

A Real-time 3D Image Refinement using Two-line Buffers

Jong-hak Kim*, Jun-dong Cho*

* Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea
jhakkim@vada.skku.ac.kr, jdcho@skku.edu

Abstract— Recently 3D image processing has been one of vibrant research field. This processing requires very high speed and performance for real-time processing. In this paper, we propose a special memory management unit with n^2 two-line buffers (distributed ram), by adjusting read and write speed to match core's speed for efficient pipelining. We synthesize on Vertex 5 from Xilinx, operating 100 MHz clock. Our core unit is a real time refinement core on erosion and dilation, using a 3 by 3 and a 5 by 5 window, respectively, in a 160 by 90 image. Our proposed method is 30 times faster processing time and slightly lower memory consumption than previous one.

Keywords— real-time image processing, two-line-buffer, 3D, memory management, refinement

I. INTRODUCTION

As 3D-TV and 3D-movie have been commercialized in recent years, 3D image processing becomes vibrant research field. This 3D image processing system that has been studied in various application requires very high speed and performance for real-time processing[1]. In addition, image processing with window operation requires that a large amount of processing is needed to analyse the image contents and recover valid data[2]. The computation takes $n^2 \times M \times N$ for an M by N image with an n by n window which is critical to overall processing time[3]. To reduce the processing time, many hardware implementation methods have been proposed previously for hardware implementation, such as memory swapping[4], parallel neighbourhood access[5] and sliding windowing operation and buffering method[6]. In this paper, we propose a new n by n two-line buffer block array structure to lesson the overall data transfer time of storing and loading time to take the image core's processing time into consideration to facilitate the seamless pipeline structure.

The rest of paper is outlined as follow. We introduce basic concept of image processing and refinement as an example application in Section II. Our memory management will be presented in Section III. Our refinement core application will be presented in Section IV. Experimental results will be given in Section V, and finally, in Section VI, we draw the conclusion.

II. BASIC CONCEPT OF REFINEMENT

Since 3D image processing is usually computed by pixel units independently so that almost all pixels are not

interrelated to neighbour pixels, and images we obtain could have noise and segmentation problems. Therefore we need refinement process using the morphological operation to reduce noise and to simplify shape and segment object from the background.

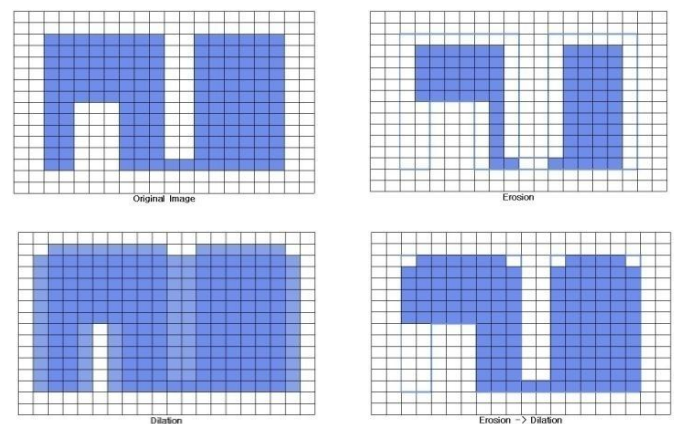


Figure 1. Result of each refinement operation

(Top-Left) Original image, (Top-Right) Erosion,

(Bottom-Left) Dilation, (Bottom-Right) Erosion followed by dilation

Morphological operations are based on erosion and dilation. Erosion operation is removing a pixel if any pixel value in an n by n window is not valid, and dilation operation is opposite, which is filling a pixel with neighbour pixel if any pixel value in an n by n window is valid. Also we define threshold to control the condition that pixel values are removed or filled. Let's assume that we apply erosion operation using a 3 by 3 window, and threshold is 3. Erosion operation will search to remove a pixel, if there are more than 3 pixels with no valid data among 8 pixels in a 3 by 3 window.

More complex morphological operations such as opening and closing are obtained by using erosion and dilation together. Defining opening and closing depend on operation order such as erosion followed by dilation (opening) or dilation followed by erosion (closing). The effect of opening is to remove pixels in regions that are too small to contain the structuring element, and the closing's effect is to fill in holes and concavities

smaller than the structuring element. Figure1 shows results of each operation.

III. THE PROPOSED MEMORY MANAGEMENT

As shown in Figure2, to compute signal processing (e.g., refinement processing as described in Section II) for one target pixel value using an n by n window, we need $n^2 - 1$ pixel values around the target pixel. To load those pixel values, we need at least M by n line buffer. The loading one pixel value from memory to core takes one clock, and to process one target pixel, entire n by n pixel array should be loaded. Therefore core's processing time is n^2 clocks per a pixel. If the storing time is faster than loading time, and data will be overwritten to memory. Therefore in this case, a whole M by N image should be stored before processing unless we decrease loading time.

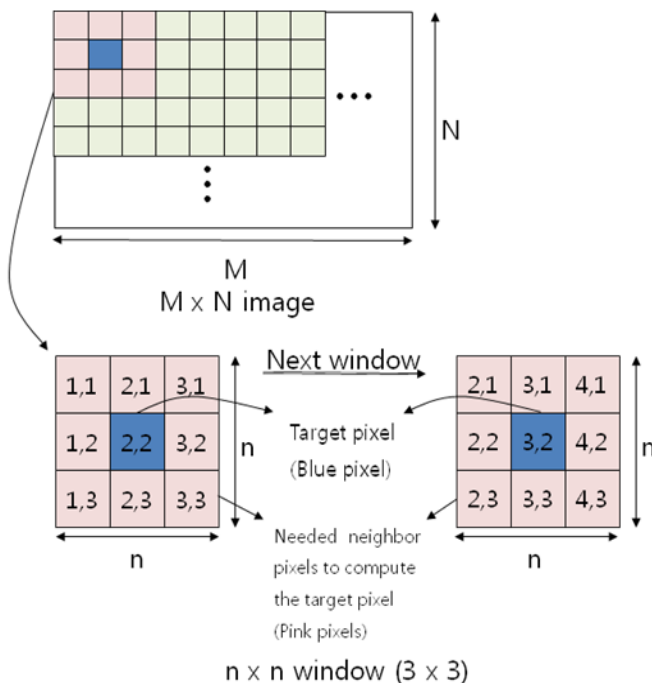


Figure 2. Example of an n by n window in the M by N image

The loading-time of an n by n window values from a memory should be matched with the delay-time of computing one pixel in the core so as to be pipelined. Therefore, to achieve real-time processing, decreasing loading-time is essential. In other words, the goal of memory management is to adjust loading-time to be matched with core's processing time. To decrease loading time for processing cores, we may attempt to obtain pixel values using plural memories. However, increasing parallelism would require excessive memory resources. Motivated by the facts, to find a tradeoff between processing time and memory resources, in this paper, we propose an n^2 two-line buffers (each two-line buffer is M by 2 line buffer) as shown in Figures 3 and 4 to obtain entire pixel value as fast as possible. Note that the n^2 two-line buffers use reasonable buffer capacity to facilitate the n^2 -way parallel processing with providing another 2-way parallelism by

simultaneously loading i -th row data to processing core and storing $(n+3)$ -th row data to buffer memory (and after n clocks later, the role will be switched each other which is so-called "ping-pong scheme" that we shall explain later).

Also, note that by adjusting a number of two-line buffers, we will be able to match loading time with delay-time of image processing core so as to provide a seamless pipeline. Therefore memory management unit usually controls data's addressing and timing along with determining the number of two-line buffers. Figure3 shows our proposed memory management unit with utilizing n^2 two-line buffers, in this case with $n=3$.

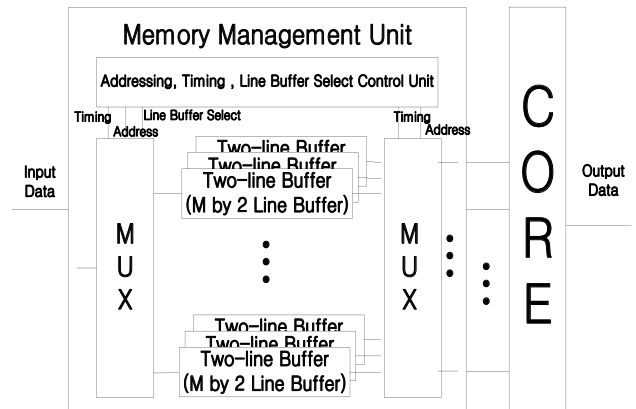


Figure 3. Memory Management Unit

As illustrated in Figure4, each two-line buffer acts as "ping-pong scheme". The upper row is loading data to the processing core, while the lower row is storing the next data. And the role will be switched each other after every n clocks. Therefore loading and storing operation are executed simultaneously for a real time processing.

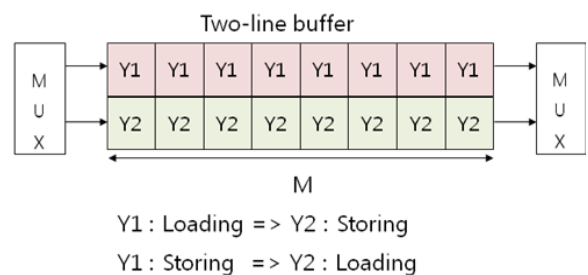


Figure 4. Operation of each Two-line buffer

Figure5 shows our buffer structure in memory management block for the example of 3 by 3 pixel array has three line buffer blocks (Y axis), and each block has n two-line buffers (Z axis). Each block corresponds to a row of the pixel array, and the three two-line buffers in each block are for loading

three pixels in the same row (upper buffer is for the first row and lower buffer is for the fourth row).

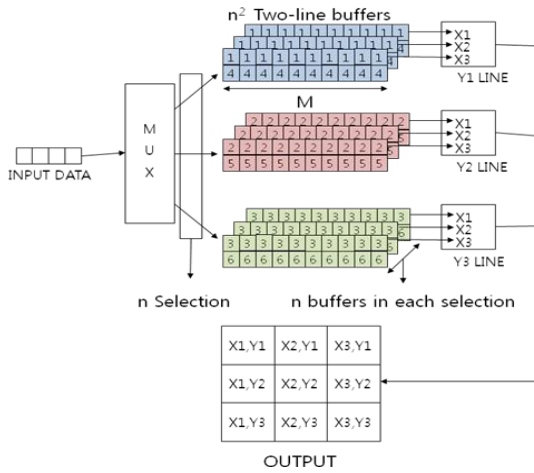


Figure 5. Example of memory management for an n by n window where $n = 3$

As shown in Figure 5, for storing input data to buffer blocks, we need n ($=3$) enable signals to select the designated i -th block. The i -th block contains three identical copies of the i -th row of the input. After loading the i -th row of the input to i -th block, the j -th copy of the i -th block that corresponds to the j -th column data at the i -th row of the input will be loaded to the processing core. In this way, we can load n by n pixel values simultaneously.

Let's assume that we have an M by N image and the image processing core consumes one clock per pixel. Therefore the total processing time denoted as T_T is equal to $T_S + T_L$, where T_S denotes the storing time of pixel values, and T_L is the loading time of pixel values. We should have stored $M \times N$ pixel values in a memory if the storing time was faster than loading time. However, since it takes a clock per one target pixel for both loading and storing and we need to processing each n by n pixel array to perform the image processing of each target pixel, we can adjust to store n lines instead of M lines, obviously, $n < M$. Also, note that we need to load an n by n pixel array consuming $n \times n$ clocks per each target pixel since each target pixel can be processed after n^2 pixel values have been loaded. Based on this facts and using our scheme with n^2 two-line buffers (n^2 pixels will be processed simultaneously and, only $M \times N$ clocks are consumed for an M by N image processing, instead of using $M \times N \times n^2$ clocks. Therefore the total processing time of 1-way is as:

$$T_T^1 = T_S^1 + T_L^1 = M \times N + (M \times N \times n^2) \quad (1)$$

And the total processing time of our proposed method with (n by n)-way parallel processing is as:

$$T_T^{nxn} = T_S^{nxn} + T_L^{nxn} = M \times n + (M \times N) \quad (2)$$

In addition, if we have 2 processing cores such as erosion and dilation with series connection including an n by n window processing, the total processing time will be increased

twice. However, by applying our method, the processing cores can be pipelined. Also, the pipelined architecture consumes only two times of storing time. Therefore the total processing will take $2 \times M \times n + (M \times N)$ clocks using our method instead of $2 \times (M \times N + (M \times N \times n^2))$ clocks originally. Note that we also could adjust the number of two-line buffers to match core's processing time with loading-time.

IV. REFINEMENT CORE

Figure6 shows the architecture of the erosion operation of refinement core. The size of input image is 160 by 90. The erosion has a 3 by 3 window and operates 9 pixels comparison. And the dilation has a 5 by 5 window and operates 25 pixels comparison. Since erosion operation uses a 3 by 3 window, we use 3 line buffer blocks, and each block has 3 two-line buffers. The core obtains 9 pixel values per each clock. And the dilation core is the same as the erosion core only obtaining 25 pixel values per each clock. Since two core's processing time and loading time is the same, the erosion core and the dilation core can be effectively pipelined.

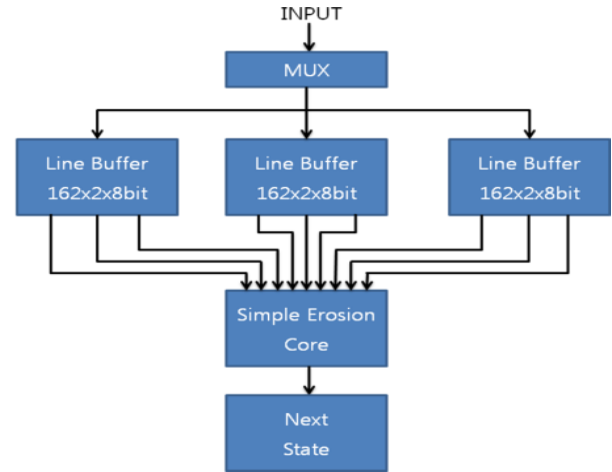


Figure 6. Architecture of the erosion operation

V. EXPERIMENTAL RESULTS

All modules are designed with Verilog and synthesized on Virtex 5-VLX330 FPGA using Xilinx's ISE. Figure7 shows the image processing results of each operation. The result of erosion operation (bottom-left) has less noise and more holes than the original image (top), and the result of dilation operation (bottom-right) has no holes with smoother shape.

Our proposed architecture focuses on real-time processing. Pipelined two core's total experimental processing time is approximately 16680 clocks, which is suitable for real time processing. Erosion core's processing time is 14880 clocks, and dilation core's processing time is 15200 clocks. The memory capacity using proposed method is 10 Kbyte. As shown in Table1, the resource utilized from our proposed

architecture consumes erosion with 864 and dilation with 2400 memory blocks, respectively. The total processing time is originally 518400 clocks, and erosion and dilation's processing time is 144000 and 374400 clocks, respectively. The memory capacity is 28 Kbyte if each core has own memory.

Therefore, if we consider a singular core, our proposed architecture's processing-time using a 3 by 3 window is 9 times faster, and using a 5 by 5 window is 25 times faster. In addition, if we consider two cores which are pipelined, our proposed architecture's processing-time using a 3 by 3 window and a 5 by 5 window is 30 times faster, and memory capacity consumption is slightly lower.

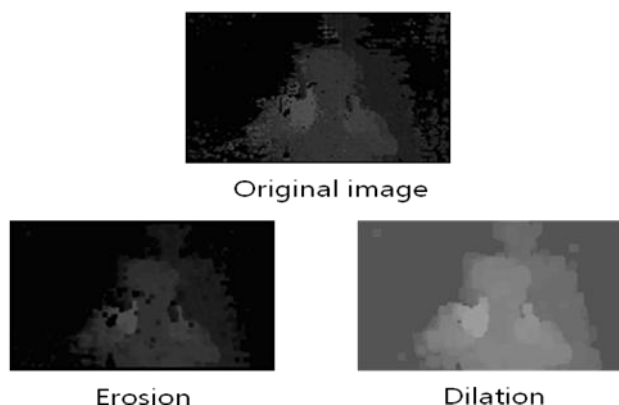


Figure 7. Actual image refinement processing results

VI. CONCLUSION

In this paper, our proposed architecture achieves real-time processing, such that loading-time is adjusted to match core's processing-time to be effectively pipelined. Also, our experimental result of performing the image refinement processing showed that our proposed architecture is suitable for real-time processing with acceptable memory capacity.

TABLE 1. COMPARE PROPOSED METHOD WITH PREVIOUS METHOD ON XC5VLX330-1FF1760

Source	Erosion operation	Dilation operation	Erosion & Dilation operation	
	Proposed	Proposed	Proposed	Previous
Slice LUTs	1008	2770	3778	4018
Block RAM/FIFO (18 Kbits memory block)	864	2400	3264	3600

ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(S-2010-1119-000)

REFERENCES

- [1] P.A. Laplante and A. D. Stoyenko, Eds, "Real-time imaging-theory, techniques and applications", IEEE Press, New York, NY, USA, 1996.
- [2] N. Ranganathan, *VLSI & Parallel Computing for Pattern Recognition & Artificial Intelligence*, Series in Machine Perception and Artificial Intelligence, Volume 18, World Scientific Publishing, 1995.
- [3] C. Torres-Huizil and M. Arias-Estrada, "Configurable Hardware Architecture for Real-Time Window-Based Image Processing", on FPL 2003, LNCS 2778, pp.1008–1011, 2003.
- [4] M.H.Seyed Javadi, H. Rafi, S.Tabatabaei and A.T.Haghighat, "An Area-Efficient Hardware Implementation for Real-time Window-based Image Filtering" sitis, pp.515-519, 2007.
- [5] J. Electron, "New hardware memory management architecture for fast neighborhood access based on graph analysis", Imaging, Vol. 11, 96 (2002).
- [6] (Haiqian Yu, Leeser M. "Optimizing Data Intensive Window-based Image Processing on Reconfigurable Hardware Boards", Signal Processing Systems Design and Implementation, 2005. IEEE Workshop on, pp491-496, Nov. 2005.