International Journal of Scientific Engineering and Technology
Volume No.6 Issue No.2, pp: 86-90

ISSN:2277-1581
01 Feb. 2017

A Volume of Medical Images Reconstruction on GPGPU Environment

E.Sudarshan[1] , Dr. S. Jumlesha[2] , Ch. Satyanarayana[3], C. Srinivasa Kumar[4], K. Seena Naik[5]
[2]Department of Computer Science and Engineering
[1] VITAE, Hyderabad;[3] JNTUCE, Kakinada ;[4] VITS, Hyderabad ;[5] SREC, Warangal

Abstract: An image before visualization the volume of data sets has to be decompressed because that has to be stored in the compressed format, so every medical imaging system must adopt a compression technique. The process of decompression takes a long time, so we are introducing a boosting method for medical volume decompression using General-Purpose Graphics Processing Units (GPGPU). This is supposed to use parallel processing on GPU architecture owing to that it's acquired less time for decompression. We improved the decompression method by introducing the Embedded Zero-tree Wavelet (EZW) technique to render the volume of images on the GPU environment. Finally, this method allows selective decompression in an alignment with the restore volume visualization and volume of additional decompression through it to obtain a better performance.

Keywords: GPGPU, CUDA, NVIDIA, DirectX, Volume Visualization.

I. INTRODUCTION

Volume visualization is a technique for generating useful image information by processing volume data [1]. Volume visualization is widely used as a method for observing three-dimensional medical images obtained from three-dimensional medical imaging apparatuses such as CT and MR. Currently, the task performing volume visualization in a number of medical imaging systems is as follows. First, the medical volume data stored in the video server are transmitted to a visualization client that performs diagnosis. At this time, medical images are generally stored in a separate storage server for security, storage capacity, and management convenience. When the transfer of the volume data is completed, the client performs necessary visualization using the visualization software.

When the capacity of the image increases, the three-dimensional medical images are stored and transmitted in a compressed form in order to save the storage space on the server and the image transmission time to the client. Therefore, the client performs volume visualization after restoring the transmitted compressed image. Compressed restore of large volume data takes a long time, so efforts to reduce it are necessary. The purpose of this study is to propose a high speed medical image decompression method suitable for 3D medical image system.

Many existing high-speed compression methods are codebook methods using vector quantization [4]. The codebook is a dictionary format of image fragments which are frequently generated by vector quantization. The image fragment to be compressed is replaced with the page number of the most similar dictionary to perform compression. The method of generating a palette and performing palette animation or using a codebook for the quadratic data, including the time change as in the related study [5] is as follows.

However, in order to perform lossless compression in this manner, the number of all cases must be stored in advance, which leads to a drawback that the compression efficiency becomes very poor and compression becomes impossible. In addition, there is a method to reduce the data without image loss, a method of removing all the transparent part of the memory by preprocessing and relocating the remaining part [6], but there is a disadvantage that if the user once determines the transparency.

In order to support lossless compression, we propose an entropy-based method [7-9] that extracts redundant information of images with high computational complexity but performs compression. The compression efficiency can be improved by separating the principal component and the residual component of frequency transformation using the trigonometric function or the wavelet, and then removing the redundant information about the residual component.

There are hundreds of parallel processors in the GPU [10], and for each device, the user can perform parallel operations by allocating a thread, which is a logical unit of work. Since each thread is executed in parallel, this study performed system design so that there is no data dependency between threads and load management is smooth. In addition, we propose a selective block restoration method based on visibility checking that compression, decompression is used for volume visualization in order to use graphics processing device efficiently.

The proposed system satisfies the following properties. First, lossless compression as well as restoration is supported. If the data size is too large or the transmission speed is too slow, a low quality reconstructed image may be generated first, but it may improve over time. That is, high-speed loss recovery and gradual improvement are possible. And a compression structure suitable for three-dimensional visualization, thereby enabling high-speed compression restoration. In Section 2, we summarize the embedded zero-tree wavelet transform, which is a compression method used in this study. Section 3 describes the proposed method. Section 4 shows the experimental results. Section 5 concludes the paper.

II. BUILT-IN ZERO-TREE WAVELET TRANSFORMS

In this paper, we propose an Embedded Zero-tree Wavelet transform (EZW) [11] based on GPU for compressing and reconstructing three-dimensional medical images. The feature of the EZW is that the important coefficients are first coded and the small coefficients are coded later. Therefore, fast loss recovery can be performed, and lossless recovery can be performed by further transmitting and restoring the coefficients.

ISSN:2277-1581
01 Feb. 2017

There are similar compression methods such as SPIHT [8] and EBCOT [9], and their compressibility is known to be higher than EZW [3], but the algorithm is more complicated and takes a long time to restore did not do it. EZW uses the self-repeatability of the wavelet transformed image to apply the observation that the change in the parent node value will be small even at the child node.

63 **P**	-34 **N**	49 **P**	10 **T**	7 **Z**	13 **Z**	-12	7
-31 **Z**	23 **T**	14 **T**	-13 **T**	3 **Z**	4 **Z**	6	-1
15 **T**	14 **Z**	3	-12	5	-7	3	9
-9 **T**	-7 **T**	-14	8	4	-2	3	2
-5	9	-1 **Z**	47 **P**	4	6	-2	2
3	0	-3 **Z**	2 **Z**	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Fig 1 EZW compression example

The details are introduced in, and this study briefly outlines the outline. A method of compressing a block in the EZW is to successively encode important coefficient values larger than the set threshold value [11]. Example, if the initial threshold is specified as 32 in Fig.1, values greater than 32 are coded 63, -34, 49, 47 sequentially the search is encoded in the P (positive) N (negative) and a value less than the threshold value are encoded with a Z (zero), you know the video picture in the form tree (quad-tree) when all the Z nodes of the tree part (sub tree) T (tree it replaces the root) are reduced by the number sign, Fig.2 shown the flowchart for significant coefficient. Each extracted because the code is one of P, N, Z, T, for each code 2bit to give of the store, larger than the critical threshold value code, adding storage to a specified threshold value from the precision parts the processing is terminated. Then, the threshold value is reduced by half, and the remaining values are successively encoded, and the detailed information is sequentially encoded.

In this study, since the number of EZWs is stored repeatedly until the threshold is 1, lossless compression and restoration are possible. The restoration process of the EZW is performed using the order of the stored codes P, N, Z, T and the precision value of the sign code. The restoration is likewise performed from a large threshold value. If the threshold value is greater than 1 and stopped in the middle, the approximate value is restored instead of the original image. If the loss recovery is to be performed in a short period of time, the restoration may be stopped while the threshold value is kept large.

III. THE PROCESS OF RESTORING THE COMPRESSED VOLUME DATA

The overall configuration of the proposed algorithm, since reconstruction of the 3D image is performed in the reverse order of the compression process, this paper focuses on the compression process, and the reconstruction process can be understood in the reverse order.

3.1 Blocking and two-dimensional map area

The 3D volume image is compressed in units of blocks. Decomposition of the entire volume data into small pieces of blocks improves the localization of the data distribution and improves the compression efficiency. However, if the size of the block is too small, the total number of blocks increases and the compression efficiency is degraded considering the header size to be stored in each block. In this study, one block is defined as a size of 8 × 8 × 4 considering compression ratio and performance efficiency, and it is transformed into 16 × 16 by arranging it as a tile. In Fig. 3, one block represented by B is a resolution of 8 × 8 × 4, and when arranging four 8 × 8 images constituting one lock side by side in two dimensions, one image of 16 × 16 sizes.

In many of the existing studies, the block size is 8x8 tiles in two - dimensional image compression, but the block size is designed to be large because three - dimensional data should consider the spatial distribution in addition to planar distribution. Then wavelet transform is applied to each generated tile image. On the other hand, decompression is a process of copying two-dimensional tile data into a certain area of volume data, and parallelization is advantageous because there is no intersection between blocks.

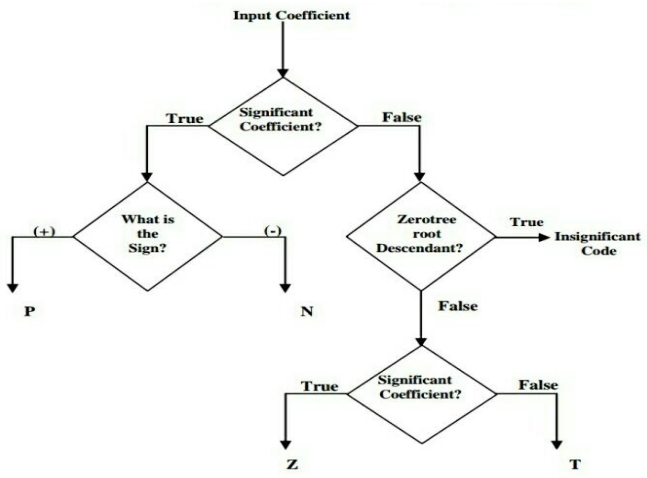

Fig 2 Flowchart for coefficient significance map

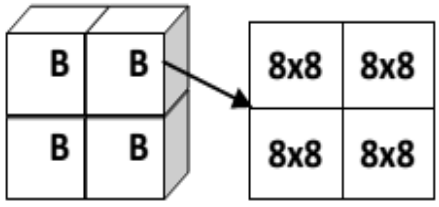

Fig.3. Blocking and two-dimensional map area

ISSN:2277-1581
01 Feb. 2017

3.2 Wavelet Transform and EZW process

Each tile image is separated into low frequency component and high frequency component by wavelet transform. The research goal is to be fast and not lose image quality. First, we chose the Haar wavelet, the simplest and fastest wavelet transforms. However, the Harr wavelet may cause image quality loss in the conversion process because errors occur when the real number value generated in the operation process is converted to an integer, or it is subjected to a quantization process. In order to support lossless compression, we used an integer lossless transform such as (Eq. 1) without quantization. In (Equation 1), the residual part *d* and the signal part *s* of the *j + 1* stage of the parent node can be obtained at the child node *j*. In the restoration process, if the (Equation 2) is repeatedly applied, the inverse transformation is implemented without loss.

The result transformed into the frequency image is output through the EZW transform and then restored in the reverse order. In the restoration process, parallel processing is possible because the operation is independent for each block in both EZW and wavelet transform.

$$d_{j+1,i} = s_{j,2i+1} - s_{j,2i}$$

$$s_{j+1,i} = s_{j,2i} + \left\lfloor \frac{d_{j+1,i}}{2} \right\rfloor \text{----------(1)}$$

$$s_{j-1,2i} = s_{j,i} - \left\lfloor \frac{d_{j,i}}{2} \right\rfloor$$

$$s_{i-1,2i+1} = d_{j,i} + s_{j-1,2i} \text{----------(2)}$$

3.3 GPU Parallelization

In this study, we implemented parallelization using NVidia's GPGPU language, CUDA, to implement parallel processing using GPUs. Since the image compression is a very complex computation step, the use of the GPGPU language is more appropriate than the expansion of graphics APIs such as DirectX [14] or OpenGL, unless it is only partially using GPUs such as wavelet transform [13]. To perform parallel processing, CUDA [12] provides a logical hierarchical structure of threads and thread blocks to facilitate parallelization.

In CUDA, the smallest logical parallel unit of work is threaded, and one thread executes the given instructions sequentially. The user can designate a certain number of threads as one thread block as shown in Fig.4. Threads belonging to the same thread block can synchronize and exchange information through shared memory and the whole task consists of many thread blocks. Each thread block is assigned to MP (multi-processor), which is a physical parallel processor built into the GPU.

This study seeks to improve the performance of the algorithm through efficient parallelization. Since the decompression process of image compression is performed independently for each block, the restoration of each block is composed of a single thread. Next, we need to solve the problem of determining the number of threads belonging to one thread block in order to construct a thread block. Since a single MP has a register with a limited capacity, if the number of threads constituting one thread block is too large, the performance is degraded due to the lack of registers, and the execution becomes impossible. On the contrary, if the number of threads constituting the thread block is too small, the processing unit constituting the MP becomes idle, thereby decreasing the efficiency. In this study, the performance change is observed while changing the number of threads per thread block, and an optimal point is found, so that it is helpful to determine the proper number of threads in future research.

3.4 The Conjunction of Effective Decompression in Volume Visualization

Restored volume data are used for volume visualization, and that is also performed in GPGPU. In this study, ray tracing method [1], which is the most commonly used volume visualization algorithm, is applied and parallelization is performed by assigning threads to each ray [15]. Since compression, decompression and visualization are performed on the GPU, this study is efficient because it does not need to transmit the restored data. If you restore compression to the CPU, additional bus transfer time is required to transfer the restored data to the GPU.

On the other hand, there is an empty space leap method as an important way to accelerate volume visualization [16,17]. The empty space leap is a method of storing positional information of a region that is considered to be transparent in volume data through preprocessing and speeding up by omitting sampling of transparent region in the visualization step. Whether or not a portion of the volume data is transparent depends on the opacity transition function you specify [1]. The user defines the transparency for a range of possible density values. For example, if you want to observe the skeletal system of the human body, you specify opacity of 0 for density values for muscle and subcutaneous fat. In this study, the maximum density of each block is obtained by preprocessing and stored. When the opacity of each block's density range is found to be 0, the empty space jump is performed. In this case, when we consider the empty space hopping technique in conjunction with decompression restoration, we conclude that the transparent block does not affect the image quality even if it is excluded from compression decompression. In this study, we improve the performance by restoring only blocks that are considered to be opaque.

In order to implement this selective restoration, this study included the maximum value of each block in the header of the compressed data, and analyzed the header for the opaque transition function defined by the user, so that it can judge whether or not to restore the compression. Thereafter, when the opacity transition function is changed, the compression is further restored in response to the change, so that image quality deterioration does not occur. Implementing such an idea with GPGPU requires additional consideration. Since GPGPU is a parallel computing device, there exists a parallel execution unit that performs the same instruction on other data [12]. If only some of the blocks in one parallel execution unit are restored, the arithmetic unit that is responsible for the transparent blocks that do not need to be restored is designed to wait for synchronization. Therefore, this study changed the addressing of

the block to the indirect method in order to increase the utilization of the GPU computing device. The compression, decompression index is generated and it is determined that it is opaque. Therefore, only the address of the block to be decompressed is stored separately, and only the block of the address stored in the decompressed index is restored in the GPGPU.

In order to implement such a selective restore, this study is included in the header of the compressed data for the maximum values of each block and user-defined functions for the opacity transition analyzes the header can be determined in advance whether decompression was possible. When the opacity after the transfer function is changed, restoring the compression quality so as to correspond to the additional degradation does not occur.

Implementing these ideas further with GPGPU is necessary considerations. Since GPGPU is a parallel computing device, there is a parallel execution unit (warp) that performs the same instruction on other data. If only some of the blocks in one parallel execution unit are to be restored, the arithmetic units responsible for the transparent blocks that do not need to be restored are designed to wait for synchronization. Therefore, in order to increase the utilization of GPU computing devices, this study changed the addressing of blocks to an indirect method after decompression index is generated, only the address of the block to be decompressed is stored separately, and only the address block stored in the decompressed index is restored in the GPGPU to further improve the performance.

IV. EXPERIMENTAL RESULTS

The experiment was performed on a personal computer equipped with Intel's Core2Duo CPU and nVidia's GeForce GTX 470 GPU. The GeForce GTX 470 has 14 MPs, 448 (= 14x32) parallel processing units and It has 1.2 GB of dedicated memory. The software implementation uses Visual Studio C ++ and the CUDA SDK 3.2 for GPGPU development. The volume data were 256 × 256 × 225 head data and 512 × 512 × 300 abdomen data, and each data was adjusted to have 16-bit precision. The visualization results using lossless decompression, which is exactly the same as the result of visualization of uncompressed data. And we measured the time taken to decompress using GPU parallelism. As described in Section 3.3, the compression recovery time is measured by changing the thread per block (TPB) per thread block in the parallel hierarchical structure of CUDA, and is shown in Table 1. In addition, the CPU-based EZW compression sleep time was also measured and presented.

In this experiment, it is observed that the optimum performance is obtained when TPB is 64. The result shows that the performance of the head data is about 36.9 times (= 4437 / 120.1) and that of the abdomen data is about 44.7 times (= 65865 / 1472.7). Since there is no previous work on EZW decompression with GPU, it is difficult to directly compare this result with related studies. However, it has been reported that GPU parallelization achieves 35-50 times faster speed than the CPU in a related study [18] that implemented Huffman lossless compression with the GPU, and lossless compression for double precision data in a previous study comparing performance, it was reported that the GPU using 320 high-speed cores is 4.5 times faster than the parallel CPU of 8 cores, and the performance improvement obtained by applying the GPU of this study is judged to be appropriate.

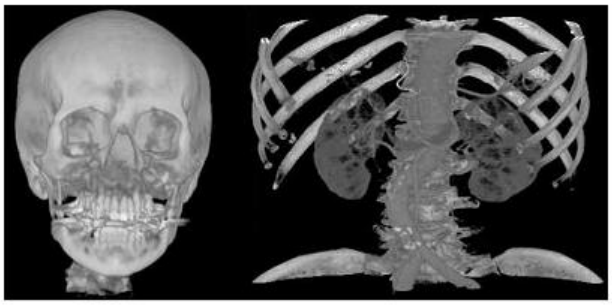

Fig.4. Optional decompression in conjunction with the opacity transfer function input image.

As can be seen in Table 1, it can be seen that the performance varies greatly depending on the size of the TPB. If the TPB is smaller than 32, which is the number of arithmetic units constituting one MP, a part of the parallel processing unit becomes idle and the performance is greatly degraded. When the TPB is greater than 32, the performance is further improved because the context exchange is performed by other operations when the operation upper limit is idle by an operation such as global memory access. However, in an overly large TPB, a lack of registers degrades performance.

In this study, compression, decompression is a complicated operation using a lot of registers, so it can be confirmed that the optimal TPB is determined to be close to 32, which is the number of arithmetic units constituting MP. It is concluded that it is better to increase the optimal TPB than 64 if the characteristics of the algorithm executed by each thread in the future study becomes simpler than this study. On the other hand, if the TPB is too small, the number of the entire blocks increases. Therefore, the abdomen data having a relatively large capacity cannot be decompressed at one time and the numerical values are not recorded (N/A). The experiment was performed based on the good TPB 64. EZW can perform loss recovery in a relatively short time after completion of the restore process. The recovery time was measured by setting the interrupt thresholds, which are the error limits, to 1 and 2. It can be confirmed that the speed is gradually increased up to 1.5 times, according to the image quality loss, and the compression can be restored in a short period of time if the image quality is lost. Finally, performance improvement using selective block restoration proposed in Section 2.5 was confirmed. The transparency transition function is set to be transparent in the abdomen data, and the header data is set to be transparent in a relatively small amount. The selective restoration method was applied to measure the compression, restoration time, and the compression, restoration index method was added to further improve the GPU efficiency, and the compression restoration time was again measured and shown in Table 3.

The selective block recovery scheme proposed in this study shows a remarkable performance improvement of about

ISSN:2277-1581
01 Feb. 2017

1.5 times (= 120.1 / 81.9) to 3.5 times (= 1472.7 / 417.5) compared with the existing restoration method. The effect of the selective block restoration is different depending on the opacity transition function designated by the user. It can be confirmed that the abdominal data having a high ratio of the transparent region omit more blocks and shows a high effect. Additionally, by improving the performance of the GPU through indirect addressing, a speed increase of about 10% is achieved as a final result, if a clinically used opacity transition function is specified, it takes only 380ms less to decompress practical data of 512x512x300 size.

On the other hand, the selective block restoration has an excellent effect when there are many vacant spaces, and there is a limitation that the performance improvement cannot be obtained when the vacant space is set in the user-defined opacity transition function.

V. CONCLUSIONS

In this study, the EZW reconstruction algorithm is implemented as GPGPU for compression and restoration of medical images. In order to improve the efficiency, we proposed a fast decompression algorithm by combining the empty space leap method of volume visualization. As a result, for practically sized 512 × 512 × 300 data, compression reconstruction in conjunction with volume visualization could be performed at an instantaneous time of 380 ms. Although the application of transparency is only possible for direct volume visualization and depends on opacity transition functions, the measured result indicates that decompression can be applied immediately to the user's command. As a limitation of this study, since the arithmetic coding process which further improves the compression efficiency is omitted, the practical result can be obtained by adding this. As Lindstrom et al reported the results of variable-length coding, it is necessary to store compressed size information together, which is inefficient, and it is necessary to study effective compression methods to overcome this problem. As a future research, it is expected that JPEG2000, which is widely used for lossless compression of medical images, can be applied to CUDA for commercial systems.

Data	CPU	TPB (GPU) (unit of time: ms)								
		1	2	4	8	16	32	64	128	256
Head	443	1000.0	604.7	392.3	273.1	210.3	157.4	120.1	140.7	139.8
Stomach	65865	N/A	N/A	N/A	2676.8	1972.0	1528.8	1472.7	1750.5	1737.8

Table 1. The decompression time, according to the number of threads (TPB) of threads per block

Image Type	Lossless Restoration	Loss Restoration (Threshold value 1)	Loss Restoration (Threshold value 2)
Head	120.1	93.8	82.5
Stomach	1472.7	1238.4	992.4

Table 2. Comparison of decompression time of loss and lossless restoration. (Time units: ms)

Image Type	Lossless Restoration	Optionally blocks Restore	Optionally blocks Restoring + Index
Head	120.1	81.9	71.2
Stomach	1472.7	417.5	380.3

Table 3. Performance of restoration method selectively blocks results. (Time unit: ms)

References

i. *M. Levoy, "Display of Surfaces from Volume Data," IEEE Computer Graphics and Application, Vol.8, No.3, pp. 29-37, 1988.*

ii. *L. Bottou, P. Haffner, P. Howard, P. Simard, Y. Bengio, and Y. LeCun, "High quality document image compression using DjVu," Journal of Electronic Imaging, Vol.7, No.3,pp. 410-425, 1998.*

iii. *Z. Xiong, X. Wu, S. Cheng, and J. Hua, "Lossyto-Lossless Compression of Medical Volumetric Data using Three-Dimensional Integer Wavelet Transforms," IEEE Transactions on Medical Imaging, Vol.22, No.3, pp. 459-470, 2003.*

iv. *N. Fout and K.L. Ma, "Transform Coding for Hardware-accelerated Volume Rendering," IEEE Transaction on Visualization and Computer Graphics, Vol.13, No.6, pp. 1600-1607, 2007.*

v. *E.B. Lum, K.L. Ma, and J. Clyne, "Texture Hardware Assisted Rendering of Time-Varying Volume Data," Proc. of the Conference on Visualization '01, pp. 263-270, 2001.*

vi. *M. Kraus and T. Ertl, "Adaptive Texture Maps," Proc. of the ACM SIGGRAPH / EUROGRAPHICS Conference on Graphics Hardware, pp. 7-15, 2002.*

vii. *M. Pratt, C. Chu, and S. Wong, "Volume Compression of MRI Data using Zerotrees of Wavelet Coefficients," Proc. SPIE Wavelet Applications in Signal and Image Processing IV, Vol.2825, pp. 752-763, 1996.*

viii. *A. Said and W. Pearlman, "A New, Fast and Efficient Image Codec Based on Set Partitioning," IEEE Trans. Circuits Syst. Video Technol., Vol.6, No.3, pp. 243-250, 1996.*

ix. *D. Taubman, "High Performance Scalable Image Compression with EBCOT," IEEE Trans. Image Processing, Vol.9, No.7, pp. 1158- 1170, 2000.*

x. *K. Engel, M. Hadwiger, J.M. Kniss, C. RezkSalama, and D. Weiskopf, Real-Time Volume Graphics, Wellesley, Massachusetts, 2006.*

xi. *J.M. Shapiro, "Embedded Image Coding using Zerotrees of wavelet Coefficients," IEEE Transactions on Signal Processing, Vol.41, No.12, pp. 3445-3462, 1993.*

xii. *http://www.nvidia.com/object/cuda_home_new.html*

xiii. *Lee, Man-Hee, Park, In-gyu, Won Seok-jin, Cho, "Fast Computation of DWT and JPEG2000 using GPU , "Electronics Journal of Applied Physics, Vol. 44, No. 6, pp. 9-15, 2007.*

xiv. *http://msdn.microsoft.com/en-us/directx, 2012.*

xv. *Hee-Won Kang, Jun-Ho Kim, "Acceleration techniques for GPGPU-based Maximum Intensity Projection," Journal of Multimedia, 14th Issue 8, pp. 981-991, 2011.*

xvi. *M. Levoy, "Efficient Ray Tracing of Volume Data," ACM Transactions on Graphics, Vol.9, No.3, pp.245-261, 1990.*

xvii. *W. Li, K. Mueller, and A. Kaufman, "Empty Space Skipping and Occlusion Clipping for Texture-Based Volume Rendering," Proc. of IEEE Visualization Conference, pp. 317- 324, 2003.*

xviii. *A. Balevic, "Parallel Variable-Length Encoding on GPGPUs," Proc. of the 2009 International Conference on Parallel Processing, Springer-Verlag, Berlin, Heidelberg, pp. 26-35, 2009.*