Scattered Data Coding in Digital Image Compression

Laurent Demaret and Armin Iske

Abstract. This paper concerns digital image compression using adaptive thinning algorithms. Adaptive thinning is a recursive point removal scheme which works with decremental Delaunay triangulations. When applied to digital images, adaptive thinning returns a scattered set of most significant pixels. This requires efficient and customized methods for coding these pixels (but not their connectivities). To this end, we propose a hierarchical coding scheme, which works with recursive subdivisions of octtree cells, and which takes advantage of the statistical data correlation. The good performance of the resulting compression scheme is shown in comparison with the well-established wavelet-based method SPIHT.

§1. Introduction

Digital image compression has recently gained enormous popularity in a variety of applications. This is especially due to the rapid development of multimedia technologies. Digital image compression is concerned with the conversion of digital image data into a *bitstream*, which is e.g. transmitted across the internet. In this case, for the sake of transmission speed, the length of this bitstream message is required to be as short as possible, while maintaining a reasonable quality of the image. In this process, the following five tasks are to be performed, one after the other.

(1) Data reduction;
(2) Encoding of the reduced data at the sender;
(3) Transmission of the encoded data from the sender to the receiver;
(4) Decoding of the transmitted data at the receiver;
(5) Data reconstruction.

Curve and Surface Fitting: Saint-Malo 2002
XXX (eds.), pp. 1–11.

ISBN XXX.

Many of the well-established methods in digital image compression, including JPEG2000 [8] and SPIHT [7], are based on wavelets and related techniques. For a comprehensive introduction to image compression by using wavelets, we refer to the survey [3]. Tensor product wavelets provide, in combination with well-adapted compression methods, high compression rates while maintaining most of the visual features of the image. At low bit rates, however, such methods typically fail to capture certain characteristic features of the image, such as sharp edges. This is mainly due to high oscillations around discontinuities which leads to undesired visual artifacts. Therefore, current research is focussing on the modelling of sharp edges. Recent work on this includes nonlinear decomposition techniques [1,2,5].

This paper proposes an alternative concept for *lossy* compression of digital image data, by using adaptive thinning methods in combination with scattered data coding. Adaptive thinning [4], when applied on a digital image, leads to a reduction of the original data, step **(1)**. This is accomplished by recursively deleting pixels of the image, so that adaptive thinning returns a *scattered* subset of *most significant* pixels. These most significant pixels are then used in step **(5)** for the reconstruction of the given image. We remark that adaptive thinning provides a class of *data-dependent* filtering operators. In contrast to data-independent filtering methods, the data-dependent approach taken in this paper copes very well with the visual perception of the image. This is supported by the numerical results in Section 5. Further details concerning the application of adaptive thinning on digital images, especially the relevant background for the steps **(1)** and **(5)**, are briefly explained in the following Section 2.

The performance of the intermediate steps **(2)**-**(4)** requires an efficient scheme for scattered data coding. This is in order to *encode* and *decode* the most significant pixels. To this end, we have designed a customized scattered data coding scheme, which serves to convert the most significant pixels into a bitstream message, to be transmitted in step **(3)**. This coding scheme is subject of the discussion in Section 3.

We remark that the resulting compression algorithm, proposed in this paper, incorporates a flexible representation of the *gridded* image data by using a *scattered* set of most significant pixels. This, in combination with the customized scattered data coding scheme, allows us to capture sharp edges and related image features reasonably well, at small coding costs and at small computational costs. The latter is supported by the analysis concerning the complexity of the compression and decompression, which is done in Section 4. Finally, the good performance of the proposed compression scheme is confirmed by numerical examples in Section 5, where we compare our method with the well-established wavelet-based image compression algorithm SPIHT.

§2. Adaptive Thinning on Digital Images

Thinning algorithms are recursive point removal schemes for scattered data. The purpose of *adaptive thinning*, as recently suggested in [4], is to approximate a bivariate function from scattered data samples by a piecewise linear function over the Delaunay triangulation of a subset of *most significant* sample points. This is accomplished by working with decremental Delaunay triangulations. At each removal step of the adaptive thinning algorithm, one vertex of the current Delaunay triangulation is deleted. This is done by using one specific (adaptive) criterion for the vertex removal (see [4] for three different such removal criteria).

This section explains how we use adaptive thinning for the data reduction step **(1)** and the subsequent reconstruction in step **(5)**. We remark at this point that the focus of this paper is more on the scattered data coding scheme, to be discussed in Section 3, rather than on adaptive thinning, which is subject of the previous paper [4]. Therefore, we refrain from expanding lengthy details on possible removal criteria for adaptive thinning.

Be it sufficient for the purpose of this paper to say that adaptive thinning is a *greedy* data-dependent removal scheme, which relies on a specific *local* error indicator. This error indicator measures, for any current vertex x, a local approximation error that would result from the removal of x, termed the *anticipated error* of x. Now, at any step of the adaptive thinning algorithm, a vertex x^*, which minimizes the anticipated error, is removed from the current Delaunay triangulation. In this sense, x^* is in the current situation considered as a *least significant* vertex point, and so adaptive thinning generates subsets of most significant points.

In view of the application of adaptive thinning on digital images, let us be more specific about their representation, and fix some notations. A digital image is a rectangular grid of *pixels*. Each pixel bears a color value or greyscale *luminance*. For the sake of simplicity, we restrict ourselves in the following discussion to greyscale images. In this case, the range of the integer luminance values has usually the form $[0..2^r - 1]$. Typically, one works with 256 different greyscale values, in which case $r = 8$, i.e., the range is $[0..255]$.

The image can be viewed as a matrix $Z = (z(i,j))_{i,j}$, whose entries $z(i,j) \in [0..2^r - 1]$ are the luminance values at the pixels. Each pixel position (i,j) is a pair of non-negative integers i and j. For the range of the pixel positions (i,j), we assume the form $[0..2^p - 1] \times [0..2^p - 1]$, for some positive integer p. Hence, the dimension of the square *image matrix* Z is 2^p-by-2^p.

Adaptive thinning, when applied on digital images, recursively removes pixels from a given digital image. Adaptive thinning returns after n removals, $n \in [1..2^{2p}]$, an index set $\mathcal{I} \subset [0..2^p - 1] \times [0..2^p - 1]$

of size $|\mathcal{I}| = 2^{2p} - n$, corresponding to the *most significant* pixel positions. In addition, for every pixel position $(i, j) \in \mathcal{I}$, an approximation $\tilde{z}(i, j) \in [0..2^r - 1]$ of the luminance value $z(i, j)$ is returned, so that $\tilde{z}(i, j) \approx z(i, j)$ for every $(i, j) \in \mathcal{I}$.

In order to explain the reconstruction of the image in step **(5)**, let $\mathcal{D}_\mathcal{I}$ denote the Delaunay triangulation over the planar set of most significant pixel positions, given by $\mathcal{I}$. Moreover, let $L(Z, \mathcal{D}_\mathcal{I})$ denote the piecewise linear function over $\mathcal{D}_\mathcal{I}$ satisfying $L(Z, \mathcal{D}_\mathcal{I})(i, j) = \tilde{z}(i, j)$ for all $(i, j) \in \mathcal{I}$. This piecewise linear function is used at the receiver in step **(5)** in order to reconstruct the entire image Z. To this end, an approximation $\hat{Z} = (\hat{z}(i, j))_{ij}$ of the image matrix Z is computed, where we let

$$\hat{z}(i, j) = [L(Z, \mathcal{D}_\mathcal{I})(i, j)] \approx z(i, j)$$

for every pixel position $(i, j) \in [0..2^p - 1] \times [0..2^p - 1]$, and where by $[x]$ we denote the nearest integer to $x \in \mathbb{R}$.

§3. Scattered Data Coding

In this section, we explain how the construction of the bitstream in step **(2)** is done. This is concerning the coding of the most significant pixels. The bitstream will contain the pixel positions in $\mathcal{I}$ and corresponding *quantized symbols* for the luminance values. Since we work with Delaunay triangulations, no connectivity coding is done. This helps us to keep the bitstream short. Before we explain details on our coding scheme, let us make some remarks.

First of all, note that the *uncompressed* code for these data would consist of a header containing the dimension of the image matrix Z, followed by the corresponding pixel positions $(i, j) \in \mathcal{I}$ and luminance values $\tilde{z}(i, j) \in [0..2^r - 1]$. Thus, when sending uncompressed data, the total coding costs (without the costs for the header) are $(2p + r) \times |\mathcal{I}|$ bits, where $|\mathcal{I}|$ is the number of indices in $\mathcal{I}$. This naive way of coding is too costly. In order to reduce the coding costs, we take advantage of the statistical data distribution.

We remark that classical wavelet methods usually consider the complete set of coefficients and decide, according to some suitable threshold, whether a value is significant or not. Then, sophisticated techniques take advantage of the dependencies between the locations and the magnitudes of significant coefficients. This is done by clustering the non-significant coefficients (*zerotrees* [7]), or by using *context-based arithmetic coding* [8].

Our coding strategy exhibits some similarities to this. Indeed, the selected pixel positions in $\mathcal{I}$ are classified as most significant, whereas the remaining pixels are regarded as non-significant. On the other hand, the values $\tilde{z}(i, j)$ of the most significant pixels are not proportional to

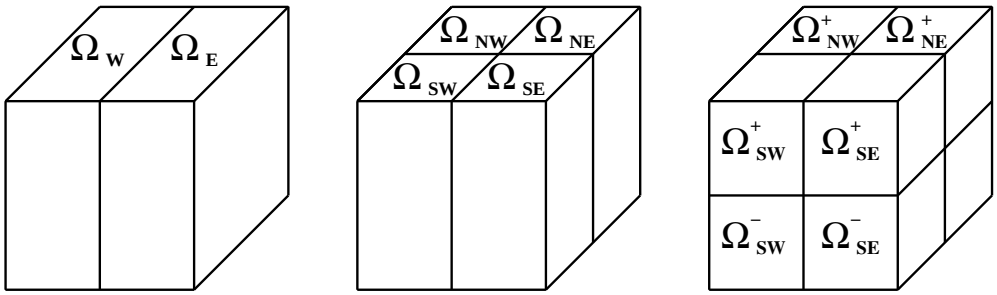

Fig. 1. Splitting of the cell Ω into eight subcells in three stages.

their significances. This requires taking into account the local correlations between the most significant pixels $(i, j, \tilde{z}(i, j))$, $(i, j) \in \mathcal{I}$.

Moreover, we apply a uniform *quantization* on the luminance values $\tilde{z}(i, j)$, yielding quantized symbols $Q(\tilde{z}(i, j))$ for all $(i, j) \in \mathcal{I}$, where the quantization step depends on a specific target rate. This reduces the range of the luminance values, from previously $[0..2^r - 1]$ to $[0..2^s - 1]$ for the quantized symbols, where $s < r$. Now the pixels (positions and quantized symbols) can be viewed as a set of tridimensional points $(i, j, k) \in \Omega$, where we let

$$\Omega = [0..2^p - 1] \times [0..2^p - 1] \times [0..2^s - 1]$$

denote the *bounding cell* of the data, containing $m = |\mathcal{I}|$ pixels. For any such data point $(i, j, k) \in \Omega$, we have $k = Q(\tilde{z}(i, j))$ and $(i, j) \in \mathcal{I}$. This data representation can also be viewed as a *binary* tridimensional matrix $M = (m_{ijk})_{i,j,k}$, of dimension 2^p-by-2^p-by-2^s, whose entries are given by

$$m_{ijk} = \begin{cases} 1, & \text{if } (i, j) \in \mathcal{I} \text{ and } k = Q(\tilde{z}(i, j)), \\ 0, & \text{otherwise.} \end{cases}$$

In what follows, we propose an efficient coding scheme for the *sparse* matrix M. Note that for coding M, it is sufficient to localize the nonzero entries of M, the *fill-ins* of M. To this end, we work with a hierarchical subdivision of cells. Initially, the bounding cell Ω is split into eight subcells. When splitting Ω, the number of pixels in the resulting subcells are progressively coded. Initially, we code $m = |\Omega|$, the total number of pixels in the bounding cell Ω. Since these are at most $2^p \times 2^p$ pixels, the coding of m requires $2p$ bits, yielding the first $2p$ bits in the bitstream. Then, the splitting of Ω is done in three stages as follows, see Figure 1.

In the first stage, Ω is split into the two subcells

$$\Omega_W = [0..2^{p-1} - 1] \times [0..2^p - 1] \times [0..2^s - 1],$$
$$\Omega_E = [2^{p-1}..2^p - 1] \times [0..2^p - 1] \times [0..2^s - 1]$$

of equal size across the *i-axis* (Figure 1, left). The number $m_W = |\Omega_W|$ of pixels contained in the cell Ω_W is coded. These are at most $m = |\Omega|$ pixels. Therefore, for coding the number m_W we need $\lceil \log_2(m+1) \rceil$ bits. Since the number $m_E = |\Omega_E|$ of pixels in Ω_E is given by $m_E = m - m_W$, we do not code m_E.

In the second stage, each of the two subcells Ω_W and Ω_E is split across the opposite *j-axis* (see Figure 1, middle), yielding the four subcells

$$\begin{aligned}
\Omega_{SW} &= [0..2^{p-1}-1] \times [0..2^{p-1}-1] \times [0..2^{s}-1], \\
\Omega_{NW} &= [0..2^{p-1}-1] \times [2^{p-1}..2^{p}-1] \times [0..2^{s}-1], \\
\Omega_{SE} &= [2^{p-1}..2^{p}-1] \times [0..2^{p-1}-1] \times [0..2^{s}-1], \\
\Omega_{NE} &= [2^{p-1}..2^{p}-1] \times [2^{p-1}..2^{p}-1] \times [0..2^{s}-1].
\end{aligned}$$

The two numbers $m_{SW} = |\Omega_{SW}|$ and $m_{SE} = |\Omega_{SE}|$ are coded one after the other. Since the cell Ω_{SW} contains at most $m_W = |\Omega_W|$ pixels, we need $\lceil \log_2(m_W+1) \rceil$ bits for coding m_{SW}. Likewise, coding the number m_{SE} requires $\lceil \log_2(m_E+1) \rceil$ bits. The two numbers m_{NW} and m_{NE} do not need to be coded. Indeed, this is because $m_{NW} = m_W - m_{SW}$ and $m_{NE} = m_E - m_{SE}$, i.e., the numbers m_{NW} and m_{NE} follow from the previous information in the bitstream.

In the third stage, each of the four subcells $\Omega_{SW}, \Omega_{NW}, \Omega_{SE}$, and Ω_{NE} is split across the *k-axis* (see Figure 1, right), each into two halves of equal size, which yields the eight subcells

$$\begin{aligned}
\Omega^-_{SW} &= [0..2^{p-1}-1] \times [0..2^{p-1}-1] \times [0..2^{s-1}-1], \\
\Omega^-_{NW} &= [0..2^{p-1}-1] \times [2^{p-1}..2^{p}-1] \times [0..2^{s-1}-1], \\
\Omega^-_{SE} &= [2^{p-1}..2^{p}-1] \times [0..2^{p-1}-1] \times [0..2^{s-1}-1], \\
\Omega^-_{NE} &= [2^{p-1}..2^{p}-1] \times [2^{p-1}..2^{p}-1] \times [0..2^{s-1}-1], \\
\Omega^+_{SW} &= [0..2^{p-1}-1] \times [0..2^{p-1}-1] \times [2^{s-1}..2^{s}-1], \\
\Omega^+_{NW} &= [0..2^{p-1}-1] \times [2^{p-1}..2^{p}-1] \times [2^{s-1}..2^{s}-1], \\
\Omega^+_{SE} &= [2^{p-1}..2^{p}-1] \times [0..2^{p-1}-1] \times [2^{s-1}..2^{s}-1], \\
\Omega^+_{NE} &= [2^{p-1}..2^{p}-1] \times [2^{p-1}..2^{p}-1] \times [2^{s-1}..2^{s}-1],
\end{aligned}$$

whose union is Ω. The four numbers $m^-_{SW} = |\Omega^-_{SW}|$, $m^-_{NW} = |\Omega^-_{NW}|$, $m^-_{SE} = |\Omega^-_{SE}|$, and $m^-_{NE} = |\Omega^-_{NE}|$ are coded.

Altogether, by splitting the bounding cell Ω into eight subcells, the sequence

$$m_W \,|\, m_{SW} \,|\, m_{SE} \,|\, m^-_{SW} \,|\, m^-_{NW} \,|\, m^-_{SE} \,|\, m^-_{NE}$$

of seven numbers is coded. This requires

$$\begin{aligned}
&\lceil \log_2(m+1) \rceil + \lceil \log_2(m_W+1) \rceil + \lceil \log_2(m_E+1) \rceil \\
&+ \lceil \log_2(m_{SW}+1) \rceil + \lceil \log_2(m_{NW}+1) \rceil \\
&+ \lceil \log_2(m_{SE}+1) \rceil + \lceil \log_2(m_{NE}+1) \rceil
\end{aligned}$$

bits in total, to be appended to the bitstream.

This splitting (including the updates in the bitstream) is then recursively applied to those subcells which are not empty. A cell $\omega \subset \Omega$ is said to be *empty*, iff it contains no pixel, and thus $|\omega| = 0$. On the most elementary recursion level, we encounter cells of the form $\omega = [2i, 2i+1] \times [2j, 2j+1] \times [k]$, provided that $s < p$. Such *atomic* cells are not split. This is in order to save additional costs in terms of bits. Instead of this, the coding of the pixels in atomic cells is accomplished as follows.

We merely discuss the coding of the atomic cell $\omega = [0,1] \times [0,1] \times [0]$, all the other atomic cases are treated in an analogous manner. Note that the atom $\omega = [0,1] \times [0,1] \times [0]$ may contain zero, one, two, three or four pixels. In case ω contains either four or zero pixels, no additional information (in terms of coding costs) is required. If ω contains exactly one pixel, then its position $(i,j) \in \{0,1\} \times \{0,1\}$ is coded by using two bits, one for the index i and one for the index j. Likewise, if ω contains three pixels, then the other position in ω (which contains no pixel) is coded by using two bits. In the remaining case, where ω contains exactly two pixels, there are six different possibilities for the distribution of the two pixels in ω. These six different cases are coded according to the *Huffman code* $(1,1), (1,0), (0,0,1), (0,0,0), (0,1,1), (0,1,0)$.

This requires only $2/6 \times 2$ bits $+ 4/6 \times 3$ bits $= 8/3$ bits in average, provided that the probabilities for each of the six cases are equal. This is cheaper than coding each of the two pixels separately, which would require 2×2 bits $= 4$ bits.

§4. Computational Complexity

In this section, we analyze the computational complexity of the proposed image compression scheme. To this end, we determine the computational costs required for the performance of the steps **(1)**,**(2)**,**(4)**, and **(5)** in the introduction. Recall that step **(1)** is done by using adaptive thinning. But the complexity of the adaptive thinning algorithm is already well-understood. According to [4], we require only at most $\mathcal{O}(N \log(N))$ operations for the removal of n pixels from a number of $N = 2^p \times 2^p$ pixels.

Now let us turn to the complexity of step **(5)**. In this step, the Delaunay triangulation $\mathcal{D}_\mathcal{I}$ is first constructed from the $m = N - n$ most significant pixel positions, before the corresponding piecewise linear function $L(Z, \mathcal{D}_\mathcal{I})$ is used in order to compute the n luminance values at the deleted pixel positions. Recall that building the Delaunay triangulation $\mathcal{D}_\mathcal{I}$ costs $\mathcal{O}(m \log(m))$ operations [6]. The subsequent reconstruction of the n luminance values costs $\mathcal{O}(n)$ operations, which is $\mathcal{O}(N)$ for large n.

As to the remaining two steps, **(2)** and **(4)**, note that these are symmetric. In fact, the asymptotic complexity of the encoding in step **(2)** is the same as the asymptotic complexity of the decoding in step **(4)**.

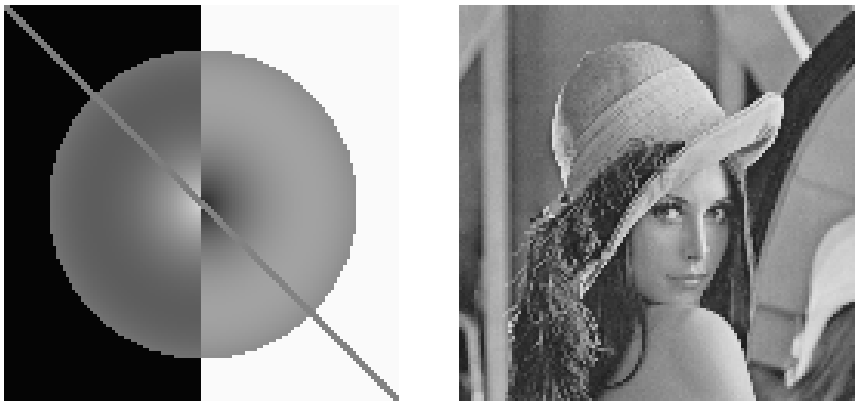

Fig. 2. The two test cases, `Reflex` (left) and `Lena` (right).

Therefore, we restrict ourselves to the analysis of the computational costs required for the performance of the encoding. To this end, recall from Section 3 that step **(2)** relies on the recursive splitting of the bounding cell Ω, containing $m = |\Omega|$ pixels. Therefore, we need to determine the computational costs required for the construction of the entire octtree-like data structure. Now note that the initial splitting of Ω into the two subcells Ω_W and Ω_E costs m operations in the first stage. Indeed, these m operations are required for counting the number m_W of pixels in Ω_W. But the subsequent splitting of Ω_W and Ω_E in the second stage costs also m operations, namely m_W for splitting Ω_W and m_E for splitting Ω_E, and so altogether $m_W + m_E = m$. By recursion, the splitting at each level ℓ costs exactly m operations. Now since the tree comprises $2p + s = \log_2(N) + s$ levels, this leads to $m \times (\log_2(N) + s) = \mathcal{O}(m \log(N))$ operations for the performance of step **(2)**.

Altogether, this shows that we require asymptotically at most

$$\mathcal{O}(N \log(N)) + \mathcal{O}(m \log(N)) = \mathcal{O}(N \log(N))$$

operations for the compression, in steps **(1)** and **(2)**, and at most

$$\mathcal{O}(m \log(N)) + \mathcal{O}(m \log(m)) + \mathcal{O}(N) = \mathcal{O}(m \log(N)) + \mathcal{O}(N)$$

operations for the decompression in steps **(4)** and **(5)**.

§5. Numerical Examples

We have applied our compression scheme, according to the steps **(1)**-**(5)** in the introduction, on two different test cases. One test case is a geometric image called `Reflex`, which is displayed in Figure 2 (left), the other one is the well-known test case `Lena`, shown in Figure 2 (right). In both cases, the image size is 128-by-128 (i.e., $p = 7$), and the greyscale values of the luminances $z(i, j)$ are in [0..255], i.e., $r = 8$.

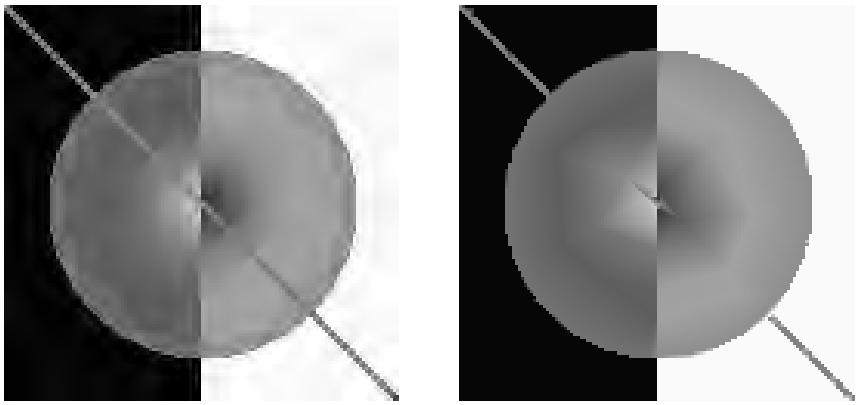

Fig. 3. `Reflex`: Reconstruction by SPIHT (left) and our method (right).

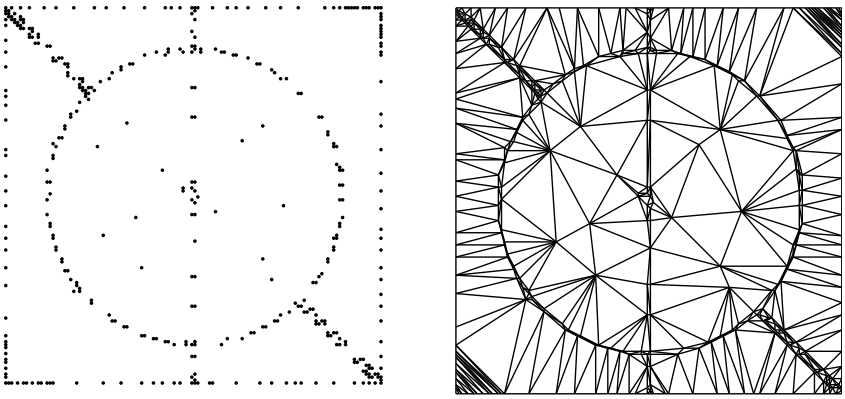

Fig. 4. `Reflex`: 407 most significant pixels and their Delaunay triangulation.

For the data reduction in step **(1)**, we have used one variant of the adaptive thinning algorithm **AT1** of [4]. The subsequent encoding of the bitstream in step **(2)** is done by using the coding scheme of the previous Section 3, with quantization step 8, so that we have [0..31] for the range of the quantized symbols $Q(\tilde{z}(i,j))$, $(i,j) \in \mathcal{I}$, i.e., $s = 5$.

In this section, we compare the performance of our compression scheme with that of the wavelet-based compression scheme *Set Partitioning Into Hierarchical Trees* (SPIHT) [7]. We remark that the good compression rate of SPIHT is, for small images such as `Reflex` and `Lena`, and at low bit rates, comparable with that of the powerful method *EBCOT* [8], which is the basis algorithm of the standard *JPEG2000*.

We measure the compression rate in *bits per pixel*, `bpp`. Moreover, the quality of the reconstruction $\hat{Z} = (\hat{z}(i,j))_{ij}$ is evaluated by its *Peak Signal to Noise Ratio* (PSNR), measured in `dB`,

$$PSNR = 10 \times \log_{10} \left(\frac{2^r \times 2^r}{MSE} \right),$$

Fig. 5. `Lena`: Reconstruction by SPIHT (left) and our method (right).

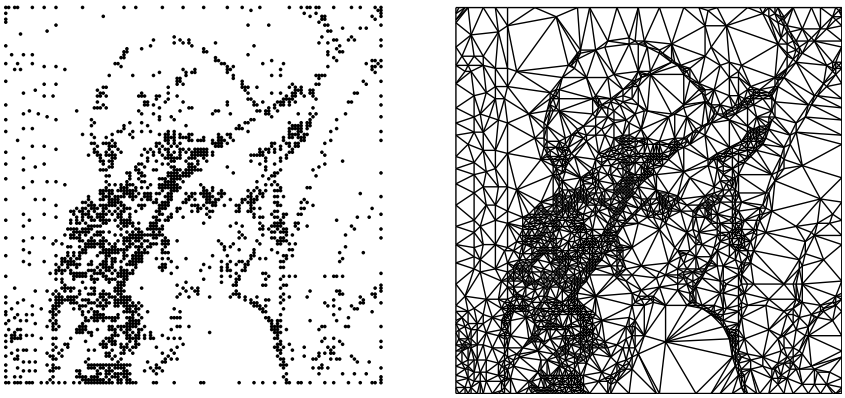

Fig. 6. `Lena`: 2205 most significant pixels and their Delaunay triangulation.

where

$$MSE = \frac{1}{2^{2p}} \sum_{i,j} |z(i,j) - \hat{z}(i,j)|^2$$

denotes the *Mean Square Error*.

In each of the two test cases, we consider using a fixed compression rate. Therefore, we compare our method with SPIHT by regarding the differences of their PSNR values.

In the test case of `Reflex`, we fix the compression rate by 0.28 `bpp`. The resulting reconstruction of our method is, in comparison with that of SPIHT, displayed in Figure 3. Our method yields the PSNR value 32.54 `dB`, whereas SPIHT provides the PSNR value 31.33 `dB`. Hence, with respect to this quality measure, our method is better. Note that our method manages to localize the sharp edges in this image. Moreover, undesired oscillations around the edges are avoided, see Figure 3. This is due to the well-adapted distribution of the 407 most significant points, which are, along with their Delaunay triangulation, displayed in Figure 4.

Now let us turn to the test case `Lena`, where we fixed the compression rate by 1.24 `bpp`. In this case, we obtain the PSNR value 31.45 `dB` for our compression scheme, whose reconstruction is displayed in Figure 5 (right). This reconstruction is obtained by using the 2205 most significant points, which are, along with their Delaunay triangulation, displayed in Figure 6. Note that by the distribution of these 2205 most significant points, the main features of the image, such as sharp edges and silhouettes, are captured very well. On the other hand, our reconstruction fails to render some of the textures in the image.

The resulting reconstruction of SPIHT is also shown in Figure 5 (left). In comparison, the method SPIHT yields the better PSNR value of 33.26 `dB`. Given the *visual* quality of the two reconstructions in Figure 5, however, we believe that our compression method is quite competitive.

References

1. Cohen, A., W. Dahmen, I. Daubechies, and R. DeVore, Tree approximation and optimal encoding, Appl. Comput. Harmonic Anal. **11** (2001), 192–226.
2. Cohen, A., and B. Matei, Nonlinear subdivision schemes: applications to image processing, in *Tutorials on Multiresolution in Geometric Modelling*, A. Iske, E. Quak, and M. S. Floater (eds.), Springer-Verlag, Heidelberg, 2002, 93–97.
3. Davis, G. M., and A. Nosratinia, Wavelet-based image coding: an overview, in *Applied and Computational Control, Signals, and Circuits*, B. N. Datta (ed.), Birkhauser, Boston, 1999, 205–269.
4. Dyn, N., M. S. Floater, and A. Iske, Adaptive thinning for bivariate scattered data, J. Comput. Appl. Math. **145** (2002), 505–517.
5. Le Pennec, E. and S. Mallat, Image compression with geometrical wavelets, Proceedings of ICIP 2000, September 2000, Vancouver.
6. Preparata, F. P. and M. I. Shamos, *Computational Geometry*, 2nd edition, Springer, New York, 1988.
7. Said, A. and W. A. Pearlman, A new, fast, and efficient image codec based on set partitioning in hierarchical trees, IEEE Trans. Circuits and Systems for Video Technology **6** (1996), 243–250.
8. Taubman, D., High performance scalable image compression with EBCOT, IEEE Trans. on Image Processing, July 2000, 1158–1170.

Laurent Demaret, Armin Iske
Zentrum Mathematik
Technische Universität München
D-85747 Garching, GERMANY
`demaret@ma.tum.de`, `iske@ma.tum.de`