후정렬 병렬 가시화 클러스터를 위한 저비용의 하드웨어 영상 합성기

타로파 에마뉴엘, 이원종, 바슨 스리니, 한탁돈
연세대학교 컴퓨터 과학과
{etaropa°, airtight}@yonsei.ac.kr
{srini, hantack}@cs.yonsei.ac.kr

A Cost-Effective Hardware Image Compositor for Sort-Last Parallel Visualization Clusters

Emanuel Taropa[0], Won-Jong Lee, Vason P. Srini, Tack-Don Han
Department of Computer Science, Yonsei University

Abstract

Real-time 3D visualization of large datasets imposes a distributed architecture of the rendering system and dedicated hardware for image composition. Previous work on this domain has relied on prohibitively expensive cluster systems with hardware composition done by complicated schemes. In this paper we propose a low-cost hardware compositor for a high performance visualization cluster. We show the system's design and the results obtained using Simulink [1] for our image composition scheme.

1. Introduction

Interactive 3D visualization is necessary from online high detail design systems to scientific computing, geo-science and medical imaging applications. All these applications require processing of *huge datasets* at *interactive speeds*, while being *computationally intensive*.

A viable solution for addressing the above requirements is to use clustering systems that offer a flexible, scalable and relatively cost-effective environment [2], [3]. The common technique used is to divide the dataset to be rendered as equally as possible among the rendering PCs, to render each subdivision in parallel and finally to obtain the whole image by composing the rendered subdivisions.

The scalability of the system and hence its performance is limited by bottlenecks that appear from initial image division algorithm to the synchronization among the rendering nodes and the communication network.

In our opinion, the most limiting of these bottlenecks occurs in the image composition stage. As composition is applied over the transformed original dataset, its computation time over data size factor is of utmost importance for the overall system's throughput.

Image composition has been previously implemented both in software [4], [5] and in hardware [6], [7]. The software approach has the advantage of a reduced cost but its computation time dependence with the number of communicated pixels makes it very slow for something else but the smallest data sizes. The hardware implementation minimizes the number of transmitted pixels and can efficiently compose sub-images in parallel, thus offering a minimum delay time. Its main disadvantage remained until now its high cost.

In this paper we propose an inexpensive hardware compositor for high performance visualization cluster. The hardware image compositor is detailed by section 2. Section 3 contains the results on few datasets using Simulink [1]. The paper concludes with section 4 where conclusions are drawn and future research directions are identified.

2. System Design

The principle used for image composition is Binary Space Partitioning [8]. Thus, the inner structure of the hardware image compositor is binary tree-shaped, where parent nodes are doing image composition for their two children nodes. The compositor design process has started from the model given for the Mitsubishi Precision Compositor (MPC) [7], but its implementation is done using less expensive hardware and reducing the gate count.

Simulink was used both as a design and as a simulation environment for proving the correctness of the compositing circuit and for performing initial performance estimations. From Simulink, VHDL code has been obtained that will be used to configure the ML310 FPGA board [9]. The Simulink implementation of the hardware compositor is given in Figure 1.

2.1 Priority Assignment

Using a BSP algorithm, we need to realize the blending of images according to their order in the final image. For specifying ordering among images, we use the notion of *priority*. The higher the priority, the closer to the observer the respective sub-image is.

As the composition happens in parallel for all pairs of sub-images of each frame, the priorities need to be known at the beginning of each composition step. The image division algorithm used for distributing the image to the rendering nodes can verify the order of sub-images in the final image and thus assign a priority for each sub-image of each frame. The rest of priorities for partially composed sub-images of the same frame will be determined using *maximum propagation*,

$$Pr(I1 \mid I2) = Max(Pr(I1), Pr(I2)) \quad (1)$$

where $Pr(I)$ represents the priority associated with sub-image I.

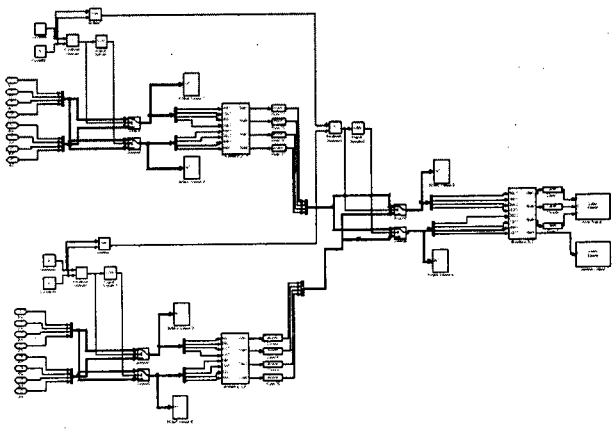


Figure 1: Hardware Image Compositor for a rendering cluster of 4 PCs

The source and destination for the blending circuit can be easily selected among the two children of the respective blending node. The blending equation controls how the source and destination images are combined in the final image.

2.2 Alpha Blender Architecture

The design of the alpha blender can be done in two ways: *generic* – supporting all the blending functions defined in OpenGL [10] and *specialized* – implementing a single blending function. The generic circuit is easily re-configurable but has a higher gate count. The specialized circuit has a lower gate count but needs to be redesigned if the blending function is changed.

Because efficiency is our utmost concern, we have chosen the second option for our blender architecture. The alpha blending will be performed in *back to front* order, implementing OpenGL's *blend (1, 1 – Source Alpha)* for each channel. The resulting blending circuit is represented in Figure 2.

Our design started from the standard alpha blending equations expressed for the blending mode *(1, 1 – Source Alpha)* (2) and (3), where S_A represents the source alpha channel, A_S the scaling factor, C_S and C_D represent the initial channel values for source and destination images.

$$C = C_S + C_D * D_C \quad (2)$$

$$D_C = 1 - S_A * A_S \quad (3)$$

Factoring out the constants from (2) and (3) and using *back-to-front rendering* we were able to reduce the gate count for processing each pixel from 16 multipliers and 12 adders to 5 multipliers and 8 adders, while maintaining the same level of parallelism. The equivalent channel expression is given in (4).

$$C = C_S + C_D - (D_C * C_D) \quad (4)$$

We have used rounding blocks for restricting the computed values to the interval [0, 255]. For domain changes from byte to double precision real and back to byte standard converters were used.

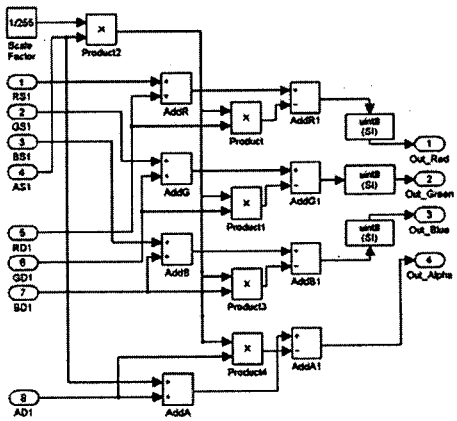


Figure 2: Alpha Blender Architecture

With a low gate count and using only simple blocks, the resulting blending circuit is highly effective, having both a small delay and power consumption.

2.3 Data Initialization and Loading

The input data for the compositing hardware has been generated using volume rendering with per fragment lighting and post shading. The distribution of data on each rendering node has been done using a load balancing algorithm and parallel communication primitives.

Each sub-image contained four channels: R, G, B and alpha. In this initial stage of the system, each sub-image data has the same size as the final image, regardless of the total number of divisions initially performed.

Therefore, in all but the final step of composition *image to empty space* blending or *empty to empty space* blending occurs. Possible directions for optimization will be detailed in a future paper.

The hardware compositor designed in Simulink imported the image data from Matlab [11] workspace. The matrices corresponding to each channel were read as two dimensional time-invariant arrays of signals. For verifying the correctness of the composition algorithm we displayed resulting sub-images at each step of the composition using Matlab's Video and Image Processing Blockset.

2.4 Scalability

The circuit presented in Figure 1 is realized for a cluster composed by 4 rendering PCs and 1 display node. Using a very simple design for the image compositor, we can easily extend the binary tree image composition architecture to support any number of large rendering nodes. The basic principle of priority inference using maximum propagation given in (1) remains valid for any dimension of the rendering cluster.

The components number has a linear dependence with the doubling size of the rendering cluster: knowing that the number of interior nodes in a complete binary tree with n leaves is n-1 we can give an upper margin in the form of *ceiling [(n-1)/2]* for the added number of hardware blenders required for the image composition.

There is an obvious tradeoff between the simplicity of the composition scheme and the scalability of the circuit. For our initial design, the composition process is greatly simplified by considering each sub-image to be of equal size with the resulting image. Therefore the compositing nodes are interconnected in a straightforward manner allowing for efficient data

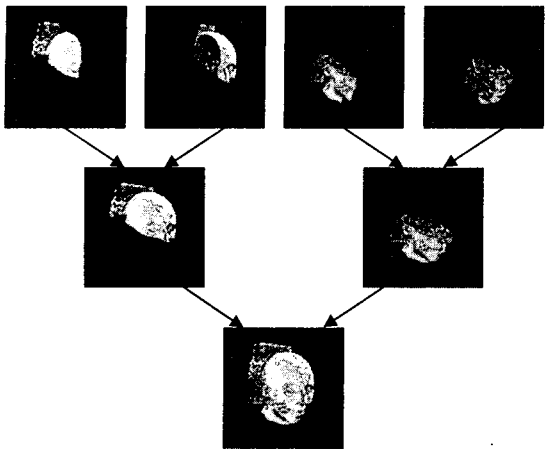


Figure 3: Human skull dataset composition for a rendering cluster consisting of 4 PCs and one display node. The first level is the input data from each of the rendering PCs to the compositing hardware. The second and third levels represent composition steps.

transmission between the compositor's inner nodes. It remains to be experimented if advanced heuristics for reducing the sub-image size resulting in complicated compositor architecture will yield a higher performance by diminishing the number of transmitted pixels but limiting the scalability of the circuit.

3. Experimental Results

We verified our composition scheme by performing several experiments on few standard datasets. Figure 3 shows the results obtained on a 4 PCs rendering cluster with one display node for a CT skull dataset (256 x 256x 256), courtesy of University of North Carolina.

Having implemented an easily scalable and flexible image composition scheme, we tested it for other rendering cluster dimensions. In Figure 4 results are showed for a rendering cluster formed by only 2 PCs and one display node. For the simulation we used the "lobster" dataset (301 x 324 x 56), courtesy of SUNY Stony Brook.

The results were obtained using Simulink simulation environment. The data were represented as time-invariant matrices of signals and were imported from Matlab workspace into Simulink environment.

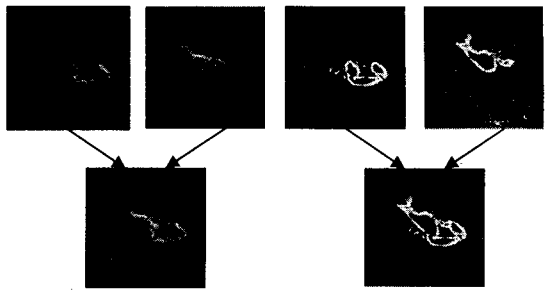
When composing two images, the simulation is ran for each pair of channels (R, G, B and alpha) of source and destination images sequentially. In an FPGA implementation of the circuit, the parallelism will be better exploited.

Thus we can work in parallel on a large portion of the input images and we can pipeline our computation stages.

4. Conclusions and Future Work

We have presented an inexpensive hardware image compositor for a high performance rendering cluster system. The design we implemented has a good scalability and is extensible, the number of added components having a linear dependence with the doubling of the rendering nodes.

The hardware compositor design implemented in Simulink will be converted using Xilinx System Generator [12] to be mappable on the Xilinx ML310 board. Currently we are building the rendering cluster, using high performance PCs with high-end GPUs. This system will serve as testbed for advanced visualization algorithms and applications.



(a) RGB composition (b) Alpha composition

Figure 4: Lobster dataset composition for a rendering cluster consisting of 2 PCs and one display node. The first level is the input data from each of the rendering PCs to the compositing hardware. The second level represents the output of the compositing hardware.

REFERENCES

[1] The MathWorks – Simulink Simulation and Model Based Design http://www.mathworks.com/products/simulink/

[2] S. Muraki, E.B. Lum, K.L Ma, M. Ogata, X. Liu – A PC cluster system for simultaneous interactive volumetric modeling and visualization *Proceedings of Parallel Visualization and Graphics* '03 pp. 95-102

[3] M. Strengert, M. Magallon, D. Weiskopf, S. Guthe, T. Ertl – Hierarchical visualization and compression of large volume datasets using GPU clusters *Proceedings Symposium on Parallel Graphics and Visualization* '04, pp. 1-7, 2004

[4] K.L. Ma, J.S. Painter, C.D. Hansen, M.F. Krogh – Parallel volume rendering using binary-swap image composition *IEEE Computer Graphics and Applications*, 14(4) pp. 59-68, 1994

[5] A. Stompel, K.L. Ma, E. Lum, J.Ahrens, J. Patchett – SLIC: Scheduled Linear Image Compositing for Parallel Vollume Rendering *Proceedings of Parallel Visualization Graphics '03*, 2003

[6] M. Magallon, M. Hopf, T. Ertl – Parallel volume rendering using PC graphics hardware *Proceedings of Pacific Graphics '01*, pp. 384 - 389

[7] M. Ogata, S. Muraki, X. Liu, K.L. Ma – The design and evaluation of a pipelined image compositing device for massively parallel volume rendering. *Proceedings of Workshop on Volume Graphics '03*, pp. 61-68, 2003

[8] H. Fuchs, Z. Kedem, and B. Naylor – On Visible Surface Generation by A Priori Tree Structures *Proceedings of the ACM SIGGRAPH '80 Conference*, pp. 124-133, 1980

[9] Xilinx - ML310 Board Documentation http://www.xilinx.com/products/boards/ml310/current/

[10] OpenGL – The OpenGL Graphics System: A Specification Version 2.0 http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf, 2004/10/22

[11] The MathWorks – Matlab The Language of Technical Computing http://www.mathworks.com/products/matlab/

[12] Xilinx – Xilinx System Generator Users Guide Version 7.1 http://www.xilinx.com/products/software/sysgen/app_docs/user_guide.htm