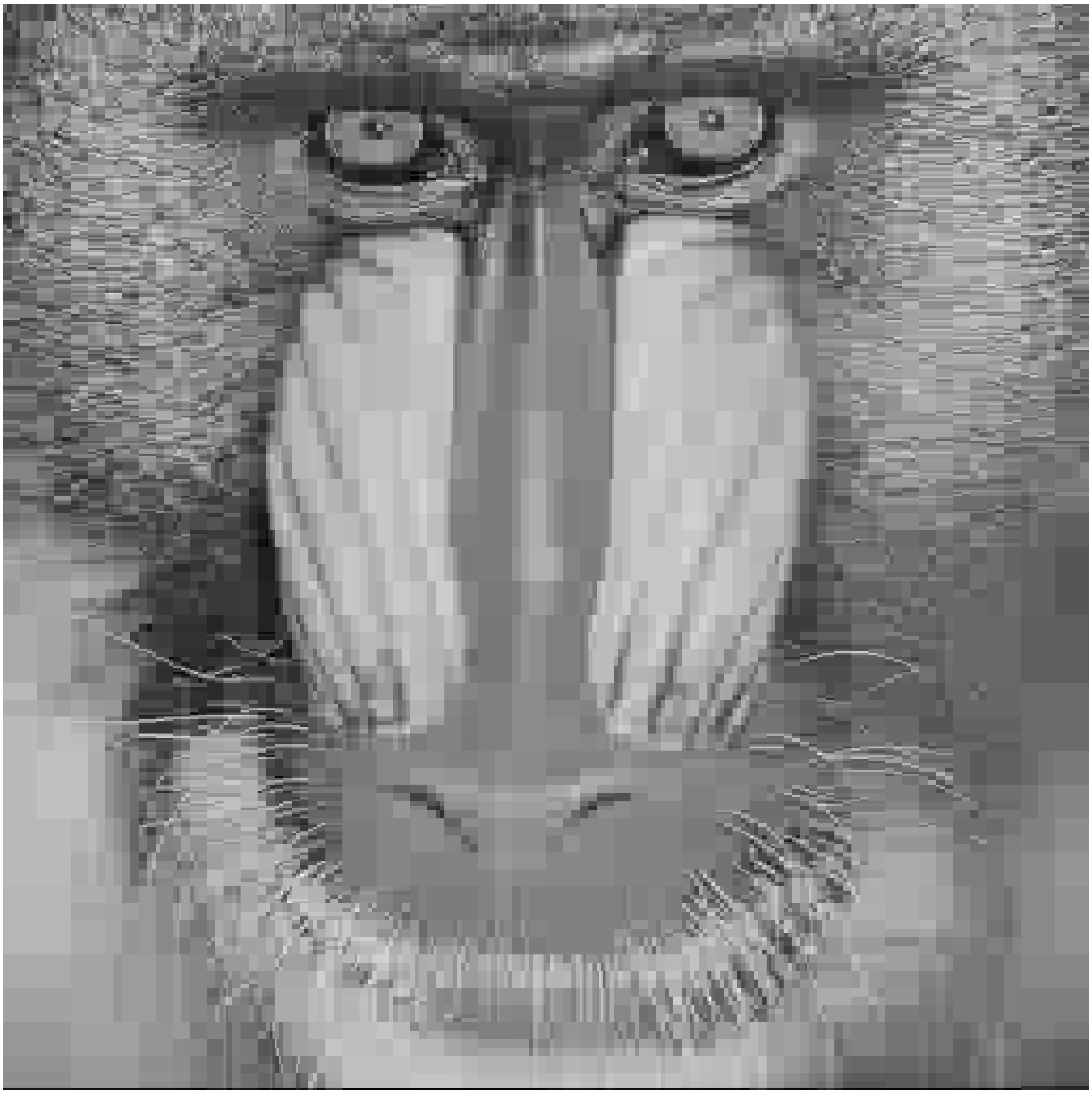

Figure 18: Baboon,Haar, 5 transforms, 98% removed,size=10852 bytes

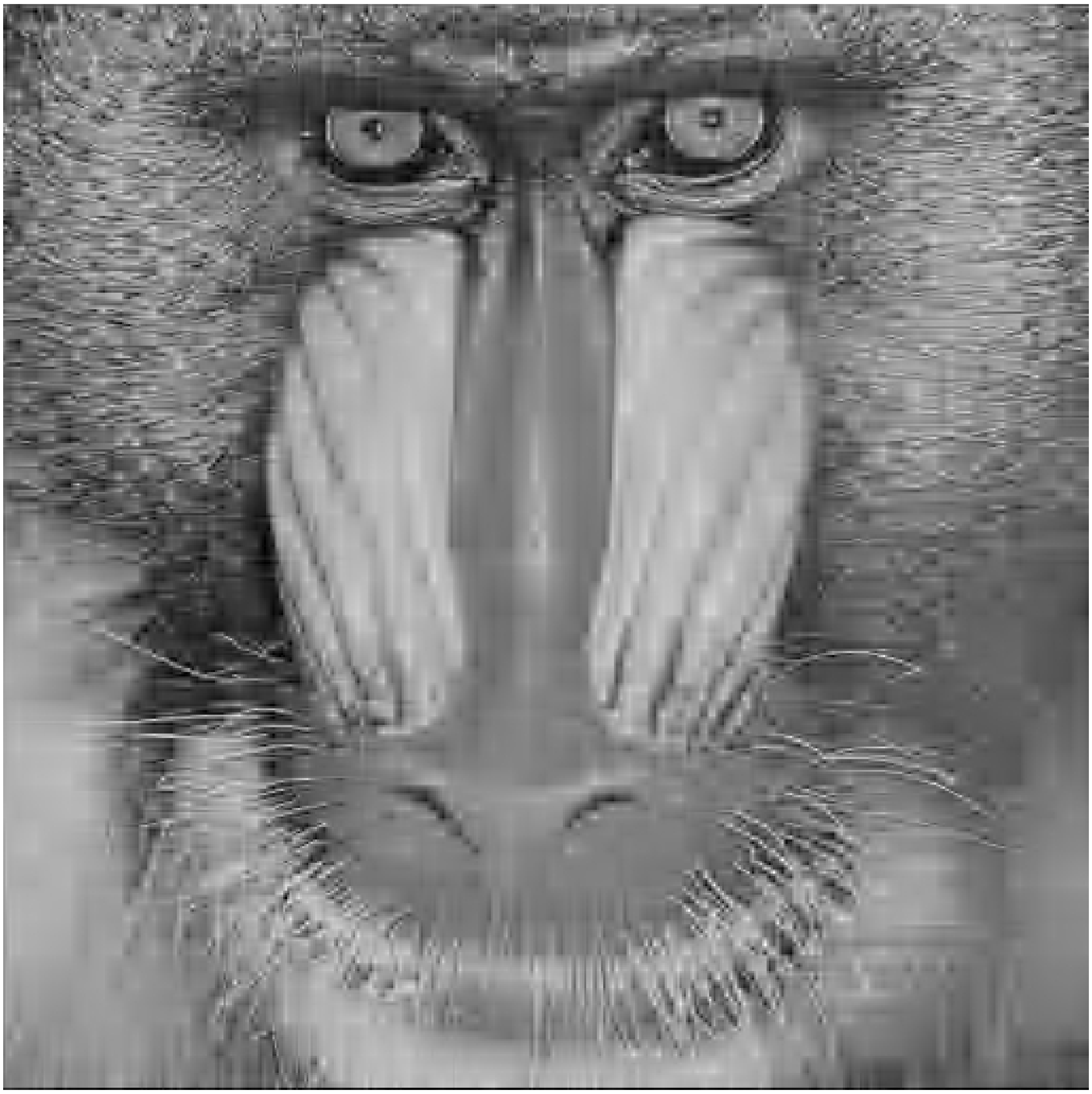

Figure 19: Baboon,Daubechies-4, 5 transforms, 98% removed,size=10719 bytes

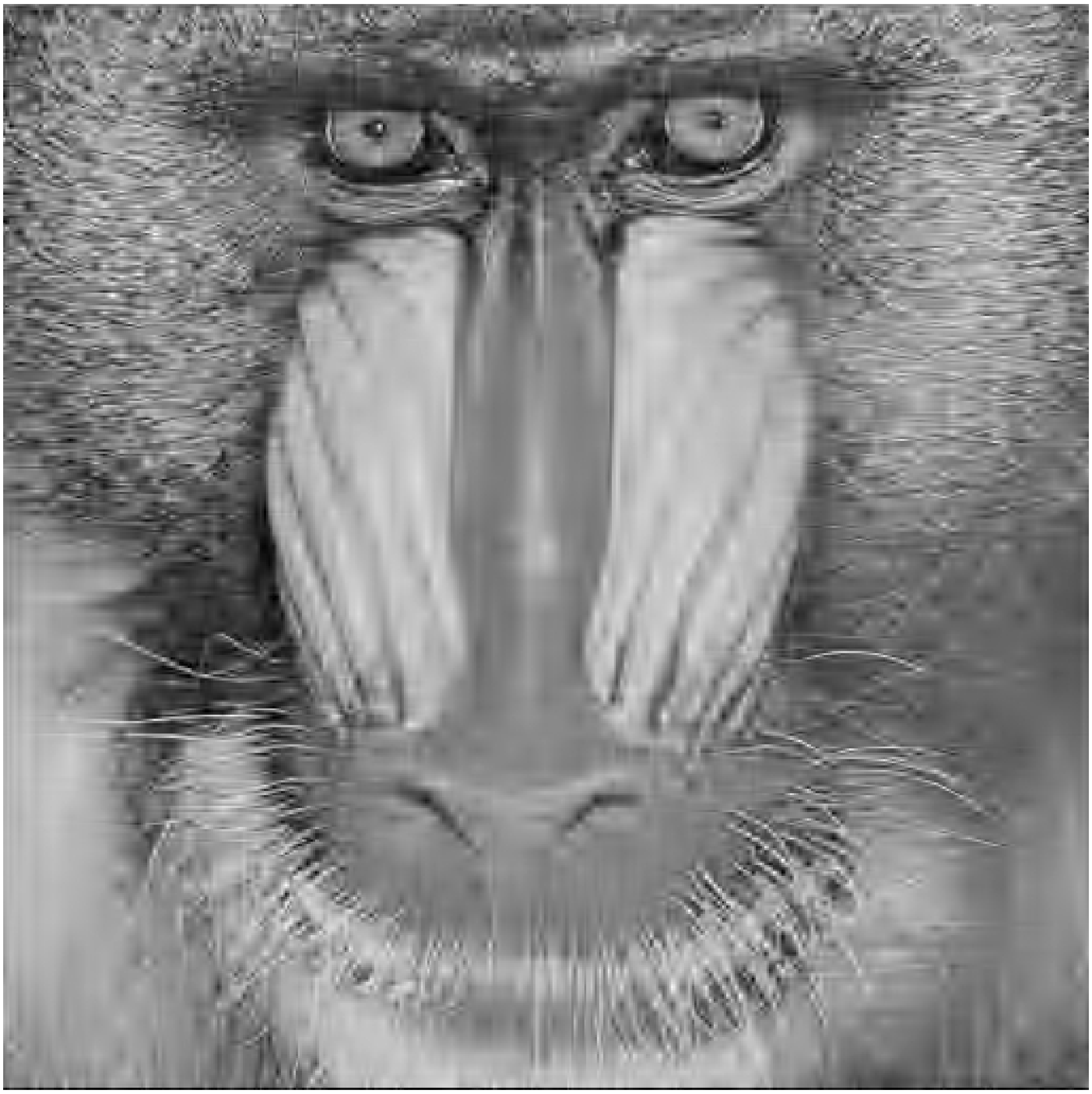

Figure 20: Baboon,Daubechies-6, 5 transforms, 98% removed,size=10725 bytes

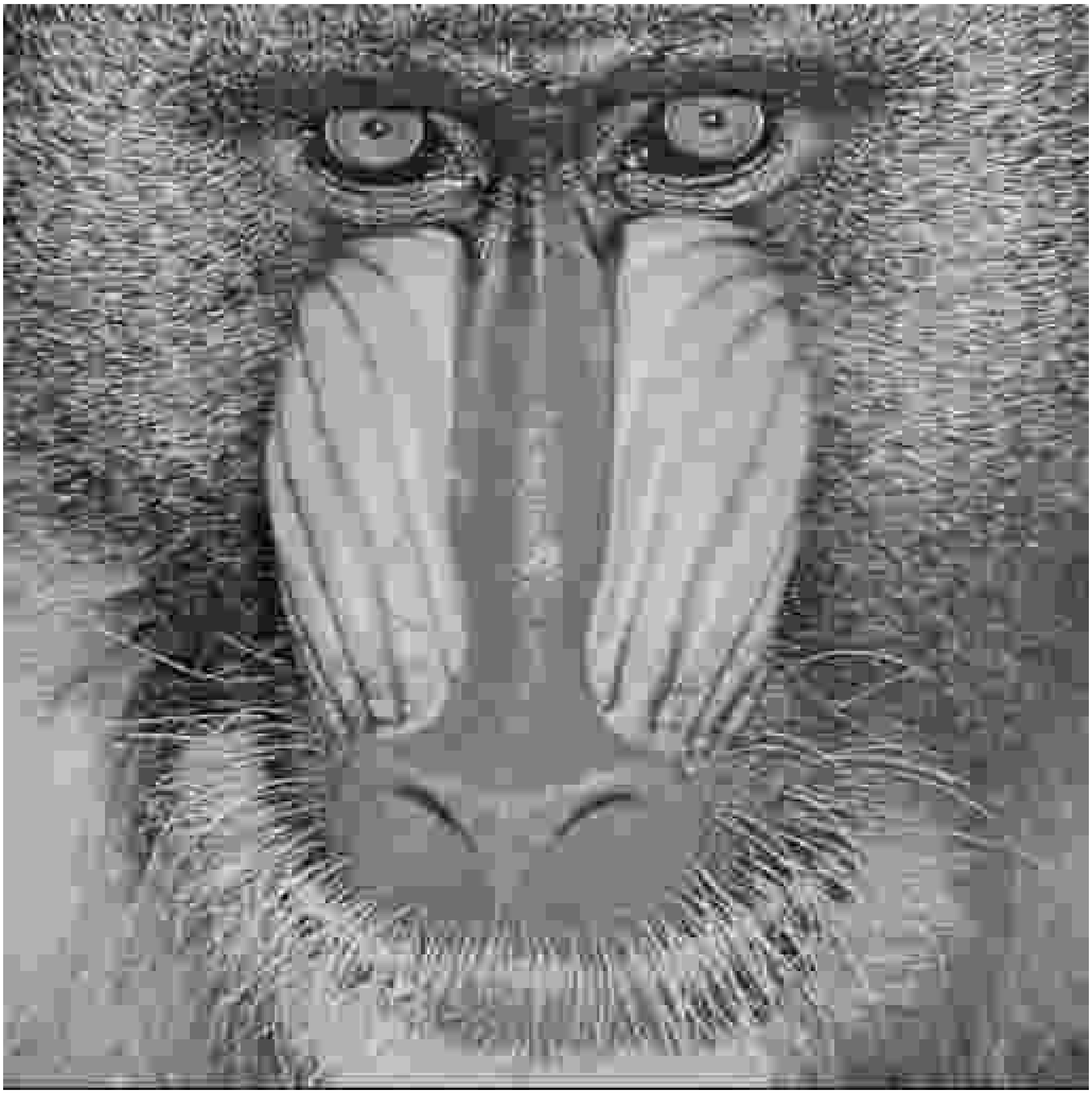

Figure 21: Baboon, JPEG corresponding to: 5 transforms, 98% removed, size=9823 bytes

C.4 Pictures with 97% compression

Figure 22: Lena,Haar, 5 transforms, 97% removed,size=15256 bytes

Figure 23: Lena,Daubechies-4, 5 transforms, 97% removed,size=15014 bytes

Figure 24: Lena,Daubechies-6, 5 transforms, 97% removed,size=14699 bytes

Figure 25: Lena, JPEG corresponding to: 5 transforms, 97% removed, size=14440 bytes

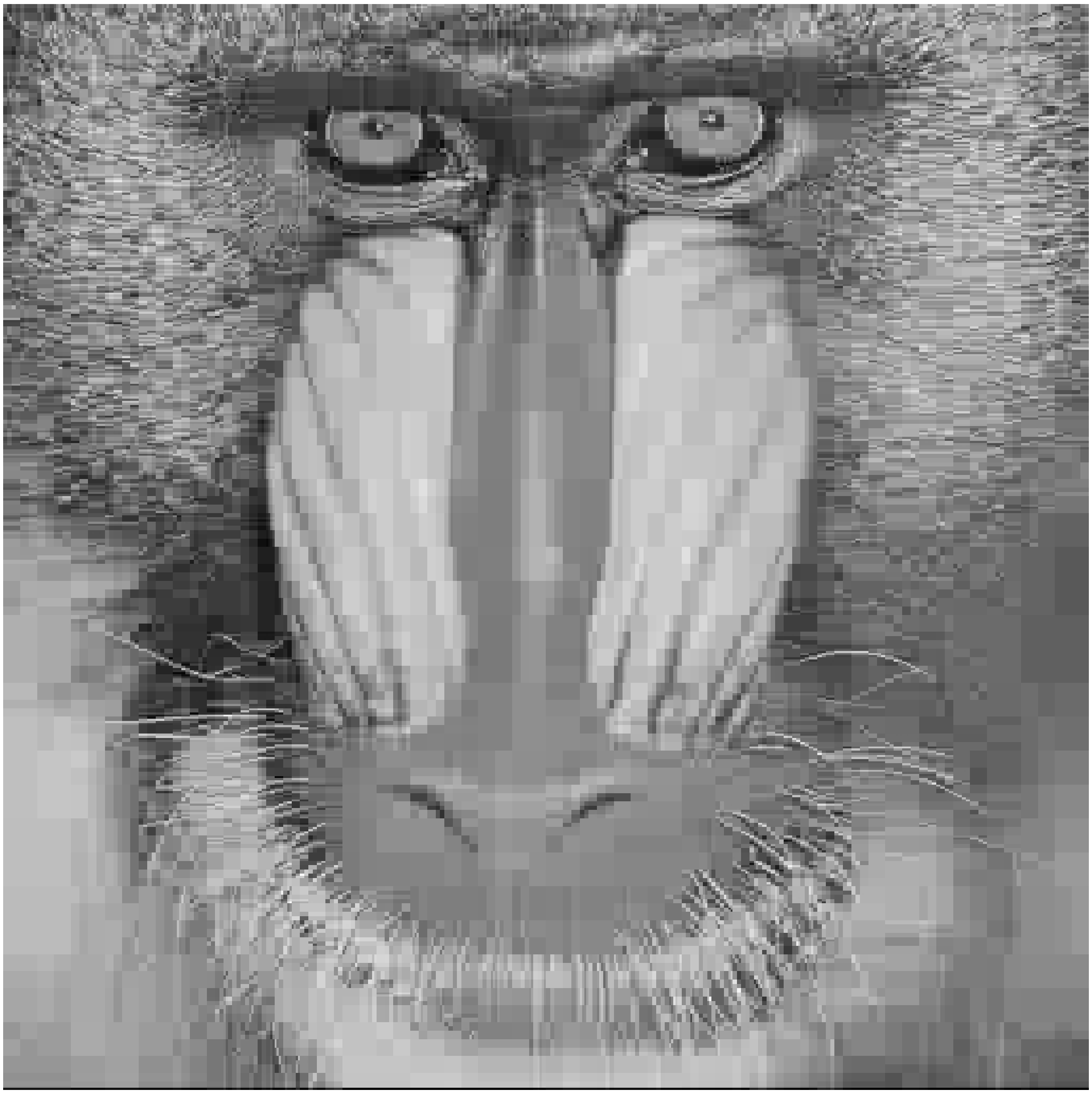

Figure 26: Baboon,Haar, 5 transforms, 97% removed,size=15346 bytes

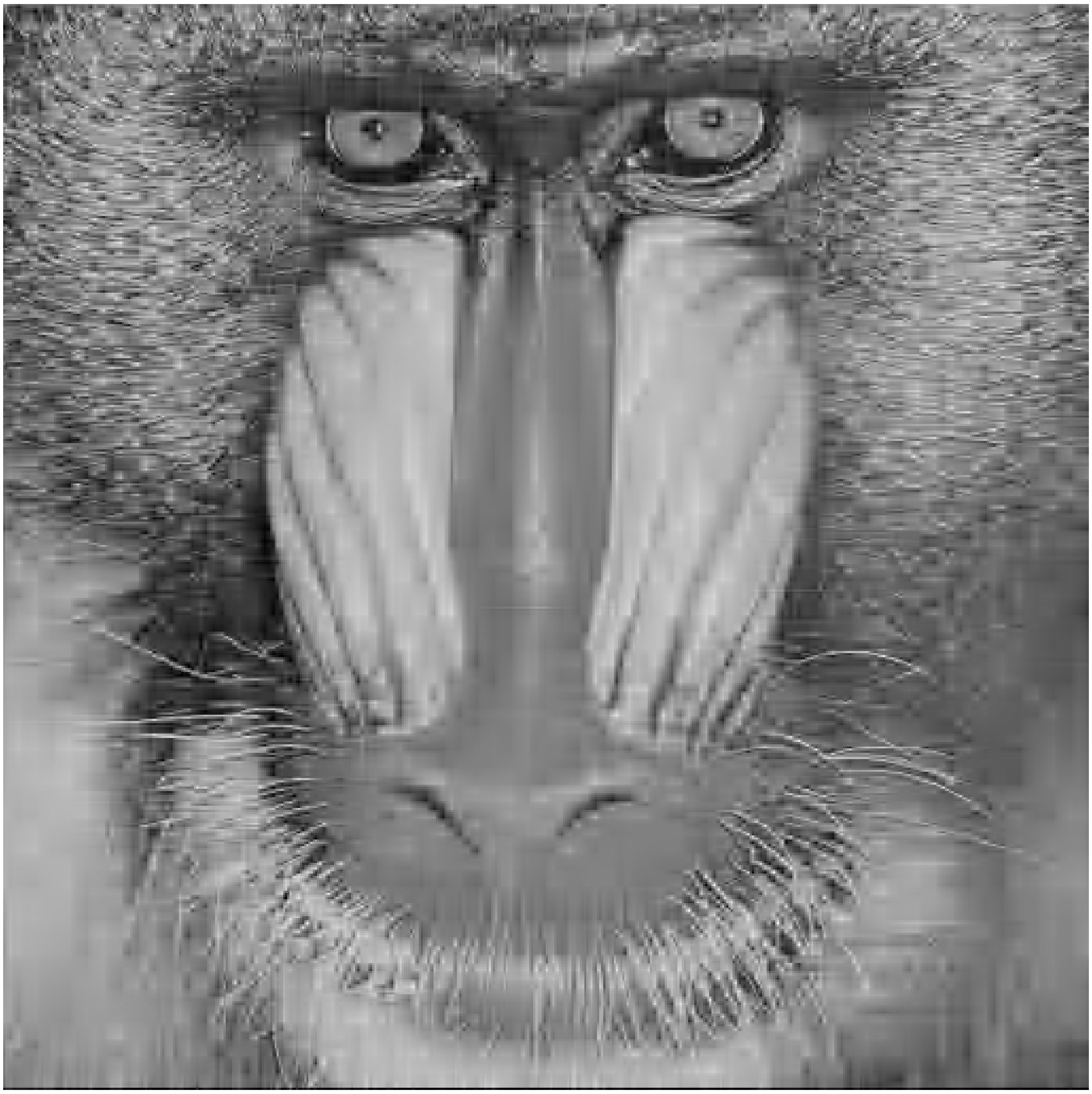

Figure 27: Baboon,Daubechies-4, 5 transforms, 97% removed,size=15230 bytes

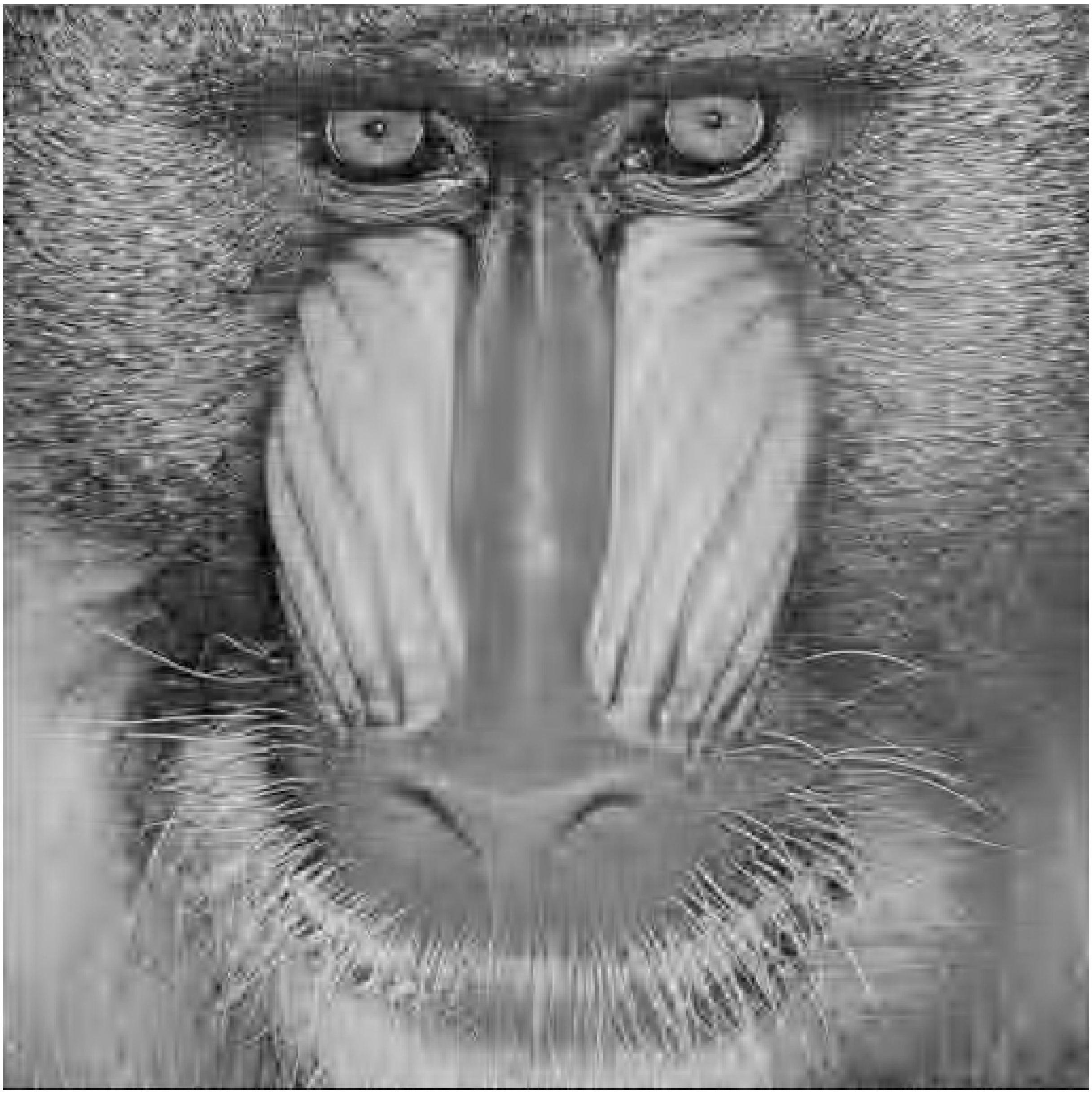

Figure 28: Baboon,Daubechies-6, 5 transforms, 97% removed,size=15234 bytes

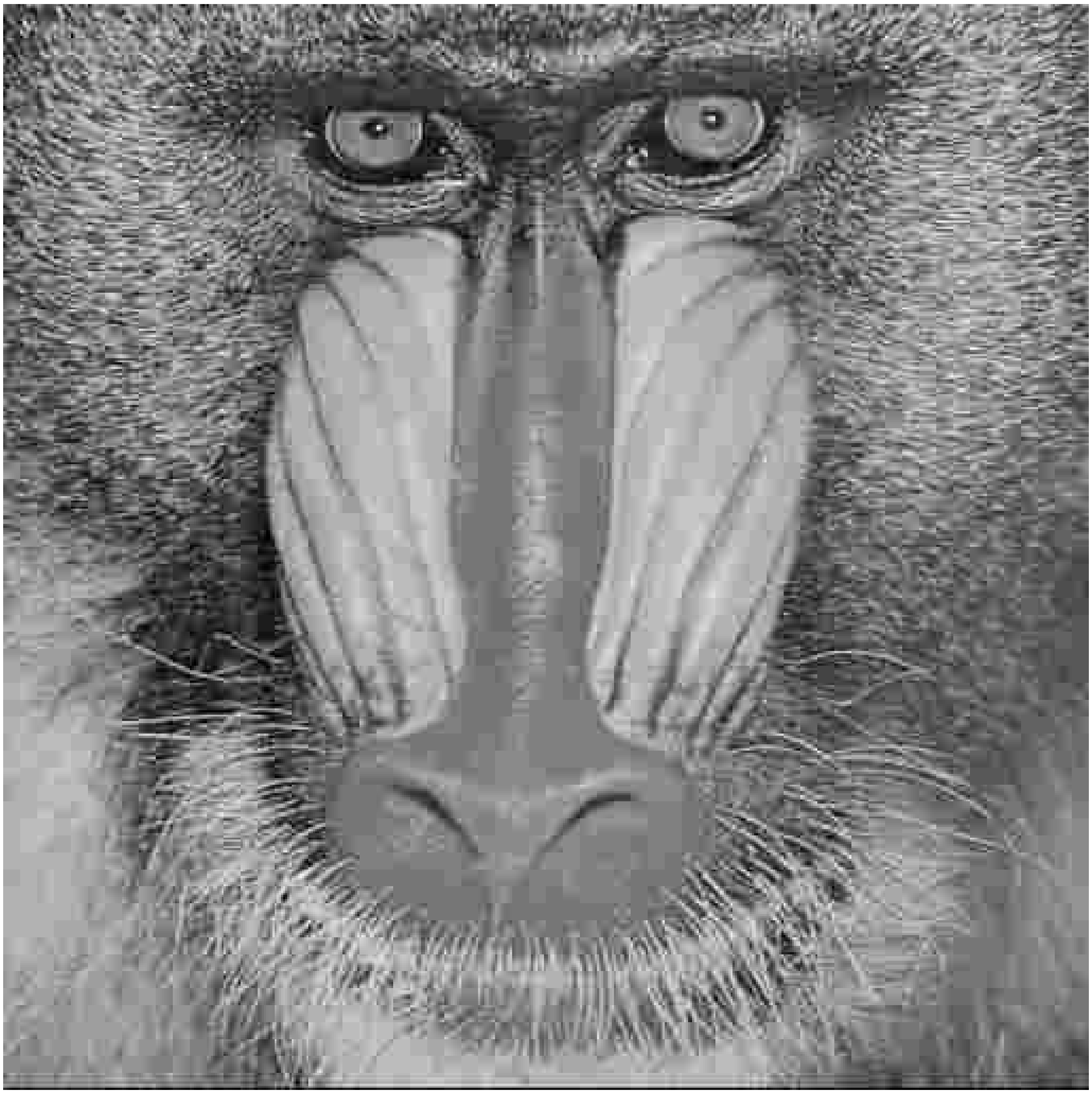

Figure 29: Baboon, JPEG corresponding to: 5 transforms, 97% removed, size=15004 bytes

D The Haar Wavelet

The scaling function for the Haar wavelet looks like this

$$\varphi(x) = \begin{cases} 1 & \text{for } 0 \le x \le 1, \\ 0 & \text{elsewhere.} \end{cases} \tag{17}$$

The coefficients for the Haar wavelet are: $h_0 = h_1 = 1/\sqrt{2}$. Using only h_0 the matrices for the transform look like this:

Matrix for decomposition of picture

$$\begin{pmatrix} h_0 & h_0 & & & & & & \\ & & h_0 & h_0 & & & & \\ & & & & \ddots & & & \\ & & & & & & h_0 & h_0 \\ h_0 & -h_0 & & & & & & \\ & & h_0 & -h_0 & & & & \\ & & & & \ddots & & & \\ & & & & & & h_0 & -h_0 \end{pmatrix}$$

Matrix for reconstruction of picture

$$\begin{pmatrix} h_0 & & & & h_0 & & & \\ h_0 & & & & -h_0 & & & \\ & h_0 & & & & h_0 & & \\ & h_0 & & & & -h_0 & & \\ & & \ddots & & & & \ddots & \\ & & & h_0 & & & & h_0 \\ & & & h_0 & & & & -h_0 \end{pmatrix}$$

E The Daubechies-4 Wavelet

The coefficients for the Daubechies-4 wavelet are:
$h_0 = 0.4829629$
$h_1 = 0.8365163$
$h_2 = 0.2241439$
$h_3 = -0.1294095$

Matrix for decomposition of picture

$$
\begin{pmatrix}
h_0 & h_1 & h_2 & h_3 & & & & & & & \\
 & & h_0 & h_1 & h_2 & h_3 & & & & & \\
 & & & & & & \ddots & & & & \\
 & & & & & & & h_0 & h_1 & h_2 & h_3 \\
h_2 & h_3 & & & & & & & & h_0 & h_1 \\
h_3 & -h_2 & h_1 & -h_0 & & & & & & & \\
 & & h_3 & -h_2 & h_1 & -h_0 & & & & & \\
 & & & & & & \ddots & & & & \\
 & & & & & & & h_3 & -h_2 & h_1 & -h_0 \\
h_1 & -h_0 & & & & & & & & h_3 & -h_2
\end{pmatrix}
$$

Matrix for reconstruction of picture

$$
\begin{pmatrix}
h_0 & & & & h_2 & h_3 & & & & h_1 \\
h_1 & & & & h_3 & -h_2 & & & & -h_0 \\
h_2 & h_0 & & & & h_1 & h_3 & & & \\
h_3 & h_1 & & & & -h_0 & -h_2 & & & \\
 & h_2 & & & & & h_1 & & & \\
 & h_3 & & & & & -h_0 & & & \\
 & & \ddots & & & & & \ddots & & \\
 & & & h_0 & & & & & h_3 & \\
 & & & h_1 & & & & & -h_2 & \\
 & & & h_2 & h_0 & & & & h_1 & h_3 \\
 & & & h_3 & h_1 & & & & -h_0 & -h_2
\end{pmatrix}
$$