

Real-Time Architecture for High-Resolution Volume Visualization

Hanspeter Pfister and Arie Kaufman
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400, USA

February 14, 1995

Abstract

This paper describes a high-performance special-purpose system, the Cube-3 machine, for displaying and manipulating high-resolution volumetric datasets in real-time. Cube-3 will allow scientists, engineers, and biomedical researchers to interactively visualize and investigate their static high-resolution sampled, simulated, or computed volumetric dataset. Furthermore, once acquisition devices or mechanisms are capable of acquiring a complete high-resolution dynamic dataset in real-time, Cube-3, tightly coupled with them, will be capable of delivering real-time 4D (spatial-temporal) volume visualization, a task currently not possible with present technologies.

1 Introduction

Visualization of scientific, engineering, or biomedical multidimensional data has become a key technology in computer science and in the related applications. Often the natural or computational objects or phenomena being studied are spatially or temporally volumetric. Unlike traditional computer graphics techniques, which represent 3D objects as geometric surfaces and edges approximated by polygons and lines, volume data are 3D entities that may have information inside them. These volumetric entities might not consist of surfaces and edges at all, or they may be too voluminous to be represented geometrically.

Volume visualization is concerned with the representation, manipulation, and rendering of volumetric data [12]. A volumetric dataset is typically represented as a 3D regular grid of voxels (volume elements), each is a quantum unit of volume that has a value(s) associated with it representing some property of the object or phenomenon. This 3D dataset is commonly stored in a regular Cubic Frame Buffer (CFB), which is a large 3D array of voxels (e.g., 128M voxels for 512^3) and is being displayed on raster screen using a direct volume rendering technique [15], [12]. Alternatively, the dataset may be represented as a sequence of cross-sections or as an irregular grid.

Applications of volume visualization include sampled, simulated, and computed datasets in confocal microscopy, astro- and geophysical measurements and analysis, molecular structures, finite element models, computational fluid dynamics, and 3D reconstructed medical data, to name just a few (see [12] Chapter 7). As with other display methods of 3D objects, the provision of visual cues and dynamic data manipulation is essential for the understanding of the 3D dataset. Real-time volume visualization rates, typically defined to be more than 10 frames per second, make

object manipulation a natural way for the understanding of 3D *static* data. Furthermore, in many *dynamic* applications, real-time visualization is a necessary component of an integrated acquisition-visualization system. Examples are the real-time analysis of an in-vivo specimen under a confocal microscope or the real-time study of in-situ fluid flow or crack formation in rocks under Computed Microtomography (CMT).

The main objective of the proposed Cube-3 architecture is to develop a real-time volume visualization system that will support these type of applications. The availability of such a system will revolutionize the way scientists, engineers, and biomedical personnel conduct their studies.

2 System Overview

Figure 1 depicts the overall organization of two real-time volume visualization environments. The host computer controls the entire environment and runs the Cube software. The acquisition device is either a sampling device such as a confocal microscope, microtomograph, ultrasound, MRI or CT scanner that produces 2D cross sections or 3D reconstructed volume, or a computer running either a simulation model (e.g., computational fluid dynamics) or synthesizing (voxelizing) a voxel-based geometric model from a display list (e.g., CAD) [12]. The sampled, simulated, or computed datasets are either a sequence of cross sections, a regular 3D reconstructed volume, or an irregular sample that can be converted into a regular volume. All these formats can be maintained and archived by the filing module of the Cube software. The Cube-3 machine accepts either a sequence of cross sections or a 3D regular volume.

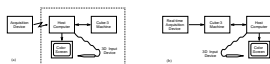


Figure 1: Volume Visualization Environments.

Figure 1 (a) shows an environment in which the acquisition and reconstruction stage will be performed in several seconds to several minutes (depending on the acquisition device), while the visualization and manipulation will be running on the Cube-3 machine in true real-time. Figure 1 (b) shows the ultimate environment, in which Cube-3 is tightly-coupled with the real-time acquisition device to create an integrated acquisition-visualization system that would allow the real-time 4D (spatial-temporal) visualization of dynamic systems.

In addition to controlling the Cube-3 volume visualization engine the host machine also runs the Cube software system, called VolVis [3], which is currently under development at Stony Brook, and complements the Cube-3 machine hardware.

3 Cube-3 Architecture

Figure 2 shows a block diagram of the Cube-3 machine hardware. Cube-3 is a highly-parallel pipelined architecture. The CFB is a 3D memory organized in n memory modules, each with n^2 voxels and its own independent dual-access and addressing unit. A special 3D skewed organization enables the conflict-free access to any beam (i.e., a ray parallel to a main axis) of n voxels (see Section 3.1). In order to generate a parallel or perspective projection, each Projection Ray Plane (PRP) is handled separately, generating a complete scan-line of pixels for each PRP (see Section 3.2).

The PRP is fetched beam after beam conflict free from CFB and stored in the 2D skewed buffer (2DSB), which is organized as a 2D skewed memory. Actually, quadruple 2DSBs are used to support

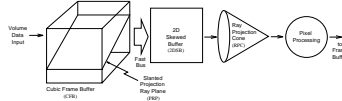


Figure 2: Cube-3 Architecture Overview.

interpolation and compositing, gray-level shading, and pipelining. The Fast Bus allows the fast de-skewing and alignment of a beam from the CFB modules into the 2DSB modules (see Section 3.3). Each projection ray is then fetched conflict-free from the 2DSB and placed at the leaves of the Ray Projection Cone (RPC). The RPC is a folded binary tree that performs interpolation and shading calculations at the leaves and generates in parallel and in a pipelined fashion the associated pixel using a variety of projection schemes on the cone nodes (see Sections 3.4 and 3.5). The resulting pixel is being post-shaded, splatted, and 2D transformed to be stored in the 2D frame-buffer (see Section 3.6).

The next sub-sections describe several aspects of the system in more detail.

3.1 Parallel Cubic Frame Buffer Organization

A special 3D skewed organization of the n^3 voxel CFB enables conflict-free access to any beam of n voxels [13], [11]. The CFB consists of n memory modules, each with n^2 voxels and its own independent access and addressing unit. A voxel with space coordinates $\langle x, y, z \rangle$ is being mapped onto the k -th module by:

$$k = (x + y + z) \bmod n \quad 0 \leq k, x, y, z \leq n - 1 \quad (1)$$

Since two coordinates are always constant along any beam, the third coordinate guarantees that one and only one voxel from the beam resides in any one of the modules. The internal mapping (i, j) within the module is given by: $i = x, j = y$.

When scanning the CFB beam after beam for viewing, the internal order of the modules along the beams is changing. Actually the module index, which is the distance of a voxel along a beam from the viewing position (i.e., the voxel depth), is either incremented or decremented by 1 (modulo n) when moving to the next voxel or beam. Consequently, even the simple arithmetic involved in the memory scheme of Equation 1 is avoided during the bulky viewing process and only a trivial incremental step is used instead. Equation 1 is employed, however, when accessing a single voxel, a single beam, or a segment of it.

This skewing scheme has successfully been employed in Cube-1 [13] and Cube-2 [4], first and second generation prototype architectures built at SUNY Stony Brook. They employ a sequence of n Processing Units (PUs) which team up to generate the first opaque projection along a beam of n voxels in $O(\log n)$ time, using a Voxel Multiple-Write Bus (VMWB) [7], [13]. 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. However, in this technique projections can be generated from only a finite number of predetermined directions and not from any arbitrary direction [5].

3.2 Architecture for Arbitrary Viewing

We propose here a new architecture, as an extension of the Cube-2 orthographic projection mechanism. The new architecture also processes rays instead of voxels, but, unlike Cube-2, the rays are not necessarily parallel to a main axis. Furthermore, the entire system works in a pipeline fashion, and thus a ray is processed in a constant time, enabling arbitrary parallel and perspective projection in a time complexity of $O(n^2)$.

All the rays belonging to the same scan line of the 2D frame-buffer reside on the same plane, termed the Projection Ray Plane (PRP). For every parallel and perspective projection, all the PRPs can be made parallel to one major axis by fixing a degree of freedom in specifying the projection parameters. For example, in parallel projection the projection plane can be rotated about the viewing axis which is performed by the pixel processing unit after projection. Since there is no direct way to fetch arbitrary discrete rays from the CFB conflict free, a whole PRP of beams (which are now parallel to an axis) is instead fetched in n cycles, beam after beam, 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 moved from the CFB to the 2DSB using the Fast Bus, it undergoes a 2D shearing (either to the left or to the right) to align all the viewing rays into beams along a direction parallel to a 2D axis (e.g., vertical). See Figure 3. This step is a de-skewing step that is accomplished by the arbitrary bus routing. Once the viewing rays are aligned vertically within the 2D memory, they can be individually fetched and placed at the leaves of the ray projection mechanism. Since there may be up to $2n - 1$ parallel viewing rays entering the PRP, the 2DSB size is $2n \times n$ voxels.

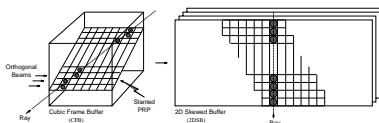


Figure 3: Arbitrary Viewing Mechanism.

The 2DSB thus supports conflict-free storage of *horizontal* beams coming from the CFB and conflict-free retrieval of *vertical* viewing rays. The 2DSB is divided into n modules, each with $2n$ voxels, and is skewed such that any module appears exactly once in every row and every column. To achieve this, the (i, j) voxel is mapped onto module $(i + j) \bmod 2n$ in location i (see also [14] which is a hardware solution for 90° rotation and mirroring of bitmaps).

3.3 Modular Fast Bus

The Fast Bus is an interconnect bus that allows the transfer of data from the n CFB modules to the n 2DSB modules in any arbitrary distribution. This enables the arbitrary shearing necessary for parallel projections and the de-fanning necessary for perspective projections. A set of fast pipelined multiplexers and demultiplexers and a fast multi-channel bus are used to accomplish the data transfer speeds necessary for real-time rendering.

The Fast Bus speeds of 128 nsec per beam/ray will support 25Hz rendering of a 512^3 16-bit per voxel datasets. This speed has been used by other researchers [17], and technologies and driving chip sets already exist for such bus speed requirements, such as the Futurebus+ [2], [20], [16], [18].

3.4 Ray Projection Mechanism

We propose a pipelined hardware mechanism for ray projection called Ray Projection Cone (RPC) that can generate a projection in $O(1)$ time using a rich variety of projection schemes. The cone is a folded (circular) cross-linked binary tree with n leaves which can be dynamically mapped onto a tree with its leftmost leaf at any arbitrary end-node on the cone. This allows the processing of a ray of voxels starting from any leaf of the cone. This in turn allows the cone to be hard-wired to the outputs of the 2DSB modules containing the voxels. Such a configuration eliminates the need for a set of n n -to-1 switching units or a barrel shifter in the connection. The beginning levels of the

cone may also be located at the site of the associated modules. The conic structure requires only an additional set of configurable interconnections and does not require any additional tree nodes over a conventional binary tree.

The cone accepts as input a set of n voxels along the viewing ray and produces the final value for the corresponding pixel. The cone is a hierarchical pipeline of $n - 1$ primitive computation nodes called Voxel Combination Units (VCU). Each VCU accepts two voxel values as input and combines them into an output voxel value in a constant time τ . At any given snapshot the cone is processing $\log n$ rays simultaneously in a pipelined fashion, producing a new pixel color every τ time units.

Each VCU is capable of combining its two input voxels in a variety of ways in order to implement viewing schemes, such as first or last opaque projection, maximum or minimum voxel value, weighted summation, and compositing projection. A VCU accepts as input two voxel values V_L and V_R (left and center or center and right inputs), each one consisting of color C , opacity α , and the depth index I of the voxel along the ray. The opacity of the voxel is either prestored with every voxel or provided through a look-up table of a transfer function at the leaves of the cone. The VCU produces an output voxel V' by performing one of the following operations:

$$\begin{aligned} \text{First opaque:} & \quad \text{if } (\alpha_L \text{ is opaque}) \quad V' = V_L \\ & \quad \text{else} \quad V' = V_R \\ \text{Maximum value:} & \quad \text{if } (C_L < C_R) \quad V' = V_R \\ & \quad \text{else} \quad V' = V_L \\ \text{Weighted sum:} & \quad C' = C_L + W_k C_R \end{aligned}$$

where W is the weighting factor and k is the cone level. W_k is precomputed and preloaded into the VCUs. Weighted sum is useful for depth cueing, bright field, and x-ray projections.

$$\begin{aligned} \text{Compositing} \quad C' &= C_L + (1 - \alpha_L)C_R \\ \alpha' &= \alpha_L + (1 - \alpha_L)\alpha_R \end{aligned}$$

where the first level VCUs compute $C_I = C_I \alpha_I$, assuming the values are gray-levels or RGB. This is the parallel implementation of the front-to-back (or back-to-front) compositing [15].

Similarly last opaque, minimum value projection, and sum (x-ray) projection can be implemented. The cone can support a richer set of projection paradigms and a larger variety of shading techniques, and by using a pipelined organization, it has the potential of supporting higher projection speeds.

The VCUs will be implemented in VLSI where three VCUs will be incorporated into one chip. A single set of three VCUs, representing a three node subcone, can be arranged in a planar organization such as to maximize space and wiring efficiency [1]. The leaves of the ray projection cone are more complex nodes, supporting trilinear interpolation and gray-level gradient computation and segmentation based on transfer functions [15].

3.5 Fast 3-D Interpolation

When sampling in non-grid locations along the ray for compositing [15], the sampled value is trilinearly interpolated from the values of the eight voxels (called a cube) around the sample point. Note that this kind of sampling does not necessarily require a regular isotropic dataset and slice data can be accommodated as well.

A 3D interpolation unit computes the interpolated data values of the samples on the projection ray when it traverses through the volume data. Suppose the relative 3D coordinate of a sample point within a cube with respect to the corner voxel closest to the origin is $\langle a, b, c \rangle$ and the data values associated with the corner voxels of the cube are P_{ijk} , where $i, j, k = 0$ or 1 , then the

interpolated data value associated with the sample point, P_{abc} , is computed through a tri-linear interpolation process as follows:

$$\begin{aligned}
P_{abc} = & P_{000} \times (1 - a)(1 - b)(1 - c) + P_{100} \times a(1 - b)(1 - c) + \\
& P_{010} \times (1 - a)b(1 - c) + P_{001} \times (1 - a)(1 - b)c + \\
& P_{101} \times a(1 - b)c + P_{011} \times (1 - a)bc + \\
& P_{111} \times abc + P_{110} \times ab(1 - c)
\end{aligned} \tag{2}$$

A brute-force implementation of this formula requires about 13 multiplications and 20 additions for *each* sampled point that is not a voxel. We solve this problem by making the observation that a tri-linear interpolation is actually equivalent to a linear interpolation followed by two bi-linear interpolations, and by replacing time-consuming arithmetic operations with a table look-up.

From Equation 2, it is clear that the only part that allows pre-computation is the intermediate values involving a , b , and c . By substituting two bi-linear interpolations and a linear interpolation for a tri-linear interpolation, the look-up table size shrinks to 64 KBytes. The price we pay for this design decision is that two more multiplications are needed than the straightforward tri-linear interpolation design. Fortunately, the performance overhead associated with these additional multiplications can be minimized by exploiting parallelism and pipelining.

To a first approximation, a parallel multiplier is nothing more than a two-dimensional array of single-bit *carry-save adders*. Therefore, it is possible to integrate a multiplication and an addition operation by inserting an extra row of carry-save adders. Moreover, one can pipeline multiple multiply-add operations through such an augmented parallel multiplier to reduce the hardware cost. Consequently, it becomes feasible to implement the entire 3D-interpolation function in one chip and because of the highly pipelined structure the targeted 128 nsec cycle time is comfortably within the reach of this design.

3.6 Volumetric Shading Mechanisms

A prominent object-based volumetric shading method is *gray-level gradient* shading [9]. It uses the gradient of the data values (gray-level values) as a measure for surface inclination. More formally, denote by V the gray-level function. The normal is obtained from the gradient vector by $\left\langle \frac{\delta V}{\delta x}, \frac{\delta V}{\delta y}, \frac{\delta V}{\delta z} \right\rangle$. The partial derivatives are approximated (in voxel space) by the differences between the gray values of the current voxel and its immediate neighbors.

Cube-3 has four 2DSB memories that store three active consecutive RPRs, the current RPR and one just above and one just below, as well as an inactive RPR which is concurrently being loaded from the CFB. The central voxel and up to 26 neighboring voxels (9 from each PRP) needed for the gray-level calculation, or the 8 voxels surrounding the current sample, may then be extracted and transferred in parallel for processing in the interpolation units.

We have developed [6] a hardware design for the implementation of an image-based post-shading method termed *congradient* shading. In this method the surface normal is obtained from the depth gradient vector where the partial derivatives are approximated (in pixel space) by the differences between the depth values of the current pixel and its immediate neighbors in the depth buffer [8]. The post-shading is performed as part of the pixel processing unit.

An alternative image-based post-shading method is the *context-sensitive* shading method [22], [23], [21], which is based on a mechanism for discontinuity detection, which divides the image space into contexts each of which is a surface segment that exhibits a high level of uniformity, that is, it is a continuous surface (C^0 continuity) with a gradually changing tangent (C^1 continuity). The

hardware implementation of the context-sensitive shading [21] is slightly more complex than that of the congruient shading.

We have developed and analyzed several volumetric shading techniques in an attempt to address the problem of image quality in volume rendering. Our initial observation, which is shared by others [10], is that more than one technique is necessary for the same image component, for different components, for different images, or different applications, and that blending of the colors calculated by several methods at each pixel results in superior quality images [21], [19]. Consequently, Cube-3 supports a large variety of shading techniques and schemes.

3.7 Parallel Input

Emerging real-time scanning devices, such as computed microtomograph, confocal microscopes, or ultrasound scanners, will be able to acquire data at very high rates. Similarly, computers and especially supercomputers can generate in real-time large volumetric datasets through simulation or modeling. In order to transfer the huge amounts of data from the acquisition device to the CFB in real-time, an efficient, reliable and fast input system will be developed and implemented. We assume that the acquisition (and reconstruction) device will supply the data in parallel channels in the skewed order of the CFB.

To achieve high input performance of multiple 512^3 datasets per second we propose the use of parallel high speed optical links to connect the acquisition device and Cube-3. Providing more than gigabit per second transfer bandwidth, these optical links provide a reliable and fast interface to Cube-3. We plan to use the Gazelle optical interfaces, currently supporting 1 gigabit per second transfer rate. These interfaces are in use in our department for the implementation of a high performance eager-sharing system. Assuming, for example, 16 dual-ported CFB modules per CFB-board for a total of 32 CFB-boards and one optical link per board, it would be possible to load 8 MBytes on each board at least 25 times per second. This assumes 32 corresponding output ports on the acquisition device.

4 Summary

The Cube-3 machine is a full-scale, high-resolution 512^3 16-bit per voxel hardware prototype of a highly-parallel, pipelined, real-time volume visualization engine. It has the following capabilities: viewing from any perspective and parallel direction, specifying the projection type (e.g., first opaque, max value, x-ray, compositing), shading, sectioning and slicing, programmable color segmentation and thresholding, and controlling translucency, which are supported by the hardware and can be performed in real-time.

In addition, the following functions run in software, not necessarily in real-time: data acquisition, 3D reconstruction, voxelizing synthetic models, filing and database handling, loading the Cube-3 machine, transforming (translation, scaling, and rotation), manipulating (voxel-by-voxel operations, e.g., 3D filtering) the dataset, and probing and measurements (e.g., distance, surface, volume), and 3D interaction for direct and natural interaction and navigation.

We have simulated the Cube-3 architecture in C and in Verilog, and have designed the general layout of the 512^3 16-bit voxel prototype implementation. Current design calls for a system with 49 boards, 16 CFB-boards with about 100 chips per board (for 32 3D memory modules per board), 32 2D-boards with about 115 chips per board (for 16 2D memory modules per board), and one master board. This board layout and chip count may change depending on off-the-shelf chip availability, pin count and package size for the custom-designed chips, bus interface technology, and real estate on the boards.

5 Acknowledgements

This work was partially supported by the National Science Foundation under grant MIP 88-05130. The authors would like to thank all the members of the Cube-3 team that contributed to this research, especially Reuven Bakalash, Robert Pacheco, and Adam Xuejun. Special thanks are due to Tzi-cker Chiueh for proposing the fast interpolation mechanism.

References

- [1] G.S. Almasi and A. Gottlieb. *Highly Parallel Computing*. Benjamin/Cummings Publishing Company, 1989.
- [2] W. Andrews. 32-bit buses contend for designer's attention. *Computer Design*, 28(11):78–96, November 1989.
- [3] R. Avila, L. Sobierajski, and A. Kaufman. Towards a comprehensive volume visualization system. In *Visualization '92 Proceedings*, pages 13–20. IEEE Computer Society Press, October 1992.
- [4] R. Bakalash, A. Kaufman, R. Pacheco, and H. Pfister. An extended volume visualization system for arbitrary parallel projection. In *Advances in Computer Graphics Hardware, V, Proceedings of the 1992 Eurographics Workshop on Graphics Hardware*, 1992.
- [5] D. Cohen and A. Kaufman. A 3D skewed memory organization for conflict free ray casting. Tr 92.09.11, Department of Computer Science, State Univeristy of New York at Stony Brook, September 1992.
- [6] D. Cohen, A. Kaufman, R. Bakalash, and S. Bergman. Real-time discrete shading. *The Visual Computer*, 6(1):16–27, February 1990.
- [7] R. Gemballa and R. Lindner. The multiple-write bus technique. *IEEE Computer Graphics & Applications*, 2(7):33–41, September 1982.
- [8] D. Gordon and R. A. Reynolds. Image space shading of 3-dimensional objects. *Computer Vision, Graphics, and Image Processing*, 29:361–376, 1985.
- [9] K. H. Hoehne and R. Bernstein. Shading 3D-images from CT using gray-level gradients. *IEEE Transactions on Medical Imaging*, MI-5(1):45–47, March 1986.
- [10] K. H. Hoehne, M. Bomans, A. Pommert, M. Riemer, C. Schiers, U. Tiede, and G. Wiebecke. 3D-visualization of tomographic volume data using the generalized voxel model. *The Visual Computer*, 6(1):28–37, February 1990.
- [11] A. Kaufman. The CUBE 3D workstation. *EG Workshop on Graphics Hardware*, August 1986.
- [12] A. Kaufman. *Volume Visualization*. IEEE Computer Society Press Tutorial, Los Alamitos, CA, 1990.
- [13] A. Kaufman and R. Bakalash. Memory and processing architecture for 3D voxel-based imagery. *IEEE Computer Graphics & Applications*, 8(6):10–23, November 1988.
- [14] C. Korenfeld. The Image Prism: A device for rotating and mirroring bitmap images. *IEEE Computer Graphics & Applications*, 7(5):21–30, May 1987.

- [15] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.
- [16] J. Martinez. BTL transceivers enable high-speed bus designs. *EDN*, 37:107, August 1992.
- [17] S. Molnar, J. Eyles, and J. Poulton. Pixelflow: High-speed rendering using image composition. *Computer Graphics*, 26(2):231–240, July 1992.
- [18] National Semiconductor. Transceiver chip set gets BTL treatment. *Electronic Engineering*, 63(4):11–12, April 1991.
- [19] I. Spector, A. Kaufman, R. Yagel, and R. Bakalash. 3D visualization of actin cytoskeleton. *Annual Meeting of the American Society of Cell Biology*, December 1990.
- [20] D.M. Taub. Clockless synchronization of distributed concurrent processes. In *IEEE Proceedings*, volume 139, pages 88–91, January 1992.
- [21] R. Yagel, D. Cohen, and A. Kaufman. Context sensitive normal estimation for volume imaging. Tr 90.05.15, Department of Computer Science, State University of New York at Stony Brook, May 1990.
- [22] R. Yagel, D. Cohen, and A. Kaufman. Context sensitive normal estimation for volume imaging. In N.M. Patrikalakis, editor, *Scientific Visualization of Physical Phenomena*, pages 211–234. Springer-Verlag, 1991.
- [23] R. Yagel, D. Cohen, and A. Kaufman. Normal estimation in 3D discrete space. *The Visual Computer*, pages 278–291, June 1992.