Pattern Recognition 44 (2011) 1859–1873

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Edge-based compression of cartoon-like images with homogeneous diffusion

Markus Mainberger [a,*], Andrés Bruhn [a], Joachim Weickert [a], Søren Forchhammer [b]

[a] Mathematical Image Analysis Group, Faculty of Mathematics and Computer Science, Campus E1.1, Saarland University, 66041 Saarbrücken, Germany
[b] DTU Fotonik, Department of Photonics Engineering Coding and Visual Communication, Ørsteds Plads, Building 343, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark

ARTICLE INFO

Available online 12 August 2010

Keywords:
Image compression
Partial differential equations (PDEs)
Laplace equation
Contour coding
Cartoon-like images
Second-generation coding
Multigrid

ABSTRACT

Edges provide semantically important image features. In this paper a lossy compression method for cartoon-like images is presented, which is based on edge information. Edges together with some adjacent grey/colour values are extracted and encoded using a classical edge detector, binary compression standards such as *JBIG* and state-of-the-art encoders such as *PAQ*. When decoding, information outside these encoded data is recovered by solving the Laplace equation, i.e. we inpaint with the steady state of a homogeneous diffusion process. For the discrete reconstruction problem, we prove existence and uniqueness and establish a maximum–minimum principle. Furthermore, we describe an efficient multigrid algorithm. The result is a simple codec that is able to encode and decode in real time. We show that for cartoon-like images this codec can outperform the *JPEG* standard and even its more advanced successor *JPEG2000*.

1. Introduction

Edges play an important role not only in human visual perception, but also in image processing and computer vision where their importance is indisputable. Edges build the basis of various algorithms and can be understood as an intermediate step from a pixel-based to a semantic image representation. The aim of the so-called second-generation image coding methods [1,2] is to incorporate properties of the human visual system into image coding. They try to extract visually significant areas of the image and neglect of visually insignificant data. Thus, those methods are in general lossy. In contrast to well-established methods such as *JPEG* [3] or state-of-the-art codecs as *JPEG2000* [4], second-generation coding methods usually do not rely on basis transforms.

The fact that human beings are able to understand cartoons and line-drawings indicate that edges provide meaningful data in the sense of second-generation image coding. Since it is more compact to describe an image by a few contours than by many pixels, it seems natural that edge information can be exploited for image compression.

There have been numerous theoretical and experimental papers [5–16] which have shown that reconstructions from edge data is in general possible. It has been experienced that the locations of edges do not suffice to reconstruct an image. The mentioned papers make different suggestions about which data should additionally be added in order to obtain "complete" reconstructions. Some suggest to incorporate gradient information, others the grey values adjacent to the edges. It is also possible to consider subsampled image data that lies not directly at the edge or to include some scale information. Not all the mentioned papers used their reconstruction approaches for image compression. However, those who did were not able to come up with competitive results to compression standards such as *JPEG*.

Moreover, the world wide web contains plenty of cartoon-like images that most often are compressed by *JPEG*. None of the standard codecs offers a specialised method for the lossy compression of such images.

Our contribution. The goal of the present paper is to address this problem. We show that it is possible to obtain high quality reconstructions for cartoon-like images when edges are stored in combination with adjacent grey/colour values. Missing information is obtained by computing the steady state of a homogeneous diffusion process, i.e. by solving the Laplace equation. In the literature, the last step is often referred to as PDE-based inpainting or PDE-based interpolation.

Furthermore, we compare different state-of-the art techniques for encoding the edge locations and the adjacent grey/colour values. Thereby our method can even outperform the quality of leading compression standards such as *JPEG2000* in terms of compression rate.

We provide a proof of existence and uniqueness for the solution of the underlying interpolation scheme and show that it satisfies a maximum–minimum principle. Moreover, we develop

* Corresponding author. Tel.: +49 681302 57353; fax: +49 681302 57342.
E-mail addresses: mainberger@mia.uni-saarland.de (M. Mainberger), bruhn@mia.uni-saarland.de (A. Bruhn), weickert@mia.uni-saarland.de (J. Weickert), sofo@fotonik.dtu.dk (S. Forchhammer).

0031-3203/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2010.08.004

an efficient solver for computing the steady state of the corresponding diffusion process. This allows us to encode and decode images in real time.

Organisation of the paper. Our paper is structured as follows: Section 2 describes the encoding method, including edge detection, edge location encoding, and pixel value encoding. In Section 3 we explain how the encoded image can be decoded. The most essential part in this section is the reconstruction of missing pixels between edges with the Laplace equation. In Section 4 we derive the linear system of equations which arises from the discretised Laplace equation. We prove that the solution of this linear system exists, is unique, and satisfies a maximum–minimum principle. Section 5 gives a detailed explanation how the steady state of the corresponding diffusion process can be computed efficiently with a suitable multigrid algorithm. After an experimental evaluation in Section 6 we conclude our paper with a summary in Section 7.

Related work. Let us now briefly mention some related work which has not been discussed so far.

Since our method can be seen as a representative of second-generation coding [1,2], it is related to compression methods that exploit perceptually relevant features. Of course, such features often incorporate edge information. A good survey of different second-generation image coding methods is given in [2].

The idea to use inpainting for image compression has been also exploited in [17–19] where the so-called structure and texture inpainting ideas are integrated in standard codecs such as *JPEG*.

More closely related are papers such as [20–22] which use PDE-based interpolation for compression. In contrast to our approach, they do not rely on edges but use a sparse point mask. For efficiency reasons this point mask is restricted by a binary tree structure such that the position of the interpolation points cannot be chosen optimally. Hence, such methods require more sophisticated interpolation functions based on nonlinear anisotropic diffusion processes.

Recently Köstler et al. [23] developed multigrid methods for the approach proposed in [20]. They could show that it is possible to use these to encode videos in real time on a *Playstation* 3 with a CELL multicore processor. By using homogeneous diffusion, our method is considerably simpler and faster than these ones that are based on nonlinear anisotropic diffusion processes. We will see that it provides real-time performance already on a singlecore CPU.

Our semantic image compression approach can be regarded as a specific implementation of a recent result on optimal point selection for compression with homogeneous diffusion: In [24] it is proven that for interpolation with homogeneous diffusion one should select the interpolation data in proportion to the modulus of the Laplacian. Thus, the modulus of the Laplacian describes how important a certain pixel is. For a piecewise smooth image, as it is the case for cartoon-like images, those pixels are given by the pixels on both sides of an edge. Moreover, since individual points may create unpleasant singularities in the solution of the Laplace equation, it is reasonable to prefer whole edges. Furthermore, edges can be encoded more efficiently than the same number of individual pixels.

This manuscript is an extension of a conference publication [25]. Differences are the evaluation of different bi-tonal encoders and entropy coders. Furthermore, we have improved the strategy to collect the pixel values along the edges and have introduced a step for presmoothing these. We complement this paper by a detailed chapter on the discrete theory for the inpainting problem with homogeneous diffusion and present an efficient way how to solve the arising Laplace equation. Eventually, we give a detailed evaluation of the compression and the run time capabilities of our codec.

2. Encoding

In this section we explain the encoding phase, which essentially consists of three steps. First of all, edges are extracted which encode the location of adjacent grey/colour values. The second step is to encode these locations efficiently. The last step addresses the encoding of the grey/colour values.

2.1. Edge detection

The encoding of an image starts with the detection of edge information. Until today many edge detectors have been developed. In this paper we focus on one of the most classical edge detectors, namely the Marr–Hildreth edge detector [26]. Edges are defined as zero-crossings of the Laplacian of a Gaussian presmoothed image. For a multichannel image $\boldsymbol{u}=(u_1,\dots,u_M)^\top$ we define the Laplacian as the sum of the Laplacians over all M channels:

$$\Delta \boldsymbol{u} = \sum_{m=0}^{M} \Delta u_m = \Delta\left(\sum_{m=0}^{M} u_m\right). \tag{1}$$

In order to remove zero-crossings that have no obvious perceptual significance we combine the Marr–Hildreth edge detector with hysteresis thresholding as suggested by Canny [27]. That means we first define the edge magnitude as the length of the vector $(|\nabla u_1|,\dots,|\nabla u_M|)^\top$, i.e. as $\sqrt{|\nabla u_1|^2+\cdots+|\nabla u_M|^2}$, where the gradient ∇u_m in channel m is computed using Sobel operators. Then we identify edge candidates as pixels where the edge magnitude exceeds a given threshold T_1. All edge candidates with an edge magnitude that is larger than a threshold $T_2 > T_1$ become seed points for relevant edges and are considered to be final edge pixels. In order to keep edge pixels connected as much as possible, we recursively add all edge candidates that are adjacent to final edge pixels.

We are not necessarily bound to this specific edge detector. However, comparisons with the Canny edge detector [27] and an edge detection method based on the Tobbogan watershed segmentation [28] have shown that for cartoon-like images most often the zero-crossings-based edge detector gives the best results.

2.2. Encoding the contour location

In the previous section we have seen how to detect edges. The edges tell us where the pixel values lie that will later on be encoded. We explain now how to encode these locations efficiently.

The edges of an image can be visualised as a bi-level image using black colour for edge pixels and white for background pixels. In the following we denote such an image as edge image. For the compression of bi-level images several methods exists. We will consider the *JBIG* (*Joint Bi-level Image Experts Group*) standard [29], the *JBIG2* standard [30,31] and the *DjVu* file format [32]. Let us now briefly sketch these methods.

JBIG. The *JBIG* standard has been developed as a specialised method for the compression of bi-level images, mainly with respect to fax transmission images which contain textual and line drawing information. Our edge image actually provides quite similar data.

In this paper we use the *JBIG-KIT*[1] [33], which is a free C implementation of the *JBIG* coder. To get optimal compression results for our data, we apply the method in its non-progressive

[1] Available at: http://www.cl.cam.ac.uk/~mgk25/jbigkit/.

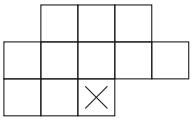

Fig. 1. The static template as used by *JBIG* in our codec. The × marks the pixels which is encoded next.

mode that also automatically excludes the so-called *deterministic prediction*. Furthermore, we disable the *typical prediction* step of *JBIG*. Usually the *JBIG* standard allows to subdivide an image into stripes and encodes these stripes separately. We prevented the routine from doing this such that the image is encoded in its entirety. Finally we set the maximum adaptive template pixel offset to 0. With these settings, the routine essentially becomes a context-based arithmetic coder using a static template for prediction as it is depicted in Fig. 1.

JBIG2. *JBIG2* is the successor of *JBIG* and can for example be used in PDF files versions 1.4 and above.

In this paper we use the open source encoder `jbig2enc`[2] (Version 0.27) as well as the open source decoder `jbig2dec`[3] (Version 0.10).

JBIG2 offers a so-called symbol mode. In this mode it tries to group the (textual) data into symbols that are stored in a dictionary. This dictionary is encoded using context-dependent arithmetic coding. The image itself is coded by describing which symbols appear at which position. Note that for the open source coder `jbig2enc` this mode is lossy, since similar looking symbols are represented by a single bitmap only. `jbig2enc` allows also to disable the symbol mode so that the coder is applied in its generic form. Then it typically just relies on a context-based arithmetic coding algorithm as *JBIG* does, and is lossless.

DjVu. DjVu is a computer file format [32]. It was designed to store scanned documents. This comprises a combination of text and line drawings, but also photographs. For pure bi-tonal images the open source DjVu-library *DjVuLibre*[4] offers an encoder that is called `cjb2`. For decoding we use `ddjvu` which is also included in DjVuLibre.

The method used by `cjb2` is called JB2 and is similar to the symbol mode of *JBIG2*. `cjb2` provides a lossless and a lossy mode which allows small changes on the input image in order to improve the compression ratio. We tested `cjb2` in its lossless mode as well as its highest possible lossy setting (i.e. losslevel = 200).

Table 1 gives a comparison of the methods, using a test set of five different edge images (see Fig. 2). The results suggest to favour *JBIG* for the encoding of our data. Even the lossy mode of DjVu cannot beat its compression rates. In addition, *JBIG* is most often the fastest of all presented methods. For the results shown in Section 6 we restrict ourselves to the *JBIG* coder. In general our codec allows to choose any of the presented edge coders by setting a flag in the file header.

Table 1
Comparison of different bi-tonal coders regarding their compression ratio and time when encoding the edge images from Fig. 2.

Image	JBIG	JBIG2	JBIG2S	DjVu	DjVuL
Compression ratio in bits per pixel (bpp)					
comic	**0.099**	0.109	0.113	0.105	0.104
coppit	**0.124**	0.137	0.158	0.160	0.143
boats	**0.256**	0.273	0.304	0.298	0.295
svalbard	**0.044**	0.053	0.057	0.054	0.053
trui	**0.173**	0.198	0.217	0.199	0.198
Encoding/decoding time in ms					
comic	**3**/**3**	4/4	8/2	21/13	28/10
coppit	**1**/<**1**	**1**/1	7/3	16/16	15/18
boats	**4**/**2**	5/5	19/6	29/22	45/18
svalbard	**3**/3	5/**1**	5/4	13/16	17/21
trui	<**1**/1	2/<**1**	5/1	14/21	23/14

Coders from *left* to *right*: *JBIG*, *JBIG2* with generic coder (JBIG2), *JBIG2* with symbol-mode (JBIG2S), *DjVu* lossless (DjVu) and *DjVu* losslevel 200 (DjVuL). Underlying CPU: Intel Core 2 Duo T7500 at 2.20 GHz. The best result in each line is in bold face letters.

2.3. Encoding the contour pixel values

In this section we consider the grey/colour values we want to store. The locations of these pixel values are encoded by the edge location. Edges usually split areas of different brightness or colour. Thus, it is obvious that we should not store the pixel values that lie directly on the edge. Instead we store the values from both sides. In addition, we store all pixel values from the border of the image domain. This has proven to give better results when reconstructing missing pixels during decoding.

Ordering of the pixel values: Pixel values along one side of a contour usually change only gradually. In contrast pixel values of opposite sides differ by a considerable amount. This suggests to collect the pixel values by the order of their occurrence along the edges instead of row by row or column by column.

To this end, we suggest the following strategy yielding a 1-D signal $\boldsymbol{f}$ that contains the desired pixel values: we create a map M with all pixels adjacent to edges and the border pixels of the image. Then we visit all pixels row by row until the end of the image is reached and apply the following algorithm in each pixel x:

1. If x is in M, put it on a queue Q_1.
2. While Q_1 is not empty do:
   (a) Get pixel as x from Q_1
   and remove it from Q_1.
   (b) If x is not in M goto 2.
   (c) Put x on Q_2,
   remove x from M,
   add the pixel value of x to $\boldsymbol{f}$,
   and set x_{last} to x.
   (d) While Q_2 is not empty do:
      (i) Get pixel as x from Q_2
      and remove it from Q_2.
      (ii) For each pixel $x_{\mathcal{N}}$
      that is in the 4-neighbourhood of x
      and that is in M do:
         If spatial distance between $x_{\mathcal{N}}$ and x_{last}
         is larger than d_{tr}
            Put $x_{\mathcal{N}}$ on Q_1
         else
            Put $x_{\mathcal{N}}$ on Q_2.
            remove it from M,
            add the pixel value of $x_{\mathcal{N}}$ to $\boldsymbol{f}$,
            and set x_{last} to x.

Queue Q_1 contains pixels that have been detected but for which the pixel value has not yet been added to the 1-D signal $\boldsymbol{f}$. In contrast, Queue Q_2 contains pixels for which the pixel value has already been added to $\boldsymbol{f}$ but for which the neighbours still have to be explored. The row-by-row iteration over all pixels ensures that we will not miss any pixel. By removing pixels that are added to $\boldsymbol{f}$ from M the algorithm is guaranteed to terminate.

[2] Available at: http://github.com/agl/jbig2enc.
[3] Available at: http://jbig2dec.sourceforge.net/.
[4] Available at: http://djvu.sourceforge.net/.

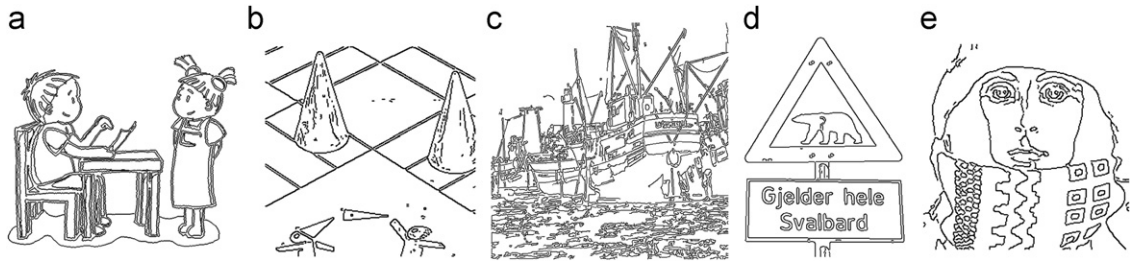

Fig. 2. Test set of edge images: (a) *comic* (512 × 512); (b) *coppit* (256 × 256); (c) *boats* (512 × 512); (d) *svalbard* (380 × 431); (e) *trui* (256 × 256).

In contrast to a pure depth-first search this strategy collects not only the direct pixel values along an edge, but also, depending on d_{tr}, reachable pixel values which lie up to a certain distance close to the edge can be found. This is advantageous if two edges lie so close to each other that they share pixel values.

Subsampling. In order to compress the obtained pixel values, first we perform data reduction by subsampling and requantisation. Thus, the pixel values along the edges are stored in a lossy manner, since we explicitly sparsify data along the edges and reduce the resolution of the co-domain.

For subsampling we consider the 1-D pixel value signal as found in the previous section. On this 1-D signal, we perform uniform subsampling. That means we introduce sampling parameters $d_m \in \{1,\dots,255\}$, where m denotes the m-th channel, and store only every d_m-th value. As already mentioned in the previous section, the pixel values along an edge change only marginally and so they do in our 1-D signal. Thus, missing pixels can be reconstructed in a reasonable way by using linear interpolation. However, this reconstruction fails, as soon as linear interpolation is performed between pixels of distinct edges. Those pixel values can differ by a significant amount. It is indispensable to subsample the pixel values of distinct edges separately.

To this end, we consider how the pixel values along edges have been collected (see previous section). Whenever the recursive search is started (that means whenever a pixel is found by searching row by row for unvisited pixels or a pixel is retrieved from the queue), we assume the previously collected pixels to belong to the same edge segment. Thus, we get for each edge segment a separate 1-D signal. Note that this approach does not demand to store any additional information since it fully relies on the stored edge image.

According to the sampling theorem the quality of a reconstructed signal can be improved by presmoothing the original signal. In our method we suggest to smooth the separated 1-D signals by a Gaussian convolution with standard deviation 1, assuming pixels to have size 1×1. This removes small variations, which also improves the compression rate of the entropy coder later on. Furthermore, smoothing includes some neighbourhood information into the sampled pixels.

Requantisation. The second data reduction step is a requantisation of the pixel values. Originally our image consist of 256 different pixel values per channel. By requantising we reduce the co-domain to q different pixel values. One of the simplest approaches is the so-called *midtread quantisation*. This is a uniform quantisation that allows to reconstruct the minimal and maximal value of the original range.

Let $f_i \in \{0,\dots,255\}$ be a given pixel value of the original signal, and let $a := 255/(q-1)$. Then the quantised value is given by

$$g_i = \left\lfloor \frac{f_i}{a} + \frac{1}{2} \right\rfloor, \tag{2}$$

where $g_i \in \{0,\dots,q-1\}$ and $\lfloor\cdot\rfloor$ denotes the floor function, i.e. $\lfloor x \rfloor$ is the largest integer that does not exceed x. In order to reconstruct the value we compute

$$f_i \approx a \cdot g_i. \tag{3}$$

Note that this means we subdivide our original range into q intervals. All intervals have the width a, except the last and first one which have width $a/2$. After reconstruction, all pixel values of the first interval have been mapped to 0, whereas the pixels of the last interval have been mapped to 255. The pixel values of the other intervals have been mapped to the central pixel value of the interval. For colour images our method allows to requantise each channel separately, i.e. we have q_m with m denoting the m-th channel.

Instead of using a uniform quantisation method, it can pay off to use a non-uniform method as for example the Max–Lloyd quantiser [34,35]. The goal is to adapt the width of the intervals to the probability with which the pixel values occur in the original signal: We start with uniform quantisation. Instead of mapping the values of an interval to the central value we map it to the centre of mass within this interval. We will refer to these points as reconstruction points. Then the interval boundaries are adapted such that they lie again exactly between two reconstruction points.

Starting from this setting we repeat the whole procedure until the interval boundaries do not change anymore. Using the obtained reconstruction points, we achieve better reconstructions. However, for this quantiser we have to store the reconstruction points additionally to reconstruct the pixel values when decoding.

In our case Max–Lloyd quantisation did not pay off, unless for very few quantisation levels. Thus, our codec uses uniform quantisation in channel m if $q_m > 8$ and Max–Lloyd quantisation otherwise.

Entropy coding. Now that we have the subsampled and requantised pixel values, we want to apply an entropy coder. The most classical entropy coders are Huffman coding[5] [36] and arithmetic coding (see footnote 5) [37]. However, meanwhile there are much more sophisticated compression methods available, which often rely on Huffman or arithmetic coding as part of their coding chain. Such compression methods are for example given by *gzip*[6] (Version 1.3.12) or *bzip2*[7] (Version 1.0.5).

One of the best compressors available so far are given by the *PAQ* data compressors[8] [38]. *PAQ* describes a whole family of lossless, GPL-licensed data compression archivers. They are based on a context mixing algorithm, which is related to prediction by partial matching (PPM) [39]. More precisely, *PAQ* uses a predictor that is provided with a large number of models conditioned on different contexts, often even tuned to special file formats. This

[5] Sources from: http://michael.dipperstein.com.
[6] See also: http://www.gzip.org.
[7] See also: http://www.bzip.org.
[8] See also: http://mattmahoney.net/dc/.

Table 2
Comparison of different entropy coders regarding their compression ratio and time.

Orig	HC	HCc	ACs	ACa	*gzip*	*bzip*	*LPAQ2*	*PAQ8o6*
File size in bytes								
12606	7058	7160	6944	7066	4300	4016	3112	**2842**
9006	4164	4336	4087	4241	2222	2095	1658	**1495**
29298	27398	26380	26714	25972	21655	24163	17974	**14766**
96063	48808	48900	48243	42168	8093	7281	5504	**4967**
13128	2178	2420	1491	1710	697	746	**467**	468
5514	6342	5439	5800	5253	5223	4981	4300	**3949**
Encoding/decoding time in ms								
	3/<**1**	3/4	3/2	11/10	<**1**/3	4/<**1**	42/37	1411/1200
	1/3	2/4	2/3	3/4	2/2	2/**1**	31/34	686/694
	4/5	4/19	9/9	16/16	**3**/**3**	7/4	86/89	2470/2438
	5/**5**	6/26	19/18	57/51	11/9	68/**5**	168/190	9041/8377
	1/<**1**	<**1**/<**1**	<**1**/1	6/5	5/2	1/<**1**	29/29	941/825
	1/2	1/4	1/3	4/2	<**1**/1	3/<**1**	39/34	693/711

Considered are six randomly chosen pixel value data files, created by our codec. From *left* to *right*: original file (orig), Huffman coding (HC), Huffman coding using canonical codes (HCc), arithmetic coding with static model (ACs), arithmetic coding with adaptive model (ACa), *gzip* (Version 1.3.12), *bzip2* (Version 1.0.5), *LPAQ2*, and *PAQ6o8*. Underlying CPU: Intel Core 2 Duo T7500 at 2.20 GHz. The best result for each file is in bold face letters.

predictor is applied in combination with arithmetic coding. As in our conference paper [25], we consider the *PAQ8o6* release.

Unfortunately, the *PAQ8o6* compressor is rather slow. This suggests to use the single file compressor *LPAQ* instead, more precisely the *LPAQ2* release. *LPAQ* is a modified version of *PAQ*, which is faster at the expense of compression.

Our codec allows to choose between all mentioned compressors. A comparison of their compression capabilities regarding our data is depicted in Table 2. For almost all examples *PAQ* outperforms the other methods regarding the compression rate. However, it is also by far the slowest method. Huffman and arithmetic coding as well as *gzip* and *bzip2* are up to small differences the fastest methods. Depending on the application it is up to the user which entropy coder should be chosen. For highest compression, *PAQ* is recommended. For fastest compression, *bzip2* or *gzip* should be used. *LPAQ* is a reasonable trade-off between high quality compression and run time. Thus, we decided to set *LPAQ* as default pixel value encoder in our codec.

2.4. File format

After this last step we have two encoded data parts, namely the encoded edge image and the encoded grey/colour values. Including header data, our encoded image consists of the following parts:

- Size of edge data, needed to split the encoded edge image from the encoded grey/colour values (requires 4 bytes).
- Edge image coder, indicating how the edge image has been encoded; default: *JBIG* (3 bit).
- Colour bit, indicating if a greyscale or colour image has been encoded (1 bit).
- Entropy coder, indicating which entropy coder has been used; default: *LPAQ2* (3 bit).
- Parameter d_{tr} for recursive search of grey/colour values; default: 1 (1 byte).
- Numbers of quantisation intervals q_m; default: 25 (1 or 3 bytes).
- Sampling distances d_m; default: 10 (1 or 3 bytes).
- encoded edge image (variable length, given in first 4 bytes of header).
- Encoded pixel values (variable length).

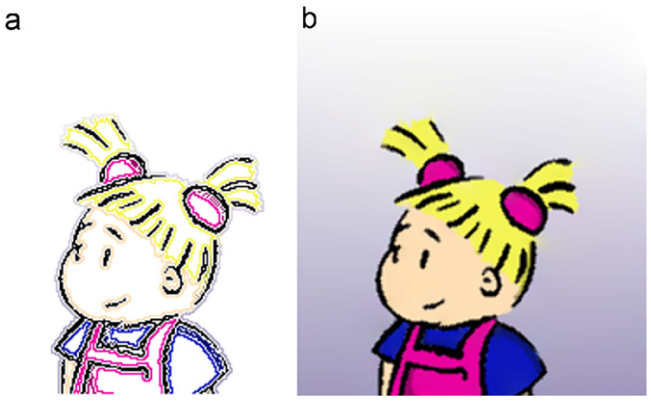

Fig. 3. (a) Zoom into reconstructed colour pixels adjacent to the edges of the test image *comic*. (b) Zoom into corresponding reconstruction with homogeneous diffusion inpainting.

3. Decoding

To decode an encoded image, we reverse all steps done in the encoding chain. However, the last step of the decoding chain differs from encoding: to get the final image, missing pixels are reconstructed by the steady state of a homogeneous diffusion process.

3.1. Decoding the contour location and pixel values

We start our decoding approach by reading the header information. We split the encoded edge image from the encoded grey/colour values and decode them with the corresponding decoders. On the one hand we obtain a binary edge image giving us explicitly the final image size. On the other hand we get grey/colour values that are quantised and have to be reconstructed (see also Section 2.3). To this end, we perform the recursive search on the edge image as done during encoding in Section 2.3 and distribute the grey/colour values. Missing pixels along the edge segments are filled in by linear interpolation. The outcome is an image which contains the decoded colours on both sides of each edge (see Fig. 3(a)). The grey/colour values of all the other pixels are still unknown and their recovery is the goal of the next section.

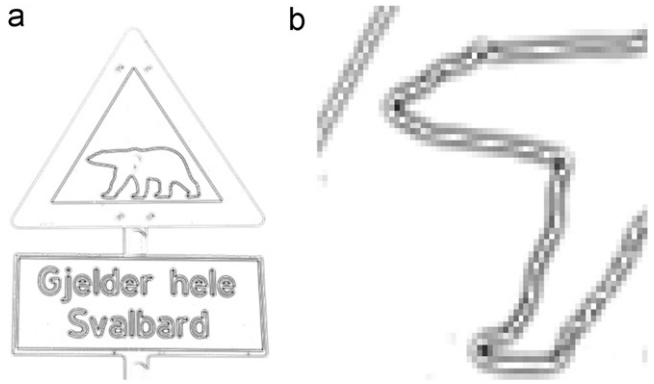

Fig. 4. (a) Modulus of the Laplacian for the test image *svalbard* normalised to [0,255]. The darker the colour, the larger the modulus of the Laplacian. (b) Detail of (a).

3.2. Reconstruction by homogeneous diffusion

In order to reconstruct missing pixels between the edges we use homogeneous diffusion for interpolation. This is the simplest and computationally most favourable inpainting approach based on partial differential equations (PDEs) [40]. We prefer this simple approach for several reasons:

First of all it is one of the analytically best understood inpainting approaches. Recently, Belhachmi et al. [24] have proven that one should choose the interpolation data proportional to the modulus of the Laplacian of the image. Thus, the modulus of the Laplacian indicates which pixels should preferentially be stored for the interpolation with homogeneous diffusion. For cartoon-like images, i.e. piecewise smooth images, this comes down to the pixels left and right of an edge contour (see Fig. 4). Hence, by choosing homogeneous diffusion for inpainting and the pixels adjacent to edges, we meet this theory to a certain extent.

Furthermore, filling-in from image edges resembles a classical finding in biological vision: Already in 1935 Werner made the hypothesis that a contour-based filling-in process is responsible for the human perception of surface brightness and colour [41].

Finally, for inpainting with homogeneous diffusion an efficient implementation is possible. Section 5 provides such an algorithm for this crucial decoding step.

To reconstruct the missing values between the edges by homogeneous diffusion, we consider given pixel values to form Dirichlet boundaries. Then we compute the steady state ($t\to\infty$) of the following diffusion process (also known as heat equation) [42]:

$$\partial_t u = \Delta u. \tag{4}$$

Already given data are preserved whereas reconstructed data satisfy the Laplace equation $\Delta u = 0$ (see also [10]). Fig. 3(b) shows an exemplary result for the image *comic*. The discrete theory behind this inpainting process is discussed in detail in the next section.

4. Discrete theory

In this section we derive a suitable discretisation of the homogeneous diffusion process and discuss important discrete properties such as the existence of a unique solution and the maximum–minimum principle. To this end, we consider two formulations that describe the steady state of the inpainting process with homogeneous diffusion: a reduced formulation and an extended formulation. The latter is used for the development of a fast hierarchical algorithm in Section 5.

4.1. Reduced vs. extended formulation

Let us start by discussing the *reduced* formulation of the inpainting problem. To this end we consider a grey value image $f(\boldsymbol{x})$, where $\boldsymbol{x}=(x,y)^\top$ denotes the location within a rectangular image domain Ω. Furthermore we assume that the grey values of this image are only known in a subset $\Omega_K \subset \Omega$ of the image domain. Using homogeneous diffusion, the inpainted image $u(\boldsymbol{x})$ can then be computed as the solution of the Laplace equation

$$\Delta u(\boldsymbol{x}) = 0 \quad \text{for } \boldsymbol{x} \in \Omega\backslash\Omega_K, \tag{5}$$

with homogeneous (reflecting) Neumann boundary conditions across image boundaries, i.e.

$$\partial_n u = 0 \quad \text{on } \partial\Omega. \tag{6}$$

All stored points are removed from the set of unknowns and used as Dirichlet boundary conditions:

$$u(\boldsymbol{x}) = f(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \Omega_K. \tag{7}$$

As a consequence, the solution is only computed in the inpainting domain $\Omega\backslash\Omega_K$. Evidently, this domain is no longer rectangular. From a numerical viewpoint, however, it can be desirable to have a domain of the solution that has a regular shape. This issue becomes particularly important, since we will develop a hierarchical solver that considers the original problem at different scales. Thus, we propose an *extended* formulation of the inpainting problem, where the solution is computed on the entire image domain Ω. This formulation is given by

$$c(\boldsymbol{x})(u-f)-(1-c(\boldsymbol{x}))\Delta u = 0 \tag{8}$$

with homogeneous Neumann boundary conditions across the image boundaries:

$$\partial_n u = 0 \quad \text{on } \partial\Omega. \tag{9}$$

The function $c(\boldsymbol{x})$ serves as binary mask that specifies whether a grey value has been stored at a certain location or not, i.e.

$$c(\boldsymbol{x}) = \begin{cases} 1 & \text{for } \boldsymbol{x} \in \Omega_K, \\ 0 & \text{for } \boldsymbol{x} \in \Omega\backslash\Omega_K. \end{cases} \tag{10}$$

By evaluating Eq. (8) for the two possible values of $c(\boldsymbol{x})$ one can easily verify that its solution is equivalent to the one of the reduced formulation in (5)–(7). The main difference is that stored pixels are now "recomputed" instead of being used as Dirichlet boundary conditions.

4.2. Discretisation

In order to compute the solution of the reduced problem (5)–(7) or the extended problem (8)–(10) numerically, we discretise all occurring expressions by means of finite differences [43,44]. To this end, we consider the problem on a rectangular grid with $N^{\boldsymbol{h}} = N_x^{\boldsymbol{h}} \times N_y^{\boldsymbol{h}}$ pixels, where $N_x^{\boldsymbol{h}}$ and $N_y^{\boldsymbol{h}}$ denote the number of pixels in x- and y-direction, respectively, and $\boldsymbol{h}=(h_x,h_y)^\top$ is a joint index that describes the corresponding grid spacing in both directions. Furthermore, we index all pixels consecutively, and we denote by $\Omega^{\boldsymbol{h}}$ and $K^{\boldsymbol{h}}$ the set of indices of all pixels and known pixels, respectively. Then we obtain the following two linear systems of equations: while the system for the reduced formulation reads

$$\sum_{l\in\{x,y\}} \sum_{j\in\mathcal{N}_l(i)} \frac{u_i^{\boldsymbol{h}} - (1-c_j^{\boldsymbol{h}})u_j^{\boldsymbol{h}}}{h_l^2} = \sum_{l\in\{x,y\}} \sum_{j\in\mathcal{N}_l(i)} \frac{c_j^{\boldsymbol{h}} f_j^{\boldsymbol{h}}}{h_l^2} \tag{11}$$

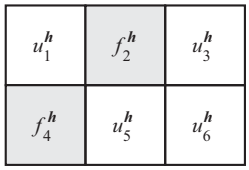

Fig. 5. Inpainting example of size 3×2 with two known pixels.

for $i \in \Omega^{\boldsymbol{h}} \backslash K^{\boldsymbol{h}}$, the system for the extended formulation is given by

$$c_i^{\boldsymbol{h}} u_i^{\boldsymbol{h}} - (1-c_i^{\boldsymbol{h}}) \sum_{l \in \{x,y\}} \sum_{j \in \mathcal{N}_l(i)} \frac{u_j^{\boldsymbol{h}} - u_i^{\boldsymbol{h}}}{h_l^2} = c_i^{\boldsymbol{h}} f_i^{\boldsymbol{h}} \tag{12}$$

for $i \in \Omega^{\boldsymbol{h}}$, i.e. $i = 1, \ldots, N^{\boldsymbol{h}}$. Here, $u_i^{\boldsymbol{h}}$, $f_i^{\boldsymbol{h}}$, and $c_i^{\boldsymbol{h}}$ are the approximations of the corresponding functions at location i, and $\mathcal{N}_l(i)$ denotes the set of neighbours of pixel i in the direction of axis l (which contains up to two elements per direction).

4.3. Structure of the linear systems

Let us now formulate both equation systems in traditional matrix–vector notation $A^{\boldsymbol{h}} \boldsymbol{u}^{\boldsymbol{h}} = \boldsymbol{b}^{\boldsymbol{h}}$, where $A^{\boldsymbol{h}}$ denotes the system matrix, $\boldsymbol{u}^{\boldsymbol{h}}$ is a vector that contains all unknowns, and $\boldsymbol{b}^{\boldsymbol{h}}$ stands for the right hand side. To make things more transparent, we illustrate the structure of the resulting system by the example of a 3×2 image, where $f_2^{\boldsymbol{h}}$ and $f_4^{\boldsymbol{h}}$ are known, and the grid spacing has been chosen to be $h_x = h_y = 1$ (see Fig. 5). While the linear system of the extended formulation is given by

$$\underbrace{\left(\begin{array}{ccc|ccc} 2 & -1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 0 & -1 & 2 \end{array}\right)}_{A_{\text{ext}}^{\boldsymbol{h}}} \underbrace{\begin{pmatrix} u_1^{\boldsymbol{h}} \\ u_2^{\boldsymbol{h}} \\ u_3^{\boldsymbol{h}} \\ u_4^{\boldsymbol{h}} \\ u_5^{\boldsymbol{h}} \\ u_6^{\boldsymbol{h}} \end{pmatrix}}_{\boldsymbol{u}_{\text{ext}}^{\boldsymbol{h}}} = \underbrace{\begin{pmatrix} 0 \\ f_2^{\boldsymbol{h}} \\ 0 \\ f_4^{\boldsymbol{h}} \\ 0 \\ 0 \end{pmatrix}}_{\boldsymbol{b}_{\text{ext}}^{\boldsymbol{h}}},$$

the linear system for the reduced formulation reads

$$\underbrace{\left(\begin{array}{cc|cc} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & -1 \\ \hline 0 & 0 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{array}\right)}_{A_{\text{red}}^{\boldsymbol{h}}} \underbrace{\begin{pmatrix} u_1^{\boldsymbol{h}} \\ u_3^{\boldsymbol{h}} \\ u_5^{\boldsymbol{h}} \\ u_6^{\boldsymbol{h}} \end{pmatrix}}_{\boldsymbol{u}_{\text{red}}^{\boldsymbol{h}}} = \underbrace{\begin{pmatrix} f_2^{\boldsymbol{h}} + f_4^{\boldsymbol{h}} \\ f_2^{\boldsymbol{h}} \\ f_2^{\boldsymbol{h}} + f_4^{\boldsymbol{h}} \\ 0 \end{pmatrix}}_{\boldsymbol{b}_{\text{red}}^{\boldsymbol{h}}}.$$

At first glance the matrices $A_{\text{ext}}^{\boldsymbol{h}}$ and $A_{\text{red}}^{\boldsymbol{h}}$ may look quite similar, since they consist mainly of the linear operator that discretises the negative Laplacian. However, a closer look shows that the treatment of the known pixels is different: While they are still present in the extended system and are hence "recomputed", they have been eliminated completely in the reduced system. As a consequence, the corresponding grey values f_i are shifted from the right hand side of the known pixels to the right hand side of their neighbours.

Despite these differences, both equation systems have two things in common: (i) They have the same solution in the unknown pixels. Thus, proofs for existence, uniqueness and a maximum–minimum principle carry over. (ii) For a given grid spacing $\boldsymbol{h}$, they are fully described by the binary mask $\boldsymbol{c}^{\boldsymbol{h}} = (c_1^{\boldsymbol{h}}, \ldots, c_N^{\boldsymbol{h}})^\top$ and the values of $\boldsymbol{f}^{\boldsymbol{h}} = (f_1^{\boldsymbol{h}}, \ldots, f_N^{\boldsymbol{h}})^\top$. This is important for Section 5, where we consider the systems at different scales.

4.4. Discrete well-posedness

Now that we have established discrete formulations of our reduced and extended interpolation problems, let us show that these problems have a unique solution that remains within the convex hull of the specified pixel data:

Theorem 1 (*Discrete well-posedness*). *Let $K^{\boldsymbol{h}}$ be nonempty. Then the linear systems* (11) *and* (12) *have a unique solution. In the unknown pixels $i \in \Omega^{\boldsymbol{h}} \backslash K^{\boldsymbol{h}}$ it satisfies the maximum–minimum principle*

$$\min_{j \in K^{\boldsymbol{h}}} f_j^{\boldsymbol{h}} \leq u_i^{\boldsymbol{h}} \leq \max_{j \in K^{\boldsymbol{h}}} f_j^{\boldsymbol{h}}. \tag{13}$$

Proof. It is sufficient to prove existence and uniqueness for the reduced problem (11), since both formulations are equivalent. Thus, let us show that the system matrix $A_{\text{red}}^{\boldsymbol{h}}$ is invertible. Since $A_{\text{red}}^{\boldsymbol{h}}$ is symmetric, its eigenvalues are real. Moreover, by inspecting the Gerschgorin disks of $A_{\text{red}}^{\boldsymbol{h}}$, it follows that all eigenvalues are nonnegative. However, we have to exclude that 0, which can lie on the boundary of some Gerschgorin disks, is an eigenvalue. To this end, we apply a result by Feingold and Varga [45, Theorem3]: If A is block irreducible and λ is an eigenvalue of A that lies on the boundary of the union of all Gerschgorin disks, then it must lie in all Gerschgorin disks. It is easy to verify that our matrix $A_{\text{red}}^{\boldsymbol{h}}$ is block irreducible. Let us now consider some pixel $i \in \Omega^{\boldsymbol{h}} \backslash K^{\boldsymbol{h}}$ that has at least one pixel $j \in K^{\boldsymbol{h}}$ in its 4-neighbourhood. Then its Gerschgorin disk G_i does not contain 0. Thus, it follows that 0 cannot be an eigenvalue of $A_{\text{red}}^{\boldsymbol{h}}$ and the inverse of $A_{\text{red}}^{\boldsymbol{h}}$ exists.

To prove the maximum–minimum principle, it is more convenient to consider the extended discrete model (12). Since $A_{\text{red}}^{\boldsymbol{h}}$ is invertible, we know from the equivalence of both models that also the inverse of $A_{\text{ext}}^{\boldsymbol{h}}$ exists.

First we show that the inverse of $A_{\text{ext}}^{\boldsymbol{h}}$ is nonnegative. To this end we note that $A_{\text{ext}}^{\boldsymbol{h}}$ has nonpositive off-diagonal entries, positive diagonal entries, nonnegative row sums, and at least one positive row sum. Hence, $A_{\text{ext}}^{\boldsymbol{h}}$ is an M-matrix. It is well known that the inverse of a nonsingular M-matrix is a nonnegative matrix.

Second we prove that each inpainted value $u_i^{\boldsymbol{h}} \in \Omega^{\boldsymbol{h}} \backslash K^{\boldsymbol{h}}$ can be written as a convex combination of the specified grey values $\{f_j^{\boldsymbol{h}} | j \in K^{\boldsymbol{h}}\}$. Let us consider the vectors $\boldsymbol{e}, \boldsymbol{g} \in \mathbb{R}^N$ with $e_i = 1$ for all $i \in \Omega^{\boldsymbol{h}}$, and

$$g_i = \begin{cases} 1 & \text{for } i \in K^{\boldsymbol{h}}, \\ 0 & \text{else.} \end{cases} \tag{14}$$

If $i \in K^{\boldsymbol{h}}$, then the i-th row sum of $A_{\text{ext}}^{\boldsymbol{h}}$ is 1, while for $i \in \Omega^{\boldsymbol{h}} \backslash K^{\boldsymbol{h}}$ the corresponding row sum is 0. Thus, we have

$$A_{\text{ext}}^{\boldsymbol{h}} \boldsymbol{e} = \boldsymbol{g}. \tag{15}$$

With $D := (A_{\text{ext}}^{\boldsymbol{h}})^{-1}$ this gives

$$\boldsymbol{e} = D\boldsymbol{g} \tag{16}$$

and therefore

$$\sum_{j \in K^{\boldsymbol{h}}} d_{i,j} = 1 \tag{17}$$

for all $i \in \Omega^{\boldsymbol{h}}$. Since

$$u_i^{\boldsymbol{h}} = \sum_{j \in K^{\boldsymbol{h}}} d_{i,j} f_j^{\boldsymbol{h}} \tag{18}$$

for all $i \in \Omega^{\boldsymbol{h}}$, and D is nonnegative, we know that $u_i^{\boldsymbol{h}}$ is in the convex hull of $\{f_j^{\boldsymbol{h}} | j \in K^{\boldsymbol{h}}\}$. Thus, our maximum–minimum principle is satisfied. This concludes the proof. □

5. Efficient numerical solvers

After we have shown existence and uniqueness as well as the maximum–minimum principle for our solution, let us consider the extended formulation only and show how the corresponding linear system $A^{\boldsymbol{h}}_{\text{ext}}\boldsymbol{u}^{\boldsymbol{h}}_{\text{ext}} = \boldsymbol{b}^{\boldsymbol{h}}_{\text{ext}}$ can be solved efficiently. To this end we develop a so-called full multigrid method—a hierarchical iterative technique, which belongs to the fastest methods for solving the Laplace equation [46,47]. This is done in four steps. Firstly, we select a simple non-hierarchical solver that forms the basis of our multigrid implementation. Secondly, we show how this solver can be embedded in a two-grid cycle that performs useful correction steps at a coarser resolution. Thirdly, we focus on advanced multigrid strategies that extend this hierarchical concept to more than two grid levels. Finally, we discuss specific details of our implementation that are related to the use of sparse data. For simplifying the notation, we skip the ext-index from now on. Furthermore, as in the previous section we restrict our explanations to grey value images. The extension to colour images is straightforward and comes down to applying the algorithm to each channel separately.

5.1. Basic solver

A common solver in the context of linear systems as given by the extended formulation (12) is the classical Gauß–Seidel method [48,49]. The corresponding iteration step is given by Eq. (19), where $|\mathcal{N}_l(i)|$ denotes the number of neighbours of pixel i in the direction of the axis l, $\mathcal{N}_l^-(i) := \{\mathcal{N}_l(i) | j < i\}$ is the set of neighbouring pixels in this direction that have already been processed, while $\mathcal{N}_l^+(i) := \{\mathcal{N}_l(i) | i < j\}$ stands for the pixels that yet have to be updated.

$$u_i^{\boldsymbol{h},k+1} = \frac{c_i^{\boldsymbol{h}} f_i^{\boldsymbol{h}} + (1-c_i^{\boldsymbol{h}}) \sum_{l \in \{x,y\}} \frac{1}{h_l} \left(\sum_{j \in \mathcal{N}_l^-(i)} u_j^{\boldsymbol{h},k+1} + \sum_{j \in \mathcal{N}_l^+(i)} u_j^{\boldsymbol{h},k} \right)}{c_i^{\boldsymbol{h}} + (1-c_i^{\boldsymbol{h}}) \sum_{l \in \{x,y\}} \frac{1}{h_l} \sum_{j \in \mathcal{N}_l(i)} |\mathcal{N}_l(i)|(i)}$$

$$= \begin{cases} f_i^{\boldsymbol{h}} & \text{for } c_i^{\boldsymbol{h}} = 1, \\ \dfrac{\sum_{l \in \{x,y\}} \sum_{j \in \mathcal{N}_l^-(i)} \frac{1}{h_l} u_j^{\boldsymbol{h},k+1} + \sum_{l \in \{x,y\}} \sum_{j \in \mathcal{N}_l^+(i)} \frac{1}{h_l} u_j^{\boldsymbol{h},k}}{\sum_{l \in \{x,y\}} \frac{1}{h_l} \sum_{j \in \mathcal{N}_l(i)} |\mathcal{N}_l(i)|} & \text{for } c_i^{\boldsymbol{h}} = 0. \end{cases} \tag{19}$$

5.2. Bidirectional multigrid

Unfortunately, iterative solvers such as the presented Gauß–Seidel method have one decisive drawback: due to the local coupling of neighbours in the iteration scheme, it may take thousands of iterations to spread information over large distances. As a consequence, only high frequencies of the error are reduced, while low frequencies remain almost undamped. This leads to a convergence rate that is very fast at the beginning, but slows down significantly after a few iterations already.

In order to overcome this problem, bidirectional multigrid methods [46,47,50–52] make use of coarser levels where they obtain useful correction steps. How this works exactly will be described in detail by the example of the following two-grid cycle that forms the basis of our implementation.

(1) *Presmoothing relaxation*: First, we perform a few iterations with the Gauß–Seidel method given by Eq. (19). This allows us to reduce the high frequency components of the estimation error.

(2) *Coarse grid computation*: Since the first step only gives us an approximation $\tilde{\boldsymbol{u}}^{\boldsymbol{h}}$ of the correct solution $\boldsymbol{u}^{\boldsymbol{h}}$, we are interested in computing the error $\boldsymbol{e}^{\boldsymbol{h}} = \boldsymbol{u}^{\boldsymbol{h}} - \tilde{\boldsymbol{u}}^{\boldsymbol{h}}$ to correct our result. Unfortunately, this error cannot be determined directly. However, it is possible to compute the residual $\boldsymbol{r}^{\boldsymbol{h}} = \boldsymbol{b}^{\boldsymbol{h}} - A^{\boldsymbol{h}}\tilde{\boldsymbol{u}}^{\boldsymbol{h}}$ that is related to the error via the following equation:

$$A^{\boldsymbol{h}}\boldsymbol{e}^{\boldsymbol{h}} = A^{\boldsymbol{h}}\boldsymbol{u}^{\boldsymbol{h}} - A^{\boldsymbol{h}}\tilde{\boldsymbol{u}}^{\boldsymbol{h}} = \boldsymbol{b}^{\boldsymbol{h}} - A^{\boldsymbol{h}}\tilde{\boldsymbol{u}}^{\boldsymbol{h}} = \boldsymbol{r}^{\boldsymbol{h}}. \tag{20}$$

The basic idea of bidirectional multigrid methods is now to transfer this so-called residual equation $A^{\boldsymbol{h}}\boldsymbol{e}^{\boldsymbol{h}} = \boldsymbol{r}^{\boldsymbol{h}}$ to a coarser grid with grid spacing $\boldsymbol{H} > \boldsymbol{h}$ by restricting the entries of $A^{\boldsymbol{h}}$ and $\boldsymbol{r}^{\boldsymbol{h}}$. Apart from the reduced computational effort on the coarse grid, this strategy offers the advantage that low frequencies on the fine grid reappear as higher ones on the coarse grid. Hence, they can be efficiently attenuated by applying the same iterative solver that was already used in the presmoothing relaxation step. Transferring the residual equation to the coarse grid yields the following system of equations:

$$c_i^{\boldsymbol{H}} e_i^{\boldsymbol{H}} - (1-c_i^{\boldsymbol{H}}) \sum_{l \in \{x,y\}} \sum_{j \in \mathcal{N}_l(i)} \frac{e_j^{\boldsymbol{H}} - e_i^{\boldsymbol{H}}}{H_l} = c_i^{\boldsymbol{H}} r_i^{\boldsymbol{H}} \tag{21}$$

for $i = 1, \ldots, N^{\boldsymbol{H}}$. As one can see, this system has the same structure as the original one from the fine grid given by Eq. (11). Moreover, we observe that it is sufficient to restrict the entries of $\boldsymbol{c}^{\boldsymbol{h}}$ and $\boldsymbol{r}^{\boldsymbol{h}}$ to set up the coarse grid equation systems, since the discretisation of the Laplacian follows directly from the new grid spacing $\boldsymbol{H}$. In order to solve this system we may use the Gauß–Seidel method from (19) or, if the number of pixels is small enough, a direct solver such as Gaussian elimination [53].

(3) *Coarse grid correction*: After we have solved the residual equation system on the coarse grid, we have to transfer the computed error back on the fine grid to correct our previous approximation. This correction step is given by $\tilde{\boldsymbol{u}}^{\boldsymbol{h}}_{\text{new}} = \tilde{\boldsymbol{u}}^{\boldsymbol{h}} + \boldsymbol{e}^{\boldsymbol{h}}$.

(4) *Postsmoothing relaxation*: Finally, we perform again a few iterations with the original Gauß–Seidel method from (19). This step allows us to remove high frequency errors that have been introduced by the interpolation of the coarse grid error.

5.3. Advanced multigrid strategies

Instead of solving the residual equation system at the coarse grid directly, it is much more efficient to use a third, even coarser grid that provides a correction step for the second one. Such a hierarchical application of the two-grid cycle is called *V-cycle*. Visiting each coarse grid twice per level yields the so-called W-cycle, which offers better convergence rates at the expense of slightly increased computational costs.

Additionally, one can speed up the computation by starting with a reasonably good initialisation. To this end, we embed the W-cycle in a coarse-to-fine estimation framework: Starting from a very coarse grid, we successively refine the problem, where solutions from coarser levels serve as initialisation for finer ones.

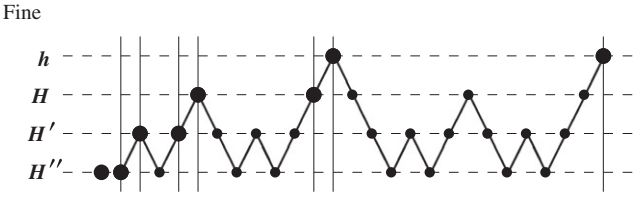

Fig. 6. Full multigrid scheme for four levels with increasing resolution from $\boldsymbol{H}''$ to $\boldsymbol{h}$ (decreasing grid size). At each level one W-cycle is used to solve the resulting system.

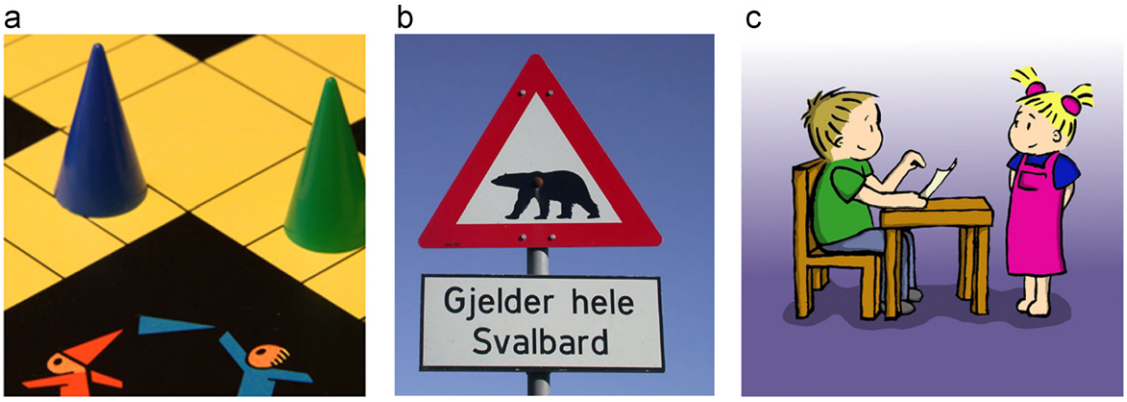

Fig. 7. Different test images: (a) *coppit* (256×256, real world); (b) *svalbard* (380×431, real world); (c) *comic* (512×512, synthetic).

At each level, the previously explained W-cycle is used to solve the resulting linear system. This combination of error correction steps and hierarchical initialisation yields the multigrid method with the best performance: *full multigrid*. A sketch that illustrates how the different grids are successively traversed is shown in Fig. 6.

5.4. Implementation details

In our implementation, we use a full multigrid scheme with one W-cycle per level, where the number of pre- and post-smoothing iterations is set to 2. The transfer between the different grids is realised by non-dyadic versions of area-based averaging and area-based interpolation as proposed in [54]. However, since these operators are not designed to deal with sparse data filled up with zeros, we have to normalise the result after restriction such that values averaged with zeros still make sense. To this end, we propose a strategy similar to normalised convolution known from scattered data interpolation [55].

The key idea in this context is to exploit the fact that all averaging effects become explicit in the restricted version $\tilde{\boldsymbol{c}}^{\boldsymbol{H}}$ of the binary mask $\boldsymbol{c}^{\boldsymbol{h}}$. In particular, this restricted version may contain values in the complete range from 0 to 1. Assuming that we have obtained the coarse grid result $\tilde{\boldsymbol{v}}^{\boldsymbol{H}}$ by the restriction of the corresponding fine grid data $\boldsymbol{v}^{\boldsymbol{h}}$, we thus propose the following subsequent normalisation:

$$v_i^{\boldsymbol{H}} = \begin{cases} \tilde{v}_i^{\boldsymbol{H}} / \tilde{c}_i^{\boldsymbol{H}} & \text{for } \tilde{c}_i^{\boldsymbol{H}} \neq 0, \\ 0 & \text{for } \tilde{c}_i^{\boldsymbol{H}} = 0. \end{cases} \tag{22}$$

This normalisation is applied to all sparse data after restriction, i.e. to the image $\boldsymbol{f}^{\boldsymbol{h}}$, the residual $\boldsymbol{r}^{\boldsymbol{h}}$, and the mask $\boldsymbol{c}^{\boldsymbol{h}}$ itself (which makes it binary again).

When interpolating the solution $\boldsymbol{u}^{\boldsymbol{H}}$ or the error $\boldsymbol{e}^{\boldsymbol{H}}$ from coarser to finer grids, we can exploit the fact that the result is known at certain positions, i.e. where $\boldsymbol{c}_i^{\boldsymbol{h}} = 1$. At those locations, we simply set back the data to the known values.

6. Experiments

Let us now investigate the capabilities of the suggested codec. To this end we first compare the codec presented in [25] with our new, improved version. Then we give a comparison to well-established and state-of-the-art compression methods, namely *JPEG* and *JPEG2000*. Finally we identify the limitations of our method.

For quantitative comparison, we use the *peak-signal-to-noise ratio* (PSNR), a common error measure for the comparison between compressed images: let max be the maximal possible pixel value, which is 255 in our case. Furthermore, let N be the number of image pixels, M be the number of channels and $(f_{m,i})_{i=1..N}$ and $(u_{m,i})_{i=1..N}$ the pixel values of the original image in channel m and its reconstructed/decompressed version, respectively. The PSNR is defined via the *mean squared error* (MSE):

$$\mathrm{PSNR} := 10 \cdot \log_{10}\left(\frac{\max^2}{\mathrm{MSE}}\right)[\mathrm{dB}], \tag{23}$$

with

$$\mathrm{MSE} := \frac{1}{M \cdot N} \sum_{m=1}^{M} \sum_{i=1}^{N} (f_{m,i} - u_{m,i})^2. \tag{24}$$

For runtime measurements we use only one core of an Intel Core 2 Duo T7500 at 2.20 GHz CPU and measure the elapsed time.

The underlying test images are shown in Fig. 7. Whereas *coppit* and *svalbard* are cartoon-like real world images, *comic* is a pure synthetic image.

6.1. Comparison with the old version of the codec

Let us first briefly compare our new codec with the previously suggested codec of our conference paper [25]. To this end we use the same test images (see Fig. 7) and same underlying edge images as in [25] (see Fig. 8). The edge images were created by choosing the thresholds for hysteresis thresholding and the standard deviation for presmoothing such that we obtain visually pleasant reconstructions. Moreover, we demand that the edge images look reasonable, that means we avoid false edges by choosing appropriate thresholds. Parameters available in both methods are chosen as in [25]. Our new codec provides additional parameters which are set to their default values. That means the parameter d_{tr} for the recursive search of the pixel values is set to 1, and the standard deviation for Gaussian presmoothing of the pixel value signals is set to 1 as well. The edge images are encoded by *JBIG* again. However, as entropy coder we use *LPAQ2* instead of *PAQ8o6*. Table 3 shows the PSNR and compression in bits per pixel for both versions of the codec. Furthermore, the encoding and decoding time is depicted.

Regarding the compression rate the new codec is at least as good as the old one, even though the faster *LPAQ2* method is used instead of *PAQ8o6*. Moreover, all reconstructions are better than the old ones. These improvements are a result of both the new strategy for finding the pixels along edges as well as the Gaussian

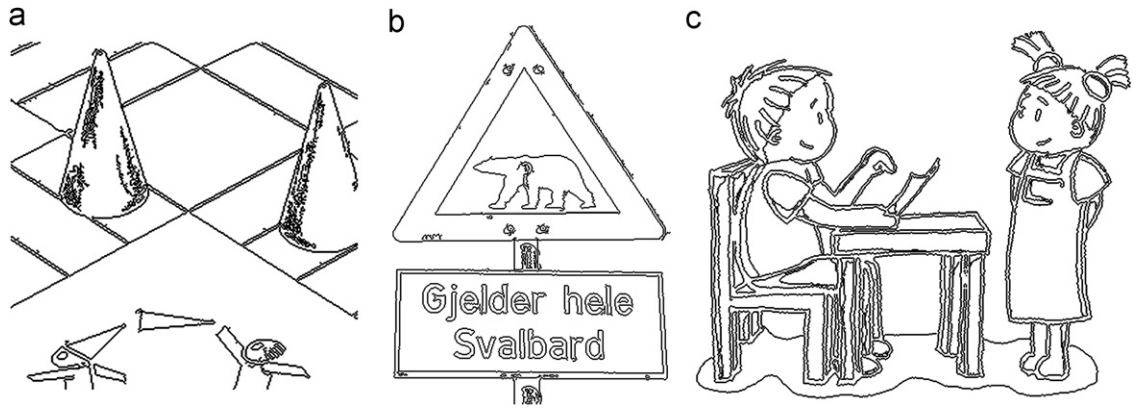

Fig. 8. Edge images obtained by zero-crossings-based edge detection such that visually pleasant reconstructions are obtained. Edge detector parameters are chosen such that false edges are avoided. Edge image for (a) *coppit* ($T_1=4.3$, $T_2=23$, $\sigma=0.5$), (b) *svalbard* ($T_1=3.5$, $T_2=20$, $\sigma=0.6$) and (c) *comic* ($T_1=20$, $T_2=20$, $\sigma=0.47$).

Table 3
Comparison of the improved codec with the old one given in our conference paper [25].

Image	*coppit*		*svalbard*		*comic*	
Approach	Old	**New**	Old	**New**	Old	**New**
Compression rate (bpp)	0.34	**0.34**	0.23	**0.22**	0.21	**0.21**
PSNR (dB)	30.16	**30.47**	30.21	**30.37**	30.31	**30.37**
Encoding time (s)	0.90	**0.08**	1.10	**0.12**	3.01	**0.16**
Decoding time (s)	18.60	**0.15**	148.81	**0.33**	495.93	**0.47**

Sampling distances are $d_m=5$ for all test images ($m\in\{1,2,3\}$). The quantisation parameters are $q_m=16$ for *coppit* and $q_m=32$ for the others. The additional parameters of the new method are set to their default values. The best results are in bold face letters.

presmoothing of the collected pixel values before subsampling. By these changes, the new 1-D grey/colour data contain less fluctuations. Thus, it can be compressed in a more efficient manner by the entropy coder. Furthermore, it allows better reconstructions when subsampled pixel values are recovered by linear interpolation.

Table 3 also proves that our new codec has real-time capabilities, not only due to *LPAQ2*, but in particular due to the fast multigrid implementation, which was presented in Section 5.

6.2. Comparison with JPEG and JPEG2000

Next we compare our codec with the well-established *JPEG* standard and with the more advanced *JPEG2000* codec. To this end, we use the encoders provided by the image processing tool *imagemagick*.[9] We fix all parameters of our codec to their default values. As edge images we use again the results that are depicted in Fig. 8.

Fig. 9 shows the different test images and their compressed versions using *JPEG*, *JPEG2000* and our codec. A cropped detail for each image is depicted in Fig. 10. The compression rates for all images lie between 0.16 and 0.37 bits per pixel (bpp). This corresponds to compression ratios between roughly 145:1 and 65:1, provided the original colour images use 3 bytes per pixel.

[9] See http://www.imagemagick.org.

Beside this visual comparison, Table 4 gives the corresponding quantitative comparison. It allows an evaluation of the compression capabilities for different methods by means of the PSNR. Furthermore, it depicts the encoding and decoding times for all images.

The quantitative and the visual analysis illustrate that our approach can be better than *JPEG* and even *JPEG2000*. For the test image *comic* we observe a remarkable difference of more than 3 dB between *JPEG2000* and our approach. Also visually, crucial differences become apparent (see Fig. 10). *JPEG* as well as *JPEG2000* suffer from severe ringing artifacts. These are a consequence of their quantisation step in the corresponding frequency/wavelet domains and the following inverse transforms. Moreover, *JPEG* applies the cosine transform in 8×8 blocks and thus suffers from unpleasant block artifacts. Especially edges are highly distorted. In contrast, our method stores edges explicitly and gives clean reconstructions. In addition, *JPEG* is not able to preserve the smooth gradient in the background of *svalbard* or *comic*. Our codec interpolates between the quantised colours so that the smooth gradient can be reconstructed almost perfectly. Thereby also for a greatly reduced grey/colour range, quantisation artifacts are hardly visible. Another drawback of the transform-based approaches can be discovered in the cropped detail image of *svalbard*. The screw in the bear completely vanishes for both *JPEG* as well as *JPEG2000*. Our approach stores the edges of such details so that they are well preserved.

The previous results have been obtained with the default settings of our codec. By changing the parameters, we can influence the compression rate and quality of an image. Most obviously, we can get higher compression rates when larger sampling distances or less quantisation intervals are chosen. Fig. 11 demonstrates what is possible if this is carried to the extreme: we reduce the number of edge pixels by choosing $T_1=15$ instead of 3.5. For each channel only six different values are used. The sampling distance along the edges is set to 40 and for the recursive search along edges $d_{\mathrm{tr}}=10$.

Using the lowest quality parameter provided by *imagemagick*, *JPEG* reaches only a compression ratio of 200:1. Its visual and quantitative quality is already far below the PDE-based result. Moreover, our codec exhibits a compression ratio of 340:1. For *JPEG2000 imagemagick* is able to provide a compression ratio of 340:1. Showing a PSNR of 22.83 dB, the quality of this image is comparable to the *JPEG* result. Considering the extreme compression rate, the PSNR of the PDE-based result (27.25 dB) is quite abundantly.

As we have seen, the compression rate can also be influenced by the underlying edge image. By choosing $T_1=15$ instead of $T_1=3.5$ we

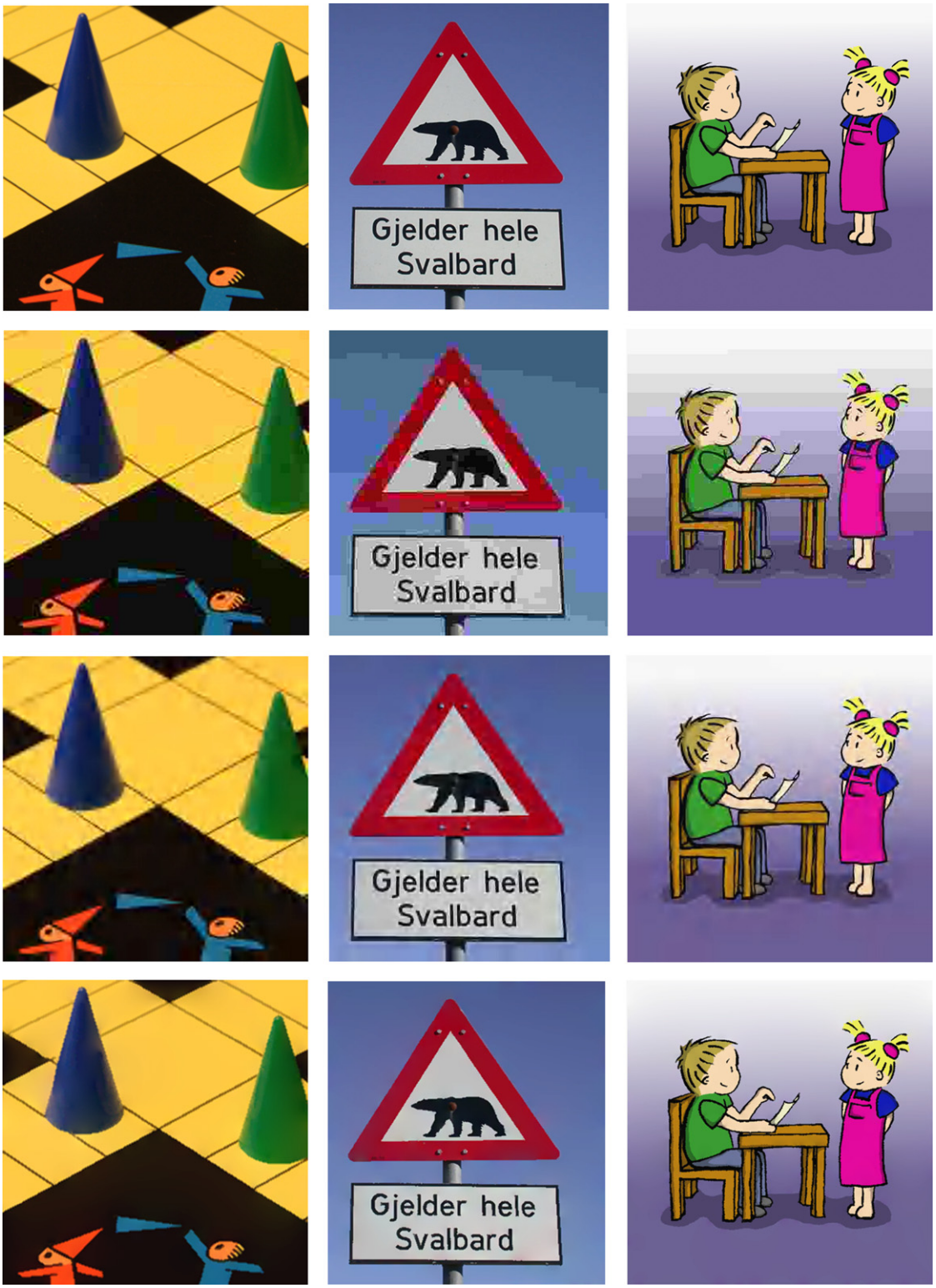

Fig. 9. Comparison of compression methods for different test images at 0.37, 0.16 and 0.19 bits per pixel (bpp) (*from left to right*). *Columns from left to right: coppit* (256×256), *svalbard* (380×431), *comic* (512×512). *Rows from top to bottom:* original image, *JPEG*, *JPEG2000*, and our codec with default parameters and edge images as depicted in Fig. 8.

removed some edge pixels enabling a higher compression rate. Note that these edge pixels should usually not be removed in order to guarantee quality. For cartoon-like images the corresponding edge image is up to small variations more or less unique.

Finally the compression rate can be increased by using *PAQ6o8* instead of *LPAQ2* at the expense of run time. Vice versa, a faster compression with less compression is obtained when *LPAQ2* is replaced for example by *bzip2*.

Fig. 10. Cropped detail (64×64) for each image depicted in Fig. 9.

6.3. Limitations

At the end of this section we briefly mention the limitations of our codec.

Obviously, our method is not well suited for images that contain textured areas. Fig. 12 gives an example where our method is not competitive to conventional compression methods anymore. We detect so many texture edges that our method

requires too much storage to get results of reasonable quality. For this example and default settings our codec gives a compression rate of 0.54 bpp and a PSNR of 30.16 dB. At the same compression rate *JPEG* leads to a PSNR of 36.02 dB and *JPEG2000* to a PSNR of 37.24 dB.

In this sense, our codec has been optimised for cartoon-like images. Those, however, can be compressed in good quality and with high compression ratios.

Another limitation is the low robustness of our decompression algorithm against corrupted files which typically result from transmission channels being exposed to noise. Since pixel locations and colours are encoded separately, errors in either part might result in a wrong assignment of colours to mask points, or to pixels being reconstructed at arbitrary locations in the image. However, such problems can already be handled in the transmission of the file: modern digital communication channels are typically equipped with fast error correction algorithms and retransmission mechanisms that assure file integrity.

Table 4
Comparison of the PSNR and elapsed encoding/decoding time for the different test images (see Fig. 9) and different compression methods.

Image	*coppit*	*svalbard*	*comic*
Compression rate (bpp)	0.37	0.16	0.19
Error measure (PSNR in dB)			
JPEG	26.61	23.38	24.25
JPEG2000	28.13	27.68	26.77
Our method	**30.31**	**30.14**	**30.20**
Encoding/decoding time in ms			
JPEG	**8.66**/7.14	**13.80**/13.35	**19.34**/24.20
JPEG2000	57.51/28.50	93.47/49.39	164.42/74.20
Our method	117.08/172.78	146.79/316.94	212.55/458.40

The best results are in bold face letters.

7. Conclusion

In this article we have presented a conceptually simple, but highly efficient way to compress cartoon-like images. By extracting edges and adjacent pixel values, encoding them efficiently and using homogeneous diffusion for reconstruction, we have created a new codec, which can even beat *JPEG2000*. This was out of reach for the previous methods, even after almost three decades of intensive research on image reconstruction from zero-crossings of the Laplacian.

Our results clearly indicate that cartoon-like images need a specialised treatment as offered by our codec. By storing edges explicitly, small details and sharp discontinuities are well preserved.

Moreover, we have analysed the discrete theory behind inpainting with homogeneous diffusion and have provided a fast multigrid algorithm for solving the inpainting problem. Thereby our codec is not only able to encode but also to decode images in real time.

In our ongoing research we are interested in implementations for distributed architectures. Furthermore, we are investigating extension that allow real-time video compression for animated cartoon movies.

Fig. 11. Compression methods driven to the extreme; (a) original image; (b) *JPEG* with minimal quality parameter at 0.12 bpp (i.e. approx. 200:1) with PSNR 21.68 dB, (c) *JPEG2000* at 0.07 bpp (i.e. approx. 340:1) with PSNR 22.83 dB, (d) our codec with zero-crossings based edge detector ($T_{1=15}$, $T_2=20$, $\sigma=0.6$) and default settings except for $q_m=6$, $d_m=40$ ($m=1\ldots3$) and $d_{\mathrm{tr}}=10$ gives 0.07 bpp (i.e. approx. 340:1) with PSNR 27.37 dB.

Fig. 12. (a) Test image *trui* (256 × 256). (b) Edge image, obtained by zero-crossings-based edge detector ($T_1=1.1$, $T_2=2.6$, $\sigma=1.2$). (c) Reconstructed colours adjacent to the edges. (d) Inpainting result (PSNR of 30.16 dB); compression rate of encoded image: 0.54 bpp.

Acknowledgement

We thank Anna Mainberger for providing the test image *comic*.

References

[1] M. Kunt, A. Ikonomopoulos, M. Kocher, Second-generation image-coding techniques, Proceedings of the IEEE 73 (1985) 549–574.
[2] M.M. Reid, R.J. Millar, N.D. Black, Second-generation image coding: an overview, ACM Computing Surveys 29 (1997) 3–29.
[3] W.B. Pennebaker, J.L. Mitchell, JPEG: Still Image Data Compression Standard, Springer, New York, 1992.
[4] D.S. Taubman, M.W. Marcellin (Eds.), JPEG 2000: Image Compression Fundamentals, Standards and Practice, Kluwer, Boston, 2002.
[5] B.F. Logan Jr., Information in the zero crossings of bandpass signals, Bell System Technical Journal 56 (1977) 487–510.
[6] A.L. Yuille, T.A. Poggio, Scaling theorems for zero crossings, IEEE Transactions on Pattern Analysis and Machine Intelligence 8 (1986) 15–25.
[7] S.R. Curtis, A.V. Oppenheim, J.S. Lim, Reconstruction of two-dimensional signals from threshold crossings, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 10, Tampa, FL, pp. 1057–1060.
[8] Y. Zeevi, D. Rotem, Image reconstruction from zero-crossings, IEEE Transactions on Acoustics, Speech, and Signal Processing 34 (1986) 1269–1277.
[9] S. Chen, Image reconstruction from zero-crossings, in: Proceedings of the Eighth International Joint Conference on Artificial Intelligence, vol. 2, Milan, Italy, 1987, pp. 742–744.
[10] S. Carlsson, Sketch based coding of grey level images, Signal Processing 15 (1988) 57–83.
[11] R. Hummel, R. Moniot, Reconstructions from zero-crossings in scale space, IEEE Transactions on Acoustics, Speech, and Signal Processing 37 (1989) 2111–2130.
[12] P. Grattoni, A. Guiducci, Contour coding for image description, Pattern Recognition Letters 11 (1990) 95–105.
[13] S. Mallat, S. Zhong, Characterisation of signals from multiscale edges, IEEE Transactions on Pattern Analysis and Machine Intelligence 14 (1992) 720–732.
[14] L. Dron, The multiscale veto model: a two-stage analog network for edge detection and image reconstruction, International Journal of Computer Vision 11 (1993) 45–61.
[15] C. Saloma, P. Haeberli, Two-dimensional image reconstruction from Fourier coefficients computed directly from zero crossings, Applied Optics 32 (1993) 3092–3093.
[16] J.H. Elder, Are edges incomplete? International Journal of Computer Vision 34 (1999) 97–122.
[17] D. Liu, X. Sun, F. Wu, S. Li, Y.-Q. Zhang, Image compression with edge-based inpainting, IEEE Transactions on Circuits, Systems and Video Technology 17 (2007) 1273–1286.
[18] S.D. Rane, G. Sapiro, M. Bertalmio, Structure and texture filling-in of missing image blocks in wireless transmission and compression applications, IEEE Transactions on Image Processing 12 (2003) 296–302.
[19] Z.W. Xiong, X.Y. Sun, F. Wu, S.P. Li, Image coding with parameter-assistant inpainting, in: Proceedings of the 2007 IEEE International Conference on Image Processing, vol. 2, San Antonio, TX, pp. 369–372.
[20] I. Galić, J. Weickert, M. Welk, A. Bruhn, A. Belyaev, H.-P. Seidel, Towards PDE-based image compression, in: N. Paragios, O. Faugeras, T. Chan, C. Schnörr (Eds.), Variational, Geometric and Level Set Methods in Computer Vision, Lecture Notes in Computer Science, vol. 3752, Springer, Berlin2005, pp. 37–48.
[21] I. Galić, J. Weickert, M. Welk, A. Bruhn, A. Belyaev, H.-P. Seidel, Image compression with anisotropic diffusion, Journal of Mathematical Imaging and Vision 31 (2008) 255–269.
[22] C. Schmaltz, J. Weickert, A. Bruhn, Beating the quality of JPEG 2000 with anisotropic diffusion, in: J. Denzler, G. Notni, H. Süße (Eds.), Pattern Recognition, Lecture Notes in Computer Science, vol. 5748, Springer, Berlin2009, pp. 452–461.
[23] H. Köstler, M. Stürmer, C. Freundl, U. Rüde, PDE based video compression in real time, Technical Report 07-11, Lehrstuhl für Informatik 10, Univ. Erlangen-Nürnberg, Germany, 2007.
[24] Z. Belhachmi, D. Bucur, B. Burgeth, J. Weickert, How to choose interpolation data in images, SIAM Journal on Applied Mathematics 70 (2009) 333–352.
[25] M. Mainberger, J. Weickert, Edge-based image compression with homogeneous diffusion, in: X. Jiang, N. Petkov (Eds.), Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns, Lecture Notes in Computer Science, vol. 5702, Springer, Berlin2009, pp. 476–483.
[26] D. Marr, E. Hildreth, Theory of edge detection, Proceedings of the Royal Society of London, Series B 207 (1980) 187–217.
[27] J. Canny, A computational approach to edge detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 8 (1986) 679–698.
[28] J. Fairfield, Toboggan contrast enhancement for contrast segmentation, Proceedings of the 10th International Conference on Pattern Recognition, vol. 1, IEEE Computer Society Press, Atlantic City, NJ1990, pp. 712–716.
[29] Joint Bi-level Image Experts Group, Information technology—progressive lossy/lossless coding of bi-level images, ISO/IEC JTC1 11544, ITU-T Rec. T.82, Final Committee Draft 11544, 1993.
[30] P.G. Howard, F. Kossentini, B. Martins, S. Forchhammer, W.J. Rucklidge, The emerging JBIG2 standard, IEEE Transactions on Circuits and Systems for Video Technology 8 (1998) 838–848.
[31] Joint Bi-level Image Experts Group, Information technology–lossy/lossless coding of bi-level images, ISO/IEC 14492, ITU-T Rec. T.88, Final Committee Draft 14492, 1999.
[32] L. Bottou, P. Haffner, P.G. Howard, P. Simard, Y. Bengio, Y. Le Cun, High quality document image compression with DjVu, Journal of Electronic Imaging 7 (1998) 410–425.
[33] M. Kuhn, Effiziente Kompression von bi-level Bilddaten durch kontextsensitive arithmetische Codierung, Studienarbeit, Institut für Mathematische Maschinen und Datenverarbeitung, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany, 1995.
[34] J. Max, Quantisation for minimum distortion, IEEE Transactions on Information Theory 6 (1960) 7–12.
[35] S. Lloyd, Least squares quantization in PCM, IEEE Transactions on Information Theory 28 (1982) 129–137.
[36] D.A. Huffman, A method for the construction of minimum redundancy codes, Proceedings of the IRE 40 (1952) 1098–1101.
[37] J.J. Rissanen, Generalized Kraft inequality and arithmetic coding, IBM Journal of Research and Development 20 (1977) 198–203.
[38] M. Mahoney, Adaptive weighing of context models for lossless data compression, Technical Report CS-2005-16, Florida Institute of Technology, Melbourne, FL, 2005.
[39] A. Moffat, Implementing the PPM data compression scheme, IEEE Transactions on Communications 38 (1990) 1917–1921.
[40] S. Masnou, J.-M. Morel, Level lines based disocclusion, in: Proceedings of the 1998 IEEE International Conference on Image Processing, vol. 3, Chicago, IL, pp. 259–263.
[41] H. Werner, Studies on contour, The American Journal of Psychology 47 (1935) 40–64.
[42] T. Iijima, Basic theory on normalization of pattern (in case of typical one-dimensional pattern), Bulletin of the Electrotechnical Laboratory 26 (1962) 368–388 (in Japanese).
[43] A.R. Mitchell, D.F. Griffiths, The Finite Difference Method in Partial Differential Equations, Wiley, Chichester, 1980.
[44] G. Evans, J. Blackledge, P. Yardley, Numerical Methods for Partial Differential Equations, Springer, Berlin, 1999.
[45] D.S. Feingold, R.S. Varga, Block diagonally dominant matrices and generalizations of the Gerschgorin circle theorem, Pacific Journal of Mathematics 12 (1962) 1241–1250.
[46] A. Brandt, Multi-level adaptive solutions to boundary-value problems, Mathematics of Computation 31 (1977) 333–390.
[47] W. Hackbusch, Multigrid Methods and Applications, Springer, New York, 1985.
[48] D.M. Young, Iterative Solution of Large Linear Systems, Academic Press, New York, 1971.
[49] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, Philadelphia, 2003.
[50] W.L. Briggs, V.E. Henson, S.F. McCormick, A Multigrid Tutorial, second ed., SIAM, Philadelphia, 2000.
[51] U. Trottenberg, C. Oosterlee, A. Schüller, Multigrid, Academic Press, San Diego, 2001.
[52] P. Wesseling, An Introduction to Multigrid Methods, R. T. Edwards, Flourtown, 2004.
[53] H.R. Schwarz, Numerische Mathematik, fourth ed., Teubner, Stuttgart, 1997.
[54] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, C. Schnörr, Variational optic flow computation in real-time, IEEE Transactions on Image Processing 14 (2005) 608–615.
[55] H. Knutsson, C.-F. Westin, Normalized and differential convolution: methods for interpolation and filtering of incomplete and uncertain data, in: Proceedings of the 1993 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Computer Society Press, New York, NY1993, pp. 515–523.

Markus Mainberger received a B.Sc. degree in Computer Science and an M.Sc. degree in Visual Computing from Saarland University in 2007 and 2008, respectively. He joined the Mathematical Image Analysis Group as a Ph.D. student in October 2008. His scientific interests focus on PDE-based image compression.

Andrés Bruhn received a diploma degree in Computer Engineering from the University of Mannheim in 2001 and a Ph.D. degree in Computer Science from Saarland University in 2006. After working as a postdoctoral researcher from 2006 to 2007, he currently holds the position of an assistant professor at the Mathematical Image

Analysis Group at Saarland University. His research interests include optical flow estimation, stereo reconstruction, diffusion filtering and fast numerical schemes for image processing and computer vision.

Joachim Weickert is professor of Mathematics and Computer Science at Saarland University where he heads the Mathematical Image Analysis Group. He received a diploma and a Ph.D. degree in Mathematics from the University of Kaiserslautern (1991, 1996), and a habilitation degree in computer science from the University of Mannheim (2001). He worked as research assistant at the University of Kaiserslautern, as post-doctoral researcher at the universities of Utrecht and Copenhagen, and as assistant professor at the University of Mannheim. He performs research in image processing, computer vision and scientific computing.

Søren Forchhammer received the M.Sc. and Ph.D. degrees from the Technical University of Denmark (DTU) in 1984 and 1988, respectively. Since 1992 he has been an associate professor at DTU, currently as head of Coding and Visual Communication at DTU Fotonik. His interests include source coding, image and video coding, visual communication and two-dimensional information theory.