Journal of Electronic Imaging 11(2), 236–245 (April 2002).

Using Hough transform for context-based image compression in hybrid raster/vector applications

Pasi Fränti
Eugene Ageenko
University of Joensuu
Department of Computer Science
P.O. Box 111
FIN-80101 Joensuu, Finland
E-mail: franti@cs.joensuu.fi

Saku Kukkonen
Heikki Kälviäinen
Lappeenranta University of Technology
Department of Information Technology
P.O. Box 20
FIN-53851 Lappeenranta, Finland

Abstract. *In a hybrid raster/vector system, two representations of the image are stored. Digitized raster image preserves the original drawing in its exact visual form, whereas additional vector data can be used for resolution-independent reproduction, image editing, analysis, and indexing operations. We introduce two techniques for utilizing the vector features in context-based compression of the raster image. In both techniques, Hough transform is used for extracting the line features from the raster image. The first technique uses a feature-based filter for removing noise near the borders of the extracted line elements. This improves the image quality and results in more compressible raster image. The second technique utilizes the line features to improve the prediction accuracy in the context modeling. In both cases, we achieve better compression performance.* © *2002 SPIE and IS&T.* [DOI: 10.1117/1.1455014]

1 Introduction

In a hybrid raster/vector storage system, both raster and vector representations of the images are encoded and stored,[1,2] see Fig. 1. The raster representation provides an exact digitized replica of the original image. The vector representation contains semantic information extracted from the image. It benefits from vector editing capabilities and is suitable for further image processing and semantic analysis.[3,4] The compressed file consists of the extracted line features and the compressed raster image.

The advantage of raster representation is that the images can be easily digitized and stored compactly using latest compression technology. Reproduction of the image is easy, and lossless compression guarantees that an exact replica of the original image can always be restored. Vector representation, on the other hand, allows better editing capabilities and resolution independent scaling and reproduction. Complete raster-to-vector conversion, however, is not a realistic solution because the conversion systems are of high complexity and they cannot capture all possible vector features reliably without human interaction. Either the file will be filled by huge number of small vector elements, or some of the undetected information will be lost.

We consider the storage problem of hybrid raster/vector systems. In an ideal situation, all background features would be in raster format and all line features in vector format. In practice, hybrid raster/vector representation means that a lot of new data will be stored in the vector format without any saving in the storage of the raster image. The raster image itself can be stored compactly using context-based image compression.[1] The question is that how utilize the existence of the vector features for compressing the raster image more efficiently.

In this paper, we introduce two novel techniques for utilizing the vector features in context-based image compression of the raster image. In both techniques, we use Hough transform[5,6] for extracting the line features from the raster image. Hough transform can be applied to arbitrary parametrized shape but only line features are general enough to model most of the images in practice.

The first technique uses a feature-based filter for removing the noise near the borders of the extracted line *elements*. This improves the image quality and results in a more compressible raster image. The line features are used only in the compression phase and therefore they need not be stored in the compressed file. The filtering is based on a simple noise removal procedure using a mismatch image

Paper 20012 received Feb. 14, 2000; revised manuscript received July 9, 2001; accepted for publication Nov. 20, 2001.
1017-9909/2002/$15.00 © 2002 SPIE and IS&T.

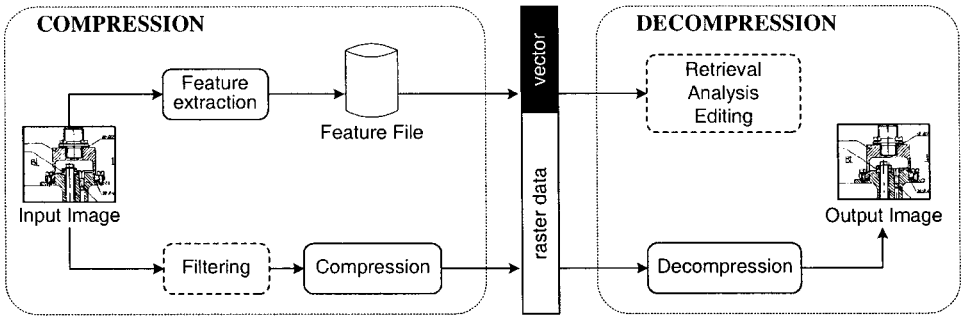


Fig. 1 Hybrid raster/vector storage system.

between the original and the feature image. Isolated mismatch pixels (and groups of two mismatch pixels defined by 8-connectivity) are detected and the corresponding pixels in the original image are reversed. The method is near-lossless because the amount of changes is controlled—only isolated noise pixels are reversed.

The second technique utilizes the line features to improve the prediction accuracy in the context modeling. The compressed file consists of the extracted line features and the compressed raster image. The vectorizing should be good both in terms of quality and compactness. Low quality vector elements causes that less information can be filtered out from the raster image. A large number of small elements, on the other hand, increases the size of the vector file and typically requires more space than the corresponding part of the raster image.

The rest of the paper is organized as follows. Previous work on context-based compression is summarized in Sec. 2. The Hough transform (HT)-based feature extraction is described in Sec. 3. Two methods for utilizing the feature information are then introduced in Secs. 4 and 5. Experiments of the proposed methods are reported in Sec. 6, and conclusions are drawn in Sec. 7.

2 Context-Based Compression

Binary images are favorable source for context-based image compression because of the spatial dependencies between neighboring pixels.[7,8] In context-based compression, the pixels are coded on the basis of their probability conditioned on the *context*. The context is determined by the combination of the color values of already coded neighboring pixels within the local template.

JBIG is the current international standard for compression of the bilevel images in communications.[9,10] In JBIG, the image is coded by default in raster scan order using context-based probability modeling and arithmetic coding, namely the *QM-coder*.[11] The emerging standard JBIG2[12–14] improves the compression of text images using pattern matching technique for extracting symbols from the image. This enhancement, however, is of limited usage in the case of line-drawing images, as they do not contain large number of text elements.

The context modeling of JBIG can be improved using variable-size context template.[15,16] The contexts are stored in the leaves of a variable-depth binary tree, referred as *context tree*. The use of variable-size context model enables selective context expansion and utilizes larger context templates without overwhelming the learning cost.

Another way to improve compression is to filter the image for noise removal. Filtering reduces irregularities in the image caused by noise, and in this way, makes the image more compressible without degrading the image quality. Noise appears in the image as randomly scattered noise pixels (additive noise), and as content-dependent noise distorting the contours of printed objects (lines, characters) by making them ragged. The noise level may be low enough not to significantly detract from the subjective quality but it introduces unnecessary details that decrease the compression performance.

Several methods have been considered in literature for image preprocessing before the compression.[17–20] These filtering methods work by analyzing local pixel neighborhood defined by a filtering template and include logical smoothing, variations of median filtering, isolated pixel removal, and morphological filters.[21] All of these use a set of rules to accept or reject the pixel, such as predefined masks or quantitative description of the local neighboring area. Recent research in mathematical morphology have shown that morphological filtering can be used as an efficient tool for pattern restoration in environment of heavy additive noise.[22–25] Such approaches, however, are not necessarily suitable for filtering content-dependent quantization noise. Another problem is that the filtering may destroy fine image structures carrying crucial information if the amount of filtering is not controlled.

3 Feature Extraction Using Hough Transform

Hough transform is used for extracting the vector features from the image,[5,26,27] as summarized in Fig. 2. The motivation is to find rigid fixed length straight lines in the image. The extracted line segments are represented as their end points. A feature image is reconstructed from the line segments and it is utilized in the compression phase. The extracted line segments are also stored in the compressed file.

3.1 *Hough Transform*

The lines are first detected by the HT as follows:

1. Create a set of coordinates from the black pixels in the image.
2. Transform each coordinate (x,y) into a parameterized curve in the parameter space.
3. Increment the cells in the parameter space determined by the parametric curve.

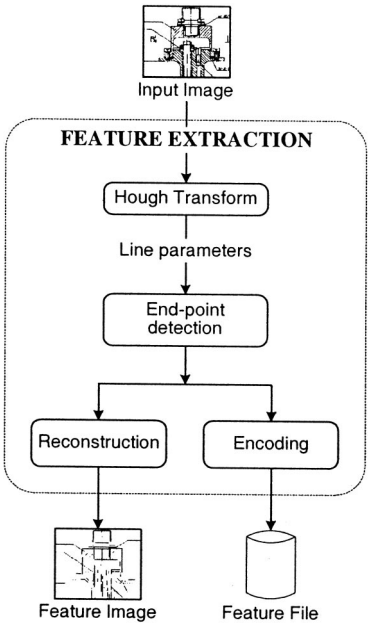


Fig. 2 Block diagram of the feature extraction.

4. Detect local maxima in the accumulator array. Each local maximum may correspond to a parametric curve in the image space.
5. Extract the curve segments using the knowledge of the maximum positions.

The parameter space is a $k \times k$ accumulator array where k can be tuned according to the image size, e.g., k=the size of the image. Usually, the (ρ, θ) parameterization is used but the (a,b) one, corresponding to $y = a \cdot x + b$, is also possible. The accumulation matrix is quantized with equal intervals.

3.2 *End-Point Detection*

The Hough transform is capable to determine the location of a line (as a linear function) but it cannot resolve the end points of the line. In fact, HT does not even guarantee that there exists any finite length line in the image but it only indicates that the pixels (x,y) along $y = a \cdot x + b$ may represent a line. The existence of a line segment must therefore be verified. The verification is performed by scanning the pixels along the line and checking whether they meet certain criteria. We use the scanning width, the minimum number of pixels, and the maximum gap between pixels in a line as the selection criteria. If predefined threshold values are met, a line segment is detected and its end points are stored for later use.

3.3 *Reconstruction of the Feature Image*

A feature image of equal size is created from the extracted line segments to approximate the input image. The image is constructed by drawing one-pixel width straight lines using the end points of the line features. The Hough transform does not determine the widths of the lines but wider lines are represented by a bunch of collinear line segments, see Fig. 3. The line segments may also be deviated from their original direction and/or have one-pixel positional error be-

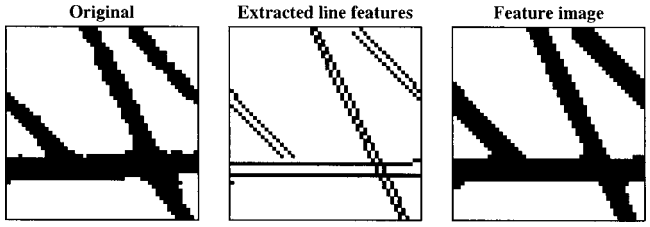


Fig. 3 Illustration of the feature image for an image sample of size 50×50 pixels.

cause of the quantization of the accumulation matrix. Therefore, we do not utilize the feature image directly but process it first by consequent operations of morphological *dilation* and *closing*.[23] These operations make the lines one pixel thicker in all directions (dilation) and fill gaps between the line segments (closing). We apply a symmetric 3×3 structure element (*block*) for the dilation, and a 3×3 cross structure element (*cross*) for the closing, see Fig. 4. The cross element is chosen to minimize the distortion in line intersections caused by closing.

3.4 *Storing the Line Segments*

The extracted line segments are stored as $\{(x_1,y_1),(x_2,y_2)\}$ representing the end points of the line. A single coordinate value takes $[\log_2 n]$ bits where n is the dimension of the image. For example, a line in an image of 4096×4096 pixels takes $4 \times 12 = 48$ bits in total. A more compact representation could be achieved if the line segments are sorted according to their first coordinate x_1. Instead of storing the absolute value, we could store the difference between two subsequent x_1's. Most of the differences are very small (about 40% of them are in the range 0–2). An improvement of about 7 bits (from 12 to 5 bits) was estimated when entropy coding was applied to these difference values. In the present implementation, this idea was not applied.

4 Feature-Based Filtering

We apply feature-dependent filtering for improving the image quality. The method can be used as a preprocessing step before the compression, as outlined in Fig. 5. The filtering removes noise by the restoration of the line contours and therefore it results in better compression performance. The line features are used only in the compression phase and therefore they need not be stored in the compressed file. The filtered image can then be compressed by the JBIG

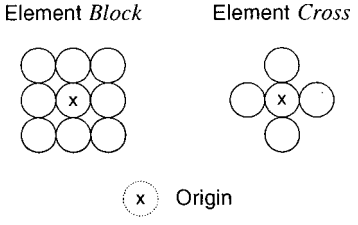


Fig. 4 Structure elements *block* and *cross*.

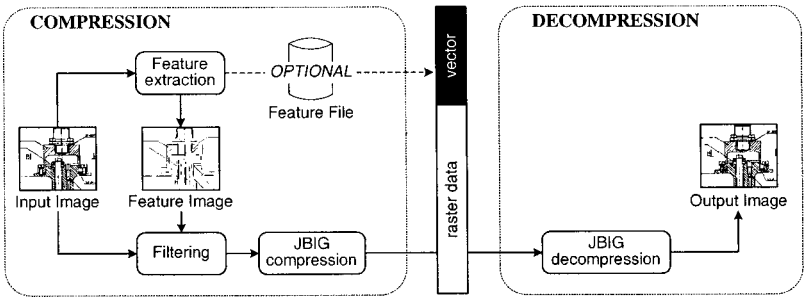


Fig. 5 Block diagram of the near-lossless compression system.

without any modifications. The method is denoted further as HTF–JBIG. Decompression is exactly the same as the JBIG.

The filtering is based on a simple noise removal procedure, as shown in Fig. 6. A difference (mismatch) image between the original and the feature image is constructed. Isolated mismatch pixels (and groups of two mismatch pixels defined by 8-connectivity) are detected and the corresponding pixels in the original image are reversed. This removes random noise and smoothes edges along the detected line segments. The method is near-lossless because the amount of changes is controlled—only isolated noise pixels are reversed. Undetected objects (such as text characters) are left untouched allowing their lossless reconstruction.

The noise removal procedure is successful if the feature image is accurate. However, the feature extraction of HT does not always provide exact width of the lines. The noise removal procedure is therefore iterated three times as shown in Fig. 7. The first stage applies the feature image as such, but the feature image is dilated in the second stage and eroded in the third stage before input into the noise removal procedure. This compensates inaccuracies in the width detection.

The stepwise process is illustrated in Fig. 8. Most of the noise is detected and removed in the first phase. However, the rightmost diagonal line in the feature image is too wide and its upper contour is therefore filtered only in the third stage where the feature image is eroded. The results of the entire filtering process are illustrated in Fig. 9. In these examples, pixel-level noise is mainly filtered out but some of the roughness remains along the lines. These consist of larger groups of noise pixels and therefore are not filtered by the method. Symbols and other nonlinear elements are not completely detected by Hough transform, and therefore parts of them may have not been processed.

It is noted that the noise removal procedure does not guarantee the retention of connectivity of the lines. It is therefore possible that very thin lines may be broken apart because of a pixel removal. Although the situation is rare, the retention of connectivity can be important in some application. In this case, an additional procedure must be applied to check whether pixel removal would break connectivity. For example, the method in Ref. 28 can be

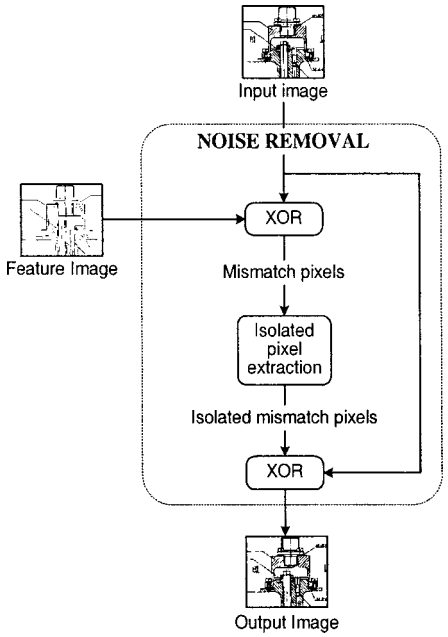


Fig. 6 Block diagram of the noise removal procedure.

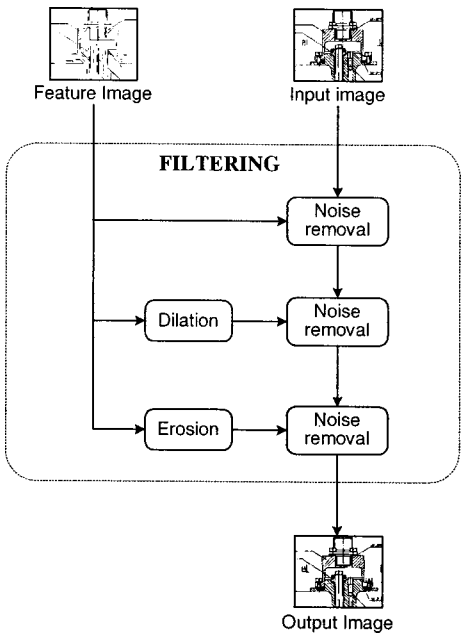


Fig. 7 Block diagram of the three-stage filtering procedure.

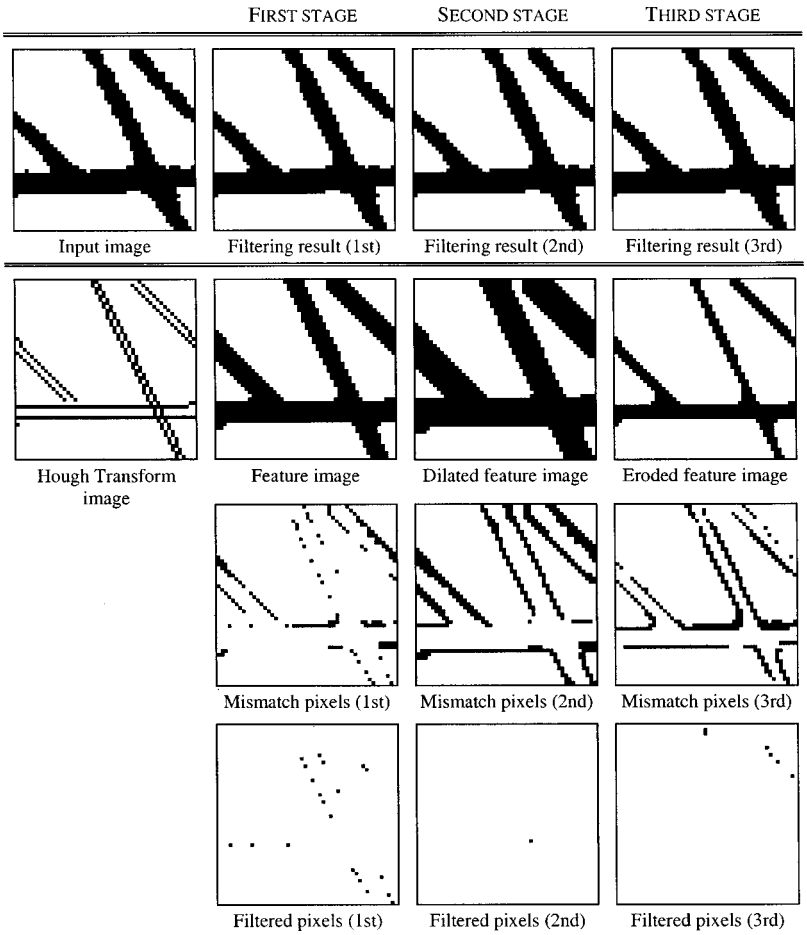


Fig. 8 Illustration of the three-state filtering procedure.

implemented by a simple look-up table consisting of the pixels within the 3×3 neighboring.

5 Feature-Based Context Modeling

If the features are stored in the compressed file, they can be used to improve the prediction accuracy in the context modeling. There are two basic approaches for utilizing the feature image: (1) lossless compression of the residual between the original and the feature image, (2) compression of the original image using the feature image as side information. The first approach does not work in practice because taking the residue destroys spatial dependencies near the borders of the extracted line features. The residual image is therefore not any easier to compress than the original one. On the other hand, the effectiveness of the second approach has been proven in practice in the case of text images.[10,12] We thus adopt the same idea here for line drawing images.

The compression method denoted further as HTC is outlined in Fig. 10. In HTC, the original image is compressed using the JBIG-like technique, based on *two-layer context modeling*. The idea of two-layer context modeling is to utilize information from an additional image, a feature image in our case. The difference in construction of two-layer and the standard single-layer context template is that an additional template is used for the selection of the pixels from the feature image. The idea is that any pixel of the feature image can be used in the context, whereas the selection of the pixels from the current image being compressed is limited only to the previously encoded pixels. This is because the line features are stored in the compressed file and the same information is therefore available also in the decompression.

We use a context template composed of ten pixels from the original image and five pixels from the feature image as illustrated in Fig. 11 (left). The example of Fig. 11 (right) illustrates how the additional information is utilized. In this case, we have no evidence of the vicinity of the line if we examine only the pixel colors inside the 10 pixel standard template. We would therefore assign an unnecessarily high probability for the white color. The additional information from the feature image, on the other hand, indicates the vicinity of a line. Although the feature image does not match pixel-by-pixel to the original image, it gives additional information so that more accurate statistical model can be constructed.

6 Test Results

The performance of the proposed methods is tested by compressing the set of test images shown in Fig. 12. Four different features sets were constructed from each image with different amount of line segments. The number of extracted lines was controlled by varying the parameters in the Hough transform, as shown in Fig. 13.

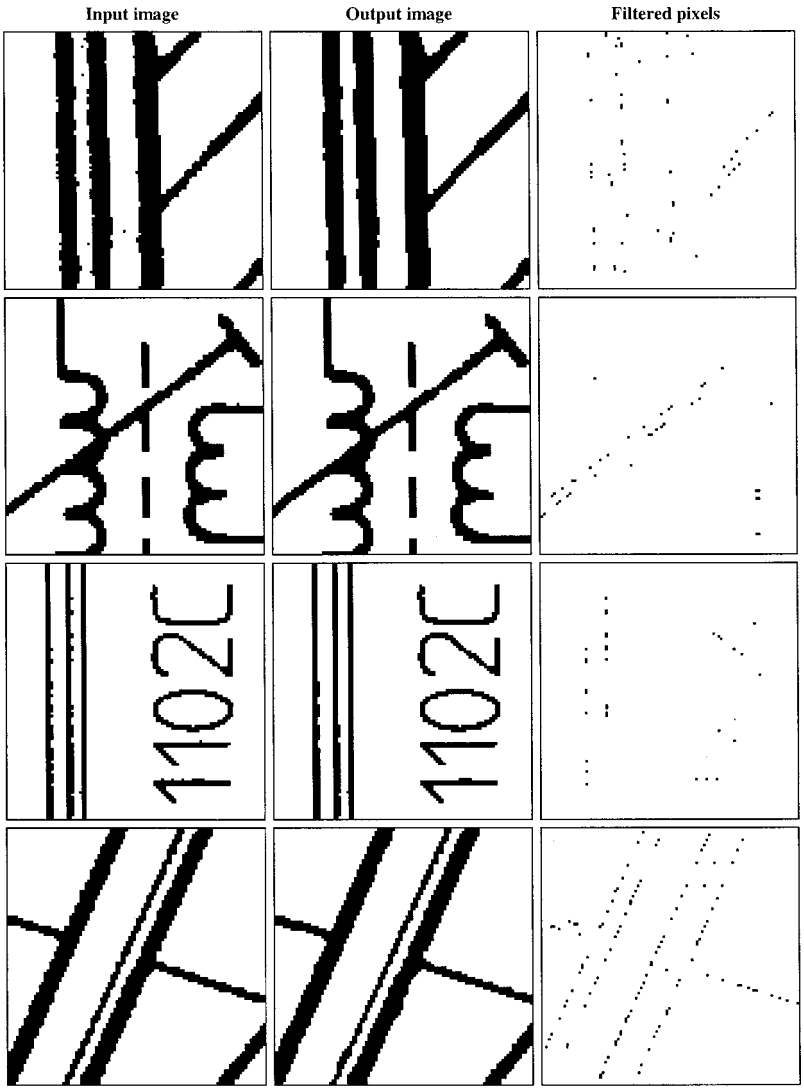


Fig. 9 Filtering examples from left to right: sample from the original image, from the filtered image, and their difference.

The effect of the feature-based context modeling on the file size is shown in Fig. 14. The feature-based context modeling improves the compression of the raster image of about 1%–10%, depending on the image and the number of extracted line elements. The amount of saving, however, is too small to compensate the overhead required by the feature file. For example, the vector data requires 6.3 kbyte for the image *bolt*, and the size of the raster data can be reduced from 12.7 to 11.2 kbyte. In total, the file takes 17.5 kbyte.

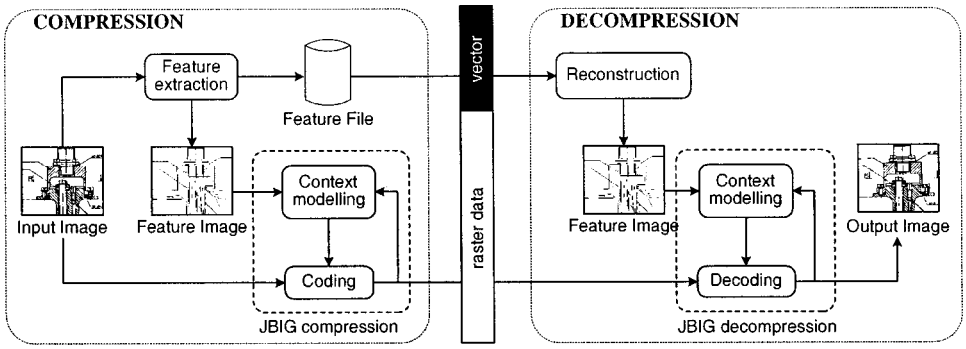


Fig. 10 Block diagram of the new hybrid compression system.

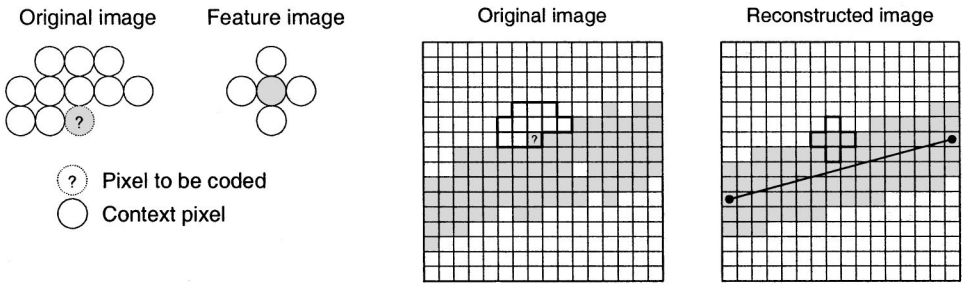


Fig. 11 Two-level context template (left), and an example of a situation, in which the information of the feature image is beneficial (right).

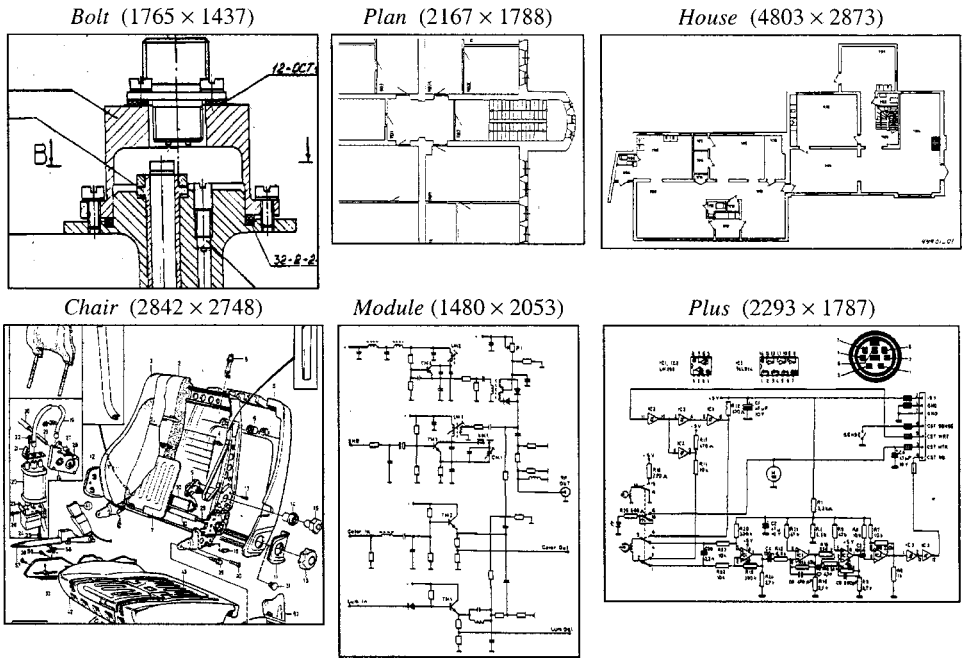
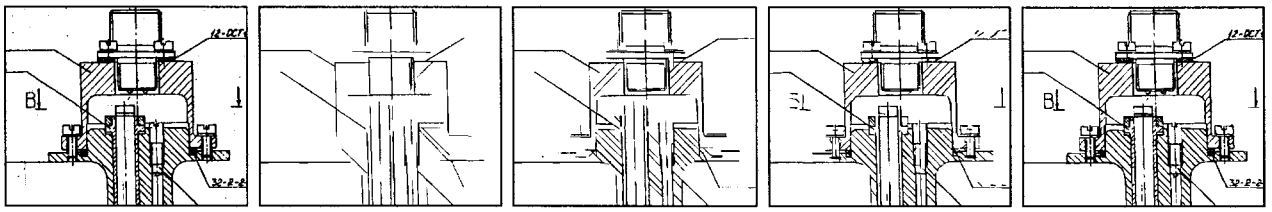


Fig. 12 Set of test images.



number of segments	117 line segments	289 line segments	752 line segments	1073 line segments
min segment length	150	70	30	15
max segment width	1	1	1	1
max length of single gap	2	2	3	2
accumulator threshold	20	20	17	10

Fig. 13 Example of the feature images from test image *bolt* made using different parameter setup.

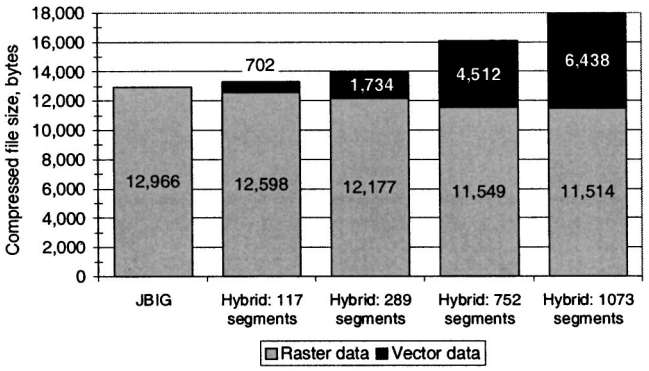


Fig. 14 Illustration of the compressed file sizes for the test image *bolt* with variable amount of extracted line elements.

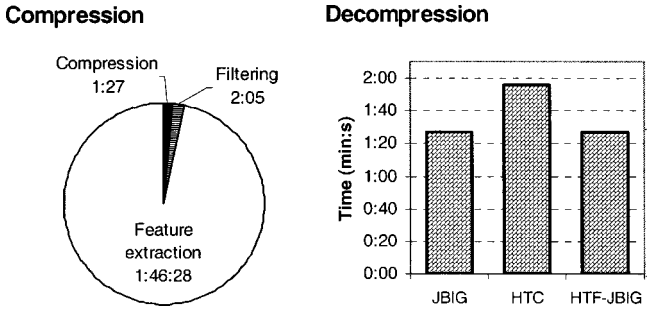


Fig. 15 Running times of the HT-based compression.

The second method (HTF–JBIG) applies feature-based filtering for noise removal and standard JBIG for image compression. In this case, the number of extracted line features does not affect the file size because the features are not stored. It is therefore better to use as many features as can be reliably detected. In our case, the set with most line segments gives the best results among the three tested sets. The method improves the compression performance of about 12% on average, in comparison to JBIG. For example, the image *bolt* requires 10.3 kbyte in comparison to 12.7 of JBIG, or 17.5 of the hybrid compression.

The storage sizes of the two proposed methods are summarized in Table 1. In a hybrid compression both raster and vector data are included in the compressed file. The column ''vector'' contain the storage size required by the vector features. The two columns ''raster'' refers to the two alternatives for compressing the raster image: using JBIG and using the proposed HTC method with feature-based context modeling. The table includes also results (last column) where the new techniques have both been utilized at the same time. In the case of image *bolt*, the size of the raster decreases to 9.1 kbyte but in this case, the vector data must also be stored. To sum up, the benefit of utilizing feature-based information in the context modeling and in the filtering is moderate at best, and cannot compensate the increase in the storage size caused by the inclusion vector features. The use of hybrid raster/vector file structure must therefore serve for other purposes, e.g., indexing or hybrid editing.

The running times of the proposed methods in total for the test set using Pentium-200 machine are shown in Fig. 15. The feature extraction dominates the running time in the compression phase and makes it an order of magnitude slower than JBIG. The method is therefore suitable only for applications where compression can be made off-line. Decompression of filtered images in HTF–JBIG is performed using the standard JBIG routines and therefore is as fast as JBIG. In the hybrid HTC compression, the decompression phase is about 35% slower because of the processing of the vector features.

7 Conclusions

Two methods are introduced for improving compression performance by utilizing global dependencies in hybrid raster/vector applications. The utilization of the extracted features is twofold: (1) we can improve context-modeling by using the stored line features; and (2) the features can be used for improving image quality in a preprocessing step prior to the compression.

In the first case, reconstructed feature image were used for improving prediction accuracy in the local context model. The compressed file consists of the extracted line features and the compressed raster image. The raster image is compressed 1%–10% more efficiently when using the feature image as side information. The improvement, however, is too small to compensate the overhead required by the feature file. The main reason for the small compression improvement is that the information of the extracted line features is mainly in all-black neighborhoods, inside the line segments. These are the pixels that are already compressed well by JBIG and therefore only small improvement can be achieved. On the contrary, most of the information (output bits) originates from the boundaries of the

Table 1 Summary of the storage sizes of the different methods (in bytes).

	Hybrid compression			Filtering only	Filtering + hybrid
Image	Vector	Raster (JBIG)	Raster (HTC)	(HTF–JBIG)	(HTF–HTC)
Bolt	6 438	12 966	11 514	10 536	9 287
Plan	2 370	5 097	4 578	4 325	3 786
House	13 398	15 681	13 961	13 336	11 553
Chair	16 710	52 708	50 140	51 529	48 023
Module	3 468	7 671	7 222	6 431	6 057
Plus	5 268	17 609	17 132	16 273	15 739
Total	47 652	111 732	104 547	102 430	94 445

objects. These areas are not well predicted by the local modeling of JBIG but global information could be useful, especially if the input image is noisy. This emphasizes the importance of the exactness of the feature extraction.

In the second case, the features are not stored and the feature extraction process is performed only for removing noise from the original image. It improves the image quality and, therefore, results also in better compression performance of about 12% compared to JBIG. The technique utilizes global spatial dependencies in the image unavailable to traditional filtering methods and enables efficient removal of content-dependent noise. At the same time, the quality of the decompressed images is visually the same (or even better) because the reversed pixels are mainly random noise or scanning noise near the line segments. The advantage of the method is that it utilizes any given (even nonperfect) feature information as such, and overcomes the problems of inaccuracies in the line detection.

A drawback of using Hough transform for feature extraction is the high complexity. A straightforward implementation of HT requires $0(kn)$ time, where n is the image size and $k \times k$ is the size of the accumulator matrix. The decompression phase, on the other hand, is identical to the JBIG and therefore equally fast. Thus, the method is suitable for off-line applications (like image archival) where the images are compressed only once but decompressed often. Moreover, HT can also be made significantly faster using the randomized Hough transform (RHT).[6] Instead of processing individual pixels, the image is randomly sampled by selecting pairs of pixel. Each pair determines only one value in the parameter space. The sampling is repeated until an evident maximum is emphasized in the parameter space. RHT reduces the size of the parameter array and it decreases the computation time since only a part of the pixels, possibly a small part, is needed to be transferred into the array. The compression performance of using RHT is only about 1% worse than that of using HT.

Overall, the benefit of utilizing feature-based information in the context modeling and in the filtering was moderate at best, and cannot compensate the increase in the storage size caused by the inclusion vector features. We therefore conclude that we must either give up the requirement of preserving exact replica of the original image, or improve the quality of the vectorizing drastically if the we want to store the hybrid file structure as efficiently as the original raster image. The vectorizing should also be good both in terms of quality and compactness. Low quality vector elements causes that less information can be filtered out from the raster image. A large number of small elements, on the other hand, increases the size of the vector file and typically requires more space than the corresponding part of the raster image.

To sum up, the purpose of storing the features must be different from the compression. For example, the features can be used in content-based image matching as proposed in Ref. 29. Otherwise, the features should be used only for image enhancement prior to compression, and the features not saved in the compressed file.

References

1. D. J. Wilson, S. E. A. Your paper. Scan Edit Archieve Initiative, White paper (1999). http://www.sea-initiative.org/
2. P. Fränti, E. I. Ageenko, H. Kälviäinen, and S. Kukkonen, "Compression of line drawing images using Hough transform for exploiting global dependencies," *Proc. 4th Joint Conf. on Information Sciences (JCIS'98)*, pp. 433–436, RTP, USA, IV (1998).
3. R. Kasturi *et al.*, "A system for interpretation of line drawings," *IEEE Trans. Pattern Anal. Mach. Intell.* **12**(10) 978–992 (1990).
4. R. Kasturi and L. O'Gourman, "Document image analysis: A bibliography," *Mach. Vision Appl.* **5**, 231–243 (1992).
5. P. C. Hough, "Methods and means for recognizing complex patterns," U. S. Patent No. 3,069,654 (1962).
6. H. Kälviäinen, P. Hirvonen, L. Xu, and E. Oja, "Probabililistic, non-probabilistic Hough transforms: overview and comparisons," *Image Vis. Comput.* **13**, 239–251 (1995).
7. G. G. Langdon and J. Rissanen, "Compression of black-white images with arithmetic coding," *IEEE Trans. Commun.* **29**(6), 858–867 (1981).
8. P. E. Tisher, R. T. Worley, A. J. Maeder, and M. Goodwin, "Context-based lossless image compression," *Comput. J. (UK)* **36**, 68–77 (1993).
9. ISO/IEC International Standard 11544, ISO/IEC/JTC1/SC29/WG9; also ITU-T Recommendation T.82. Progressive Bi-level Image Compression, 1993.
10. I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York (1994).
11. W. B. Pennebaker and J. L. Mitchell, "Probability estimation for the Q-coder," *IBM J. Res. Dev.* **32–6**, 737–759 (1998).
12. P. G. Howard, "Text image compression using soft pattern matching," *Comput. J. (UK)* **40**, 146–156 (1997).
13. P. G. Howard, F. Kossentini, B. Martins, S. Forchammer, and W. J. Rucklidge, "The emerging JBIG standard," *IEEE Trans. Circuits Syst. Video Technol.* **8**(7), 838–848 (1998).
14. Final Committee Draft for ISO/IEC International Standard 14492, 1999. http://www.jpeg.org/public/jbigpt2.htm
15. P. Fränti and E. I. Ageenko, "On the use of context tree for binary image compression," *IEEE Proc. Int. Conf. on Image Processing (ICIP'99)*, Vol. 3, pp. 752–756, Kobe, Japan, (1999).
16. B. Martins and S. Forchhammer, "Bi-level image compression with tree coding," *IEEE Trans. Image Process.* **7**(4), 517–528 (1998).
17. D. Ting and B. Prasada, "Digital processing techniques for encoding of graphics," *Proc. IEEE* **68**(7), 757–769 (1980).
18. R. Bernstein, "Adaptive nonlinear filters for simultaneous removal of different kinds of noise in images," *IEEE Trans. Circuits Syst.* **CAS–34**(11), 1275–1291 (1987).
19. V. R. Algazi, P. L. Kelly, and R. R. Estes, "Compression of binary facsimile images by preprocessing and color shrinking," *IEEE Trans. Commun.* **38**(9), 1592–1598 (1990).
20. Q. Zhang and J. M. Danskin, "Bitmap reconstruction for document image compression," *Proc. SPIE* **2916**, 188–199 (1996).
21. J. Serra, *Image Analysis and Mathematical Morphology*, Academic, London (1982).
22. D. Schonfeld and J. Goutsias, "Optimal morphological pattern restoration from noisy binary images," *IEEE Trans. Pattern Anal. Mach. Intell.* **13**(1), 14–29 (1991).
23. H. J. A. M. Heijmans, *Morphological Image Operators*, Academic Boston (1994).
24. L. Koskinen and J. Astola, "Soft morphological filters: a robust morphological filtering method," *J. Electron. Imaging* **3**, 60–70 (1994).
25. *Nonlinear Filters for Image Processing*, E. R. Dougherty and J. Astola, Eds., SPIE, Bellingham, WA (1997).
26. V. F. Leavers, "Survey: which Hough transform," *CVGIP: Image Understand.* **58**(2), 250–264 (1993).
27. J. R. Parker, *Algorithms for Image Processing and Computer Vision*, Wiley, New York (1996).
28. S. Yokoi, J. Toriwaki, and T. Fukumura, "Topological properties in digitized binary pictures," *Systems Computer Controls* **4**, 32–39 (1973).
29. P. Fränti, A. Mednonogov, V. Kyrki, and H. Kälviäinen, "Content-based matching of line-drawing images using the Hough transform," *Int. J. Document Analysis and Recognition* **3**(3), 117–124 (2000).

Pasi Fränti received his MSc and PhD degrees in computer science in 1991 and 1994, respectively, from the University of Turku, Finland. From 1996 to 1999 he was a postdoctoral researcher with the University of Joensuu (funded by the Academy of Finland), where he has been a professor since 2000. His primary research interests are in image compression, vector quantization, and clustering algorithms.

Eugene Ageenko received his MSc degree applied mathematics and software engineering in 1995 from Moscow State University, Russia, and PhD degree in computer science in 2000, from the University of Joensuu, Finland. Currently he works as a senior assistant for the Computer Science Dept., University of Joensuu, Finland, and a principal scientific advisor for Arbonaut Ltd. His research interests include line-art image processing and compression, mobile imaging and location-based systems. He is a member of IEEE and ISO/IEC JTC 1/SC 29/WG1 (JBIG).

Saku Kukkonen was a MSc (diploma engineer) student in the Department of Information Technology at the Lappeenranta University of Technology in 1995–1999. During the academic year 1999–2000 he was in the University of Delaware where he received MSc degree in computer and information sciences, in Spring 2000. His research interests include image processing, machine vision, and soft computing. Currently he is working as a research and teaching assistant with the Information Technology Dept. in the Lappeenranta University of Technology, Finland.

Heikki Kälviäinen received his MSc and PhD degrees (Doctor of Technology) in computer science in 1989 and 1994, from Department of Information Technology in Lappeenranta University of Technology (LUT), Lappeenranta, Finland. Since 1988 he has been working in the Lappeenranta University of Technology. Since 1996 Kälviäinen has been a professor of Computer Science and a director of Laboratory of Information Processing in LUT. Since 1999 he has been a director of the East Finland Universities Graduate School in Computer Science and Engineering (ECSE). From 1995 to 1996 Kälviäinen was a visiting research fellow in Center for Vision, Speech, and Signal Processing (CVSSP) in the University of Surrey, United Kingdom. Currently he is a visiting professor in the same university. His primary research interests include pattern recognition, image processing, and applications of machine vision and neural computing.