

Short Paper

A Quadtree-based Progressive Lossless Compression Technique for Volumetric Data Sets

GREGOR KLAJNSEK, BORUT ZALIK, FRANC NOVAK* AND GREGOR PAPA*

Faculty of Electrical Engineering and Computer Science

University of Maribor

Smetanova 17, 2000 Maribor, Slovenia

**Jozef Stefan Institute*

Jamova 39, 1000 Ljubljana, Slovenia

An efficient technique for progressive lossless compression of volumetric data is described. It is based on the quadtree data structure and exploits the expected similarities between the neighboring slices. The proposed approach has better compression ratio than the octree method. A small amount of memory is required since only two slices need to be located in the memory at a time, which makes it suitable for hardware implementation. A built-in compression in volumetric scanners could considerably reduce the transfer and storage requirements for volumetric medical data.

Keywords: volumetric data, data compression, progressive visualization, lossless data compression algorithm, hardware implementation

1. INTRODUCTION

Volumetric data (also called voxel data) are becoming more and more popular form of representation of interiors of 3D geometric objects. Their elemental particle voxel is a volume element, which represents a value in 3D space. It is analogous to a pixel, which represents data in 2D space.

In different applications in medicine, biology, physics and geology, voxel representation is used for exploration and visualization of volumetric data. In medicine, for example, computerized tomography (CT) or magnetic resonance imaging (MRI) scanners generate data as a set of slices – two dimensional raster images. Vertical mesh slices are used for seismic analysis in structural geology. Physical phenomena like fluid simulation or terrain erosion modeled by fractals are also often presented by slices. A perfect description of the examined object is obtained by arranging the slices one after another along the spatial axis. Up to now, the main focus has been on the visualization of the data sets [1]. Different algorithms were developed (*i.e.*, ray casting [2], shear-warp factorization [3]). Due to the significant progress of computer technology, real-time visualization of the voxel data has become possible recently [4].

The problems that still remain in practice are related to storage of voxel data and its transfer over networks. A typical voxel data set may occupy a few hundred MB of disk

Received October 5, 2006; revised February 27, 2007; accepted June 11, 2007.

Communicated by Liang-Gee Chen.

space, which makes data compression a necessity. Usage of general-purpose compression techniques such as ZIP could be the first option. However, these techniques do not exploit particular properties of voxel data such as coherence of the data and homogeneity of individual regions, which could significantly improve the efficiency of the compression. Besides, they do not support progressive transmission and visualization of data. Consequently, development of special-purpose compression algorithms adapted to voxel data is needed.

The compression of volume data with quantization was introduced by Hesselink and Ning in 1992 [5]. They used a very simple approach that is based on the reduction of the number of bits used for a representation of a value of single voxel. Clearly, this is a lossy compression method. In 1993 Muraki introduced the usage of wavelet coding as an alternative way of describing voxel data [6], but he did not think of it as a compression technique. Despite this, his approach can be regarded as an important contribution towards the development of new compression methods for voxel data. In 1997 Zhu, Machiraju, Fry and Moorhead presented a method that at first divides the voxel space into subspaces using octree structure [7]. This method is lossless and its additional advantage is that it can also easily be transformed into a lossy method with controllable quality. Many authors have later used this technique in connection with visualization algorithms. In 2001 Ma and Shen described the usage of this method for compression of time-varying (dynamic) volume data [8].

The above methods can be roughly classified as vector quantization and octree compression. Vector quantization is a lossy method and hence not appropriate for applications where volumetric data needs to be visualized to the smallest detail (*e.g.*, in medical diagnostics). Octree compression is based on recursive uniform division of volumetric space into subspaces. In order to perform the compression, the whole voxel space has to be loaded into memory, which represents a serious drawback in practice.

In this paper, an alternative approach to lossless compression of the volumetric data is presented. It is based on the quadtree data structure [9] and exploits the expected similarities between the neighboring slices. In video compression [10] the interframe difference trees are used for compressing the video stream. However, the applicability is different due to the fact that video compression is a lossy method and does not support progressive visualization. The proposed approach is similar to the work of F. F. Rodler [11] in that it compresses volumes using a hierarchical (Harr/quadtree) decomposition of individual slices of a volume with encoding that takes advantage of coherence with neighboring slices. While Rodler's technique is indeed more sophisticated supporting random access and compression with an added thresholding step, the algorithm is rather complex which makes it difficult to implement in hardware. In our approach, a small amount of memory is required since only two slices need to be located in the memory at a time, which makes it suitable for hardware implementation. Besides, Rodler's approach is lossy, which restricts its use in some applications in practice (*i.e.*, medical diagnosis).

2. QUADTREE COMPRESSION

In octree compression, volumetric data set is associated with a cube recursively split into subcubes up to the smallest nondivisible element of a digital volume – voxel. By analogy, data structure in the quadtree compression refers to a square area recursively

divided into subsquares up to the smallest nondivisible element of a digital image - pixel. The third dimension is achieved by processing the subsequent series of slices representing the given 3D object. One of the main advantages of the octree method is the very efficient determination and encoding of homogeneous areas. The technique also has some disadvantages. The whole dataset needs to be loaded into the memory for processing and compression and most algorithms even require datasets with dyadic (power-of-two) resolution.

Our goal was to develop an algorithm for lossless compression of volumetric data that would try to overcome all the above disadvantages. The proposed quadtree based compression algorithm keeps most of the advantages of the octree based compression (efficient determination of homogenous areas, progressive reconstruction). In addition, we were looking for a solution with relatively modest resource requirements for possible implementation in a CT or MRI scanner.

In quadtree compression, the input data consist of a set of uncompressed slices. The image area of each slice is divided into four regions or blocks and the resulting areas are recursively further divided. At the lowest level, a block consists of $8 \times 8 = 64$ pixels. The homogeneity of each block at the lowest level is determined and the average value of pixels is calculated. A block is homogenous if all its pixels have equal values.

The image area of a slice is associated with a quadtree; it is a tree with four branches at each non-leaf node, and is generated by the following bottom-up procedure:

- the blocks at the lowest level are associated with the leaves of the quadtree,
- a node of an upper level of the quadtree (father) is obtained by associating the corresponding four neighboring blocks (sons) of the lower level,
- if the sons are homogenous and have equal average values of pixels, they are deleted (quadtree pruning) and the father becomes a leaf of the quadtree.

Data structure for storing the quadtree of a slice refers to a full quadtree template (FQT). A FQT consists of records corresponding to the nodes of a full quadtree. Each record carries information about the position, size, father and sons of a quadtree node. The nodes are identified by a unique index.

Consider a simple illustrative example shown in Fig. 1. A slice is depicted in Fig. 1 (a), the greyscale mapping in Fig. 1 (b) and FQT is partly sketched in the upper part of Fig. 1 (c). For convenience, the size of the slice is 16×16 pixels and the block at the lowest level has 4×4 pixels. The first record of FQT (the root) has the size of the whole slice, it has no father, its sons are nodes 2, 3, 4, 5. Similarly, node 2 has the size 8×8 , its father is node 1, its sons are nodes 6, 7, 8, 9. The position of each node is referenced to the left bottom corner of the slice.

The data of the slice is organized in a data structure shown in Fig. 1 (c) below the FQT. Each record contains: index, node type, average value, and pixel set pointer. Index refers to the corresponding record in the FQT. Node types are: father nodes (type 1) and leaves (types 2 and 3). Leaves of type 2 correspond to the homogenous blocks. Leaves of type 3 correspond to the non-homogenous blocks at the lowest level. Average value is the level of the grey scale mapping corresponding to the average value of pixels contained in the block. Pixel set pointer points to the pixel values of non-homogenous blocks at the lowest level that are stored separately.

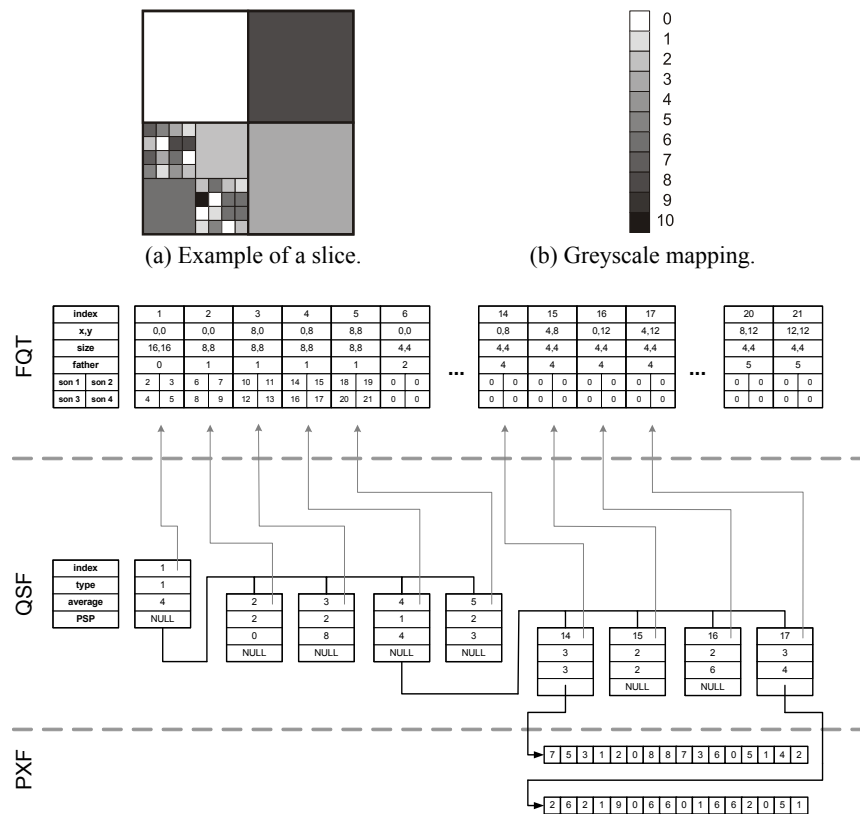


Fig. 1. Organization of the data structure.

The distinction between node types 2 and 3 is done because it is convenient in practice to store the quadtree data structure in two separate files:

- Quadtree Structure File (QSF) describing the upper levels of the quadtree, including the leaves of the homogenous blocks,
- Pixel Data File (PXF) describing pixel values of the non-homogenous blocks at the lowest level.

For a brief overview of the acquired data sets, only the first few levels in a progressive reconstruction are sufficient hence only QSF file is required. For the full reconstruction, both QSF file and PXF file are processed.

2.1 The Algorithm

The compression algorithm, suitable for hardware implementation consists of the following steps:

- Step 1: Initialization.** The first slice s_i is loaded and its quadtree is generated and stored in QSF file. If s_i includes non-homogenous leaves at the lowest level they are stored in PXF file.
- Step 2: Processing.** The slice s_{i+1} is loaded and the corresponding quadtree is generated. The difference quadtree $DQ_{s_i, s_{i+1}}$ is generated by the Boolean intersection of the quadtrees and stored. Next, slice s_{i+2} is loaded and its quadtree is generated. The difference quadtree $DQ_{s_{i+1}, s_{i+2}}$ is generated and stored. The process is repeated for all the remaining slices. The difference quadtrees are stored in QSF files and optionally in PXF files, if they include non-homogenous leaves at the lowest level.
- Step 3: Entropy encoding.** As in other compression algorithms, entropy encoding of the stored data is performed using RLE and Huffman coding, [12].

Decompression is performed in a progressive way. Data from the first quadtree level are extracted and the process is repeated for the next levels until the required level of details is achieved.

An example of progressive reconstruction and visualization of a liver dataset with 234 slices of 512×512 pixels (total size 119,808 kB) is shown in Fig. 2. The reconstruction levels 1 – 6 are performed only from the QSF file, while for the full reconstruction both QSF and PXF files are required. Table 1 shows the number of kB needed to transfer the voxel data set.

Table 1. The size of the transferred data in the reconstruction levels of Fig. 2.

Reconstruction level	Transferred data (kB)	Fig. 2
1	2	
2	6	
3	27	
4	119	(a)
5	503	(b)
6	2,113	(c)
FULL	49,497	(d)

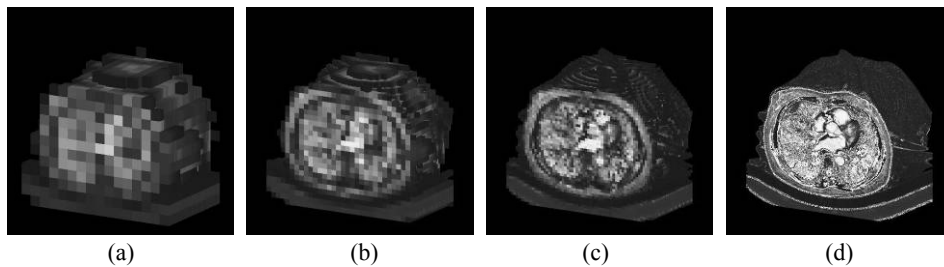


Fig. 2. Progressive reconstruction and visualization of a liver dataset.

3. HARDWARE IMPLEMENTATION

Reported data compression algorithms are typically implemented in software tools for storage and manipulation of volumetric data sets. An interesting alternative would be

to implement data compression within the system that actually generates the volumetric data. Our work was motivated by the goal of developing a data compression algorithm that could be implemented in hardware as a part of a CT or MRI scanner control logic. Hardware implementation of various algorithms allows much higher execution frequencies and widens the algorithm's usage [13].

The quadtree algorithm was implemented in Field-Programmable Gate Array (FPGA) using the development board Celoxica RC1000 [14] and development suite Celoxica DK [15]. The purpose of the prototype version was to gain experience in optimisation of algorithm structure for the prospective embedded implementations.

Celoxica RC1000 hardware platform is a standard PCI bus card equipped with a XILINX VirtexTM chip XCV2000E with 2,541,952 system gates. It has 8 Mb of SRAM directly connected to the FPGA in four 32 bit wide memory banks. The memory is also visible to the host CPU across the PCI bus as if it was normal memory. Each of the 4 banks may be granted to either the host CPU or the FPGA at any time. Data can therefore be shared between the FPGA and host CPU by placing it in the SRAM on the board. It is then accessible to the FPGA directly and to the host CPU either by DMA transfers across the PCI bus or simply as a virtual address. High speed Direct Memory Access (DMA), data buffering and clock speed control make it suitable for high speed compression applications.

The DK Design Suite software provides the environment necessary to implement the target software-compiled design described in a C-based design language Handel-C in FPGA. Sequential programs can be written in Handel-C just as in conventional C. Handel-C also includes parallel constructs that may considerably speed up the application. The compiler compiles and optimises Handel-C source code into a file suitable for simulation (netlist). The resulting file can be placed and routed on a real FPGA using XILINX ISE Foundation Kit. There are, however, some restrictions (*i.e.*, floating point variables are not supported, functions may not be recursive) that the designer must be aware of when writing the code in Handel-C or translating the source C code into Handel-C.

In practice, such an FPGA would be placed on the Printed Circuit Board (PCB) within the CT or MRI scanners, which ensures fast and effective hardware compression. With a proper selection of an FPGA we can influence the resolution of the images and their bit-depth, or if needed due to the constraints, larger FPGA can be used to fulfill the resolution demands. An example of different resolutions and their hardware constraints is presented in Table 3.

4. EXPERIMENTAL RESULTS

In practice, the proposed technique was first implemented in C++ and tested on numerous medical data. A comparison with the octree-based compression is given in [16], where in average 15 percent better compression rate has been obtained by quadtree compression. For illustration, a short summary for some typical examples in medicine and fractals is shown in Table 2.

The comparison should, of course, include other aspects such as space (memory) complexity, time (CPU) complexity, robustness, *etc.* In this paper, however, we focus primarily on the feasibility of hardware implementation. While in octree compression,

Table 2. Comparison of quadtree and octree compression algorithm.

Dataset	Resolution	Original size (kB)	Quadtree compression (kB)	Octree compression (kB)
Liver	$512 \times 512 \times 234$	119,808	49,497	54,629
Head	$512 \times 512 \times 85$	43,521	17,582	18,755
Spine	$512 \times 512 \times 60$	30,721	10,633	11,248
Teapot	$256 \times 256 \times 178$	11,392	1,291	2,264
Fractal1	$512 \times 512 \times 512$	131,072	945	1,611
Fractal2	$512 \times 512 \times 512$	131,072	303	1,504

Table 3. Implementation details for different resolutions with 16-bit data precision.

Resolution	Total equivalent gates	Occupied slices	LUTs used for RAM	LUTs as logic
256×256	2,158,344	7,868	16,370	5,356
512×512	5,866,987	14,337	51,707	8,400

the complete dataset has to be loaded into memory (which makes hardware implementation very difficult or even impossible), quadtree compression requires much smaller memory space and the task of implementing the algorithm in hardware can be easily accomplished by today's programmable devices. In our case, some effort was nevertheless necessary to translate the C++ code into Handel-C. This was mainly due to the fact that two different groups were involved: one focusing on volumetric data compression and the other experienced in programmable logic applications. Table 3 shows the resulting implementation details of the quadtree compression algorithm. Celoxica RC1000 hardware platform that was used in our preliminary case study allows implementations up to 256×256 resolution. Implementations of higher resolution can be managed by larger FPGAs.

5. CONCLUSIONS

A quadtree based approach for lossless compression of volumetric data is described. It supports progressive data transmission and visualization. Comparative case studies on medical data sets have shown that it provides in general better compression ratio than the octree compression. Since only two consecutive slices are processed at a time the method requires a small amount of memory, which makes it particularly suitable for hardware implementations. Consequently, a built-in compression in volumetric scanners could considerably reduce the transfer and storage requirements for volumetric medical data. The aim of this paper is to highlight the main points of the approach enabling the potential user to implement the compression technique in practice.

REFERENCES

1. R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*,

- 1988, pp. 65-74.
2. M. Levoy, "Efficient ray tracing of volume data," *ACM Transactions on Graphics*, Vol. 9, 1990, pp. 245-261.
3. P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, 1994, pp. 451-457.
4. K. L. Ma, E. B. Lum, and S. Muraki, "Recent advances in hardware-accelerated volume rendering," *Computers & Graphics*, Vol. 27, 2003, pp. 725-734.
5. L. Hesselink and P. Ning, "Vector quantization for volume rendering," in *Proceedings of Workshop on Volume Rendering*, 1992, pp. 69-74.
6. S. Muraki, "Volume data and wavelet transforms," *IEEE Computer Graphics and Applications*, Vol. 13, 1993, pp. 50-56.
7. Z. Zhu, R. Machiraju, B. Fry, and R. Moorhead, "Wavelet-based multiresolutional representation of computational field simulation datasets," in *Proceedings of the 8th Conference on Visualization*, 1997, pp. 151-158.
8. K. L. Ma and H. W. Shen, "Compression and accelerated rendering of time-varying volume data," in *Proceedings of Workshop on Computer Graphics and Virtual Reality*, 2001, pp. 263-270.
9. H. Samet, "The quadtree related hierarchical data structures," *Computing Surveys*, Vol. 16, 1984, pp. 187-260.
10. R. F. Chang and W. M. Chen, "Interframe difference quadtree edge-based side-match finite-state classified vector quantization for image sequence coding," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 6, 1996, pp. 32-41.
11. F. F. Rodler, "Wavelet based 3D compression with fast random access for very large volume data," in *Proceedings of Pacific Graphics*, 1999, pp. 108-117.
12. D. Salomon, *Data Compression, the Complete Reference*, Springer, New York, 1998.
13. M. Mali, F. Novak, and A. Biasizzo, "Hardware implementation of AES algorithm," *Journal of Electrical Engineering*, Vol. 56, 2005, pp. 265-269.
14. Celoxica, *RC1000 Hardware Reference Manual*, version 2.3, 2001.
15. Celoxica, *Handel-C Language Reference Manual*, version 3.1, 2002.
16. G. Klajnsek and B. Zalik, "Progressive lossless compression of volumetric data using small memory load," *Computerized Medical Imaging and Graphics*, Vol. 29, 2005, pp. 305-312.

Gregor Klajnsek is currently working as a researcher at Faculty of Electrical Engineering and Computer Science at University of Maribor, Slovenia. He obtained Ph.D. in computer science in 2005 from the University of Maribor, Slovenia. His research interests include visualization of terrain, compression of volumetric data, scientific visualization of volumetric data and multimedia algorithms.

Borut Žalik is a professor of Computer Science at University of Maribor, Slovenia. He obtained Ph.D. in Computer Science in 1993 from the University of Maribor, Slovenia. He is the head of Laboratory for Geometric modeling and multimedia algorithms. His research interests include computational geometry, geometric data compression, sci-

entific visualization and geographic information systems. He is an author or co-author of more than 50 journal and 80 conference papers.

Franz Novak gained the B.Sc., M.Sc., and Ph.D. degrees in Electrical Engineering from the University in Ljubljana in 1975, 1977, and 1988, respectively. Since 1975 he has been with the Jožef Stefan Institute, where he is currently head of Computer Systems Department. Since 2001 he is also assoc. professor, at Faculty of Electrical Engineering and Computer Science, University of Maribor. His research interests are in the areas of electronic design and test and fault-tolerant computing.

Gregor Papa is research assistant at the Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia. He received his M.Sc. and Ph.D. degrees in Electrical Engineering from the University of Ljubljana, Slovenia, in 2000 and 2002, respectively. His research interests include optimization techniques, meta-heuristic algorithms, high-level synthesis of integrated circuits, hardware implementations of high-complexity algorithms, and industrial product improvements. His work is published in several international journals.