Vector Processor Based Architecture for Gradient and Normal Computation in Real-Time Volume Rendering

Rajkiran Gottumukkal and Vijayan K. Asari
Department of Electrical and Computer Engineering
Old Dominion University, Norfolk, VA 23529

Abstract- **Volume rendering is a key technique in scientific visualization. The high computational demands of real-time volume rendering and continued technological advances in the area of VLSI give impetus to the development of special-purpose volume rendering architectures. This paper presents the architecture for the preprocessing stage of volume rendering. In the preprocessing stage the gradient and normal of all the pixels are computed. There are 16 million pixels in a 256^3 dataset; real-time volume rendering cannot be achieved by using traditional architectures to calculate the gradient and normal of such a large dataset. To achieve the goal of real-time volume rendering the preprocessing stage should be fast. An architecture using vector registers is proposed in this paper, which aims at speeding up this preprocessing step.**

1 Introduction

Volume visualization is concerned with the representation, manipulation and display of volumetric data, typically represented by a 3D grid of scalar values. Volume visualization has become a key factor in the understanding of the large amount of scientific data generated in a variety of disciplines. Examples include sampled data from biomedical and geophysical measurements, and simulated data from finite element models or computational fluid dynamics. Another source of 3D data are various imaging systems like computer tomography (CT), magnetic resonance imagery (MRI), ultrasound tomography (UST), and other systems which produce large amounts of data in short time. The physician has to work with and analyze this data for diagnostic and therapy planning purposes [1]. This huge amount of data forces the physician to switch from investigating one slice after the other to view the 3D reconstructions.

Direct volume rendering algorithms are employed to reveal the internal structure of the data. However, their high computational expense limits interactivity and real-time frame rates. The main computational aspects of volume rendering are the massive amount of data to be processed resulting in high storage, memory bandwidth, and arithmetic performance requirements. The three-dimensional volume can be represented as identical cells called *voxels,* which are arranged, in a fixed, regular, rectilinear grid. The volume can be visualized as slices of two-dimensional grid. The volume rendering architecture was implemented using the shear warp factorization algorithm. This algorithm was chosen because it was proved that it is capable of rendering large data sets effectively [2-3]. The initial step involved in this algorithm is the computation of the gradients and normals of all the voxels in the volume. The gradient shows the density changes in the volume. Using the gradient and interpolated sample value, a local shading model is applied and sample opacity is assigned. An efficient VLSI implementation of the gradient and normal

computation module for the volume-rendering unit is presented in this paper. The architecture for the gradient computation is developed based on an array of vector registers to enable faster computations and the architecture for surface normal is built in a look up table base.

2 Gradients and Normal Computation

The surface normal n (x, y, z) at any point in the volume is defined as a unit vector parallel to the local gradient of the voxel scalar value d (x, y, z). The surface normal can be obtained as:

$$n(x,y,z) = \frac{\nabla d(x,y,z)}{|\nabla d(x,y,z)|} \tag{1}$$

The gradient ∇d (x, y, z) is approximated using the central difference gradient operator and it is computed from the intensity values of the neighboring voxels as:

$$\nabla d(x,y,z) = \begin{cases} \frac{1}{2}[d(x+1,y,z) - d(x-1,y,z)]i \\ +\frac{1}{2}[d(x,y+1,z) - d(x,y-1,z)]j \\ +\frac{1}{2}[d(x,y,z+1) - d(x,y,z-1)]k \end{cases} \tag{2}$$

3 Vector Processor Based Architecture

The architecture proposed in this paper tries to exploit the parallelism available in the equations of gradient and normal computations [4]. The dataset was assumed to be initially loaded into a RAM, from where it will be transferred to the vector registers. The vector registers are a set of 8-bit registers assembled in a parallel fashion. A vector register of length *m* can be defined as *m* number of 8-bit registers in parallel. There are a total of nine such vector registers in the proposed architecture, of which six will be loaded with the gray levels of the pixels from the RAM. The nine vector registers are labeled as VR1 to VR9. The first *m* pixels in the scanline are loaded into VR1, and the next *m* pixels are loaded into VR2, where *m* is the length of the vector register. The values in these two vector registers are fed to a subtractor for the parallel computation of the first term in the gradient equation. VR3 and VR4 are loaded with the corresponding *m* pixels of the scanline above and below the current scanline. By feeding these to a subtractor the second term of the gradient equation will be computed in parallel. Similarly the corresponding *m* pixels of the slice after and before the current slice were loaded into VR5 and VR6 respectively. From VR5 and VR6 the last term of the gradient equation was computed in parallel. Fig. 1 shows the arrangement of the vector registers employed in this architecture for loading data from the RAM.

Once the vector registers are loaded, the gradient and normal for *m* pixels can be computed in parallel. The greater the length of the vector registers *m*, the greater will be the parallelism available. But, increasing the vector register length increases the time to load the vector registers

This in turn increases the time to finish the preprocessing task. The number of gates required for the hardware implementation of the architecture would also increase with the increase in m. Hence there should be a trade off to fix the length of the vector registers. The criteria to select ideal m will be discussed in the next section.

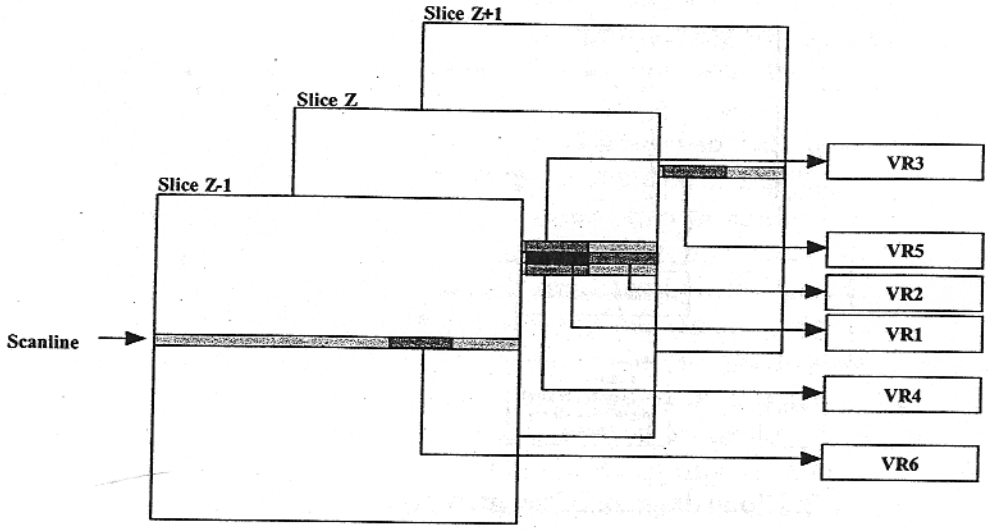

Figure 1: Loading vector registers VR1-VR6

Vector registers VR1-VR6 are slightly different form VR7-VR9. VR1-VR6 has only one 8-bit data input bus, which will feed all the registers in the vector register set. Individual registers in these vector registers are selected by the write signal. In the case of VR7-VR9 there are m 8-bit data buses, which will feed the corresponding registers in the vector register. Hence unlike VR1-VR6, VR7-VR9 can be loaded in parallel. VR7 is used to store the first term of the gradient equation, VR8 is used to store the second term of the gradient equation and VR9 is used to store the third term of the gradient equation. The block diagram for the preprocessing stage is shown in Fig. 2.

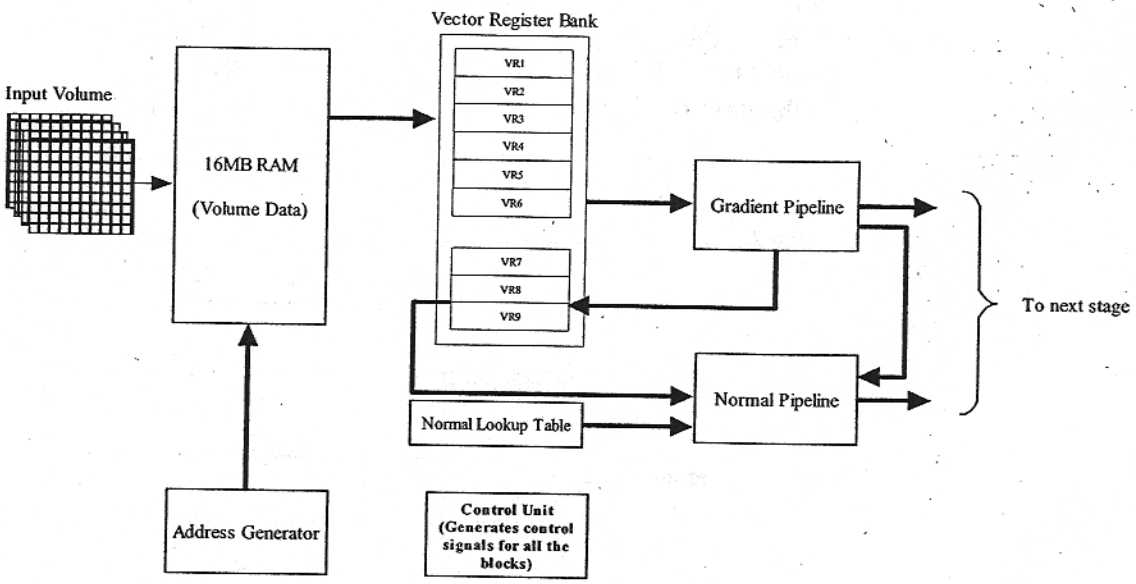

Figure 2: Block diagram of the preprocessing stage.

The gradient pipeline consists of m 8-bit subtractors and m 8-bit multipliers. Once the data is loaded into VR1 and VR2 the pipeline operation begins. The output from the subtractor asserts the overflow signal if the result is negative. When the overflow signal is asserted the result will become 2's complemented. These results are stored in VR7 and the registers in a pipelined fashion. The registers in the pipeline are used to accumulate the three terms of the gradient equation. Meanwhile VR3 and VR4 are being loaded one after the other. Once they are loaded fully the same process is performed on VR3 and VR4 and the results will be stored in VR8 and added to the registers in the pipeline and stored back in these registers. At the same time, VR5 and VR6 are loaded and the same process is performed on them. The results are stored in VR9 and added to the registers in the pipeline and stored back. The values accumulated in the registers are fed to a divider where they are divided by 2 to compute the gradient. The outputs from the registers in the gradient pipeline are also fed into the normal pipeline, where they are stored in registers of the normal pipeline. Now new set of data is loaded into VR1 and the process continues for the entire dataset.

Once the sum of the three terms in the gradient equation is accumulated in the registers of the gradient pipeline, it is loaded into the registers in the normal pipeline. The normal pipeline uses the m 8-bit divider in the gradient pipeline since it is free at this stage. The divider is used to divide the corresponding values in the registers of the normal pipeline and VR7. The result is accumulated in another set of registers in the normal pipeline. In the same way the divider is used on VR8 and VR9 and the results are added with the contents of the second set of registers and stored back in them. In the next stage of the pipeline, VR8 and the results accumulated in the second set of registers in the pipeline are fed in to a multiplier. Using the result of the multiplication as the index, the normal value is read from the lookup table. The lookup table was loaded with the normal values at the initialization phase.

4 Simulation Results and Discussion

The vector processor based architecture for the gradient and normal computation module of the volume-rendering unit was implemented in Altera Quartus II FPGA design platform for eventual programming of the Apex 20K device. The hardware design was performed for various sizes of the vector registers to study the effectiveness of the proposed design and to obtain an ideal size configuration of the architecture in terms of the VLSI area requirement and time of computation. The length of the vector registers m, was varied from 8 to 128 to obtain an optimum level of parallelism. The simulation result describing the number of flip-flops needed to implement the architecture with respect to the length of the vector registers is shown in Fig. 3. It also illustrates the time taken for the preprocessing stage with respect to the vector register size. By varying the vector register length, the number of adders, multipliers and dividers required for the implementation would also vary. For a vector register length of 64, there would be 64 adders, multipliers and dividers functioning in parallel. So the number of flip-flops used by the architecture increases exponentially with the increases in m. It was also observed that increasing m beyond 64 doesn't increase the preprocessing speed, since the pipeline would be idle for the duration of loading the vector registers. A preprocessing time of 0.74 seconds was achieved with the above implementation for a volume data of size 256^3 voxels by choosing the length of the vector registers m=64. Real-time volume rendering requires at least 30 frames per second. Hence for real-time volume rendering of the above data set, the entire process has to be completed in about 0.2 seconds. This computational speed can be achieved by using several parallel nodes consisting similar architecture.

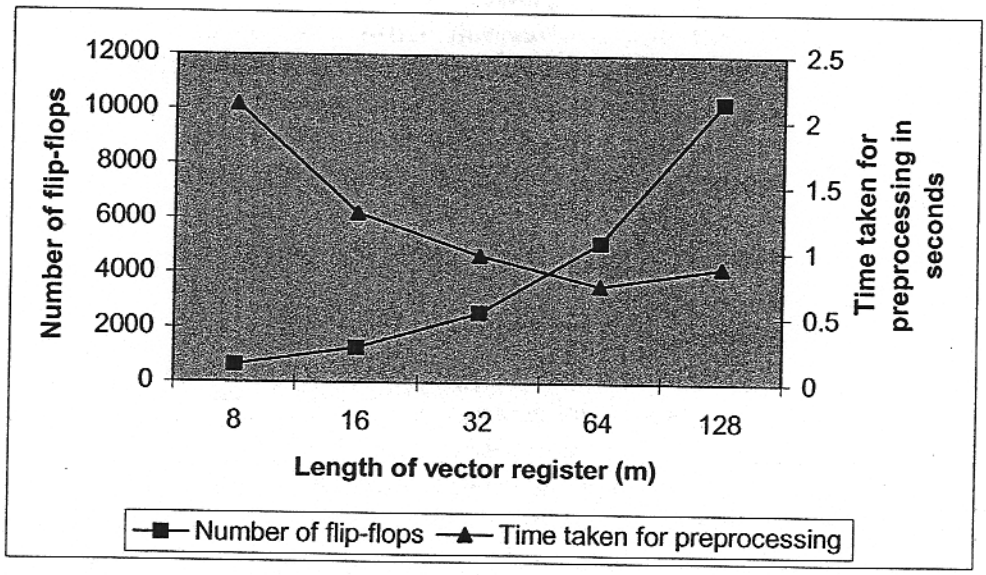

Figure 3: Time taken for preprocessing and number of flip-flops used with respect to the length of vector register.

5 Conclusion

A vector processor based architecture for the computation of gradient and normal in a real-time volume rendering system has been presented in this paper. The simulation results of the FPGA based implementation of the proposed architecture were encouraging. The speed of the preprocessing stage in real time volume rendering can be improved by providing several nodes in parallel, where each node consists of a vector processor based architecture as the one presented in this paper.

References

[1] J.A. Adam, "Medical Electronics," *IEEE Spectrum*, pp. 80-83, Jan. 1995.

[2] P. Lacroute and M. Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation," *Proc. SIGGRAPH '94*, Orlando, Florida, pp. 451-458, July, 1994.

[3] J. Hesser, R. Maenner, G. Knittel, W. Strasser, H. Pfister, and A. Kaufman, "Three Architectures for Volume Rendering," *Computer Graphics Forum,* vol. 14, no. 3, pp. 111-122, Aug. 1995.

[4] M. J. Flynn, *Computer Architecture Pipelined and Parallel Processor Design*, Jones and Bartlett Publishers, pp. 435-438, 1995.