5U−5

Applications of the Global Frame Buffer to Volume Visualization

Yukio Sakagawa†, Satoshi Nishimura‡, Yoshihisa Shinagawa†and Tosiyasu L. Kunii†‡
†The University of Tokyo, ‡Kubota Computer Inc., †‡The University of Aizu

1 Introduction

In this work, the Global Frame Buffer (GFB) of the VC-1 and a new parallelized implementation of volume visualization is developed.

VC-1 is a multiprocessor environment for computer graphics, and the GFB is used basically to isolate the image data that flows at the VC-1 from the data that is being displayed.

VC-1 [5] is a loosely-coupled multiprocessor with a frame buffer subsystem called Conflict-Free Multiport Frame Buffer (CFMFB). The CFMFB consists of *local frame buffers* (LFBs), a *pipelined image merger* (PIM), and a *global frame buffer* (GFB). Every processor element (PE) has its own LFB to hold the sub-image (including Z-values) created by it. The PIM periodically superimposes the sub-images stored in the LFBs and transfers the merged picture to the GFB.

To extend the functions of the GFB, an additional device called the merging unit for the accumulation buffer (MUACC) is introduced. This hardware device can enlarge the range of applications of the VC-1 and improves its general performance, transferring processing load from the processor elements to the GFB.

2 Global Frame Buffer

The data that come to the GFB flow in the same dataflow order as a video scan. The GFB [6] was developed with this singular characteristic in mind.

The primary role of the GFB in the VC-1 is to isolate the frequency and phase of the pipeline image merger scan rate and the CRT scan rate [4]. In the context of the VC-1, the functions of the GFB are: integration all local frame buffer, treatment of FLB overflow LFB, generation of signals to synchronize the PIM, feedback screen data, transference of digitalized image data and accumulation of frame buffer.

The GFB is hosted and managed by a PC-AT like computer. The GFB is composed of Video Frame Buffers, a Video Controller, a PIM Scan Counter Controller, a Accumulation Buffer and a Merging Unit for accumulation buffer (MUACC).

The range of applications of an accumulation buffer is very large, as to antialiasing [1], and improvement of image quality [2]. The MUACC is designed on a field programmable logic device, so the range of operations it can support is very large, and can be changed very easily.

3 Algorithm Description

3.1 Volume Rendering

Volume rendering [3] obtains the volumetric image directly from the volumetric data. The case in study is volumetric data with transparency, and the algorithms are the ray casting and the Z-planes, a new algorithm suitable for the hardware we use, due to the PIM and the MUACC.

When considering transparency, the volumetric data is represented by the pair density C (color) value and the degree of transparency, α, where $\alpha = 1$ means a complete transparent and $\alpha = 0$ means a complete opaque material.

For a given viewline that crosses several elements, the observer would have the following view:

$$V = C_0(1-\alpha_0) + \alpha_0(C_1(1-\alpha_1) + \alpha_1(\ldots \alpha_{n-2}(C_{n-1}(1-\alpha_{n-1}) + \alpha_{n-1}C_n(1-\alpha_n)))). \quad (1)$$

A higher index indicates the element is in a deeper position in the scene.

3.2 Implementation

We developed a new implementation of visualization of volumetric data with transparency, the Z-Planes algorithm, that makes use of the graphical structures (PIM, MUACC, Accumulation buffer) of the VC-1 (fig. 1).

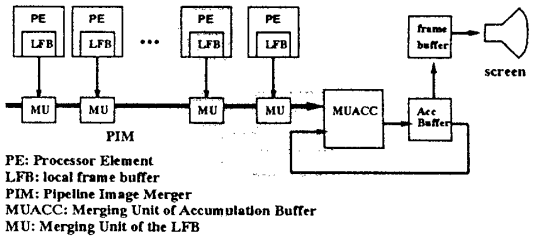

Figure 1: Simple architecture structure of VC-1

There is no need to preprocess the volumetric model database, and data space division for database distribution is adopted.

The algorithm is based on the equation 1. Consider the model database divided in planes parallel to the

screen, and at the system level, the algorithm begins with the PEs drawing the P_n (farthest plane from the screen), (C_n, α_n), and transferring it to GFB. Then the following plane P_{n-1} is drawn and transferred to GFB. The algorithm continues drawing and transferring the planes until P_0, the closest plane to the viewer (see fig. 2):

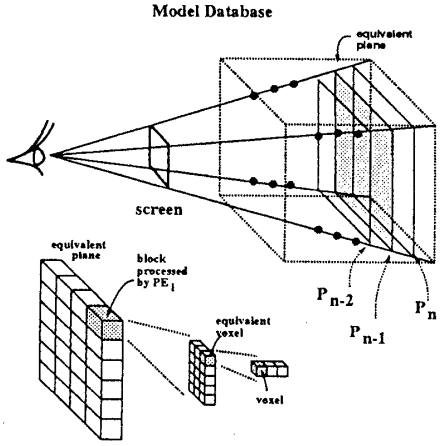

Figure 2: View of a Volumetric Data considering Transparency (3D array of voxels)

For the parallel implementation, the host computer sends the same Z (depth) position of the Image space to all the processor elements.

The GFB, when receives the (C_Z, α_Z) values, executes the equation 2, by the MUACC hardware:

$$V_{accZ} = C_Z(1 - \alpha_Z) + \alpha_Z(V_{acc(Z+1)}); \quad (2)$$

The accumulated value V_{accZ} is stored at the Accumulation Buffer.

After the last plane, (C_0, α_0), is sent, the resulting (accumulated) view is displayed on the screen.

The PIM takes 1/30s to transfer one complete Z plane to the GFB. In order to make more efficient use of the PE processing time, and at the same time to decrease the number of Z planes sent to GFB, the PE itself integrates two or more layers of Z planes in one equivalent Z plane in 1/30s, and write it at the LFB.

At PE level, another algorithm is implemented. To calculate the equivalent characteristic value of the plane, an algorithm similar to ray casting is used. V is the blended value of the voxels, and α is the accumulated attenuation factor. We start with the voxels closest to the screen with initial values $V_{old} = 0$ and $\alpha_{old} = 1$, and blend their values as follows:

$$V_{new} = V_{old} + \alpha_{old} * C_Z * (1 - \alpha_Z); \quad (3)$$

$$\alpha_{new} = \alpha_{old} * \alpha_Z; \quad (4)$$

the algorithm continues until α reaches 0 (the accumulated material becomes completely opaque), or the plane finishes. The equivalent characteristic values when the stop condition occurs is:

$$C_e = \frac{V_{new}}{1 - \alpha_e}; \qquad \alpha_e = \alpha_{new}; \quad (5)$$

If α_e is 1 when the algorithm is finished, it means that the equivalent voxel is completely transparent and whatever color it has it will not be projected on the screen.

The characteristic values of the voxels are calculated using trilinear interpolation of the voxel vertices.

We have scalability in the resolution of the image with different step values for Z, or with super-sampling of the voxels.

4 Discussion

Compared to the ray casting algorithm, the expected speed-up obtained with the implementation of the Z-Planes algorithm proposed in this work is due to two factors: 1) Processing of the plane with same Z value is distributed linearly among the processor elements, and with only a few (if any) communication among the PEs. 2) One of the heaviest operations to be executed, represented by equation 2, is executed "on the fly", as soon as the GFB receives the (C_Z, α_Z) values. It is implemented in hardware and lighten processing load necessary to integrate the several Z planes.

The advantage of the ray casting algorithm is that it can stop processing a ray as soon as it reaches transparency zero. The Z-Planes algorithm processes all the volumetric data no matter how close to the screen is the opaque object.

The best performance is obtained when the time the processor elements take to calculate the equivalent Z plane is the same as the time the PIM takes to transfer the local frame buffer data to the GFB.

The GFB also enables direct participation of the host computer in the generation of the image. Without the GFB, the participation of the host computer is limited to management of the resources of the PEs.

References

[1] Loren Carpenter. The A-buffer, an antialised hidden surface method. *ACM Computer Graphics*, 18(3):103 - 108, July 1984.

[2] P. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. *ACM Computer Graphics*, 24(4):309–318, August 1990.

[3] Marc Levoy. Display of surfaces from volume data. *IEEE CG & A*, 8:29 - 37, May 1988.

[4] Ryo Mukai. The design of a graphics system based on the Conflict-Free Multiport Frame Buffer. Master's thesis, The University of Tokyo, 1991.

[5] S. Nishimura, R. Mukai, and T.L. Kunii. A loosely-coupled parallel graphics architecture based on a conflict-free multiport frame buffer. In *Proceedings of the Third Workshop on Future Trends of Distributed Computing Systems*, pages 411–418, Taipei, Taiwan, April 1992.

[6] Yukio Sakagawa. Implementation of the Global Frame Buffer and its applications to volume visualization. Master's thesis, The University of Tokyo, 1993.