An Extended Volume Visualization System for Arbitrary Parallel Projection

R. Bakalash, A. Kaufman, R. Pacheco, H. Pfister

ABSTRACT We present a special architecture for arbitrary parallel projection for visualization of volumetric data. Using a ray-casting technique, parallel memory access, and pipelined processing of rays in a composition tree, we can achieve interactive rendering rates for a 512^3 dataset.

1.1 The Cube Architecture

Cube is a special-purpose computer architecture for volume visualization [1]. The heart of the architecture is a *Cubic Frame Buffer (CFB)*, which is a large (e.g., 128M voxels for a 512^3 CFB) three-dimensional memory of *voxels*. The voxel is a quantum unit of volume, which has a value representing some measurable properties of the real object or phenomenon, such as the color, fluorescent level, material, and translucency ratio.

Cube's processing speed is achieved by handling beams of voxels rather than single voxels. In order to access a full beam of voxels simultaneously, a 3D modular organization of the CFB has been designed [1]. The special 3D skewed organization of the CFB enables conflict-free access to a full beam (axial ray) of n voxels, in any orthographic direction.

The *3D Viewing Processor (VP3)* [2] generates 2D shaded orthographic projections of the CFB images. It casts rays into the CFB in the specified viewing direction, utilizes the CFB parallel memory organization for conflict-free retrieval of a beam and then determines the pixel projection along that beam. It employs a sequence of n processing units which team up to generate the projection along a beam of n voxels in $O(\log n)$ time for a CFB of n^3 voxels. Consequently, the time necessary to generate an orthographic projection of n^2 pixels is only $O(n^2 \log n)$, rather than the conventional $O(n^3)$ time. The VP3 also shades the projected pixels concurrently with the projection stage by employing the depth-gradient *congradient* shading technique [3].

Arbitrary parallel projections are currently created by first rotating the scene and then viewing it through a principal orthographic direction. However, the CFB image is distorted every time a rotation is executed. A major goal of the extended Cube architecture project, presented in this paper, is to develop and prototype an alternative mechanism for parallel viewing that supports real-time arbitrary viewing.

1.2 The Extended Cube Architecture

The new architecture described here is an extension of the existing orthographic projection mechanism of Cube. It enables arbitrary parallel projection in the same time complexity as orthographic projection, $O(n^2 \log n)$. Perspective projection can also be generated in a similar fashion [4], but requires a more complex architecture.

A projection ray, originating at a pixel in the projection plane and cast through the CFB in an arbitrary direction, is the basic unit of projected data. Two processing stages are concerned with this data: first there is a need to retrieve the data from the CFB, and then to obtain the projection along the ray. In the original Cube architecture, where only orthographic viewing is supported, projection rays always coincide with orthographic beams and are fetched and processed by the beam projection mechanism. However, for arbitrary viewing there is no direct way to fetch arbitrary discrete rays from the CFB in parallel. The set of projection rays belonging to the same scan line of the projected 2D frame-buffer form a slanted plane, termed the *Projection Ray Plane (PRP)*. For every parallel projection, all the PRPs can be made parallel to one major axis by fixing a degree of freedom in specifying the projection parameters, namely, by rotating the projection plane about the viewing axis. A whole PRP of beams (now parallel to an axis) is fetched in n memory cycles and stored in a 2D temporary buffer called the *2D Skewed Buffer (2DSB)*.

The direction of the viewing ray within the original PRP depends on the observer's viewing direction. When a PRP is copied from the CFB to the 2DSB, it undergoes a 2D shearing to align all the viewing rays into beams along a direction parallel to a 2D axis (e.g., vertical). This step is a de-skewing step that is accomplished by a barrel shifting mechanism (see below). Once the viewing rays are aligned vertically within the 2D memory, they can be individually fetched and treated by the ray projection mechanism.

This imposes a basic structural condition on the 2D memory. It should be capable of parallel access for storing "horizontal" beams coming from the PRP, and for parallel retrieval of "vertical" viewing rays. The structure chosen for the 2D memory is a 2D Skewed Buffer (2DSB), described below.

The retrieved "vertical" rays must pass through another de-skewing process on their way to the ray projection mechanism in order to match the physical sequential order of the modules in the projection mechanism. The latter is a *Ray Projection Tree (RPT)*, which is a hardware mechanism structured as a hierarchical pipeline, capable of implementing a variety of projection functions (see below).

The communication mechanism that bridges between the CFB and the 2DSB, and between the 2DSB and RPT and performs the de-skewing steps, is a unique beam-based barrel-shifting mechanism, termed the Conveyor [6], and is described below.

Fig. 1.1 illustrates the general architecture of the extended system for arbitrary parallel viewing comprising of the Cubic Frame Buffer (CFB), the 2D Skewed Buffer (2DSB), the Ray Processing Tree (RPT), and two Conveyors for ray de-skewing.

1.3 The 2D Skewed Buffer

The 2DSB used for storing the slanted PRP is divided into n modules and is skewed diagonally such that any module appears exactly once in every row and every column. This

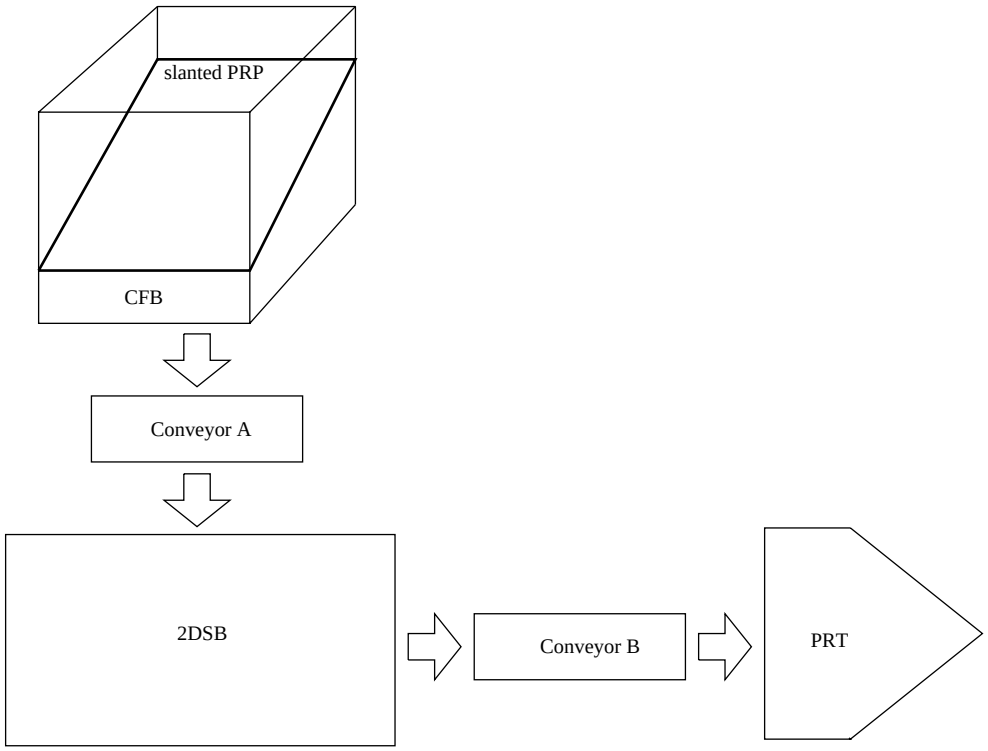

Fig. 1.1. Block diagram of the arbitrary parallel viewing architecture

enables parallel access of any row or column of voxel data without memory bus contention (see Fig. 1.2). the Module k containing the voxel located in column i and row j of the 2DSB is computed as $k = (i + j) \mathrm{mod} n$. The slanted PRP is loaded into the 2DSB one beam at a time, from the closest beam to the furthest. Each beam is shifted to the left or to the right within the 2DSB in order to align the viewing rays vertically. Since there may be $2n$-1 parallel viewing rays entering the slanted PRP (one for each voxel on the visible edges of the slanted PRP), the 2D memory must be at least $2n$-1 columns wide. Once the slanted PRP has been loaded onto the 2DSB one "horizontal" beam at a time, each vertical ray is retrieved in turn, and transferred into the RPT for processing. The rays are shifted as they are transferred into the RPT in order to ensure that the closest voxel in the ray appears at the desired position in the RPT. The voxels in each ray are processed in parallel to compute

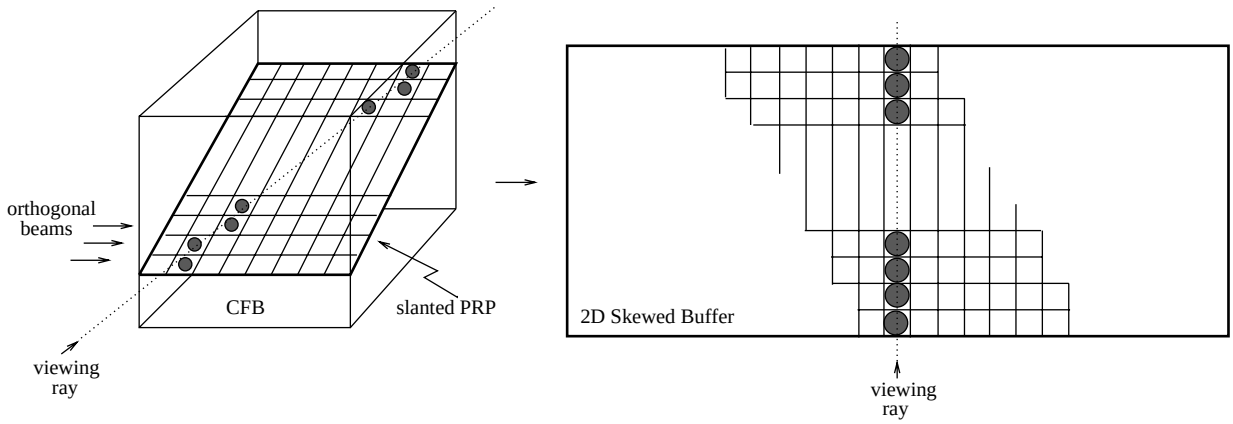

Fig. 1.2. Transfer of the projection ray plane to 2D skewed buffer

a pixel value to be displayed for that ray. Certain algorithms require that the values of the entire neighborhood of up to 26 voxels surrounding a central voxel be used in computing the pixel value. This requires information from slanted PRPs just above and just below the plane containing the ray currently being processed. This is accomplished by using multiple parallel 2DSBs and processing rays in a plane only after the succeeding and preceeding slanted PRPs have been loaded. By using a special skewing scheme, it is also possible to achieve conflict free retrieval of the entire 27 connected neighborhood of voxels [7]. The central voxel and its 26 voxel neighborhood may then be extracted and transferred to the RPT for processing in parallel. Including a fourth plane in the 2DSB allows the concurrent projection of rays and loading of the next slanted PRP. While three of the planes are used for extracting a 27 voxel

cube, the fourth plane is loaded with the next slanted PRP. This conforms to the desired pipelining scheme.

1.4 The Ray Projection Tree (RPT)

The RPT recieves a set of n voxels as input at the leaves which form a viewing ray and produces the final color for the corresponding pixel at the root. The RPT is a hierarchical binary tree of *n-1* primitive computation nodes called *Voxel Combination Units* (*VCU*) (see Fig. 1.3). Each VCU accepts two voxel values as input and combines them into an output voxel value in τ time units. The n input voxels comprising a ray are fed into $n/2$ VCUs that produce $n/2$ results after τ time units. These results are fed into the next stage of the tree (containing $n/4$ VCUs) while at the same time the next ray of n voxels is fed into the first stage. After a short period of initialization time ($\tau \log n$ time units), the tree is processing $\log n$ rays simultaneously in a pipeline fashion, producing a new pixel color every τ time units.

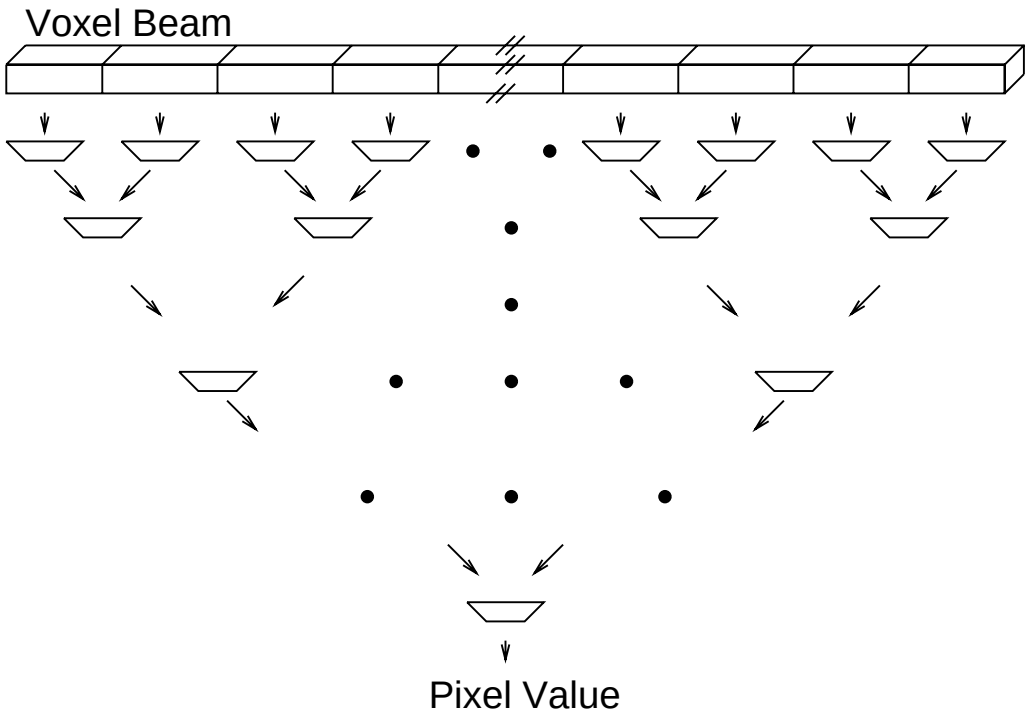

Fig. 1.3. Ray Projection Tree organization

Each VCU is capable of combining its two input voxels in a variety of schemes in order to implement first or last opaque projection, maximum or minimum voxel value, weighted summation, and compositing projection [9].

1.5 The Conveyor

The Conveyor is a modular barrel-shifting mechanism based on a special flow-through network that interconnects the n modules [6]. Its basic component is a modular barrel-switch unit which is serving s pairs of input and output clients receiving from each input, a single bit of data (s bits in total), barrel-shifting it onward, and storing it in the output client. Every two neighbors of the $m = n/s$ switch units are linked by an s-bit communication path. A general width of w bits of data requires w duplicates of the Conveyor. This layout provides modularity and flexibility of the entire network. The network is extendible in its overall length, by serially connecting arbitrary numbers of switch units, and in data width, by stacking up any number of Conveyors, one layer per data bit [5].

A method of combining the two conveyors, Conveyor A and Conveyor B, into one physical conveyor is being used to conserve resources and optimize pipelined operation. We are able to fit the two into one conveyor chip without compromising system parallelization. This is done by interleaving the use of the shift lines and sharing the data I/O lines. The output of each conveyor is connected to the input of the other. This allows data to be loaded into or read from both conveyors at the same time. The double conveyor is also designed to allow the reading and writing of new data without interfering with data already in the shifting mechanism. This is useful for minimizing the time of the pipeline stages by overlapping memory access and shifting times. This feature greatly increases the speed of the the system by contracting the pipeline segments. Cycle interleaving allows the use of multiplexed shift lines to reduce the number of interconnect lines needed for implementation with little effect on the overall shift cycle time.

1.6 Practical Limitations

A system constructed for a cube space of 512^3 requires 512 parallel processing units with 512 memory modules each using a 16 bit data bus and a 19 bit address bus as well as control lines. This amounts to an order of 18,000 lines of information operating in parallel. This is clearly a serious consideration for implementation. Another size consideration is the amount of intermodule communication required by the system. Each conveyor is connected to its neighboring equivalent by 256 lines (16 bits x 16 shift places). A third consideration involves the connection of the conveyors to the modules' data lines. The modules' data lines are grouped by module and the conveyor data lines are grouped by bit position. This presents routing considerations. One more consideration involving physical dimension is the number of chips needed to implement the system. The memory modules alone may require as many as 1,024 chips. Implementation of the system includes the use of custom ASICs as well as programmable gate arrays such as Xilinx [8]. The amount of board space necessary to contain such a system is of serious concern as well as the inter-board communication traffic.

1.7 Acknowledgments

This project has been supported by the National Science Foundation under grant MIP-8805130 and a grant from Hewlett Packard.

1.8 References

[1] Kaufman, A., and Bakalash, R., "Memory and Processing Architecture for 3D Voxel-Based Imagery", *IEEE Computer Graphics and Applications*, 8, 6, November 1988, 10-23.

[2] Kaufman, A., Bakalash, R., and Cohen, D., "Viewing and Rendering Processor for a Volume Visualization System", in *Advances in Graphics Hardware IV*, R. L. Grimsdale and W. Strasser, (eds.), Springer-Verlag, Berlin, 1991, 171-178.

[3] Cohen, D., Kaufman, A., Bakalash, R. and Bergman, S., "Real-Time Discrete Shading", *The Visual Computer*, 6, 1 (February 1990), 16-27.

[4] Kaufman, A. and Shimony, E., "Arbitrary Parallel and Perspective Projection Architecture for Voxel Images", Technical Report 89/21, Computer Science, Stony Brook, 1989.

[5] Bakalash, R. and Xu, Z., "Barrel Shift Microsystem for Parallel Processing", *Proc. Micro 23, 23rd Symposium and Workshop on Microprogramming and Microarchitecture*, Orlando, Florida, November 1990.

[6] Cohen, D. and Bakalash, R., "The Conveyor - an Interconnection Device for Parallel Volumetric Transformations", *6th EG Workshop on Graphics Hardware*, Vienna, Austria, September 1991.

[7] Chor, B., Leiserson, C. E., and Rivest, R. L., "An application of number theory to the organization of raster-graphics memory", *Conf. Rec. 23^{rd} Annual IEEE Symp. Foundations of Computer Science*, Chicago, 1982, 92-99.

[8] Xilinx, Inc., *The Programmable Gate Array Data Book*, 1989.

[9] Levoy, M., "Display of Surfaces from Volume Data", it IEEE Computer Graphics and Applications, 8, 5, May 1988, 29-37.