Spatial Indexing into Compressed Raster Images: How to Answer Range Queries Without Decompression

Renato Pajarola Peter Widmayer
Department of Computer Science
Institute of Theoretical Computer Science
ETH Zurich, Switzerland

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 to perform queries directly on the compressed version of the image. We suggest a compression technique that allows for the subsequent use of a data structure to guide a spatial search. In response to a range query, our algorithm delivers a compressed partial image. Experiments show that the new algorithm supports spatial queries on satellite images efficiently. In addition it is even competitive in terms of the compression that it achieves.

1. Introduction

Due to the enormous data volume, the efficiency of operations is still a major bottleneck in systems that handle large raster images, such as geographic information systems (GIS). A single satellite image, for instance, with its multiple data channels and high resolution quality, is easily as large as several hundred megabytes. On the one hand, efficient access is possible by storing raster data in one of several data structures, for instance quad trees [13, 14], or by ordering the raster data according to expected access patterns [15]. On the other hand, efficient access alone is not what we need: in addition, we would like to represent the data as succinctly as possible.

Raster image compression deals with the problem of finding 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 [11]. For archiving a raster image or for sending 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; in addition, 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. As raster images become larger and larger, partly because satellite snapshots can be glued together into one image easier and easier, partly because the resolution increases, it becomes less likely that even a significant 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.

A typical application where access into a compressed image appears to be efficient is the scenario where a server has to answer range queries in a satellite image archive over a wide area network (e.g. the internet). Here, the transmission over the network is usually the biggest bottleneck. With an appropriate compression and access method, the server should retrieve a part of the compressed image that contains the query range, and it should send this part of the image without decompression. The decompression should then be performed on the client side after the transmission.

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 Paradise system [4], 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 for efficiency: Large tiles return many redundant data in response to a range query, and small tiles give bad compression ratio, where tile size varies from 8 KB (very small) to 512 KB (very large) of data. We propose not to just fix tile sizes in advance, but instead to use a mechanism that effectively partitions the data adaptively in a way that is useful for both, good compression and efficient indexing.

2. The problem

We study the problem of answering a range query on a compressed version of a raster image, without decompressing the entire image quickly. 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 keeping all information that has been acquired via satellite. The range query serves as a particular, but prototypical spatial query. Ideally, we would like to find a raster data structure and compression technique with the following properties:

1. The pixels in the same storage block are close in space; more generally, the physical storage locations of pixels reflect the spatial locations.
2. A range query can be answered by accessing just those storage blocks that contain the compressed version of the data in the range.
3. The compression ratio is as good as that of any other well-known 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; see, for instance, the criteria for efficiency of spatial access structures based on aligned rectangles ([1, 2, 12]).

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 will be unable to fulfill requirement 2. Unfortunately, this is just what the best compression techniques do. On the other hand, we can hope that by exploiting local (w.r.t. a single storage block) statistics for compression, the compression ratio might be satisfactory.

We propose a solution to the compressed raster image handling problem that satisfies all of the above requirements. In Section 3, we discuss the basic issues in the design of a suitable compression method. We propose a compression algorithm in Section 4. Section 5 sketches the spatial access to the compressed image. In Section 6, we present the experimental results that we gained from an implementation of various compression algorithms and experiments with satellite images. Section 7 concludes the paper.

3. Basics of 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. The essential ingredient is that locality in space should be mapped to locality on the storage medium as much as possible. Ideally, all data in a storage cluster should be close in space. A storage cluster may stand for one physical disk block or several of these blocks, depending on the space partition that is to be achieved (most database systems support storage clusters of different sizes).

To combine compression with spatial access, two clustering strategies seem applicable: a partition of the data space based on the original pixels, 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 key to the data block(s) holding the compressed image data of the region. The second strategy builds its regions according to the amount of compressed data that fill one or more entire data blocks. However, any fixed partition of the space of original pixels leads to regions whose compressed data may not fill entire data blocks, because the compression ratio is not uniform all over the image (see Figure 1). Therefore, in a case in which the compressed data fit into one block, two blocks may need to be accessed. We propose to avoid some (but not all) of this inefficiency by scanning the image along an appropriate *Space Filling Curve* (SFC) [5, 9] and creating index regions – bounding boxes of data blocks – which cover the area of the compressed data in one data block.

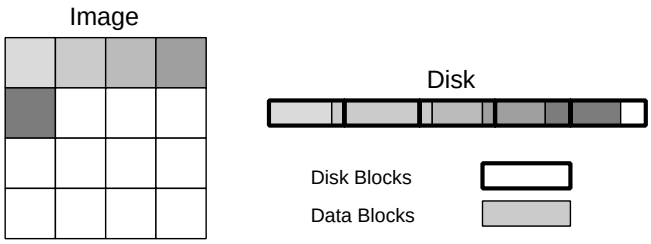

Figure 1. Logical Clustering

To permit efficient clustering, spatial access and lossless compression at the same time, an algorithm and data structure have to obey several restrictions. First, there should not be any free space in a data block, because this would affect the space usage and therefore the compression ratio negatively. Second, to keep the number of block accesses for (range-) 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 take the statistics of pixels encountered in other data blocks into account when coding the elements of the current block's region. The rea-

son 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 ranges. 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 more simpler ones. The newly proposed (highly sophisticated) standard *CALIC* [19] achieves high compression ratio based on complex prediction functions and extensive context statistics; for our problem, this collides with the requirement of history-less operations. Another sophisticated method called *FELICS* [7] is also based on image statistics, and the current standard *JPEG* [17] 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 some particular neighboring pixels; this imposes restrictions on the traversal order of the pixels in the image, and it therefore makes the construction of well-shaped pixel regions impossible.

4. A spatial compression algorithm

As discussed in [6], lossless image compression algorithms usually view a pixel value as a random variable. An algorithm then typically consists of four processing steps: first, select the location of the next pixel to be encoded (*pixel selection*); second, compute a prediction of the value of the selected pixel from the history of already encoded pixels (*pixel value prediction*); third, compute a prediction for the variance of that random variable (*variance estimation*); fourth, encode the difference between the pixel value prediction and the actual pixel value, making use of the variance estimate (*prediction error coding*).

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, while each portion (subsequence) is stored in a 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. Hence, taking the pixels in the order in which they appear along the sequence of rows in the picture (scan-line sequence) is poorly adapted to typical queries. More balanced space filling curves (as to the dimensions of space) lead to better spatial selectivity. Figure 2 shows the resulting data regions for three different pixel sequences with equal block capacities of 17 pixels; the first four resulting data regions (A, B, C and D) are shown in different shades and shapes. The Z-curve (Figure 2 a), which can have heavily overlapping regions (i.e. A and B), and scan-line (Figure 2 c) sequences may even produce data regions with multiple connected components. For the pixel sequence we propose to use the Hilbert-curve, since it generates quite compact data regions, and it also supports the prediction step quite well.

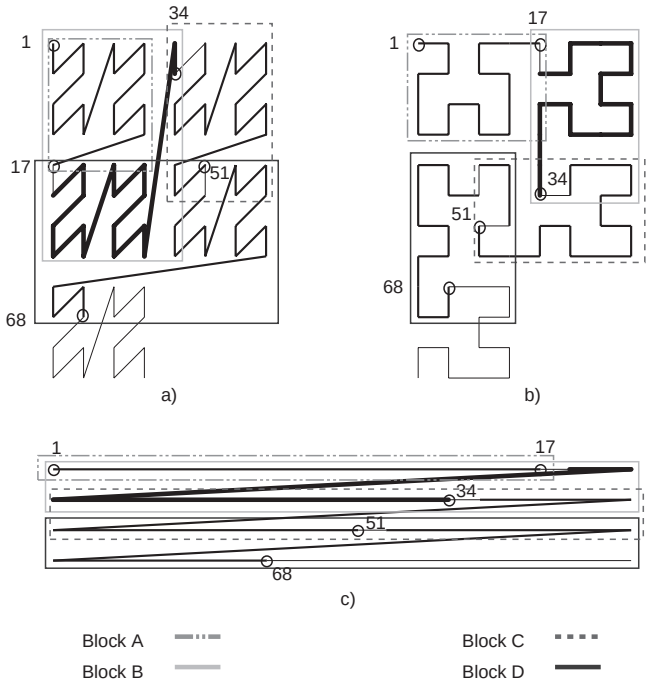

Figure 2. Clustering Sequences

Prediction is usually based on neighboring pixels in space. However, the pixels used as predictors for the current value have to be encoded before the current one, because the decompression process has to use 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 (CALIC makes use of this fact, e.g.). Space filling curves like the Z-curve or the Hilbert-curve don't conform to this behavior but are useful for effective clustering. We therefore propose to calculate prediction from a specific set of relative indices (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 describes. To get a close prediction $\hat{v}$, we propose the weighted sum of equation 1 over the

context C_v; this incorporates neighboring pixels according to the space filling curve (SFC), but doesn't rely on exact relative locations. The weights w_j resemble 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 which also have smaller relative indices on the SFC will also have a bigger influence on the prediction.

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

To achieve good data reduction, an accurate probability distribution has to be selected for the current prediction error. The *Laplace* distribution of equation 2 is normally used as a basis for statistical coding of differential signals [11, page 225]. In this case, the best compression ratio for a value x is achieved with the Laplace distribution with variance x^2. This value, however, cannot be used for compression, because it makes decompression impossible, since the value x isn't known at decompression time. Therefore, we need a good estimate for the variance.

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

Naturally, our variance estimation uses the same context C_v as the prediction step. 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 equation 3 as a normalized sum of the quadratic prediction errors in the current context C_v.

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

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. Note that *arithmetic* coding [3] would attain the theoretical entropy bound of the compression ratio for a given probability model. However, the lack of an exact link between the symbols of the input sequence and the bits of the output binary rational number would interfere with the division of the compressed data into many different data blocks. A *Huffman* entropy coder [8] 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 model. To reduce coding complexity, the Huffman tables are only computed for a specific set of variances, as suggested e.g. in [6].

Figure 3 shows these processing steps in a flowchart. Prediction and variance estimation are performed on a sliding window over the input sequence, where 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 precalculated tables to save calculation time. The decoding process is the symmetrical inversion of the encoding process. For ease of reference, let us call this *Hilbert Compression*.

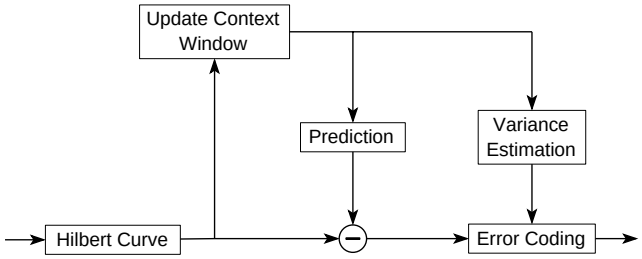

Figure 3. Coding a Pixel

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), whereas 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, the precalculated Huffman trees need $O(nl)$ space, where n is the number of symbols in the alphabet and l is the number of variances used.

5. Spatial access

Our spatial compression algorithm, Hilbert Compression, cuts the sequence of pixels of the raster image along the Hilbert curve into block size sequences, where due to compression not all subsequences have the same length on the curve. Each pixel subsequence defines a shape in space; its bounding box serves as the geometric key of its block (see Figure 2). In the example shown in Figure 2 b), the bounding boxes of subsequences A, B, C and D are the ge-

ometric 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.

In response to a rectangular range 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 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 range 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 [13, 14]. One simple and efficient structure that we feel suitable in our setting is the R*-tree [2]. Most of the other spatial data structures for aligned rectangles will also perform efficiently, due to the nature of the bounding boxes that Hilbert Compression generates. Now, operations on the compressed raster image then amount to the corresponding operations in the spatial data structure, with an additional mapping that needs to be computed between the position of a pixel in the compressed sequence and the location of the pixel in space.

6. Experimental results

We have implemented Hilbert Compression (H.1.6 in Table 1) in C, based on the *Khoros* [1] image processing system. The other compression algorithms used to compare the compression efficiency are: *gzip* (gz) and *compress* (Z), two well known general purpose compression tools available on UNIX systems; *CALIC* [19] which is a very sophisticated image compression algorithm (we used an implementation of Xiaolin Wu et al. of the University of Western Ontario); and the quasi standard *lossless JPEG* (ljpg) [18], where we used an implementation from Cornell University.

Our Hilbert Compression algorithm has been tested with lots of data from satellite sources such as *LANDSAT* and *SPOT* pictures. Several tests with derived RGB images and original (7 channel) satellite images showed all the same relative behavior between the three image compression techniques CALIC, lossless JPEG and Hilbert Compression, which used a context size of 6 pixels. Table 1 shows typical results for compression ratios *original size : compressed size* for different channels of LANDSAT satellite images. The scenes are multi-spectral images with seven different channels, each of 8 bit resolution, and the sizes are about 5800×4700 picture elements; this takes about 200 megabytes of data on disk. Figure 4 shows a small part of channel 5 of one scene, which was hard to compress.

The two compression algorithms gzip and compress are based on the *Lempel-Ziv* [10] coding algorithm. They perform well only on highly redundant pictures, such as in the present test for channel 6 of the satellite images, and sometimes the compression ratio falls below usable values, such as for the channels 4 and 5. The CALIC algorithm achieves very good compression ratios overall even for complex image data. JPEG and Hilbert Compression have quite similar results and stay within about 10% of the CALIC results for the critical images.

Note that unfortunately, the CALIC and JPEG algorithms in their optimal form are not at all a good basis for a modification towards spatial access. CALIC needs a scan-line like pixel sequence to accomplish its complex prediction function and needs considerable history information – frequency counts – to achieve its data reduction (a typical implementation needs 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 history is very important. Instead of restarting the compressor, the history could be stored wherever needed to allow partial image reconstruction as mentioned in Section 3; this however, would also reduce the compression ratio. JPEG also needs information about image statistics and the scan-line sequence has the same drawbacks as for the CALIC algorithm. Lossless JPEG could be adapted to use a better SFC and our prediction function, but the Huffman table would still have to be stored and precalculated from the complete image statistics. In contrast, Hilbert Compression does not need much history information, only a few pixels are needed after restarting the compressor to achieve best compression ratio. Therefore, using disk-block like (or bigger) cluster sizes the compression ratio will decrease minimally whatever cluster size is chosen.

7. Conclusion

We considered the problem of range queries on a compressed raster image. With the goal of avoiding a decompression of the complete image before querying, we proposed a compression algorithm, Hilbert Compression, that is oriented towards spatial clustering and that permits decompression from local information alone. We have demonstrated experimentally that Hilbert compression ratio typically is competitive with well known compression algorithms such as lossless JPEG or CALIC. We feel that our proposal is a first step towards an investigation of how to perform spatial operations (or other complex operations)

[1] available via anonymous ftp: `ftp.khoros.unm.edu`

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

Table 1. Compression Ratios for two different LANDSAT Scenes

directly on the compressed data. Since the body of compressed raster image is growing steadily, and advanced applications call for increasingly complex operations (the IEEE Computer dedicated its September 1995 issue to this topic), further investigations in this direction appear to be useful.

8. Acknowledgement

We want to thank Jürg Nievergelt and Guy Even for inspiring discussions, and Klaus Seidel, National Point Of Contact (NPOC) *Bundesamt für Landestopographie*, for providing us with satellite image data.

References

[1] B. Becker, P. Franciosa, S. Gschwind, T. Ohler, G. Thiemt, and P. Widmayer. Enclosing many boxes by an optimal pair of boxes. In *Proc. of the 9th Annual Symposium on Theoretical Aspects of Computer Science STACS*, volume 577, pages 475–486, Cachan, 1992. Springer-Verlag. Lecture Nores in Computer Science.

[2] 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. of the ACM SIGMOD Int. Conf. on the Management of Data*, pages 322–331. ACM, 1990. Atlantic City, New Jersey.

[3] J. G. Cleary, R. M. Neal, and I. H. Witten. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[4] D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel, and J.-B. Yu. Client-server paradise. In *Proceedings of the 20th VLDB Conf.*, pages 558–569, September 1994. Santiago - Chile.

[5] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *Proc. of the ACM Conf. on the Principles of Database Systems*, pages 247–252. ACM, March 1989.

[6] P. G. Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems*. PhD thesis, Department of Computer Science at Brown University, 1993.

[7] P. G. Howard and J. S. Vitter. Fast progressive lossless image compression. In *Image and Video Compression*, volume 2186, pages 98–109. SPIE, 1994.

[8] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proc. Inst. Electr. Radio Eng.*, pages 1098–1101, September 1952.

[9] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 332–342. ACM, June 1990.

[10] A. Lempel and J. Ziv. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, pages 337–343, May 1977.

[11] A. N. Netravali and B. G. Haskell. *Digital Pictures: Representation, Compression and Standards*. Plenum Press, New York and London, second edition, 1995.

[12] B. Pagel, H. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *Proc. of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 214–221, Washington DC, 1993.

[13] H. Samet. *Applications of Spatial Data Structures: computer graphics, image processing, and GIS*. Addison Wesley Publ. Co., Massachusetts, 1989.

[14] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley Publ. Co., Massachusetts, 1989.

[15] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. 10th Int. Conf. on Data Engineering '94*, pages 328–336, Houston, Texas, February 1994. IEEE.

[16] K. Szabo, P. Stucki, P. Aschwanden, T. Ohler, R. Pajarola, and P. Widmayer. A virtual reality based system environment for intuitive walk-throughs and exploration of large-scale tourist information. In *Enter95*, January 1995.

[17] G. K. Wallace. Overview of the JPEG (ISO/CCITT) still image compression standard. In *Image Processing Algorithms and Techniques*, volume 1244, pages 220–233. SPIE, February 1990.

[18] G. K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, April 1991.

[19] X. Wu, N. Memon, and K. Sayood. A context-based, adaptive, lossless/nearly-lossless coding scheme for continuous-tone images, July 1995. Proposal for ISO compression standard.

[2] ©ESA / Eurimage

Figure 4. Satellite Image of Central Switzerland[2]