An Image Compression Method for Spatial Search

Renato Pajarola and Peter Widmayer

***Abstract*—The maintenance of large raster images under spatial operations is still a major performance bottleneck. For reasons of storage space, images in a collection, such as satellite pictures in geographic information systems, are maintained in compressed form. Instead of performing a spatially selective operation on an image by first decompressing the compressed version, we propose in this paper to perform queries directly on the compressed version of the image. We suggest a compression technique that allows for the subsequent use of a spatial index structure to guide a spatial search. In response to a window query, our algorithm delivers a compressed partial image, or the exact uncompressed requested image region. In addition to the support of spatial queries on compressed continuous tone images, the new compression algorithm is even competitive in terms of the compression ratio that it achieves, compared to other standard lossless compression techniques.**

***Index Terms*—Lossless image compression, space filling curves, spatial indexing and retrieval.**

I. Introduction

THE growing interest in geoinformation systems (GIS), virtual reality (VR) environments, and multimedia applications is accompanied by the demand to process very large amounts of digital raster image data. This includes remote sensing data such as satellite images and aerial photographs. Large database systems such as image archives call for efficient access methods to provide fast query processing. Therefore, data management, physical storage, and retrieval algorithms are very important. Of particular interest, both from a theoretical as well as from a practical point of view, is providing at the same time compact storage by *data compression* and supporting fast retrieval by efficient *access algorithms*.

Due to the enormous data volume, the efficiency of operations is still a major bottleneck in systems that handle large raster images. A single digital satellite image, for instance, with its multiple data channels and high-resolution quality, is easily as large as several hundred megabytes; the images that we used in our experiments typically consisted of 25 million pixels. On one hand, efficient spatial access is possible by storing raster data in one of several data structures, e.g., quadtrees [1], or by ordering the raster data according to expected access patterns [2]. On the other hand, efficient access alone is not what we want, and we would like to represent the data as succinctly as possible.

Images are preferably maintained in compressed form not only to save space, but also to perform operations faster on the compressed images. This is a realistic objective because in compressed form, less data has to be processed. Digital image compression deals with the problem of finding such a succinct representation for a raster image. Over the years, quite a few compression techniques have been developed that aim at eliminating redundancy from raster images [3], [4]. For archiving a raster image or for transmitting an image over a network, compression is particularly useful. It has, however, an adverse effect on query processing: decompressing an image requires that the whole image is accessed on external storage. Additionally, the decompression algorithm may take a lot of computation time. Whenever a query does not refer to the full image, some of the decompression work is wasted. Raster images tend to become larger and larger, partly because of the increased application of *mosaicing*[1] techniques, and partly because the image resolution increases. Thus, it becomes less likely that a large part of an image is needed to answer a query. It seems crucial that access to a small part of an image is supported on the compressed version of the image, that is, the retrieval of a (small) subimage specified by its location should be possible without decompressing the (large) compressed image. This paper suggests a way to achieve this goal.

As an aside, note that it might also be interesting to perform the search "the other way around": We might as well search for a specific pattern (i.e., the location at which a given subimage occurs) in the compressed image. In another paper [5], we show how this can be done efficiently.

A typical application where access into a compressed image appears to be particularly efficient is the scenario where a server has to answer window queries in satellite image archives over a wide area network (e.g., the Internet). Here, the transmission over the network is usually the most severe bottleneck. With an appropriate compression and access method, the server should retrieve a part of the compressed image that contains the query window, and it should send this part of the image without decompression. The decompression should then be performed on the client side after the transmission. Furthermore, interactive visualization of large-scale terrain data involving satellite image texturing is a potential application of access to compressed images. On one hand, the texture database must maintain large amounts of image data such as satellite images and aerial photographs, so reducing storage costs calls for efficient compression methods. On the other hand, the dynamic visualization requires fast access to texture data for different image regions because the visible scene changes rapidly and covers only a fraction of the whole database.

Only the most advanced GIS systems today are able to offer both storage of compressed raster images and a spatial index. The prime example for such a system is the client–server

Manuscript received August 18, 1998; revised August 2, 1999. The associate editor coordinating the review of this paper and approving it for publication was Dr. Yoshitaka Hashimoto.

R. Pajarola is with the Department of Information and Computer Science, University of California, Irvine, CA 92697 USA (e-mail: pajarola@acm.org).

P. Widmayer is with the Institute of Theoretical Computer Science, Department of Computer Science, ETH Zürich, CH-8092 Zürich, Switzerland (e-mail: widmayer@inf.ethz.ch).

Publisher Item Identifier S 1057-7149(00)01506-2.

[1]This is a technical term for merging and gluing images together.

Paradise system [6], a database system designed for GIS-type applications. Special care has been taken in Paradise to make processing of satellite image data efficient. For this purpose, a raster image is partitioned into tiles; they form the basic processing units for indexing and for compression. It is argued that the choice of the tile size is crucial to efficiency: large tiles return many redundant data in response to a window query, and small tiles give a bad compression ratio. The tile size varies from 8 KB (very small) to 512 KB (very large) of data. Fixing tile sizes in advance requires accepting whatever efficiency results. We do not advocate this approach. Instead, we propose an adaptive mechanism that effectively partitions the data in a way that is directed toward both, good compression, and efficient indexing.

II. The Problem

We study the problem of answering a window query on a compressed version of a raster image quickly, in particular, without decompressing the entire image. A *window query* specifies an axis-parallel rectangular window in the underlying space (i.e., in the matrix of pixels) and asks for the subimage that is seen in this window. The answer to the query, therefore, is itself a raster image. For efficiency reasons, we would ideally want the answer to be a compressed raster image. We restrict ourselves to lossless compression for two reasons. The first is our interest in the theoretical foundation of the problem, and the second is the practical demand of not discarding any of the precious information that has been acquired via satellite. The window query (also known as range query in the database literature) serves as a specific, but prototypical, spatial query. Since we assume raster images to be large and to reside on disk (for both reasons of size and durability), we want to avoid loading the entire raster image from disk into internal memory in order to answer a window query. Instead, the raster image should be stored on disk in such a way that the image partition onto disk blocks reflects the locality of pixels in the image, even in its compressed version. It is obvious that this can not be achieved by simply taking any compression technique and spreading the resulting bit-sequence in order across the disk blocks. On the contrary, both the compression technique and the storage mapping of the resulting bit-sequence have to work hand-in-hand to support window queries efficiently. As a cornerstone against which we want to compare any method, let us state the desired properties of an (unachievable) ideal solution. Ideally, we would like to find a raster data structure and compression technique with the following properties:

1) pixels represented in compressed form in the same disk block are close in the image;
2) a window query can be answered by accessing just those disk blocks that contain the compressed version of the pixels in the window, and these blocks do not contain much else;
3) compression ratio is as good as that of any other applicable image compression algorithm.

Property 1 is interpreted to imply that the geometric shape of the union of all pixels in a block (or some container for the pixels, such as the bounding box) is good for efficient spatial indexing. For instance, see the criteria for efficiency of spatial access structures based on aligned rectangles [7]–[9].

These properties together call for a solution that combines the characteristics of the best available spatial data structure with those of the best available compression algorithm. The problem now lies in the fact that the compression technique must lend itself to spatial clustering. Any compression technique, for instance, that makes use of the statistics of the whole image for every pixel (and updates them) will be unable to fulfill requirement 2: these statistics could only be obtained by accessing extra blocks. Unfortunately, using statistics is just what the best compression techniques do. On the other hand, we can hope that by exploiting local (w.r.t. a single disk block) statistics for compression, the compression ratio might be satisfactory as well.

We propose a solution to the compressed raster image handling problem that satisfies all of the above requirements to a large extent. In Section III, we discuss the basic issues in the design of a suitable compression method. We propose a compression algorithm in Section IV. Section V sketches the spatial access to the compressed image. In Section VI, we present the experimental results that we gained from an implementation of various compression algorithms and experiments with satellite images. Section VII concludes the paper.

III. Basic Compressed Raster Image Handling

Our examination of suitable compression methods has to take into account the way in which spatial indexing achieves its efficiency. Spatial indexing methods cover the data space with regular shaped regions. Whenever the spatial data are points or pixels on a regular grid, a partition of the data space will suffice; the regions of the space partition will typically be aligned rectangles. The data in one region are stored in one storage cluster on disk. A storage cluster may stand for one physical disk block or a sequence of consecutive physical blocks, depending on the space partition that is to be achieved (most database systems support storage clusters of different sizes). The essential ingredient is that locality in space should be mapped to locality on disk as much as possible. Ideally, all data in a storage cluster should be close in space.

To combine compression with spatial access, two clustering strategies seem applicable: a partition of the data space based on the original pixels, regardless of compression, or a clustering based on the compressed data. The former one may use any partition of the data space into rectangular regions, where each region serves as the two-dimensional (2-D) key to the data block(s) holding the compressed image data of the region. The most well-known instance of a data structure built on this principle is the quadtree and its variants [10]–[12]. Such a partition makes little sense for our setting: data compression may lead to some savings in storage space by reducing the number of physical blocks that a storage cluster corresponds to. We can, however, not get beyond a granularity of physical blocks, and therefore, the compression ratio is reduced intolerably. Only very large storage clusters would lead to acceptable compression ratio, but they, in turn, suffer from intolerably low geometric selectivity. From now on, we therefore identify a storage cluster with a physical disk block.

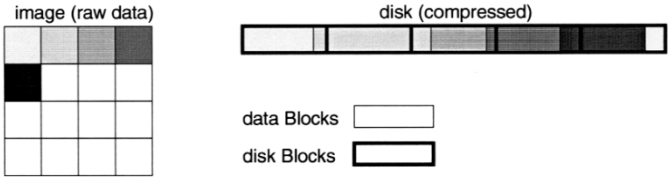


Fig. 1. Logical clustering.

The second strategy is therefore the only remaining alternative. It builds its regions according to the amount of compressed data that fill an entire block. However, this cannot be achieved by any fixed partition of the space of original pixels, because the compression ratio is not uniform across the image (see Fig. 1). Therefore, due to block boundary positions, even in the case in which the compressed data would fit into one block, two blocks may need to be accessed (see Fig. 1). Accessing extra blocks is harmless, if these blocks contain compressed pixels that are needed for the answer to the query anyway. In other words, it is helpful to preserve spatial locality of pixels not only inside blocks, but also beyond block boundaries, inside sequences of blocks. We propose to achieve this by scanning the image along an appropriate *space filling curve* (SFC) [13], [14]. The index regions—bounding boxes of data blocks—cover the area of the compressed data in one data block.

More precisely, to permit efficient clustering, spatial access, and lossless compression at the same time, an algorithm and data structure must obey several restrictions. First, there should not be any free space in a data block, because this would negatively affect the space usage, and therefore the compression ratio. Second, to keep the number of block accesses for (window-) queries low, every index region should smoothly map to a data block, and therefore the compressed data should not continue beyond data block boundaries. Third, the compression algorithm should not consider the statistics of pixels encountered in other data blocks when coding the elements of the current block's region. The reason is that this would lead to some extra block accesses for reconstructing this part of the image—a potential source of inefficiency especially for small query windows. Therefore, we restrict the compression scheme to local operations and disallow the use of statistics or history of other image parts.

Unfortunately, none of the known compression algorithms fulfills our requirements, neither the highly sophisticated nor the simpler ones. The newly proposed standard *CALIC* [15] achieves high compression ratios based on complex prediction functions and extensive context statistics (see also [16] for context based compression). For our problem, this interferes with the requirement of history-less operations. Another sophisticated method called *FELICS* [17] is also based on extensive image statistics. The current standard *JPEG* [18] has drawbacks in the need for prescanning the whole image to achieve good compression, and that a certain amount of statistics be maintained and stored with the compressed data. Furthermore, the prediction functions of CALIC, and more or less also those from JPEG and FELICS, depend on the knowledge of the location and value of some particular neighboring pixels. This spatial relation imposes restrictions on the traversal order of the pixels in the image, and it therefore makes the construction of well-shaped pixel regions impossible.

IV. A Spatial Compression Algorithm

As discussed in [19], lossless image compression algorithms usually view a pixel value as a random variable. A compression algorithm then typically consists of the following four processing steps:

1) select the location of the next pixel to be encoded (*pixel selection*);
2) compute a prediction of the value of the selected pixel from the history of already encoded pixels (*pixel value prediction*);
3) compute a prediction for the variance of that random variable (*variance estimation*);
4) encode the difference between the pixel value prediction and the actual pixel value, making use of the variance estimate (*prediction error coding*).

This scheme using differential prediction error encoding is often referred to as differential pulse code modulation (DPCM).

Since the prime goal of the compression is the support of spatial queries, the order of pixels in the sequence has to be chosen with care. The pixel sequence is cut into block-size portions. Each portion (subsequence) is stored in a data block and is therefore the unit of access. As a consequence, the geometric shape of the (bounding box of the) pixels in a block should go well with typical query populations such as squares or rectangles with no preferred direction on the average. In particular, it would be a poor choice to take the pixels in the order in which they appear along the sequence of rows in the picture (scan-line sequence). Other space filling curves (that treat the dimensions of space more uniformly) lead to better spatial selectivity. The problem of which linear order is best suited to accommodate spatial queries has been discussed extensively. There is a large body of literature on the subject from the database perspective [12], [14], [20] on the more experimental side, and from the data structure's perspective [10], [11], [21], [22], in addition to theoretical investigations that ask for lower and upper bounds on the number of consecutive sequences that a query might induce along a space filling curve [23], [24]. It turns out that whenever all possible windows (size, shape, and position) are equally likely, then all space filling curves are equally efficient for window queries on the average [22]. But since square shapes are more likely in practical applications than long and skinny windows, we might just as well pick a space filling curve that performs best in that case. This is true for space filling curves that are defined recursively, by refining the curve according to a simple rule in each of four quadrants, and by putting the four pieces together accordingly. The most well-known examples of theses curves are the Z-curve (also known as Morton encoding [25] among geographers or quadcode [10], [11] in the data structures world). A Z-curve on four quadrants is shown in Fig. 2(a), and the refinement step replaces each quadrant by a Z-curve on four quadrants and connects these, as shown in Fig. 2(b).

In spite of the popularity of the Z-curve in general, for our purpose the Hilbert curve [26], [27] appears to be more appropriate; it is defined recursively as well, with the four quadrant

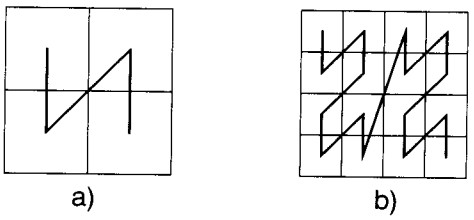


Fig. 2. Z-curve.

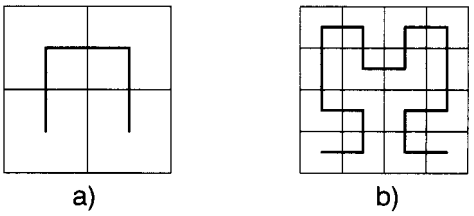


Fig. 3. Hilbert curve.

patterns rotated appropriately in the refinement step (see Fig. 3). Its advantage over the Z-curve is that it does not jump, in the sense that any two cells that are adjacent on the curve are also adjacent in space. We therefore expect it to induce quite well formed regions in space.

To see this in more detail, consult Fig. 4 that shows the data regions resulting from three different pixel sequences, where we assumed a block capacity of 17 pixels. The first four portions of 17 pixels each, i.e., sets of pixels 1 to 17, 18 to 34, 35 to 51, and 52 to 68, result in data regions A, B, C, and D that are shown in different shades and shapes. The Z-curve [Fig. 4(a)] and scan-line [Fig. 4(c)] sequences may produce data regions with multiple connected components. Furthermore, the Z-curve can have heavily overlapping regions (i.e., A and B in the example). The Hilbert-curve [Fig. 4(b)] indeed generates quite compact data regions. Additionally, it supports the prediction step quite well because of the spatial adjacency of consecutive pixels in the sequence. Note that compression along Hilbert curves has been proposed earlier [28]–[30], but with a different objective. In this paper, we focus on the interplay between compression and spatial access, since optimizing either of both alone does not support spatial queries in compressed images.

Prediction is usually based on neighboring pixels in space. However, the pixels used as predictors for any given current value have to be encoded before the current one, because the decompression process relies on the same predictor values. If the relative spatial positions of the neighbors always have the same relative index in the sequence, as it is the case for scan-lines, sophisticated combinations can be used to derive a close prediction (e.g., CALIC makes use of this fact). Space filling curves like the Z-curve or the Hilbert-curve do not conform to this behavior but are very useful for effective clustering. We therefore propose to calculate the prediction from a specific set of relative indexes (distances in the pixel sequence) of previously encountered pixels. However, the relative spatial positions of these pixels are not the same throughout the sequence. Furthermore, their influence based on the Euclidean distance differs from one space filling curve to another. Let us call the set of pixels used to predict the current pixel's value the *context*.

We propose to choose the context C_v of a pixel v to be the ordered collection of the last k pixels visited before coding the current one. We now face the problem of computing a prediction that is good in all the geometrically different situations that the context may describe. To get a close prediction $\hat{v}$, we propose the weighted sum of (1) over the context C_v. This prediction incorporates neighboring pixels according to the space filling curve (SFC), but does not rely on exact relative spatial locations. The weights w_j model an exponential function, decreasing with growing distance on the SFC, and depend on the size k of the context and on the index j of pixel v_j in the sequence. Thus the distribution of the weights w_j takes into account that pixels close in space, having smaller relative indexes on the SFC, will also have a bigger influence on the prediction.

$$\begin{aligned} w_1 &= 1 \\ w_j &= 3\left\lfloor \frac{w_{j-1}}{2} \right\rfloor + 2 \\ \hat{v} &= \left(\sum_{j=1}^{k} w_j \right)^{-1} \sum_{v_j \in C_v} w_j v_j \end{aligned} \tag{1}$$

To achieve good data reduction, an accurate probability distribution has to be selected for the current prediction error. The *Laplace* distribution of (2) is normally used as a basis for statistical coding of differential signals [4]. In this case, the best compression ratio for a value x is achieved with the Laplace distribution with variance x^2. This value x^2, however, cannot be used for compression, because it makes decompression impossible, since the value x is not known at decompression time. Therefore, we need a good estimate for the variance.

$$L(x) = \frac{1}{\sqrt{2\sigma^2}} \exp\left(-\sqrt{\frac{2}{\sigma^2}} |x - \mu| \right) \tag{2}$$

Naturally, our variance estimation uses the same context C_v as the prediction step, although other contexts might be considered too. Unlike other compression algorithms, however, we do not use this context as a key to retrieve information about earlier prediction errors within the same context. Thus no history or statistical information of occurrences of this context are used for variance estimation, we only use local operations on the values in C_v. We propose to calculate a variance approximation σ^2 according to (3) as a normalized sum of the quadratic prediction errors in the current context C_v.

$$\sigma^2 = \frac{1}{|C_v|} \sum_{v_j \in C_v} (v_j - \hat{v})^2 \tag{3}$$

The prediction errors $e = v - \hat{v}$ can now efficiently be coded with a statistical entropy coder, using the estimated variance σ^2 to calculate the probability distribution at the current position. A *Huffman* entropy coder [31] generates minimum redundancy codes which have a one-to-one relationship with the input symbols, and it also approximates the entropy bound for a given probability model. To reduce coding complexity, we use fixed precomputed Huffman tables for a specific set of variances presented in [19]. This set of variances introduced in [19] guarantees an unnoticeable loss in coding efficiency.

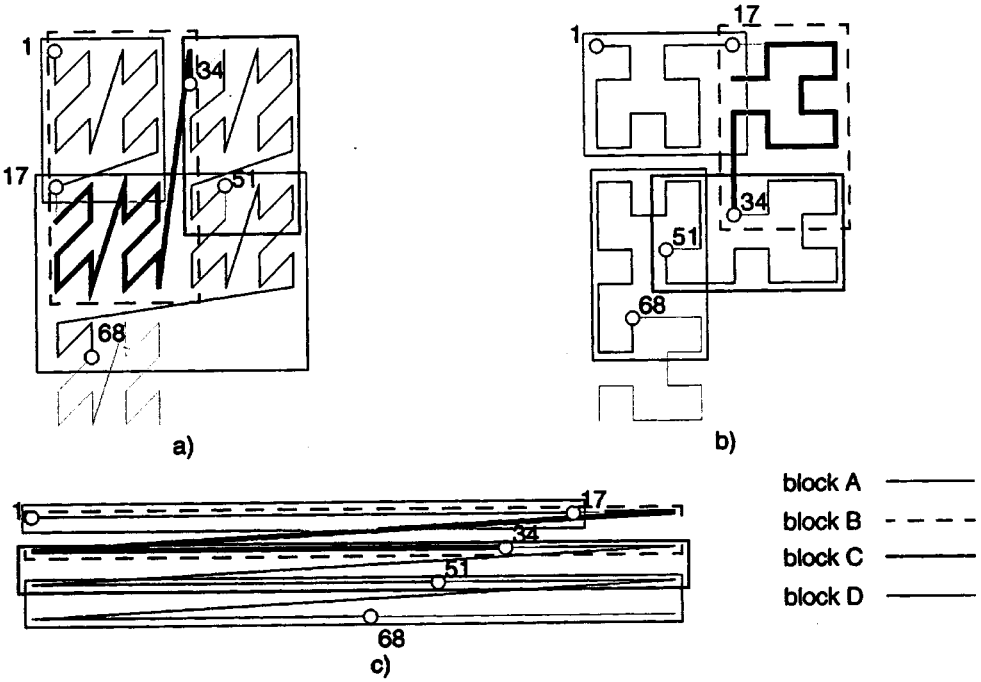


Fig. 4. Overlap of bounding boxes of space filling curve segments.

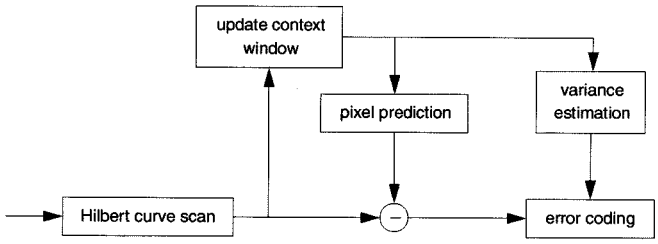


Fig. 5. Coding a pixel.

Fig. 5 shows all processing steps in a flowchart. Prediction and variance estimation are performed on a sliding window over the input sequence, the context C_v. Therefore, only local operations on a limited support area around the current picture element are used. The entropy coder which encodes the prediction errors makes use of precomputed tables to save calculation time. Given the variance estimation σ^2, the closest Huffman table is used to encode the prediction error e. The decoding process is the symmetrical inversion of the encoding process. For ease of reference, let us call this *Hilbert Compression*.

This algorithm runs fast even without code optimization. To calculate the prediction and variance estimation, only a few computation steps are necessary, $O(k)$ for a context of k pixels. Other algorithms tend to spend more time on complex prediction computation. The main work of the entropy coder is done once in the initialization step. The probabilities of symbols are computed for a specific set of variances, and for each such variance the Huffman tree is constructed. Therefore, a pixel is coded in $O(m)$ steps by outputting the m bits from the path in the Huffman tree. Not only time complexity is low, also the space requirements are small. The main part of the coding routine just uses an image buffer of size k to compute the prediction and variance estimation. Additionally, the precomputed Huffman trees need $O(nl)$ space, where n is the number of symbols in the alphabet and l is the number of variances used.

V. Spatial Access

To build a spatial index, our spatial compression algorithm Hilbert Compression cuts the sequence of compressed pixels of the raster image along the Hilbert curve into block size sequences. Due to the varying compression ratio, not all subsequences have the same length on the SFC. Each pixel subsequence defines a shape in space; its bounding box serves as the geometric key of its block (see Fig. 4). In the example shown in Fig. 4(b), the bounding boxes of subsequences A, B, C, and D are the geometric keys for maintaining the pixels numbered 1 to 17, 18 to 34, 35 to 51, and 52 to 68, respectively. As a consequence of the properties of the Hilbert curve, the shapes of these bounding boxes tend to be almost square and to have very limited overlap. To enable decompression of independent blocks, the context C_v must be known at the beginning of the pixel sequence of each block. This can either be achieved by storing the last context of the previous block with the data of the current block, or by starting each block with an empty context. Starting with an empty context will lead to bad compression for the first $k = |C_v|$ pixels, and storage of the last context requires $O(k)$ wasted space at the beginning of each block. However, both methods will insignificantly affect the compression ratio for typical parameters of $k < 10$ and block sizes of at least 1024 bytes.

In response to a rectangular window query, a pixel subsequence needs to be accessed only if the query rectangle geometrically intersects the bounding box of the subsequence. Note as an aside that even in that case, the pixel sequence does not

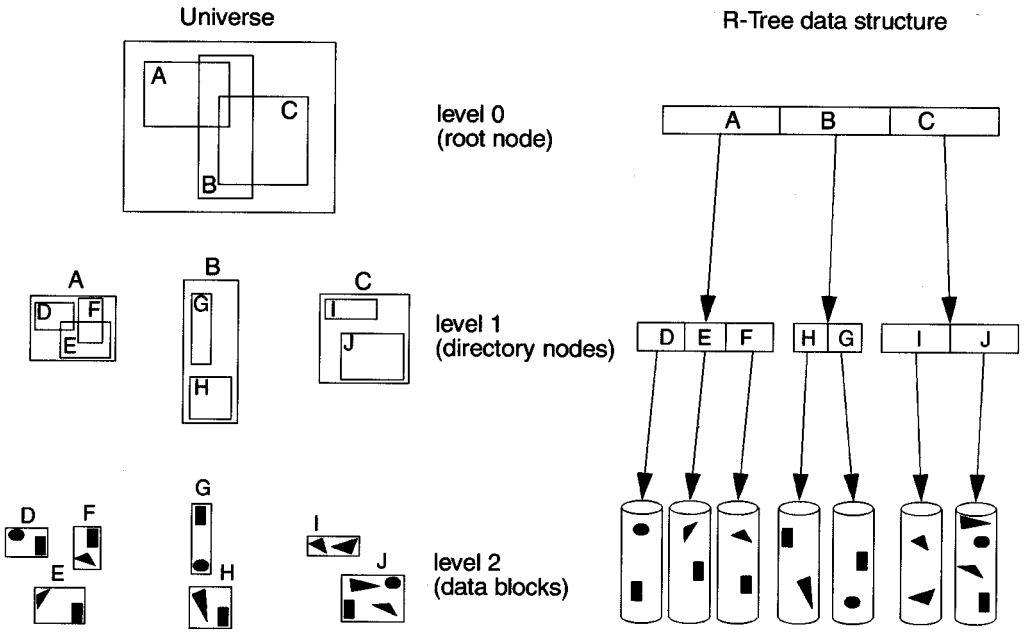


Fig. 6. R-Tree.

necessarily always contribute to the answer, because an intersection with the bounding box does not imply an intersection with the pixel sequence. In order to efficiently answer a window query, we therefore only need a way for quickly deciding which bounding boxes intersect the query rectangle. Since the number of bounding boxes is very large, the data structure that supports these rectangle intersection queries must be stored on external storage as well. This, however, is a classical problem in the field of spatial data structures, with an abundant literature and many worthwhile suggestions over the past decade [10], [11], [21], [22]. The spatial index is built using the bounding boxes of the data blocks as geometric keys. One simple and efficient structure that we feel suitable in our setting is the R*-Tree [8]. The R*-Tree is variation of the R-Tree [32], which itself is a generalization of a B-Tree [33]. The generalization concerns the dimensionality: Whereas in a B-Tree each node represents a one-dimensional interval, defined implicitly by the keys on the path from the root to the node, in an R-Tree each node represents a 2-D interval, i.e., an axis-parallel rectangle. For an example, see Fig. 6. In a B-Tree, the intervals of the nodes on a level of the tree partition the universe; therefore, a search for a key proceeds on a unique path from the root toward the leaves. In an R-Tree it is not possible to keep the rectangles of the nodes on a level disjoint. To support geometric selectivity, the R-Tree mechanism aims at keeping the overlap (and a few other measures) of index rectangles low, but depending on the data and the sequence of operations, a certain overlap is unavoidable. As a result, an exact search operation needs to visit not just one path in the tree in the worst case, but instead the search path may split (even more than once). Nevertheless, an R-Tree (and its R*-Tree version) appears to be the index structure of choice for the rectangles that Hilbert compression generates, among several others that might not necessarily perform much worse.

Now, spatially limited operations on the compressed raster image are performed by first retrieving the involved data blocks from the R*-Tree. Then the required image region can be reconstructed through decompression, and the image operations can be carried out locally. Note that a mapping needs to be computed between the position of a pixel in the compressed sequence, the SFC, and the spatial location of the pixel in image space. To do this, the spatial index maintains further information of the indexes on the SFC of the data blocks.

VI. Experiments

To evaluate the spatial selectivity, we implemented Hilbert Compression in C++ in an *ObjectStore*[2] database system. This implementation was designed to allow the use of different prediction and coding methods, as well as the use of different space filling curves. In the following experiments, we compared the spatial selectivity of the Hilbert curve and the scan-line approach when used for data clustering with fixed sized data blocks. Since we aimed at relating the spatial selectivity only to the choice of the space filling curve, we did not use any compression of the image data at all in these clustering experiments.

The input image for the spatial clustering and access tests was a 24 bit satellite image of 2334 × 2000 pixels (about 14 MB of disk space), covering a world-coordinate space of 70 × 60 km. Thus, one pixel represents approximately 30 × 30 m in the real world. The sequence of image pixels was cut into segments such that each fills a 4 KB data block in the database.

The fill-rate of a bounding box of a data block is defined as the number of pixels in the data block divided by the area, measured in pixels, of the bounding box. The higher this fill-rate is, the higher is the probability that this data block indeed contributes to the result of a window query that intersects the bounding box. The average fill-rate for the Hilbert curve was 71%, whereas the scan-line only achieved 41%. Another crucial aspect of spatial

[2]Registered trademark of Object Design, Inc.

TABLE I
SPATIAL OVERLAP OF BOUNDING BOXES

overlap	1	2	3
scan-line	0.1%	58.3%	41.6%
Hilbert	60.0%	38.8%	1.2%

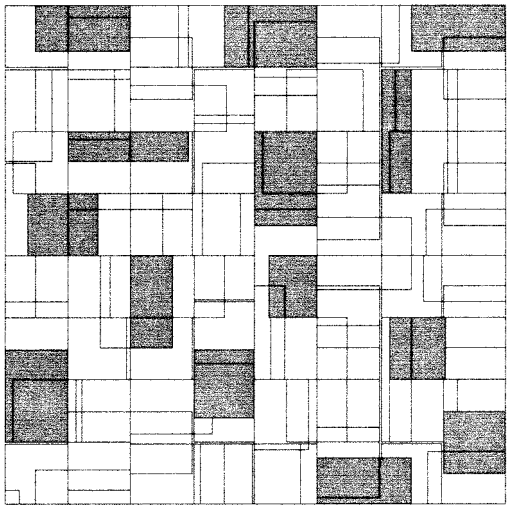


Fig. 7. Hilbert curve bounding boxes.

TABLE II
COMPARISON OF DIFFERENT COMPRESSION METHODS. SCENE1 (TOP) AND SCENE2 (BOTTOM) ARE MULTISPECTRAL LANDSAT-4 THEMATIC MAPPER SATELLITE IMAGES WITH SEVEN CHANNELS AND EIGHT BITS PER CHANNEL. SCENE1 HAS 5866×4782 PIXELS AND SCENE2 5807×5761. THE HILBERT COMPRESSION METHOD USED A CONTEXT OF SIX PIXELS AND IS DENOTED BY H 1.6

channel	gz	Z	calic	ljpg	H 1.6
1	1.70	1.66	2.30	2.08	2.11
2	2.00	2.02	2.72	2.46	2.44
3	1.80	1.79	2.44	2.18	2.18
4	1.27	1.18	1.70	1.56	1.52
5	1.23	1.15	1.68	1.54	1.51
6	16.02	4.77	4.59	6.58	3.27
7	1.54	1.50	2.09	1.91	1.87

channel	gz	Z	calic	ljpg	H 1.6
1	1.65	1.61	2.19	1.96	2.00
2	1.84	1.84	2.51	2.24	2.24
3	1.65	1.62	2.22	1.98	1.99
4	1.26	1.17	1.68	1.56	1.52
5	1.25	1.15	1.67	1.53	1.51
6	15.98	4.56	4.41	3.99	3.05
7	1.50	1.46	2.02	1.83	1.81

selectivity is the number of times a region, or pixel in our case, is covered by bounding boxes of different data blocks. The lower this number is, the fewer data blocks will have to be read for a given window query. Table I presents the overlap numbers for the scan-line and Hilbert space filling curves. Using the Hilbert curve, most of the pixels are only covered by one bounding box; about one-third of the pixels are covered by two bounding boxes, and only very few are overlapped by three or more bounding boxes. The scan-line sequence shows quite a different result for this test: most pixels are covered by several bounding boxes. Note that the fill-rate and bounding box overlap of the scan-line curve depends on the relative size of one data block compared to the width of the input image, whereas the Hilbert curve is independent of the data block size or the aspect ratio of the image.

In a second experiment, we verified these statistical properties with a set of actual window queries. The window queries were chosen to reflect a dynamic navigation and visualization system such as ViRGIS [34] which requests nonoverlapping image regions of constant and almost square shape from the image database. In our test, we queried the image database with 56 square regions of size 10 × 10 km, or about 333 × 333 image pixels, which covered the whole image space. Summed over all 56 window queries, the database which used a scan-line image sequence needed to read 23 313 data blocks—because the queries intersected 23 313 bounding boxes. On the other hand, the Hilbert curve database only had to read 4282 data blocks for the same 56 window queries. This also amounts to 95 MB read for the scan-line approach and only 17 MB using the Hilbert curve.

Fig. 7 illustrates that the bounding boxes created by the Hilbert curve have indeed an aspect ratio close to one, and that their overlap is distributed nicely over the universe. A few bounding boxes are shaded to support the illustration.

For comparing the raw compression ratio, we have implemented Hilbert Compression in C, based on the *Khoros* [39] image processing system. The other compression algorithms used to compare the compression efficiency are *gzip* (gz) and *compress* (Z), two well known general purpose data compression tools available on UNIX systems; *CALIC* [15] which is a very sophisticated image compression algorithm (we used an implementation of Wu *et al.* of the University of Western Ontario); and the quasi-standard *lossless JPEG* (ljpg) [35], where we used an implementation from Cornell University.

Our Hilbert Compression algorithm has been tested with an abundance of data from satellite sources such as LANDSAT and SPOT images. Several tests with derived RGB images and original (seven-channel) satellite images showed a similar relative behavior between the three image compression techniques CALIC, lossless JPEG, and Hilbert Compression. In the presented experiments we used a context size of six pixels for our Hilbert Compression. This context size turned out to be a good compromise between compression ratio and computation time. Table II shows typical results for compression ratios *original size: compressed size* for the different channels of two original LANDSAT satellite images. The complete raw data set for one of these images takes about 200 MB of disk space. Fig. 8 shows a small part of channel 5 of Scene1. Table III presents a comparison of compression ratios of several RGB satellite images.

The two compression algorithms gzip and compress are based on the *Lempel-Ziv* [36] coding algorithm, and were mainly designed for text compression. They perform well only on highly redundant pictures, such as the channel 6 of the satellite images Scene1 and Scene2. The CALIC algorithm achieves best compression ratios overall, even for complex image data. JPEG

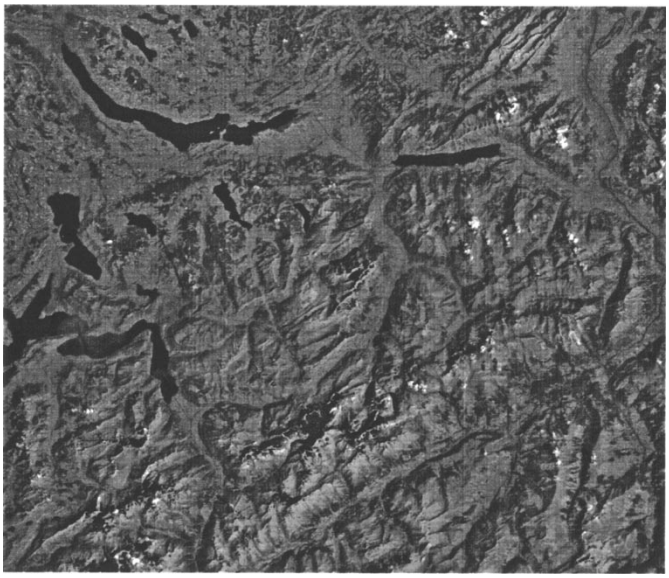


Fig. 8. LANDSAT satellite image of central Switzerland.

TABLE III
COMPARISON OF COMPRESSION RATIOS FOR DIFFERENT RGB IMAGES WHICH WERE DERIVED FROM LANDSAT AND SPOT (FELSBERG) SATELLITE IMAGES. THE HILBERT COMPRESSION METHOD USED A CONTEXT OF SIX PIXELS AND IS DENOTED BY H 1.6

image	gz	Z	calic	ljpg	H 1.6
aargau	1.17	1.02	1.37	1.29	1.29
felsberg	1.10	0.94	1.25	1.18	1.23
luzern	1.44	1.35	1.39	1.34	1.34
rigi	3.28	3.36	3.79	4.20	3.03

and Hilbert Compression provide similarly good compression results, and stay within about 10% of the CALIC results.

Note that, unfortunately, the CALIC and JPEG algorithms in their optimal form are not at all a good basis for a modification toward spatial access. CALIC needs a scan-line-like pixel sequence to accomplish its complex prediction function. Moreover, it needs considerable history information from traversed image regions to achieve its data reduction. A typical implementation maintains frequency counters for a few hundred specific contexts. Building efficient cluster regions would therefore lead to frequent restarts of the compressor which would reduce the compression ratio significantly as the global history is very important in this method. Instead of restarting the compressor, the history could be stored wherever needed to allow partial image reconstruction as mentioned in Section III. This would reduce the compression ratio due to the storage overhead incurred. Nevertheless, the implied scan-line sequence allows no efficient clustering of image data. The JPEG method needs information about global image statistics as well, and the imposed scan-line sequence has the same drawbacks as for the CALIC algorithm. Lossless JPEG could be adapted to use a better SFC and to apply our prediction function. However, the Huffman table would still have to be stored and precomputed from the overall image statistics. Furthermore, using the same encoding for the whole image is not adaptive to local variances in the image. In contrast, Hilbert Compression does not need any global history information, only a local context of a few pixels is used, and provides best support for spatial clustering. In a data block, after restarting the compressor, it achieves the optimal compression ratio after having processed only a few pixels. Therefore, partitioning the image into disk-block like (or bigger) clusters does not affect the overall compression ratio.

VII. CONCLUSION

In this paper, we considered the problem of window queries on compressed raster images. With the goal of avoiding decompression of the complete image before answering a query, we proposed a compression algorithm, together with a spatial index that supports window queries on the compressed image. Our compression method, Hilbert Compression, is oriented toward spatial clustering and permits decompression from local information alone. We have demonstrated experimentally that the Hilbert compression ratio typically is competitive with well known compression algorithms such as lossless JPEG or CALIC. Furthermore, we implemented an experimental image database that provides spatial access to compressed images. This image database can also be used as a texture server to our real-time terrain visualization system ViRGIS which is described in [34], [37], and [38].

We feel that the presented Hilbert Compression proposal is a first step toward an investigation of how to perform spatial operations (or other complex operations) directly on the compressed data, yielding a compressed image as the result, with no decompression along the way. Since the body of compressed raster images is growing steadily, and advanced applications call for increasingly complex operations (*IEEE Computer* dedicated its September 1995 issue to this topic), further investigations in this direction appear to be useful.

ACKNOWLEDGMENT

The authors would like to thank Dr. K. Seidel, National Point of Contact (NPOC), Swiss Federal Office of Topography for providing us the digital satellite images used to perform the experimental tests. We are grateful to the referees for suggestions that helped improve the paper.

REFERENCES

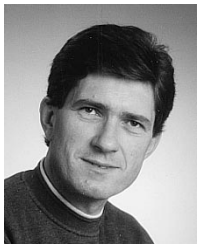
[1] H. Samet, "The quadtree and related hierarchical data structures," *Comput. Surv.*, vol. 16, pp. 187–260, June 1984.
[2] S. Sarawagi and M. Stonebraker, "Efficient organization of large multidimensional arrays," in *Proc. IEEE 10th Int. Conf. Data Engineering'94*, 1994, pp. 328–336.
[3] W. Kou, *Digital Image Compression: Algorithms and Standards*. Norwell, MA: Kluwer, 1995.
[4] A. N. Netravali and B. G. Haskell, *Digital Pictures: Representation, Compression and Standards*. New York: Plenum, 1995.
[5] R. Pajarola and P. Widmayer, "Pattern matching in compressed raster images," in *3rd South Amer. Workshop String Processing (WSP 1996)*, Ottawa, Ont., Canada, 1996, pp. 228–242.
[6] D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel, and J.-B. Yu, "Client–server paradise," in *Proc. 20th VLDB Conf.*, 1994, pp. 558–569.
[7] B. Becker, P. Franciosa, S. Gschwind, T. Ohler, G. Thiemt, and P. Widmayer, "Enclosing many boxes by an optimal pair of boxes," in *Proc. 9th Annu. Symp. Theoretical Aspects Computer Science (STACS)*. Berlin, Germany: Springer-Verlag, 1992, pp. 475–486.
[8] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proc. ACM SIGMOD Int. Conf. Management Data*, 1990, pp. 322–331.

[9] B. U. Pagel, H. W. Six, H. Toben, and P. Widmayer, "Toward an analysis of range query performance in spatial data structures," in *Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. Principles Database Systems*, 1993, pp. 214–221.
[10] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Reading, MA: Addison-Wesley, 1989.
[11] H. Samet, *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison Wesley, 1989.
[12] M. F. Worboys, *GIS: A Computing Perspective*. London, U.K.: Taylor & Francis, 1995.
[13] C. Faloutsos and S. Roseman, "Fractals for secondary key retrieval," in *Proc. ACM Conf. Principles Database Systems*, 1989, pp. 247–252.
[14] H. V. Jagadish, "Linear clustering of objects with multiple attributes," in *Proc. ACM SIGMOD Int. Conf. Management Data*, 1990, pp. 332–342.
[15] X. Wu, N. Memon, and K. Sayood, "A context-based, adaptive, lossless/nearly-lossless coding scheme for continuous-tone images," proposal for ISO compression standard, July 1995.
[16] J. Rissanen, "A universal data compression system," *IEEE Trans. Inform. Theory*, vol. IT-29, pp. 656–664, Sept. 1983.
[17] P. G. Howard and J. S. Vitter, "Fast progressive lossless image compression," *Image Video Compress.*, vol. 2186, pp. 98–109, 1994.
[18] G. K. Wallace, "Overview of the JPEG (ISO/CCITT) still image compression standard," *Image Process., Algorithms, Tech.*, vol. 1244, pp. 220–233, 1990.
[19] P. G. Howard, "The Design and Analysis of Efficient Lossless Data Compression Systems," Ph.D. dissertation, Dept. Comput. Sci., Brown Univ., Providence, RI, 1993.
[20] I. Kamel and C. Faloutsos, "Hilbert r-tree: An improved r-tree using fractals," in *Proc. 20th VLDB Conf.*, 1994, pp. 500–509.
[21] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Comput. Surv.*, vol. 30, pp. 170–231, June 1998.
[22] J. Nievergelt and P. Widmayer, "Spatial data structures: Concepts and design choices," in *Algorithmic Foundations of Geographic Information Systems*, M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, Eds. Berlin, Germany: Springer-Verlag, 1997, pp. 153–197.
[23] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer, "Space filling curves and their use in the design of geometric data structures," *Theoret. Comput. Sci.*, vol. 181, pp. 3–15, 1997.
[24] E. Bugnion, T. Roos, R. Wattenhofer, and P. Widmayer, "Space filling curves versus random walks," in *Proc. Algorithmic Foundations Geographic Information Systems*, M. van Kreveld, J. Nievergelt, T. Roos, and P. Widmayer, Eds. Berlin, Germany: Springer-Verlag, 1997.
[25] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," Tech. Rep., IBM, Ottawa, Ont., Canada, 1966.
[26] D. Hilbert, "Über die stetige Abbildung einer Linie auf ein Flächenstück," *Math. Ann.*, pp. 459–460, 1891.
[27] G. Peano, "Sur une courbe qui remplit toute une aire plane," *Math. Ann.*, pp. 157–160, 1890.
[28] S. Kamata, E. Kawaguchi, and A. Perez, "An arithmetic coding model for compression of LANDSAT images," *Vis. Commun. Image Process.*, vol. 1605, pp. 879–884, 1991.
[29] A. Lempel and J. Ziv, "Compression of two-dimensional data," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 1–8, Jan. 1986.
[30] N. D. Memon and K. Sayood, "Lossless image compression: A comparative study," in *Proc. SPIE Conf. Electronic Imaging*, 1995, pp. 8–20.
[31] D. A. Huffman, "A method for the construction of minimum redundancy codes," in *Proc. Inst. Electron. Radio Eng.*, 1952, pp. 1098–1101.
[32] A. Guttman, "A dynamic index structure for spatial searching," in *Proc. ACM 13th ACM SIGMOD Int. Conf. Management Data*, 1984, pp. 47–57.
[33] R. Bayer and C. McCreight, "Organization and maintenance of large ordered indexes," *Acta Inform.*, vol. 1, no. 3, pp. 173–189, 1972.
[34] R. Pajarola, T. Ohler, P. Stucki, K. Szabo, and P. Widmayer, "The alps at your fingertips: Virtual reality and geoinformation systems," in *Proc. IEEE 14th Int. Conf. Data Engineering (ICDE'98)*, 1998, pp. 550–557.
[35] G. K. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, vol. 34, pp. 30–44, Apr. 1991.
[36] A. Lempel and J. Ziv, "A universal algorithm for sequential data compression," *IEEE Trans. Inform. Theory*, vol. 23, pp. 337–343, May 1977.
[37] R. Pajarola, "Access to large scale terrain and image databases in geoinformation systems," Ph.D. dissertation, Dept. Comput. Sci., ETH Zürich, Switzerland, 1998.
[38] R. Pajarola and P. Widmayer, "Virtual geoexploration: Concepts and design choices," *Int. J. Comput. Geometry Applicat.*, to be published.
[39] "Khorus image processing system," [Online] ftp://ftp.khorus.unm.edu.

Renato Pajarola received the Ph.D. degree in computer science in 1998 from the Swiss Federal Institute of Technology, Zürich, Switzerland.

For one year, he was a Postdoctoral Researcher and part-time Faculty Member with the Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta. He is currently an Assistant Professor with the Department of Information and Computer Science, University of California, Irvine. His research interests include image and geometry compression, geometric data structures, 3-D graphics, multiresolution modeling, and visualization.

Dr. Pajarola is a member of the IEEE Computer Society and the ACM Special Interest Group on Computer Graphics.



Peter Widmayer received the Ph.D. degree from Karlsruhe University, Karlsruhe, Germany.

He was with the IBM T. J. Watson Research Center, Yorktown Heights, NY, for one year. He was a Professor of computer science at Freiburg University, Germany. He is now a Professor at ETH Zürich, Switzerland.