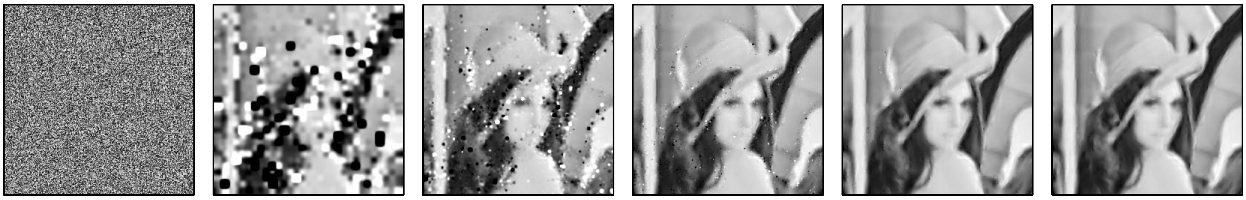
Fractal Coding by Means of Local Feature Histograms



Per-Erik Forssén and Björn Johansson

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden
{perfo,bjorn}@isy.liu.se

LiTH-ISY-R-2295
2000-09-15
ISSN 1400-3902

Fractal Coding by Means of Local Feature Histograms

Per-Erik Forssén and Björn Johansson
Computer Vision Laboratory
Department of Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden
{perfo,bjorn}@isy.liu.se

September 18, 2000

Abstract

This report describes an experimental still image coder that grew out of a project in the graduate course "Advanced Video Coding" in spring 2000.

The project has investigated the idea to use local orientation histograms in fractal coding. Instead of performing a correlation-like grey-level matching of image regions, the block search is made by matching feature histograms of the block contents. The feature investigated in this report is local orientation, but in principle other features could be used as well.

In its current state the coder does not outperform state of the art still image coders, but the block-search strategy seems promising, and will probably prove useful in several other applications.

Contents

1 Introduction **4**

2 Local orientation **4**

3 Local orientation histograms **5**

4 Matching of histograms **7**
4.1 Finding the best histogram regardless of rotation 7
4.2 Mirrored histograms 8
4.3 Finding the best rotation of a histogram 8

5 Finding a grey-level mapping **10**

6 Bit rates **11**

7 Decoding **13**
7.1 Iterations 13
7.2 Decoding performance 14

8 Experiments **14**
8.1 Varied overlap 14
8.2 Two and four grey-level parameters 14
8.3 Fractal coding of fractals 14
8.4 Comparison with other coders 17

9 Conclusions **19**

10 Acknowledgment **19**

1 Introduction

This project has investigated the idea to use local orientation histograms in fractal coding (see e.g. [2] for details about fractal coding). The search for the best transform from a part of the image to a block is usually very time consuming and is generally limited to correlation with a block and a down-sampled version of the image. If the content in every block and every local area in the image is represented by an orientation histogram then the search can be made more efficient by simply matching histograms. This match is not optimal in the least square sense, but in general the matches have similar appearence. The method sometimes gives a more perceptually satisfying result than the result from a simple least square approach.

In other words, for each image block we assume the fractal model (adopted from [2])

$$w(\begin{pmatrix} \mathbf{x} \\ I(\mathbf{x}) \end{pmatrix}) = \begin{pmatrix} \mathbf{A} & \mathbf{0} \\ p_1\ p_2 & p_3 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ I(\mathbf{x}) \end{pmatrix} + \begin{pmatrix} \mathbf{b} \\ p_4 \end{pmatrix} \quad (1)$$

The main idea to find the parameters $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{p}$ is as follows:

1. Calculate suitable local image features. In this case image gradients in several scales was chosen.
2. Calculate local feature histograms in several scales. These will represent the content in local image areas.
3. For each image block:
   (a) Find the local image area that best matches the block by comparing their histograms.
   (b) Calculate $\mathbf{A} = s\mathbf{RM}$ and $\mathbf{b}$ using the best matched histograms. s is a scale factor, $\mathbf{R}$ is a rotation matrix, and $\mathbf{M}$ is a mirroring matrix.
   (c) Calculate $\mathbf{p} = (p_1\ p_2\ p_3\ p_4)$ using weighted least square on equation 1.

2 Local orientation

Local orientation is an important feature in human vision and is also a very common feature in any kind of image. We have therefore chosen local orientation as feature in our matching procedure.

Local orientation can be estimated in a number of ways, e.g. by finding simple signals, edges, lines etc. In this project we have used a simple edge detector - calculate the image gradient $\nabla I = (\frac{\partial Im}{\partial x}, \frac{\partial Im}{\partial y})$ using derivating Gaussian filters:

$$\frac{\partial Im}{\partial x} = Im\ *\ (\frac{x}{\sigma_\nabla^2} e^{-\frac{x^2}{2\sigma_\nabla^2}})\ *\ (e^{-\frac{y^2}{2\sigma_\nabla^2}})$$

$$\frac{\partial Im}{\partial y} = Im\ *\ (\frac{y}{\sigma_\nabla^2} e^{-\frac{y^2}{2\sigma_\nabla^2}})\ *\ (e^{-\frac{x^2}{2\sigma_\nabla^2}})$$

($*$ means convolution)
Figure 1 contains the result from calculating the image derivatives in several scales on the famous image 'Lena'. The orientation is represented in *double angle representation* which is calculated as

$$\mathbf{z} = |\nabla \mathbf{I}| \mathbf{e}^{\mathbf{2i} \angle \nabla \mathbf{I}}$$

The orientation is represented by a complex number which has the interpretation:

- The phase is the double orientation angle. This means that the representation is unaffected by a rotation of the orientation by 180° (as it should because the orientation is then the same).
- The magnitude represents our confidence in the orientation.

(See [1] for a more detailed explanation).
The complex valued image in fig. 1 is visualized by colours, see fig. 2 for an explanation of their meaning.

All the gradient images in fig. 1 will be used in the matching procedure explained further on.

3 Local orientation histograms

Local histograms on the local orientation can be calculated in two steps:

1. Calculate one histogram for each pixel, $\mathbf{H}(\mathbf{x}$. Each bin in the histogram represents a certain local orientation (or rather an interval of the local orientation). We have used overlapping bins in order to get a more smooth histogram. This will hopefully make the histogram matching more robust. The overlapping function used in this project is 90° overlapping $\cos^2$-functions, see figure 3. Each orientation value will then fall into two neighbouring bins. The sum of the contributions by one orientation will always be 1, since $\cos^2(x) + \cos^2(x + \pi/2) = 1$.
2. Make local area histograms, $\mathbf{H}_{\sigma_H}(\mathbf{x})$ by summing up pixel histograms in a local region, i.e. $\mathbf{H}_{\sigma_H}(\mathbf{x}) = \sum_{\mathbf{x}} w(\mathbf{x}, \sigma_H)\mathbf{H}(\mathbf{x})$, where $w(\mathbf{x}, \sigma_H)$ is a Gaussian filter with standard deviation σ_H.

Figure 4 shows an example of an orientation histogram with 10 bins.

The histogram is a compact representation of the content in each local area, and it is also computationally efficient to calculate. The histogram can of course only describe the presence of an orientation and not the position, but hopefully this description is sufficient.

We now calculate local histograms in every orientation image in fig. 1. σ_H is chosen to be proportional to σ_∇. This means that we have a quick (and suboptimal) way to match image contents between local areas in different scales by simply comparing their histograms. The comparison can be made invariant to rotation and mirroring with a suitable choice of histogram matching algorithm. This is explained in the next section.

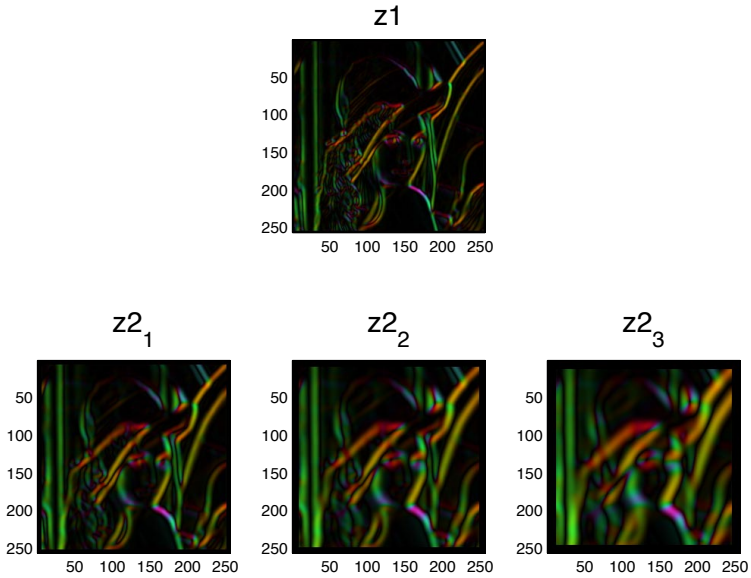


Figure 1: Image gradient in several scales. Top row: $\sigma_{\nabla} = 1$, Bottom row: $\sigma_{\nabla} = 2^1, 2^{1.5}, 2^2$

Figure 2: Left: Colour representation of complex numbers in figure 1, Right: Orientation interpretation of each colour (complex number).

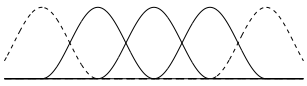


Figure 3: Overlapping bins used when computing the histograms.

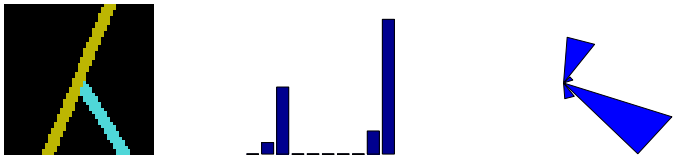


Figure 4: Example. Left: Local orientation image (same representation as in fig. 1). Middle: Orientation histogram. Right: Same orientation histogram drawn in a polar coordinate system (which may be easier to interpret).

4 Matching of histograms

The first step when finding a suitable affine mapping $\mathbf{x}_n = \mathbf{A}\mathbf{x} + \mathbf{b}$ is to compare the local orientation histograms of the destination blocks with the histograms of local image regions in three larger scales, and at all integer pixel offsets.

Finding the affine mapping is a two-stage process. First we find the most similar histogram regardless of rotation and mirroring, then we find the best rotation (modulo π, due to the *double angle* nature of the local orientation statements) for the histogram we found. This will give us four candidate mappings, which all have to be evaluated at the grey-scale level.

4.1 Finding the best histogram regardless of rotation

We can find the best matching histogram regardless of rotation and mirroring, by looking at the absolute values of the Fourier coefficients.

Since the grey-level mapping can handle scalings, we are only interested in the shape of the histograms, not in the scaling. For this reason we normalize the histograms before computing their Fourier transforms.

As a measure of how well two histograms match, we have used:

$$m_k = |\boldsymbol{W}_0||\boldsymbol{W}_k|^t \tag{2}$$

Here $\boldsymbol{W}_0$ is the Fourier transform of the target block histogram, and $\boldsymbol{W}_k$ is the Fourier transform of one of the possible source blocks.

To find the best matching histogram, we thus first generate a list of all histograms, then compute m_k for each one of them. Finally we select the histogram that corresponds to the highest m_k value.

One thing is important to note here. Taking the absolute values of the Fourier coefficients will give matches not only to histograms that differ by one global shift, but also

to histograms that have different shifts in each frequency. An important implication of this is that if two histograms have a certain m_k value, the same value will be obtained if one of them is mirrored.

4.2 Mirrored histograms

One problem when using orientation features to estimate the rotation of a block is that the orientation property is only defined modulo π. The reason for this is that the local orientation is represented with the double angle of the gradient direction. This means that an image, and the same image rotated π rad will have the same orientation histogram (see figure 5).

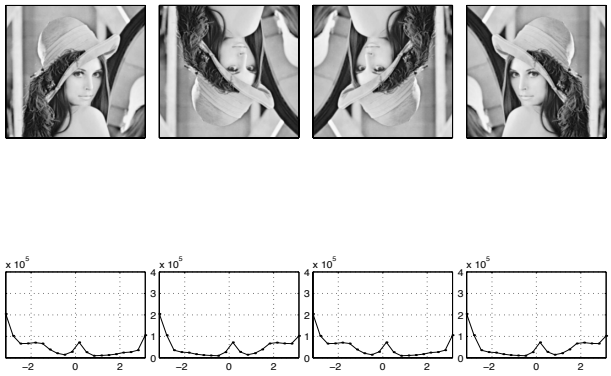


Figure 5: Four permutations of the "Lena" test image.

We can also see that if we mirror the image content with respect to the x, or y axis, the histogram will be mirrored as well.

As mentioned in section 4.1, the Fourier coefficients of these four candidates have the same absolute values. At the histogram level there is no easy way to tell which one of these four permutations is the best, so we will pass them all on to the grey-level matching stage.

4.3 Finding the best rotation of a histogram

The best way to rotate an orientation histogram in order to align it with another one can be found using the Fourier transforms of the two histograms. The method to be described produces results with a resolution considerably higher than what is indicated by the number of bins in the two histograms.

The method utilizes the Fourier coefficients in the positive half of the Fourier domain, ie. for an orientation histogram $\{h_1, h_2, \ldots, h_N\}$ we will use the coefficients

$$\boldsymbol{w}_k(h) = \sum_{n=1}^{N} h_n e^{-\boldsymbol{i} 2\pi k(n-1)/N} \quad \text{for } k = 1 \ldots \lfloor N/2 \rfloor \tag{3}$$

If we have two histograms $h_1, \ldots h_N$, and $g_1, \ldots g_N$, with similar shape, but different shifts, the phase difference of the Fourier coefficients can give an estimate of how much the histograms have been shifted. However, since only the first coefficient can give a full estimate (the others are given modulo π, modulo $\pi/2$ and so on), we will have to use it to move the others into place. The shift as seen by the first Fourier coefficients looks like this:

$$\boldsymbol{c}_1 = \boldsymbol{w}_1(h)\,\mathrm{conj}(\boldsymbol{w}_1(g)) \tag{4}$$

The next shift estimate can be moved into place (or *unwrapped*) using this estimate as follows:

$$\tilde{\boldsymbol{c}}_2 = \boldsymbol{w}_2(h)\,\mathrm{conj}(\boldsymbol{w}_2(g)) \tag{5}$$

$$\varphi_2 = \varphi_1 + \arg(\tilde{\boldsymbol{c}}_2 e^{\boldsymbol{i}2\varphi_1})/2 \tag{6}$$

$$\boldsymbol{c}_2 = |\tilde{\boldsymbol{c}}_2| e^{\boldsymbol{i}\varphi_2} \tag{7}$$

We keep the magnitude of the complex number, since it contains a measure of the two signal energies at the current frequency. Using the two shift estimates $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, we can construct an estimate that takes the energy at the two frequencies into account:

$$\theta_2 = \arg(\boldsymbol{c}_1 + \boldsymbol{c}_2) \tag{8}$$

For the next pair of Fourier coefficients it would thus be better to use this estimate to unwrap the phase. We will thus use the following algorithm:

$$\tilde{\boldsymbol{c}}_k = \boldsymbol{w}_k(h)\,\mathrm{conj}(\boldsymbol{w}_k(g)) \tag{9}$$

$$\varphi_k = \theta_{k-1} + \arg(\tilde{\boldsymbol{c}}_k e^{\boldsymbol{i}k\theta_{k-1}})/k \tag{10}$$

$$\boldsymbol{c}_k = |\tilde{\boldsymbol{c}}_k| e^{\boldsymbol{i}\varphi_k} \tag{11}$$

$$\theta_k = \arg\left(\sum_{l=1}^{k} \boldsymbol{c}_l\right) \tag{12}$$

The final value of θ_k will be our estimate of the rotation. Thus we now have four candidates for the $\mathbf{A}$ matrix in the affine transform:

- Rotation $\theta_k/2$
- Rotation $\theta_k/2 + \pi$
- Rotation $\theta_k/2$, and mirroring
- Rotation $\theta_k/2 + \pi$, and mirroring

The first two rows in figure 6 shows some examples of regions selected by the histogram matching algorithm. (The smaller of the two blocks in each image is the destination block.) The top row shows the block outlines superimposed on the original image, and the centre row shows them on the corresponding orientation images. As can be seen in the bottom row (explained further in section 5), these regions may later be mirrored, and even inverted in grey-level by the grey-level matching stage.

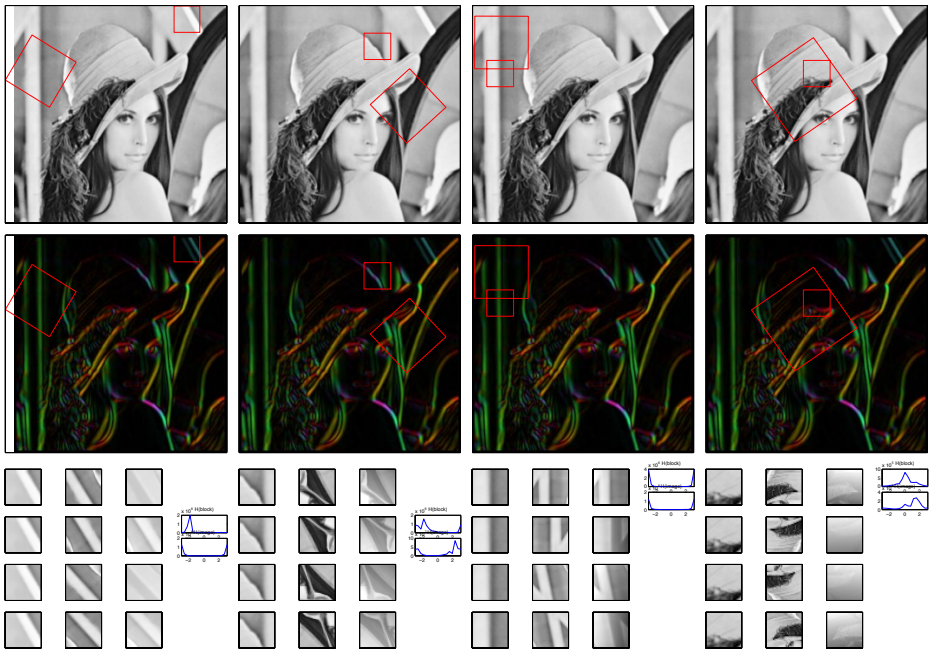


Figure 6: Example of chosen blocks.

5 Finding a grey-level mapping

To find out which of the four candidates is the best, we will now apply a grey-level mapping to the pixels in the source block. Recall the full mapping:

$$\begin{pmatrix} y_n \\ x_n \\ i_n \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ p_1 & p_2 & p_3 \end{pmatrix} \begin{pmatrix} y \\ x \\ i \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ p_4 \end{pmatrix} \tag{13}$$

We will now use the third of these equations to find the best grey-level mapping from the source block to the target block.

There are two ways to do this. Either we use all four intensity parameters $p_1 \ldots p_4$, or we only use only the DC-offset, and scaling parameters, p_3 and p_4.

In either variant, the parameters are estimated using a least squares fit. If the target blocks overlap, we will instead use a weighted least squares fit, where the pixels in the centre of the block are regarded as more important than those near the rim.

The weighted least squares fit uses a weighting function that is one in the middle of the block. At the sides, where the block overlaps it's neighbours, the function decreases as a $\cos^2(x)$ function (see figure 7). Near the corners the function will consequently decrease as $\cos^2(x)\cos^2(y)$.

The reason for using this weighting function is mainly that the decoder uses it to merge the overlapping blocks. This weighting function will turn the merging of neighbouring blocks into a simple weighted sum, due to the fact that $\cos^2(x)+\cos^2(x+\pi/2) = 1$, regardless of the value of x.

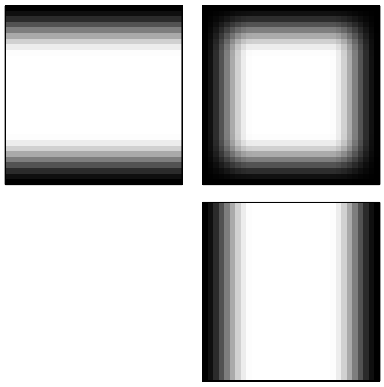


Figure 7: Weighting function.

Once we have found the (in the least square sense) best grey-level mapping, we can use it to compute a performance measure. The performance measure we will use is the sum of squares of the *residual*, ie. the difference between the intensities in the destination block, and the transformed source block. As in the weight computation, we will use a weighted sum of squares if the blocks overlap.

Of the four candidate blocks found by the histogram matching, we now select the one with the smallest residual. This selection is illustrated in the bottom row of figure 6. The first column for each image shows the target block contents, the second column shows the four variants of transformed source blocks, the third column shows the transformed source blocks, with the grey-level adjusted according to $(\ p_1 \quad p_2 \quad p_3 \quad p_4\ )$. The source and destination block histograms are plotted next to the chosen mapping.

6 Bit rates

The two alternatives in grey-level matching will give us two different bit-rates, but there are many factors that stay constant. For each block we have the following parameters:

- b_1 and b_2. For a 256×256 image these will correspond to $2 * 8 = 16$ bits.

- The rotation angle φ. This will have to have a fairly high accuracy, for instance as 8 bits.
- The block could either be mirrored, or not. This is 1 bit.
- We tested histograms in three scales, this corresponds to 2 bits (or really $\log 3/\log 2 \approx 1.585$ but we prefer being generous).
- each of the grey-level parameters $p_1, \ldots p_4$ will need 8 bits.

Thus we have the cases:

- A two-parameter grey-level mapping:
  $B_B = 2 * 8 + 8 + 1 + 2 + 2 * 8 = 43$ bits/block.
- A four-parameter grey-level mapping:
  $B_B = 2 * 8 + 8 + 1 + 2 + 4 * 8 = 59$ bits/block.

The final bit rate can be computed as:

$$\text{bitrate} = \frac{B_B N_B}{X_{im} Y_{im}} \tag{14}$$

Where (X_{im}, Y_{im}) is the image size, and N_B is the number of blocks. Assuming that the image size is a multiple of the block size, the bit-rate calculation instead becomes $B_B/(X_{bl}Y_{bl})$ where (X_{bl}, Y_{bl}) is the block size. This calculation has been done for a few block sizes in the table below:

Block size	two parameters	four parameters
8×8	0.67	0.92
16×16	0.17	0.23
32×32	0.04	0.06

Bit rates for some block sizes with no overlap between the blocks.

In general, the number of blocks can be computed from the following formula:

$$N_B = \left\lceil \frac{Y_{im} - Y_{bl}(1-\alpha)}{\alpha Y_{bl}} \right\rceil \left\lceil \frac{X_{im} - X_{bl}(1-\alpha)}{\alpha X_{bl}} \right\rceil \tag{15}$$

Where α is the fraction of a block that is overlapped by its neighbour.

If we add some overlap, the bit rates in the table above will go up, but they will always stay below the value in the cell above the current. Also note that any bit rates should naturally be judged in combination with some kind of error measure.

7 Decoding

The bit stream output from the encoder contains some representation of the three matrices:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad \mathbf{p} = \begin{pmatrix} p_1 & p_2 & p_3 & p_4 \end{pmatrix}$$

for each block in the image. The matrices $\mathbf{A}$ and $\mathbf{b}$ define a mapping from the target block to the source block

$$\begin{pmatrix} y_s \\ x_s \end{pmatrix} = \mathbf{A} \begin{pmatrix} y_d \\ x_d \end{pmatrix} + \mathbf{b} \tag{16}$$

and the $\mathbf{p}$ vector defines an intensity mapping from the source block to the destination block. In the two-parameter case this is

$$i_d = p_3 i_s + p_4 \tag{17}$$

and in the four parameter case:

$$i_d = \begin{pmatrix} y_s & x_s & i_s & 1 \end{pmatrix} \mathbf{p}^t \tag{18}$$

7.1 Iterations

The decoding starts off with an image with random content. For each destination block we then apply the two mappings in the following way:

- First the backward mapping (equation 16) is applied to each pair of coordinates in the destination block. This gives us the location where we should fetch our intensity value. This will in general be a non-integer image position, so we apply bi-cubic interpolation in order to find the correct intensity value i_s.
- For each of the resultant intensity values we then apply the intensity mapping (equation 17 or 18).

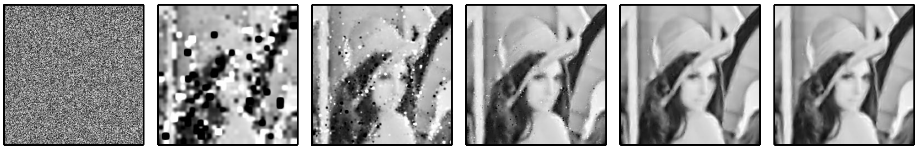


Figure 8: Illustration of the decoding process.

This process is repeated until the image content is stable, usually after 5 or 6 iterations (see figure 8).

7.2 Decoding performance

If we compare the time it takes our decoder to decode an image with that of a conventional fractal image coder, ours is clearly much slower. This is mainly due to the bicubic interpolation step. In MATLAB our implementation requires more than a minute to unpack a 256×256 image coded with 32×32 blocks. However, an efficient implementation should be possible if we could somehow make use of vectorized machine code instructions such as MMX2, or 3DNOW.

The use of large blocks seems to reduce the number of iterations required in the decoding. Especially the ratio between the sizes of the source and the target blocks seem to matter.

The $\cos^2(x)$ windowing seems to give us smooth and pleasant looking results. However, it might also introduce unnecessary blurring of the decoded image.

8 Experiments

In this section we will evaluate the presented image coding algorithm. All bit rates are computed according to equation 14. Most of the tests have been performed on the 256×256 "Lena" image. As a comparison we have also coded the 401×401 fractal image "Zapato".

8.1 Varied overlap

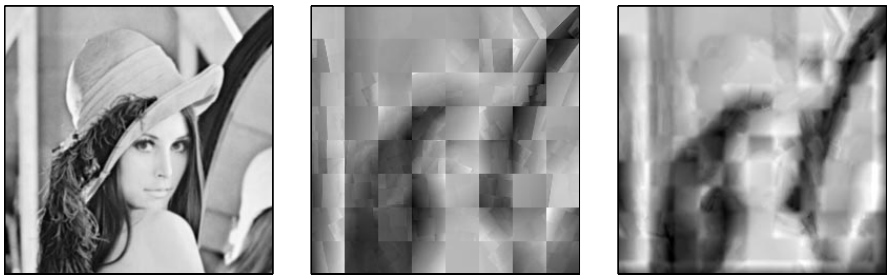
The first experiment involves varied degree of overlap. The overlap is gradually increased from no overlap, until two neighbouring blocks have exactly half their pixels in common (see figure 9). As we can see, all overlaps except the last one have visible artefacts stemming from the destination block positions.

8.2 Two and four grey-level parameters

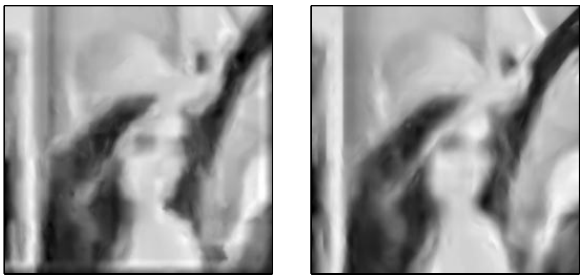
The next experiment involves smaller block sizes, and the two-parameter grey-level mapping. The result is shown in figure 10.

8.3 Fractal coding of fractals

Out of curiosity we also tried fractal coding on a fractal image. Considering that fractals actually are built up from smaller parts of themselves (which is the assumption made in fractal image coding), this should work at least as good as fractal coding on an ordinary image.



Block size	32×32	32×32
Overlap	0	8
p-params	4	4
Bit rate	0.0576	0.1089



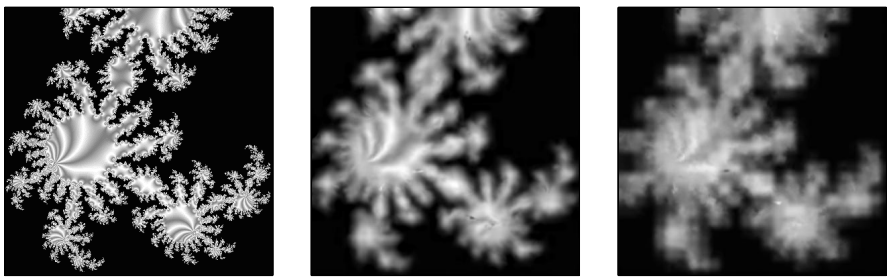
32×32	32×32
12	16
4	4
0.1521	0.2026

Figure 9: Coding of the “Lena” image with varying degree of overlap.

Block size	32×32	32×32
Overlap	16	16
p-params	4	2
Bit rate	0.2026	0.1476

Block size	16×16	16×16
Overlap	8	8
p-params	4	2
Bit rate	0.8652	0.6305

Figure 10: Varying block size, and grey-level mapping.



Block size	32×32	32×32
Overlap	16	16
p-params	4	2
Bit rate	0.2293	0.1671

Figure 11: Fractal coding of a fractal image.

8.4 Comparison with other coders

Previous approaches to fractal coding, use only a subset of the full mapping:

$$\begin{pmatrix} y_n \\ x_n \\ i_n \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ p_1 & p_2 & p_3 \end{pmatrix} \begin{pmatrix} y \\ x \\ i \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ p_4 \end{pmatrix} \tag{19}$$

The first reported implementations were made by Jacquin in 1989 and 1990. They used the following restrictions on $\mathbf{A}$:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} \pm 0.5 & 0 \\ 0 & \pm 0.5 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 0 & \pm 0.5 \\ \pm 0.5 & 0 \end{pmatrix}$$

and did not use the parameters p_1 and p_2. Jacquin's approach was extended in 1991 by Øien to use the grey-level mapping parameters p_1 and p_2 [2]. The reduced degree of freedom in block matching reduces the amount of bits per block.

Method	Bits per block, B_B
Jacquin	$3 + 2 * 8 + 2 * 8 = 35$
Øien	$3 + 2 * 8 + 4 * 8 = 51$

A comparison between the Øien coder and ours is made in table 1. The bitrates and the PSNR values in this table are plotted against each other in figure 12.

Coder	Block size	Overlap	No of p-pars	bitrate	PSNR
Our	32×32	16	4	0.2026	20.5085
Our	32×32	32	4	0.0576	18.0918
Our	32×32	24	4	0.1089	18.5946
Our	32×32	20	4	0.1521	19.0208
Our	16×16	8	4	0.8652	24.9872
Our	32×32	16	2	0.1476	17.7871
Our	16×16	8	2	0.6305	22.5896
Øien	32×32	16	4	0.1751	20.7532
Øien	16×16	8	4	0.7478	25.2421

Table 1: Table of various bitrates and PSNR values.

The PSNR values in the table have been computed using the following formula:

$$\text{PSNR} = 10 \log_{10} \frac{|x_{pp}|^2}{\sum_{ij}(x_{ij} - \hat{x}_{ij})^2} \quad \text{dB} \tag{20}$$

As can be seen in figure 12, the Øien coder outperforms our coder at present. However, there exists several possibilities to improve our coder as is mentioned in section 9.

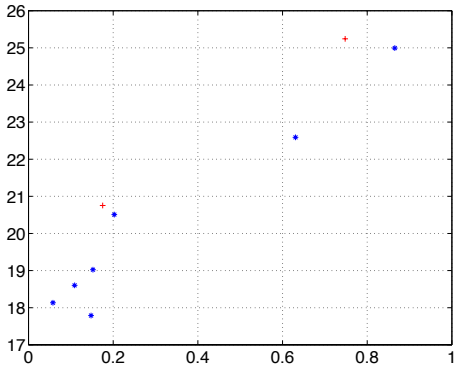


Figure 12: PSNR vs. bitrate

Plus-signs show the performance of the Øien coder.
Asterisks are our results.

One advantage with our approach is the encoding time. Even though our coder searches three scales, and arbitrary rotations of the image, the search is significantly faster. So, even if we make the search more detailed we would still have a coder that is at least as convenient to use as the Øien coder.

9 Conclusions

The idea to find similar local areas in the image by matching their feature histograms seems to work fine in some areas and worse in other areas, especially ones that contain complex patterns. There are several things that can be improved in the algorithm, some of them are:

- In the experiments above we only used the best histogram match for further processing. We could instead use for instance the 5 best matches, calculate $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{p}$ for each match and then decide which one is the best.
- Image gradients are probably not enough in areas with complex patterns. For instance the fractal image in fig. 11 containts a lot of orientations in every local area and the orientation histograms do not contain enough information about the area. To improve the algorithm we could make histograms from other features. One example is to use corners as features. The histogram can then be based on corner orientation or corner angle. Another example could be to use texture features.

The idea to match feature histograms to find similar areas should also be possible to use in video coding. We could then try to predict areas in one frame from areas in the previous one. The algorithm will be very similar to the one used in this report. Another idea could be to make local feature histograms in volumes instead of frames.

10 Acknowledgment

The authors would like to thank Robert Forchheimer for giving the graduate course and commenting on the report.

References

[1] G. H. Granlund and H. Knutsson. *Signal Processing for Computer Vision*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9530-1.

[2] L. Torres and M. Kunt, editors. *Video Coding: The Second Generation Approach*. Kluwer Academic Publishers, 1996.