UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Compression of map images for real-time applications

Pasi Fränti, Eugene Ageenko, Pavel Kopylov,
Sami Gröhn and Florian Berger

Report A-2001-1

ACM I.4.2, I.5.3
UDK 519.712
ISSN 0789-7316
ISBN 951-708-975-9

Compression of map images for real-time applications

Pasi Fränti, Eugene Ageenko, Pavel Kopylov, Sami Gröhn and Florian Berger

Department of Computer Science, University of Joensuu
Box 111, FIN-80101 Joensuu, FINLAND
Email: franti@cs.joensuu.fi

Abstract: Digital maps can be stored and distributed electronically using compressed raster image formats. We introduce a storage system for the map images that supports compact storage size, decompression of partial image, and smooth transitions between various image resolutions. The main objective of the proposed storage system is to provide map images for real-time applications that use portable devices with low memory and computing resources. Compact storage size is achieved by dividing the image into semantic binary layers, which are then compressed using context-based method. Partial image decompression is supported by tiling the image into smaller blocks and implementing direct access to the compressed blocks. In this paper, we give overview of the system architecture, describe the compression technique in detail, and discuss implementation aspects. Experimental results are given both in terms of compression ratios and image retrieval timings.

Keywords: Image compression, map images, real-time applications, personal navigation, spatial access.

1. Introduction

Real-time cartography imaging application provides user with the view of geographic map for the area surrounding the user's location [1,2]. The system may use *global positioning service* (GPS) [3] or *mobile positioning service* (MPS) [4] for obtaining the coordinates of the current location. The location can be updated in real-time (about once or twice in every seconds). The system must also support real-time *panning* (spatial movement) and *zooming* (change of resolution) on the map. By panning, we mean that the user may interactively browse (scroll) the map independent on the current location. By zooming, we mean the change of the view on the display in a closer or wider perspective.

Digital spatial library (DSL) means an electronic archive of geographic maps (topographic or road map) used in the navigation [5,6]. It consists of hierarchy of map images drawn in various scales. A separate map-sheet is usually stored in a separate image augmented with the spatial location information of the map. A typical image have only a few color tones and high spatial resolution for representing fine details such as text and graphics objects. The images are usually stored in an inefficient, uncompressed raster form, or are compressed as an entity using one of the existing compression technique.

The storage size of a map image is huge. For example, electronic library of Finnish road maps of the resolution 1:250 000 takes an entire CD (over 600 Mb) in uncompressed form [7]. In comparison, the portable viewing device, such as pocket computers, have usually about 32 Mb of the storage space, which can be expanded at present by about 96 Mb through using *compact flash* memory cards [8]. The storage requirements of the maps can therefore be a bottleneck,

especially in the case of portable devices, in which the maps share the limited memory resources with the operating system, application and other data.

A better approach is to provide the user with the images stored in compressed form [9,10]. For example, an uncompressed black-and-white topographic image of 5000×5000 pixels takes about 3 megabytes in uncompressed form. Using state-of-the-art compression technology [11], this image can be compressed usually by a factor 10:1, which corresponds to the file size of 300 kilobytes. A drawback of the existing compression techniques is that entire image must be decompressed in memory before the image can be presented to the user. This can be a problem as the device may not have sufficient resources for real time image decompression.

In this paper, we propose *Map Image Storage System* (further denoted as MISS), in which we present reasonable solutions both to the storage problem and to the real-time requirements of the system. The images in MISS are composed into semantic binary layers, which are compressed using a context-based statistical modeling and arithmetic coding. The method is basically the one used in the latest international compression standards, JBIG (*Joint Bi-level Image Group*) and JBIG2 [12-14] with a few differences described later.

To meet the real-time requirements, we provide direct access to the compressed image file. Our approach is to divide the image into $b \times b$ non-overlapping rectangular blocks and compress the blocks separately. The compressed blocks are stored in the same file, and an index table is stored in the header of the file to locate the starting points of the code blocks. In this way, direct access can be provided with the accuracy of the block size. The block size is a compromise between compression efficiency and the decoding speed. The JBIG2 file structure supports this kind of file organization, where the image is composed of several segments with direct access.

The rest of the paper is organized as follows. The proposed map image storage system is introduced in Section 2. We first recall context-based binary image compression in Section 2.1. Multi-scale representation of the image is discussed in Section 2.2. for supporting the zooming property, and the decomposition of the image into binary layers is studied in Section 2.3. The image tiling into blocks is given in Section 2.4 for supporting efficient panning. The file structure and the proposed system architecture is summarized in Section 2.5. Experimental results are given in Section 3 to demonstrate the compression performance and the decoding efficiency of the system in real-time environment. Conclusions are drawn in Section 4.

2. Map image storage system

Digital maps are usually stored in vector format in a database for accessing and retrieving the data using spatial location as the search key. Vector representation is convenient for zooming as the maps can be displayed in any resolution defined by the user. Panning of the map can be performed by retrieving the elements needed for updating the changes in the view. The use of database, however, can be impractical in mobile environment as the devices may not have enough resources to store the complete map database and the database engine.

The storage problem could be solved by generating spatial views (*mapsheets*) from the database and store the maps in a vector formats. The storage size can be reduced further by compressing the vector maps (by a factor of about 2:1), or by simplifying the vector representation. This approach, however, do not support real-time panning as separate data structures must be built for this purpose.

The biggest problem of the vector formats are that maps are not always available for the user in vector format. Moreover, the maps are stored in various formats and incompatibility between different systems can restricts the use of the maps. To sum up, vector format is a good approach if the user has sufficient hardware and software resources, and if the maps are widely available in a compatible vector format. Otherwise, raster image format is the only choice.

We introduce next a map image storage system (denoted as MISS) based on compressed raster format. The system support the following properties of the maps:

1. Compact storage size
2. Multiscale representation (*zooming*)
3. Fast scrolling ability (*panning*)

The idea is that the maps are stored in server-side database. Spatial views are generated for the client-side application using compressed raster image format organized so that it supports the zooming and panning requirements. In this way, raster format is suitable in applications, where the maps are needed for viewing purposes only.

Furthermore, the system does not depend on any database or vector format as digitized raster maps can be easily generated and reproduced from any source format, including paper maps. The conversion of the maps is therefore not a big problem. Another advantage of the system is that it requires only a modest memory and computing resources in order to be operational in real-time environment.

2.1 Image compression

Bi-level image consists of two color tones and is represented as a binary data, in which '0' stands for background color (usually white), intended to be transparent for data superimposing, and '1' stands for the foreground color. Binary images can be efficiently compressed using context-based statistical modeling and arithmetic coding.

The image is processed pixel-by-pixel in raster-scan order starting from the top leftmost pixel of the image [15]. The probabilities of the black and white pixels are conditioned on the combination of already coded neighboring pixels defined by a template, see Fig. 1.

The above approach is adopted in the international standard JBIG [11,12] and the emerging standard JBIG2 [13,14]. Furthermore, JBIG2 segments the image into regions of different types, in particular, textual, halftone and generic (other), and utilize the repetitive nature of the textual and halftone images. However, the encoding of the data other than text or halftones remains similar to JBIG with the difference that a newer version of the arithmetic coder is used, namely the *MQ-coder*.

The MQ-coder is an approximate implementation of arithmetic coding tailored for binary data. It's pre-ancestor, *Q-coder*, has similar working principles and is introduced in [16]. The sub-optimality of the coder is compensated by a sophisticated automaton-based probability estimation, which provides fast adaptation to the source data. Instead of maintaining pixel counts, the estimation process is implemented as a state automaton consisting of 94 states. The automaton is a Markov-chain containing one state for each probability estimate. The states are organized in rows that are ordered by the level of adaptation. The adaptation process starts from nearly uniform blank model represented by the state 0.

The statistical model is usually integrated with the coder. The model consists of the pointers to the automaton states, one per context. The encoder (and decoder) estimates the model dynamically during the compression adapting it to the input data. At the beginning, the coder has no a priori information about the image, and an equal initial probability distribution is assumed, represented by the *zero-state*. After the pixel is coded, the model is updated by assigning new, updated, state index for the pixel context.

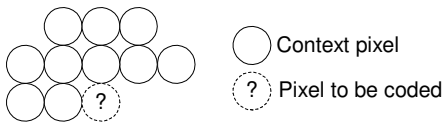


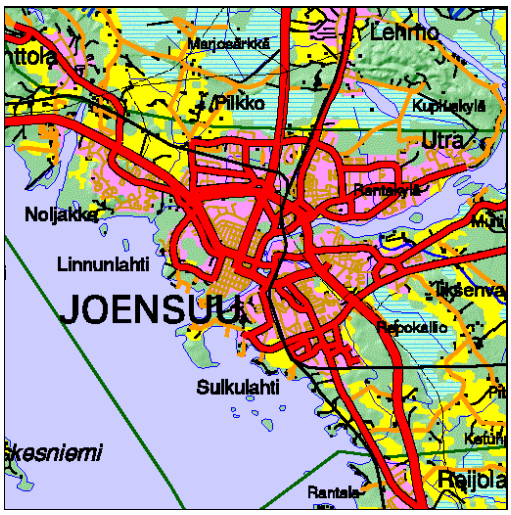
Fig. 1: Example of a ten-pixel context template.
It corresponds to the template 2 in JBIG2.

2.2 Multiscale representation

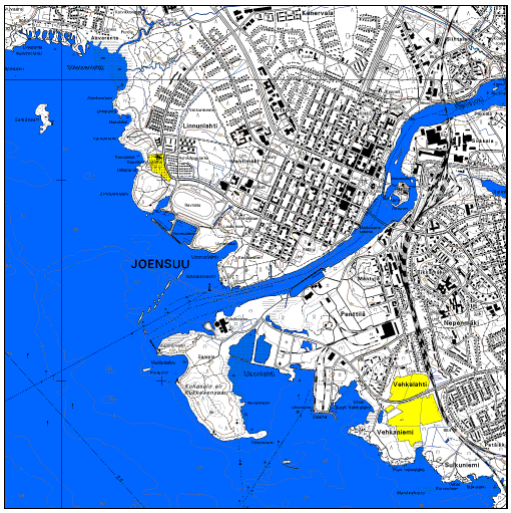
Zooming property is supported by storing several maps of different scales representing the same area. The images of different scale can have different level of details and the way to depict the spatial data. In addition to this, intermediate scales can be generated by using existing resolution reduction/enhancement algorithms. The example in Fig. 2 include two different maps (1:20 000 and 1:100 000) of the same location. An intermediate scale of 1:40 000 has been generated from the detailed map (1:20 000) in order to provide the user with smoother zooming. The image has the same level of details but the size and quality of the features might suffer because of the change in resolution. The intermediate scale images can be generated beforehand, and stored in the library. Another solution is to perform real-time zooming in the client application.

The organization of the data is illustrated in Fig. 3. In this example, the large rectangles represent the map images of four different scale. The size of the rectangles corresponds to the size of the individual images in pixels. The images have usually the same size when printed on paper or shown on display. The thin grid lines drawn across the rectangles correspond to the spatial territories shown in the maps that have same size in reality but different resolution.

Map of scale 1:100,000



Map of scale 1:40 000 (generated from 1:20 000)



Map of scale 1:20 000

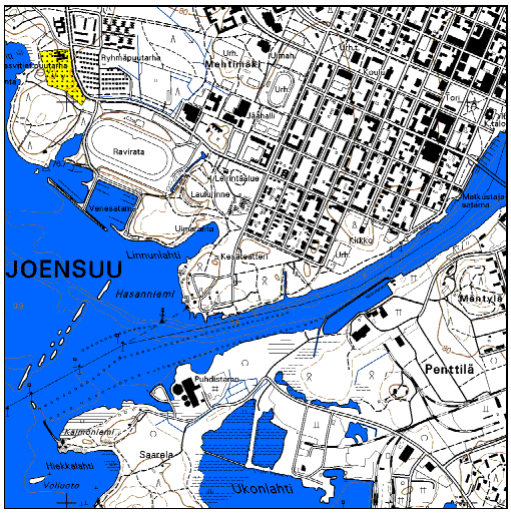


Fig. 2: Example of a map shown in three different scales. The highest and smallest scales have different representation of the content, but the intermediate scale has been generated from the map of lowest scale.

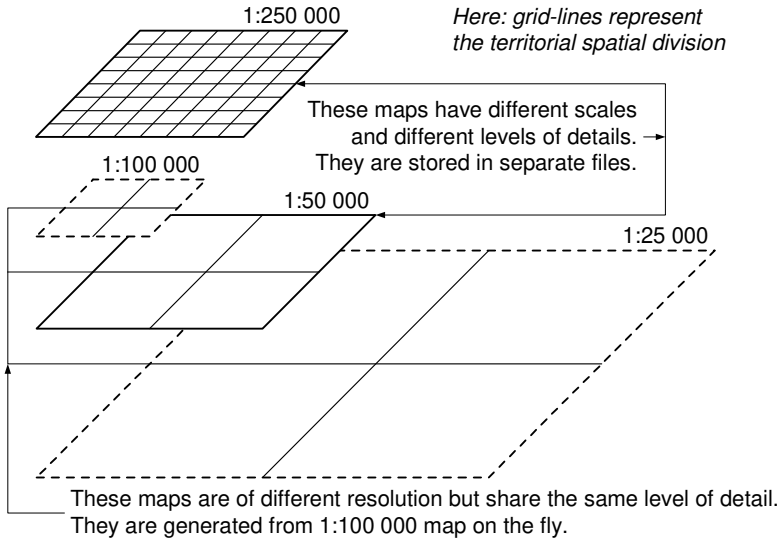


Fig. 3: Multi-scale organization of the map images.

2.3 Decomposition to binary layers

In order to utilize the context-based compression, the map images must be divided into binary layers. Each layer is then compressed separately using the context-based method described in Section 2.1, and the compressed layers are stored into the same file. There are three ways to perform the decomposition into binary layers:

1. Semantic decomposition
2. Color separation
3. Bit-level separation

Semantic decomposition is possible if the maps are obtained from a map database in vector format. The map is output into a set of binary layers each containing different semantic meaning of the location it represents. For example, the map shown in Fig. 4 consists of five layers with different semantic meaning: basic (topographic data and contours), elevation lines, fields, water and property (administrative borders). The user application can reproduce the map by plotting each layer by its own color overlapping each other in a given order. Color information can be added to the file. The image can be reconstructed as a gray scale image, or using any color set given by the user application. The benefits of the semantic separation is better compression performance in comparison to the other options, and that the application may select the layers that are shown on the display at the time.

The second approach, *color separation*, can be used when we have only raster color image as the original map, and it contains only a limited number of colors [17]. The image is divided into binary layers so that each layer represents one color in the original image. The drawback of the color separation is that information of the original semantic separation cannot be recovered. Furthermore, the color separation can create “pseudo-information” into the binary layers at the color transition points, see Fig. 5. For example, overlapping text elements breaks the continuation of the fields and lakes, which increases the complexity of the binary images and thus, the compressed file size.

The third approach, *bit-level separation*, must be applied when we have the original map only as a raster image, and the number of different colors is too high for efficient color separation. For example, the image might have been digitized from a paper copy and stored using lossy

compression method, such as JPEG [18]. In the bit-level separation, the number of colors are first reduced by quantizing the image into a limited-color representation of a 256 colors or gray-scales. The resulting pixel values are then separated into bit planes using Gray coding [19], and the image is represented as a sequence of binary images.

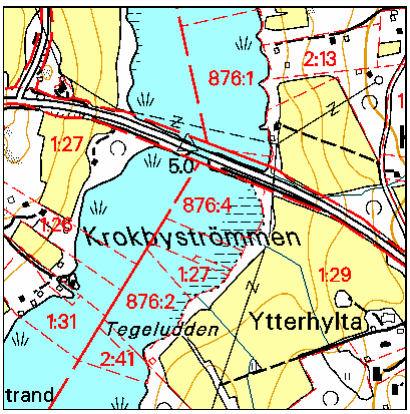


Fig. 4: Sample fragment from a color map image.

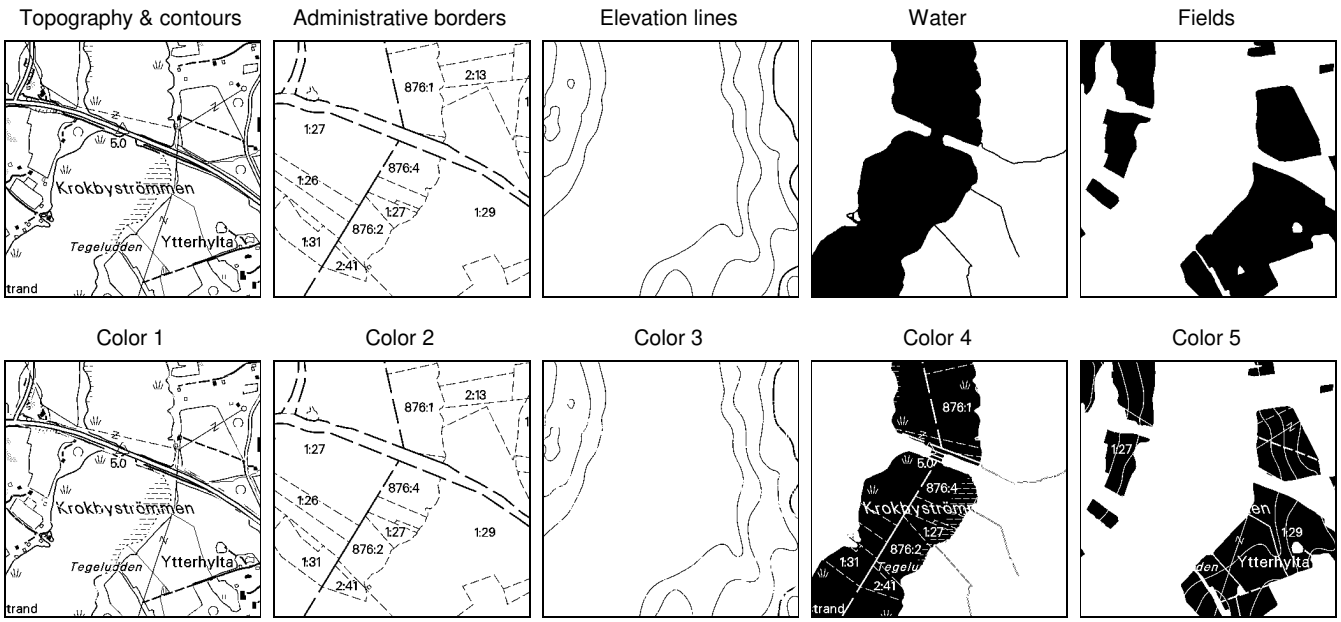


Fig. 5: Example of the image decomposed into binary layers by semantic separation (above), color separation (below).

2.4 Block decomposition for direct access

The binary layers are divided into $b \times b$ non-overlapping rectangular blocks before the compression, and each block is compressed separately from others as proposed in [20]. The compressed blocks are stored in the same file, and an index table is stored in the header of the file to locate the starting points of the code blocks, see Fig. 6. When the compressed image is accessed, a block index table indicating the location of the block in the compressed file, can

be constructed. This provides direct access to the compressed image file, and therefore, enables efficient and independent decompression of a particular image fragment.

The block decomposition has the effect that there are fewer pixels to be coded in the same run. It means that the backward-adaptive modeling has less time to adapt to the statistics of the image, which increases the learning cost. Another problem related to the compression of small blocks is the compression inefficiency near block boundaries. This is because the pixels located outside the block cannot be used in the context template. Previous studies indicate that the compression inefficiency remains tolerable if the block size is kept 256×256 pixels or higher [20].

Somewhat better compression performance can be obtained using the following two modifications. First idea is to apply a forward-adaptive variant of the compression method based on the ideas presented in [21]. The forward-adaptive variant uses a pre-calculated initial model for the encoder. The model is constructed using the statistics collected from the entire image layer. This requires an additional pass over the image but does not affect the speed of the decompression. Second idea is to use variable-size context modeling technique as described in [22]. This technique reduces the size of the model, and at the same time, allows for using larger context templates when needed. These modifications can provide about 20% improvement in the compression performance.

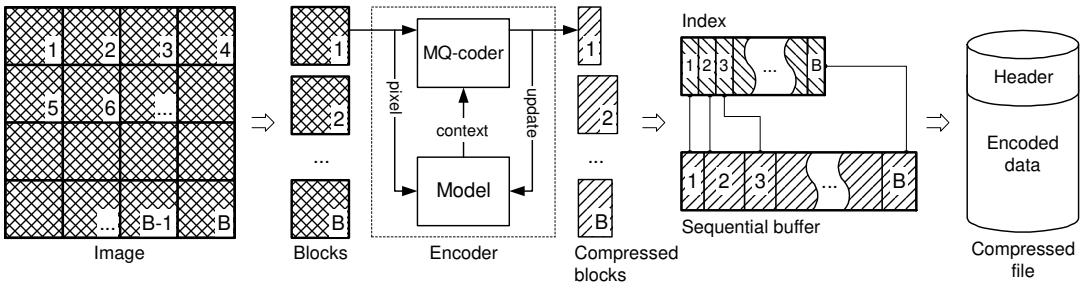


Fig. 6: Diagram of the block decomposition.

2.5 Compression phase

The compression of a single map image is performed using the following steps:

1. Layer decomposition
2. Block decomposition
3. Compression

In the first step, the image is decomposed into the binary layers (unless the semantic decomposition already exists). The color space is enumerated and a the number of required bit planes are generated. The resulting color palette is optionally stored in the compressed file. If the palette is not stored, the image will be reconstructed as a gray scale, and pixel colors will be gray values equally spaced into the range of the viewing device.

In the second step, the layers are partitioned into blocks. If the forward-adaptive modeling variant is applied, the non-empty blocks of the entire layer are analyzed and the statistical model is built and stored in the compressed file. The emptiness of a block is determined by checking whether all pixel values in the block are the *default color value*.

In the third step, the series of the bit planes are compressed and ordered highest to lowest, see Fig. 7. Each non-empty block is compressed using context-based modeling and the MQ-coding algorithm as described in Section 2.1, which is basically the same as the compression of separate generic regions in JBIG2. The context is determined by the template shown in Fig. 1 but we also permit the use of custom context-templates as described in [23]. The MQ-coder is reinitialized every time the compression of a new block starts. The initialization resets the model to the blank model (default), or to the optimized initial model (optional).

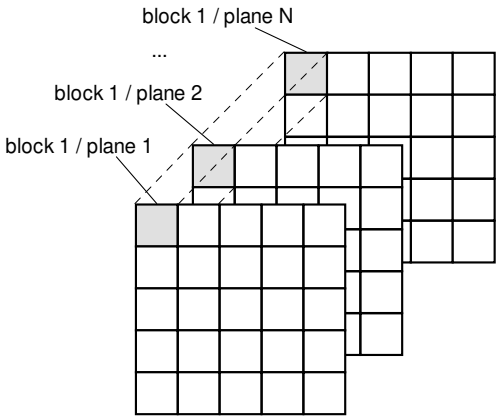


Fig. 7: Image decomposition diagram.

2.6 Use in the client device

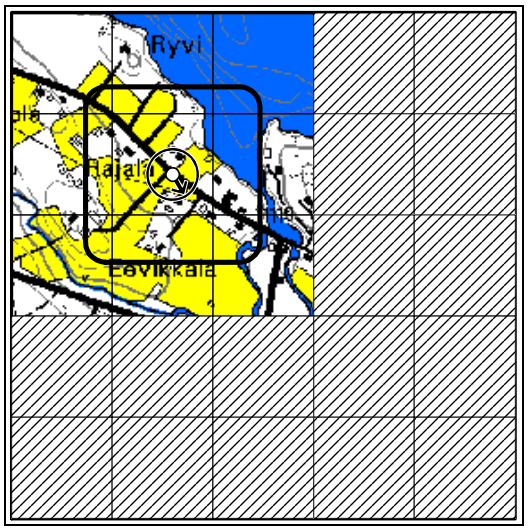
A typical use scenario of the map image storage system (MISS) is to show the area surrounding the object whose position is tracked. The scale to be used can be set by the user or it can be automatically determined on the basis of the speed, or other parameters defined in the application. The maps for the particular region are stored in the client's viewing device (flash memory, hard disk, CD, etc.), or the images can be located in a remote server and accessed via local network (internet, gsm, grps, etc.)

The system uses the following steps to show the current view on the map. First, the file possessing the map of the desired location is accessed and its header part is retrieved in memory. From the header, the type and structure of the image are determined, and the block index table is built. The table indicates the size and locations of the blocks in the compressed file. All supplementary data required for image decoding (such as initial model, context tree) are also retrieved and kept in memory until this particular map image is no longer used.

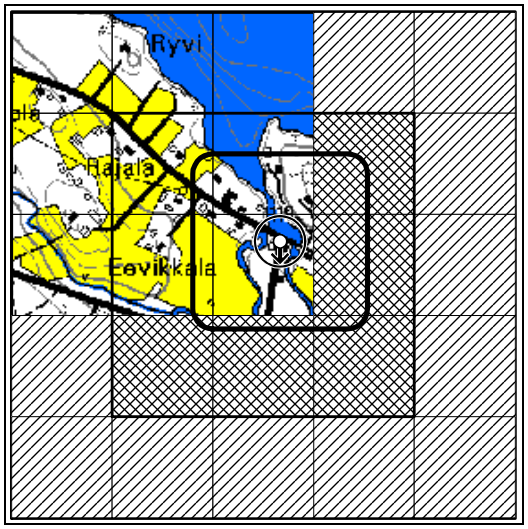
Next, depending on the requested location, the system calculates the map image fragment needed to be displayd. The blocks covering this fragment are retrieved and decoded. If the map image is accessed remotely, the retrieved blocks are also stored (in compressed form) in the local storage space for further use. When the position of the object changes, the view is updated by decoding new image blocks in the direction of movement, see Fig. 8.

To speed-up the access to the image, the system may use cache for temporary storage of the decompressed image blocks, see Fig. 9. The size of the cache can be fixed, or it can be determined by the amount of free memory. The performance may be further improved by exploiting an idle time for decoding neighboring blocks further in the direction of movement, before the blocks are actually requested. In practice, it is convenient to buffer the data according to the block boundaries.

Step 1: Nine blocks are decompressed in the cache.



Stage 2: Change of location is registered.



Stage 3: Five new blocks are decompressed.

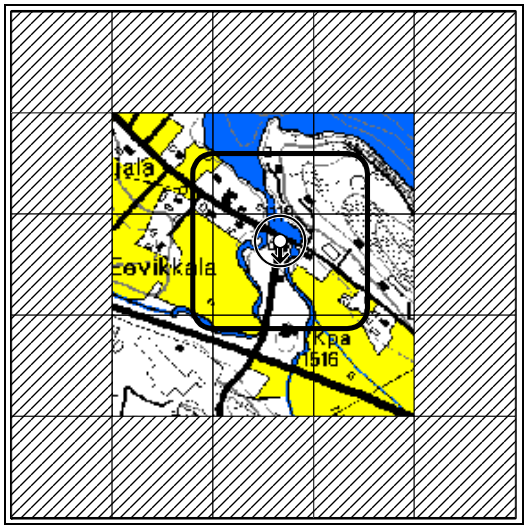


Fig. 8: Image decoding in a real-time cartography imaging system.

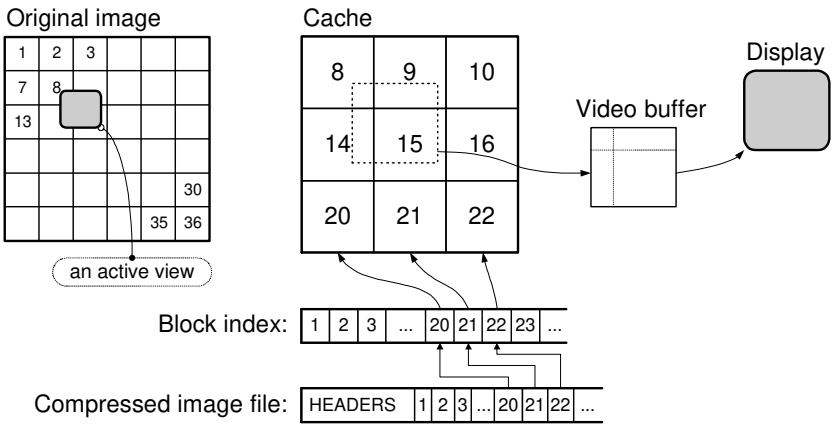


Fig. 9: Illustration of the cache operation.

3. Experimental results

We study next the compression performance and the retrieval times of the proposed storage system. The following methods are considered in the comparisons:

- MISS
- JBIG2
- TIFF G4
- GIF
- PNG
- RAW

JBIG2 [13, 14] is the latest binary image compression standard as described in Section 2.1. We compress the whole image as one region using generic coding with the 10-pixel context template. TIFF G4 [9] refers to the older *ITU-T Group 4* fax compression standard; we use the method as included in the *Tagged Image File Format* (TIFF). The *CompuServe Graphics Interchange Format* (GIF) [24] is the most widely used format in internet for palette images. It uses the dictionary-based coding known as LZW [25], which is based on class of Ziv-Lempel methods known as LZ78 [26]. The *Portable Network Graphics* (PNG) [27] differs from GIF in that it is based on LZ77 dictionary compression [28] instead of the LZ78.

In MISS, we use the following parameter setup:

- The default 10-pixel context template from JBIG2,
- Forward-adaptive statistical modeling optimized for each layer,
- Block size of 100×100 pixels,
- Other parameters same as in JBIG2.

3.1 Test image sets

We use three image sets representing the following situations:

- Set #1: layers separation by semantic decomposition,
- Set #2: layers separation by color separation,
- Set #3: layers separation by quantization and bit-level separation.

The first two sets contain topographic map images, and the third a set of two road map images. The first two sets originate from the NLS topographic database [29]. The images are of the size 5000×5000 pixels, and have the original scale 1:20 000. This corresponds to a resolution of two meters per pixel. The topographic map images are composed of five semantic layers representing topography, elevation lines, waterways, fields and administrative boundaries (when available).

The Set #1 (semantic decomposition) includes the images when separated into the five semantic layer. The Set #2 (color separation) contains the same set of images but after the following processing. Color image is first constructed from the original semantic layers, and color separation is then performed to estimate the original division. See Fig. 5 for the difference in the bit-plane separation in these two cases. The Set #3 (bit-level separation) consists of two scanned color map images including scanning noise and a high number of output colors, see Fig. 10. In the experiments, the images have then been converted to 2 bits per pixel gray-scale images corresponding to the number of colors available in a low cost mobile device. The images have been divided into two bit-planes.

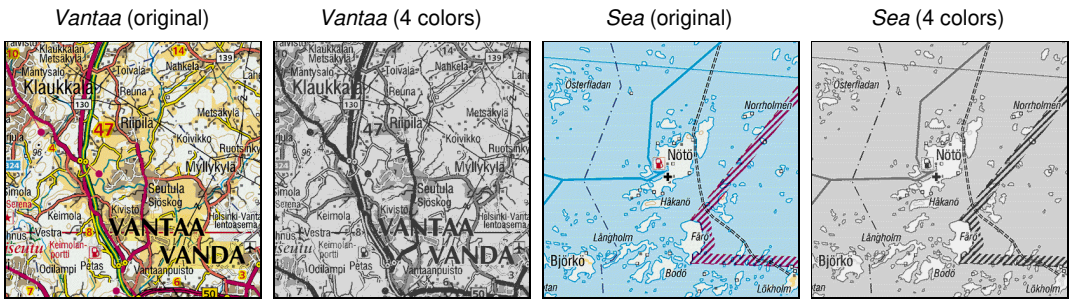


Fig. 10. Sample fragments from the images in the Set #3.

3.2 Compression performance

The compression results have been summarized in Tables 1-3. The average compression performance of MISS is about 0.20 bits per pixel but the result depends on the complexity of the image. The MISS files take about 5-15% more space than that the JBIG2 files, on average, but 50-65 % less than the comparative methods (TIFF-G4, GIF, PNG). The MISS, on the other hand, is the only method that supports direct access to the compressed file. The results in Fig. 11 shows that most of the bits (about 54 %) originates from the basic information, whereas the water and fields are rather easy to compress. Administrative boundaries were absent in the case of image 3.

The compression results between the Set #1 and Set #2 are not significant. The color separation loses information that originally appear in the image, but at the same time, it adds to the complexity of the images when the layers are compressed separately. The overall effect, however, remains small. We therefore did not consider multilevel context templates, which could be used to handle this kind of inter-layer dependencies. In the case of Set #2, the GIF and PNG methods were applied to the color images, and not the bit planes. In the case of Set #1, this is also possible but not desirable as we did not want to lose the semantic separation of the images.

The images in the Set #3 are more difficult to compress because of the noise and the level of complexity of the images. The average bit rates for these images is 0.84. The performance gap between MISS and the comparative methods (TIFF-G4, GIF, PNG) is much smaller than in the case of the Set #1 and Set #2 (MISS files take about 25% less than the PNG files). The results demonstrate the importance of having the image divided into semantic layers, or at least a clean original so that the color separation can be performed.

The effect of the block size is illustrated in Fig. 12. The optimal block size is around 350×350 to 500×500 in terms of compression performance as the model can be best optimized for images of this size. With the chosen 100×100 the files sizes are slightly bigger but it allows more dense tiling allows more accurate buffering with less memory resources, and smaller transmission and decompression delays.

Table 1: Compression results (kilobytes) for the set 1 (semantic decomposition).

	MISS	**JBIG2**	**TIFF G4**	**GIF**	**PNG**	**RAW**
Image 1	247	197	372	727	602	15259
Image 2	1109	969	2382	2866	2608	15259
Image 3	395	327	604	1142	1100	15259
Image 4	840	759	1540	2633	2534	15259
Total	2591	2252	4899	7367	6845	61035
bpp	0.21	0.18	0.40	0.60	0.56	5.00

Table 2: Compression results (kilobytes) for the set 2 (color separation). *The GIF and PNG results are obtained by compressing the corresponding color image instead of the binary layers.

	MISS	**JBIG2**	**TIFF G4**	**GIF***	**PNG***	**RAW**
Image 1	310	258	466	610	654	15259
Image 2	1269	1138	2605	2642	2644	15259
Image 3	426	360	691	1028	1074	15259
Image 4	957	875	1902	2295	2384	15259
Total	2962	2631	5664	6575	6757	61035
bpp	0.24	0.22	0.46	0.54	0.55	5.00

Table 3: Compression results (kilobytes) for set 3 (bit-level separation).

	MISS	**JBIG2**	**TIFF G4**	**GIF***	**PNG***	**RAW**
Vantaa	91	84	123	120	132	313
Sea	44	39	72	53	59	313
Total	135	123	195	173	191	625
bpp	0.86	0.79	1.25	1.11	1.22	2.00

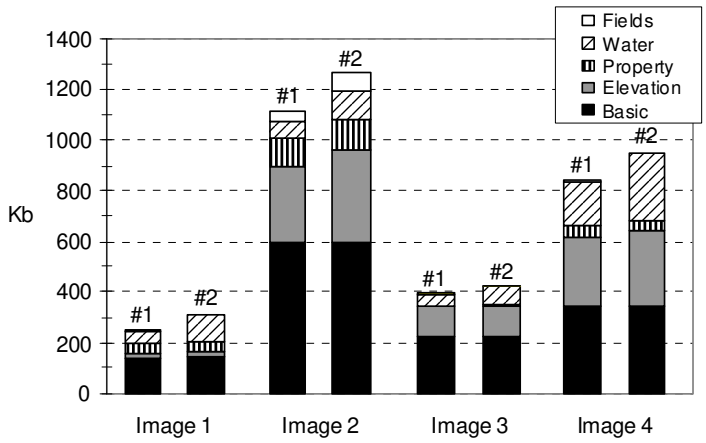


Fig. 11: The proportion of the layers in the compressed MISS files for the Set #1 (left columns), and Set #2 (right columns).

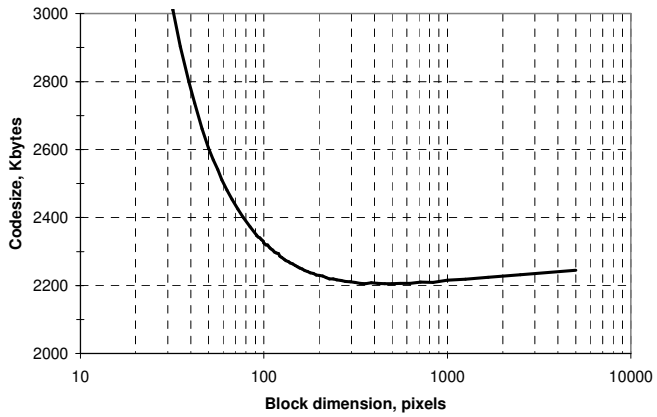


Fig. 12: The effect of the block size on the code size for the Set #1.

3.3 Retrieval timings

We consider next the retrieval performance of the proposed system by measuring time required to transmit and decompress a desired part of the map. We assume that the client device buffers the image data according to the nearest block boundaries. In other words, the client devices stores in the memory the pixels of the current view, plus additional outside pixels from the blocks across the screen boundary. In this way, we need to transmit and decompress, on average, the exact amount of pixels required to update the view.

The decompression times are summarized in Table 4 when decompressing the complete 5000×5000 images. The results show that the better compression performance of the MISS has been obtained at the cost of 10-20 times slower decoding speed in comparison to GIF and PNG. The decoding speed of MISS corresponds to 973,710 pixels per seconds for the images in the Set #1, on average. Using this result, we then calculated the sample retrieval timings for several different screen sizes with varying computing power and transmission speeds.

Table 4: Decompression times (in seconds) for the Set #1 using a processor of 1000 MIPS.

	Method:		
Image:	MISS	GIF	PNG
Image 1	25.9	2.1	1.2
Image 2	31.4	2.5	1.4
Image 3	18.4	2.2	1.1
Image 4	27	2.2	1.3
Total	102.7	9.0	5.0

We consider the following three transmission networks:

- GSM, capable of 9600 bits per second,
- high speed GSM, capable of 14400 bits per second,
- GRPS network, capable of 48 kilobits per second.

Typical hand-held devices have relatively low computing power with only a few MIPS but the computing power of these devices is developing rapidly. We consider three speeds (10, 50, 100 MIPS) that roughly correspond to the computing power of the current and forth-coming low-cost compact devices. The retrieval timings can be calculated as follows:

Transmission time = compressed data size / channel bandwidth

Decompression time = uncompressed image size / decoding speed

The results are summarized in Table 4 in case of retrieving data for a full screen. The results show that the data for a reasonable size screen can be transmitted and decompressed in real-time using existing networks and devices of reasonably low computing speed. For example, the screen size of 150×150 pixels equals to 22,500 pixels, and $0.21 \cdot 22{,}500 = 4{,}725$ bits in the compressed format. Let us then consider the GSM channel and a low-performance client device with 50 MIPS. In this case, the transmission of a full screen takes about 0.33 seconds and the decompression about 0.57 seconds. The timings varies depending on the complexity of the blocks; the most complex image regions take about twice as that of the average case.

In the case of movement, the view is scrolled and a new portion of data is retrieved. We consider a diagonal movement by amount of one block ($b \times b$ pixels). The proportion of data to be retrieved is ($xb + yb - b^2$) pixels, where $x \times y$ is the size of the screen, and $b \times b$ the size of a block. The proportion of data to be retrieved is summarized in Table 5 using various block sizes. Using the default block size of 100×100 pixels, we must decompress 64-100% of the data shown on screen size. Smaller block sizes would improve the situation but also increase the file sizes, and therefore, increase the transmission delays.

Table 4: Transmission and decompression times (in seconds) for retrieving a full screen. The processor is assumed to be a low performance processor capable of 100 MIPS, and the blocks fitting perfectly on the chosen view.

Full screen	Transmission times:			Decompression times:		
Screen size:	9600 bps (GSM)	14400 bps (hs-GSM)	48 kbps (GRPS)	10 MIPS	50 MIPS	100 MIPS
100×100	0.22	0.15	0.04	1.03	0.21	0.10
150×150	0.49	0.33	0.10	2.31	0.46	0.23
200×200	0.88	0.58	0.17	4.11	0.82	0.41
250×250	1.37	0.91	0.27	6.42	1.28	0.64

Table 5: Amount of work (decompression) when refreshing a view after a diagonal scroll of 100 pixels in each dimension (corresponds to one block). The numbers are relative to the update of full screen.

Refresh	Block size:		
Screen size:	50×50	100×100	200×200
100×100	75 %	100 %	400 %
150×150	56 %	89 %	178 %
200×200	44 %	75 %	100 %
250×250	36 %	64 %	96 %

3.3 Memory requirements

We consider next the memory requirement of the client device. Memory is needed both for the decompression software, and for storing the uncompressed image blocks. The decompression software can be fit into 64 kilobytes of object code (standard libraries not included), or in a 100 kilobytes DLL (everything included). This is hardly a bottleneck. Besides the Codeq, we need memory also for buffering the image data. The following parameters must be set:

- L = number of layers per image
- K = number of images stored
- C = number of contexts (1024 by default)

Let us suppose that we store K images of different scales in the memory, and each of the images have L layers. We can safely assume (on the basis of the discussion in Section 2.3) that the number of layers is never more than eight. Thus, we can store the layers one pixel per byte (one bit per layer). Furthermore, we assume that the chosen block size is within the range 50-100% of the screen size in both directions. For example, the block size 100×100 fulfil this requirement for screen sizes up to 200×200 pixels. The main consequence of this assumption is that we need to buffer only 3×3 image blocks to form a bounding box around the current view. Using the default 10-pixel context template, the number of contexts is 2^{10}=1024.

The memory requirements are summarized in Table 6. First, we need I/O data buffers for a single block; one buffer for the input (compressed) data and another one for the output (uncompressed) data. Image header then takes 400 bytes per image, and the coding parameters 616 bytes per layer. For each layer, we must also store the model despite the choice of static or forward-adaptive approach. The model data takes 3 bytes per context plus 56 bytes overhead.

Assuming that we have the block size of 100×100 pixels, L=5 layers per image, and K=4 images stored (e.g. images with the scale 1:20 000, 1:40 000, 1:100 000, 1:250 000), the overhead of these parts sum up to about 100 kilobytes. The data of one images takes $9 \cdot 100^2$ = 90,000 bytes, which sums up to 360,000 bytes in total.

Table 6: Memory requirements (in bytes) of the data buffering in the client device (above), and the sample numbers (below) for the parameters: B=2500, L=5, K=4, C=1024, b=100.

	per block	**per layer**	**per image**	**Total**
I/O data buffers:	---	---	---	$2 \cdot b^2$
Image header:	---	---	400	$L \cdot 400$
Coding parameters:	---	616	$L \cdot 616$	$KL \cdot 616$
Statistical model:	---	$56 + C \cdot 3$	$L \cdot 56 + LC \cdot 3$	$KL \cdot 56 + KLC \cdot 3$
Image data:	---	$N/8$	N	KN

	per block	**per layer**	**per image**	**Total**
I/O data buffers:	---	---	---	20,000
Image header:	---	---	400	2,000
Coding parameters:	---	616	3,080	12,320
Statistical model:	---	3,128	15,640	64,560
Image data:	---	---	90,000	360,000

4. Conclusions

We proposed a map image storage system (MISS) for real-time applications that use portable devices with low memory and computing resources. The system architecture is designed to minimize storage size, transmission time, and memory requirements of the user device. Compact image size is achieved by dividing the image into semantic binary layers, which are then compressed using the state-of-art context-based method. Direct access to the compressed image file allows to transmit/decompress only the necessary part of the image needed. This minimizes the transmission time and memory requirement in the user device are minimized. These property together enables the use of large map images in mobile environments, and on portable devices with limited memory and computing resources.

Acknowledgments

The work was supported by the *National Technology Agency* of Finland (TEKES) as the project *Real-time cartography imaging* (No.954/401/99). The work of Pavel Kopylov was supported by a grant from the *Centre for International Mobility* (CIMO).

References

[1] M.-J. Kraak, et al, *Cartography : Visualization of Spatial Data*, Addison-Wesley, 1996.

[2] M.-J. Kraak, A. Brown, *Web Cartography*, Taylor & Francis, 2000.

[3] E.D. Kaplan (ed.), *Understanding GPS: Principles and Applications*, Artech House Telecommunications Library, March 1996.

[4] S. Dye, S. Buckingham, *Mobile Positioning*, Mobile Lifestreams, 1999.

[5] E.A. Fox, et al. (eds.). Digital Libraries. [*Special issue of*] *Communications of the ACM* **38** (4), 1995.

[6] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing*, *GIS*. MA: Addison-Wesley, Reading, 1989.

[7] GeoData, *CD-tiekartasto Suomi 1:250* 000 (*CD Roadmap Catalog Finland*), WSOY, Helsinki, 1999. (in Finnish)

[8] H.W. Gellersen, P.J. Thomas (eds.), *Proc. Second International Symposium on Handheld and Ubiquitous Computing* (HUC'2000), Bristol, UK, Springer Verlag, September 25-27, 2000.

[9] R.B. Arps, T.K. Truong "Comparison of international standards for lossless still image compression," *Proceedings of the IEEE*, **82** (6), 889-899, June 1994.

[10] J.D. Murray, W. vanRyper, D. Russell (ed), *Encyclopedia of Graphics File Formats*, 2-nd ed, O'Reilly Associates Inc, 1996.

[11] B.G. Haskell et al., "Image and video coding – emerging standards and beyond," *IEEE Trans. Circuits and Systems for Video Technology*, **8** (7), 1998, 814-837.

[12] ISO/IEC, *Progressive Bi-level Image Compression*, 11544, ITU-T Recommendation T.82, 1993.

[13] ISO/IEC, *Final committee draft for ISO/IEC International Standard 14492*, 1999. (`http://www.jpeg.org/public/jbigpt2.htm`)

[14] P.G. Howard, F. Kossentini, B. Martins, S. Forchammer and W.J. Rucklidge, "The emerging JBIG2 standard," *IEEE Trans. Circuits and Systems for Video Technology*, **8** (7), 838-848, 1998.

[15] G.G. Langdon, J. Rissanen, "Compression of black-white images with arithmetic coding", *IEEE Trans. Communications*, **29** (6), 858-867, June 1981.

[16] W.B. Pennebaker, J.L. Mitchell, G.G. Langdon, R.B. Arps, "An overview of the basic principles of the Q-coder adaptive binary arithmetic coder," *IBM Journal of Research and Development*, **32** (6), 717-726, 1988.

[17] D. Tompkins, F. Kossentini, "Additional Extension Segments for JBIG2," ISO/IEC JTC 1/SC 29/WG1 (ITU-T SG8), Document No 1318, July 1999.

[18] W.B. Pennebaker, J.L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, 1993.

[19] M.J. Weinberger, J. Rissanen, R. Arps, "Application of universal context modeling to lossless compression of gray-scale images," *IEEE Trans. Image Processing*, **5** (4), 575-586, April 1996.

[20] E.I. Ageenko and P. Fränti, "Enhanced JBIG-based compression for satisfying objectives of engineering document management system", *Optical Engineering*, **37** (5), 1530-1538, May 1998.

[21] E.I. Ageenko, P. Fränti, "Compression of large binary images in digital spatial libraries", *Computer & Graphics*, **24** (1), 91-98, 2000.

[22] E.I. Ageenko, P. Fränti, "Forward-adaptive method for compressing large binary images," *Software Practice & Experience*, **29** (11), 943-952, 2000.

[23] E.I. Ageenko, P. Kopylov, P. Fränti**,** "Optimizing context template for compression of multi-component map images," *Proc. GraphiCon'00*, Moscow, Russia, pp. 151-156, 2000.

[24] J. Miano, *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*, (ACM Press), Addison-Wesley , Boston, 1999.

[25] T. Welch, "A technique for high-performance data compression", *IEEE Computer*, **17** (6), 8-19, June 1984.

[26] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding", *IEEE Trans. Information Theory*, **24**, 530-536, 1978.

[27] G. Roelofs, *PNG: The Definitive Guide*, O'Reilly & Associates, Cambirdge, MA, June 1999.

[28] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression", *IEEE Trans. Information Theory*, **23**, 337-343, 1977.

[29] National Land Survey of Finland, Opastinsilta 12 C, P.O.Box 84, 00521 Helsinki, Finland. (`http://www.nls.fi/index_e.html`)