An effective depth data memory system using an escape count buffer for 3D rendering processors

Woo-Chan Park[1a), **Jin-Hong Park**[2], **Woo-Nam Chung**[2], **Jeong-Soo Park**[2], **Sang-Duk Kim**[2], **Hong-Sik Kim**[3], **Young-Sik Kim**[4], **and Tack-Don Han**[2]

[1] *Department of Computer Engineering, Sejong University, 98 Gunja-dong, Gwangjin-gu, Seoul 143–747, Korea*

[2] *Department of Computer Science, Yonsei University, 134 Shinchon-dong, Seodeamun-gu, Seoul 120–749, Korea*

[3] *Advanced Design Team, R&D Division, Hynix Semiconductor Inc., San 136–1 Ami-ri Bubal-eub, Icheon-si, Gyeonggi-do 467–701, Korea*

[4] *Department of Game and Multimedia Engineering, Korea Polytechnic University, 2121 Jeongwang-dong, Siheung-si, Gyeonggi-do 429–793, Korea*

a) *pwchan@sejong.ac.kr*

Abstract: This paper proposes an effective memory system of depth data to reduce the bandwidth requirement from the external memory for low-power 3D rendering processors. For this purpose, we propose an escape count buffer that contains information about the data size for each compressed depth block. Compared to the previous scheme, experimental results show that this approach reduces the memory bandwidth requirements up to 44%.

Keywords: rendering processor, graphics hardware, memory system, compression

Classification: Integrated circuits

References

[1] M. H. Choi, W. C. Park, F. Neelamkavi, T. D. Han, and S. D. Kim, "An effective visibility culling method based on cache block," *IEEE Trans. Comput.*, vol. 55, no. 8, pp. 1024–1032, Aug. 2006.

[2] S. Morein, "ATI Radeon HyperZ technology," *Hot3D Session 2000 ACM SIGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2000.

[3] J. Deroo, S. Morein, B. Favela, and M. Wright, "Method and apparatus for compressing parameter values for pixels in a display frame," US Patent 6,476,811, 2002.

[4] J. Hasselgren and T. Akenine-Moller, "Efficient depth buffer compression," *Proc. SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pp. 103–110, 2006.

© IEICE 2011
DOI: 10.1587/elex.8.209
Received January 04, 2011
Accepted January 31, 2011
Published February 25, 2011

1 Introduction

The problem of memory bandwidth is one of the most important research topics that seeks to improve the performance of graphics processing units (GPU's) [1]. In case of mobile GPU's where hardware resources are severely restricted, it is more difficult to compensate the performance degradation due to limited memory bandwidth. In order to address this problem, many research studies, based on cache memory and lossless data compression, have been proposed.

The external memory of general GPUs consists of texture memory and frame memory that store the texture data and the pixel data (depth and color data), separately. The high temporal locality of the texture data, due to its repeated access patterns, enables the texture cache to effectively reduce the memory bandwidth requirement. However, in the case of pixel data, the spatial locality is too low to address the memory bandwidth problem using only cache memories.

Lossless data compression schemes have been applied to reduce the memory bandwidth requirement for depth data [2, 3, 4]. Depth data blocks, with a fixed square size, are encoded using a DDPCM algorithm and the encoded block data are stored in the external memory. In [2, 3], the compressed block are classified into three levels—uncompressed, poorly-compressed, and well-compressed—and are stored in the external memory. With this scheme, in the case that the variance of the compression ratios of the block data is high, a large amount of memory access to empty space could occur.

In this paper, a new compressed memory system is proposed to reduce the bandwidth requirement from the external memory by controlling the data transaction size of the compressed block data. By using an escape count buffer to store the compression levels of the depth data block, the data transaction size for the external memory access could be accurately controlled. According to the experimental results, the proposed scheme could reduce the external memory requirement by 44%.

2 DDPCM

Differential Differential Pulse Code Modulation (DDPCM) [2] is suitable for two-dimensional data compression by conducting a DPCM algorithm twice along the horizontal (x) and vertical (y) axes, respectively. After applying DDPCM, the final result consists of one original depth data, two delta values, and correction terms. Correction terms are entropy-encoded and then the results are stored in the code block and the data block. If pixels lie on one plane, correction terms are entropy-encoded to 00, 01, and 10. If pixels lie on more than one plane, correction terms are entropy-encoded to 11. This value or pixel is known as the escape value. The escape value is stored in data block. The number of escape values, referred to as the escape count, is used as the compression ratio level.

In the case of the 8×8 block compression, the escape count could be maximally 61 when none of the correction terms could be compressed. If a

© IEICE 2011
DOI: 10.1587/elex.8.209
Received January 04, 2011
Accepted January 31, 2011
Published February 25, 2011

depth is assumed to 16 bits, the compressed block size could be minimally 172 ($= 1 * 16 + 2 * 17 + 61 * 2$) bits and maximally 1024 bits. The original data size of the 8×8 block is 1024 bits and the compressed data size is larger than the original data size when the escape count is 60 or 61, so that the maximum size of the compressed block would be 1024 bits.

In [2, 3], according to the compression ratio, the data transaction of the compressed data to the external memory is classified into the following three modes: uncompressed (compressed data size $>$ original data size), poorly-compressed (compressed data size $> 1/2 *$ original data size), and well-compressed (compressed data size $> 1/4 *$ original data size). The mode of each depth block is specified with 2 bits and stored in the internal SRAM. Therefore, the bit size of the internal SRAM shall be twice that of the number of depth blocks in the screen resolution. Since the ATI approach supports only three storage modes, unnecessary memory access to the empty space could degrade the memory system performance in the case that the variance of the compression ratios of the block data is high.

The last mode is used to support the fast depth buffer clear. In traditional GPU, the depth buffer clear initializes the depth buffer for each frame, which is accompanied by a large number of accesses to the external memory. In the case of the ATI's approach, this is performed by simply setting the mode for each depth block of the internal SRAM into the last mode without any access from the external memory.

3 Proposed architecture

In this paper, a new compressed memory system is proposed to reduce the external memory bandwidth requirement by accurate calculation of the memory transaction size of the compressed depth data. Fig. 1 illustrates the proposed architecture, which consists of an ECB (escape count buffer), an ECB controller, a bus transaction controller, a compression unit, and a decompression unit. Instead of allocating 2 bits representing four modes in the ATI approach, we allow storage of the escape count into the ECB. That is, the ECB is a 2D array used to store the escape count of each depth block data. Therefore, similar to the ATI approach, the depth buffer clear can be performed by initializing the ECB without any access to the external DRAM.

The rendering process of Fig. 1 is as follows. The textured fragment is generated during the rasterization stage. In the depth read step, the depth value is retrieved from the depth cache and is, then, compared with that of the current textured fragment in the depth test step. If the depth test fails, that is, the current fragment is obscured by the previously drawn pixel, then the current fragment is excluded from the pipeline. Otherwise, the depth value of the current textured fragment is written into the depth cache in the depth write step. After that, the color blending step is carried out.

The processing flow of the proposed architecture is as follows. First, when the cache miss occurs during the depth read step, the current depth block in the cache is stored in the missed block buffer and an address of the missed

© IEICE 2011
DOI: 10.1587/elex.8.209
Received January 04, 2011
Accepted January 31, 2011
Published February 25, 2011

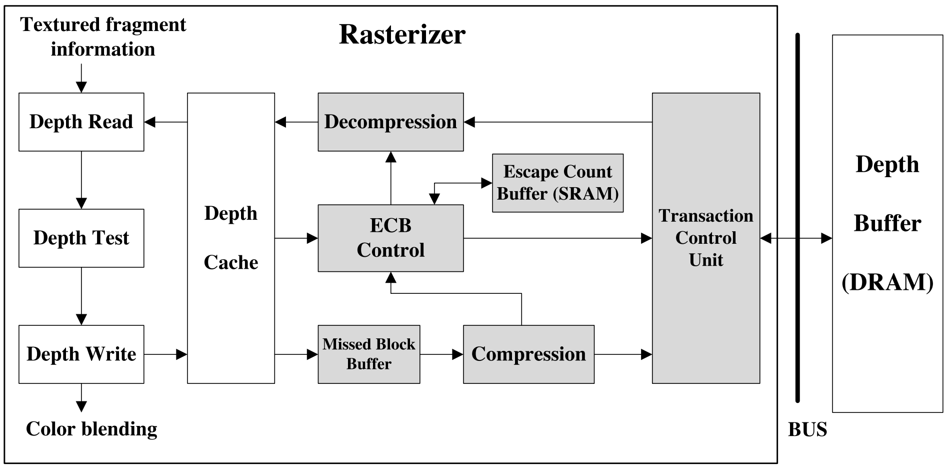

Fig. 1. Proposed Architecture

depth block is sent to the ECB controller for writing back. At the same time, the ECB controller reads the escape count of the requested depth block from the ECB, with which the size of the compressed data is calculated. With this size information, the transaction control unit issues the memory transaction, so that the compressed data, with the exact size, is retrieved from the external DRAM and is, then, sent to the decompression unit. With the compressed data retrieved from the external DRAM and its escape count retrieved from the ECB, the decompression unit restores the orignal block data to the depth cache. After processing the requested data, the missed cache block data in the missed block buffer is compressed. Finally, the compressed data for the cache missed depth block are written back to the depth buffer through the transaction control unit. At the same time, escape count of compressed block is updated into the ECB through the ECB controller.

4 Simulation and results

In order to evaluate the performance of the proposed scheme, we selected two test benches from real application programs. The test benches are Quake 3 from id Software and Unreal Tournament 2004 by Epic Games. For each test bench, 100 frames are simulated and their averages are provided. The depth buffer access patterns have been constructed using Meas3D supporting OpenGL with a 16 bit depth. The depth cache is assumed to be a 16 KB direct mapped cache with 128 entries of an 8×8 block size. Additionally, it is assumed that each test bench is processed at 30 frames per sec under VGA (640 by 480) and SVGA (800 by 600) screen resultions, and the buffer clear is performed at 30 Hz.

Experimental results are shown in Table I. We compare the proposed scheme with a no compression scheme, a compression scheme, and an ATI approach. In the no compression scheme, the uncompressed depth block is accessed from the external DRAM, while the compression scheme, the ATI approach, and the proposed scheme can store the compressed depth data into the external DRAM.

© IEICE 2011
DOI: 10.1587/elex.8.209
Received January 04, 2011
Accepted January 31, 2011
Published February 25, 2011

Table I. Bandwidth Requirements (MBytes/sec)

	Quake3		UT2004	
	VGA	**SVGA**	**VGA**	**SVGA**
No Compression	137.6	283.4	305.1	418.3
Compression	40.0	76.0	78.5	99.1
ATI	26.4	58.3	72.8	86.1
Proposed	**22.4**	**48.5**	**60.9**	**71.6**

The memory accesses can be generated by the write, the read, and the buffer clear operations. In the case of the write operation, through a process of compression, the exact size of a compressed block can be calculated, which will be written to the depth buffer. Therefore, in both the compression scheme and the ATI approach, the memory bandwidth requirement for writing each compressed block into the external DRAM equals the size of the compressed block itself. In the proposed scheme, because the escape count is stored into the internal SRAM, the memory bandwidth requirement for each compressed block is slightly decreased compared to the compression scheme and the ATI approach.

In the case of the read operation, the compression scheme would perform the read operation twice. The first time it is used to retrieve a fixed length size of the compressed data and the second time it is used to retrieve a variable length data of which the size is calculated with the escape count obtained when the first read operation is being conducted. If the escape count is zero, there is no need to perform the second read operation. In the ATI approach, a fixed length size of the compressed depth data is retrieved according to the three modes described in Section 2. Meanwhile, the exact size of the compressed data can be retrieved in the proposed scheme.

For the buffer clear operation, the depth buffer in the external DRAM should be initialized for each frame for the no compression scheme and the compression scheme. However, the ATI approach and the proposed scheme can achieve faster buffer clear for each depth block by initializing the mode bits and escape count buffer contents, respectively.

According to the experimental results shown in Table I, compared to the compression scheme, the proposed scheme could reduce the memory bandwidth requirement by about 44% and 27%, for the Quake 3 and the UT2004 test benches, respectively. Compared to the ATI approach, the proposed scheme could reduce the memory bandwidth requirement by about 16% and 17% for the Quake 3 and the UT2004 test benches, respectively.

The proposed scheme uses a total of 19.29 KB SRAM, of which 3.29 KB is used for the escape count buffer and 16 KB is used for the depth cache. For the ATI approach, 17.3 KB SRAM is used for the depth cache and the depth block mode storage. Therefore, the proposed scheme requires a larger SRAM than the ATI scheme, by only 11%.

© IEICE 2011
DOI: 10.1587/elex.8.209
Received January 04, 2011
Accepted January 31, 2011
Published February 25, 2011

5 Conclusion

In this paper, a new compression scheme is proposed and performance evaluation results are provided to improve the external memory access performance by using ECB. The proposed scheme could reduce the bandwidth requirement of the external memory access by about 17% compared to the ATI approach.

Acknowledgments

The CAD tools were provided by IDEC of Korea.

DOI: 10.1587/elex.8.209
Received January 04, 2011
Accepted January 31, 2011
Published February 25, 2011