



DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD

Cube-3: A Real-Time Architecture for High-Resolution Volume Visualization

The Harvard community has made this article openly available.
[Please share](#) how this access benefits you. Your story matters.

Citation	Pfister, Hanspeter, Arie Kaufman, and Tzi-Cker Chiueh. 1994. Cube-3: A real-time architecture for high-resolution volume visualization. In Proceedings of the 1994 symposium on Volume visualization: October 17-18, 1994, Washington, D.C., ed. A. Kaufman and W. Kruege, 75-82. New York, N.Y.: Association for Computing Machinery.
Published Version	doi:10.1145/197938.197969
Accessed	February 18, 2015 4:08:16 PM EST
Citable Link	http://nrs.harvard.edu/urn-3:HUL.InstRepos:4238997
Terms of Use	This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

(Article begins on next page)

Cube-3: A Real-Time Architecture for High-Resolution Volume Visualization

Hanspeter Pfister, Arie Kaufman, and Tzi-cker Chiueh
Department of Computer Science
State University of New York at Stony Brook

Abstract

This paper describes a high-performance special-purpose system, Cube-3, for displaying and manipulating high-resolution volumetric datasets in real-time. A primary goal of Cube-3 is to render 512^3 , 16-bit per voxel, datasets at about 30 frames per second. Cube-3 implements a ray-casting algorithm in a highly-parallel and pipelined architecture, using a 3D skewed volume memory, a modular fast bus, 2D skewed buffers, 3D interpolation and shading units, and a ray projection cone. Cube-3 will allow users to interactively visualize and investigate in real-time static (3D) and dynamic (4D) high-resolution volumetric datasets.

1. Introduction

A volumetric dataset is typically represented as a 3D regular grid of voxels (volume elements) representing some uniform or piecewise property of an 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 a 512^3 dataset) and is displayed on a raster screen using a direct volume rendering technique (see e.g., [14, 9]). 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 modeled datasets in confocal microscopy, astro- and geophysical measurements, molecular structures, finite element models, computational fluid dynamics, and 3D reconstructed medical data, to name just a few (see [9] Chapter 7). As with other display methods of 3D objects, the provision of real-time data manipulation rates, typically defined to be more than 10 and preferably 30 frames per second, is essential for the investigation of 3D *static* datasets. Furthermore, in many *dynamic* applications, 4D (spatial-temporal) 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).

High-bandwidth memory access and high arithmetic performance are key elements of real-time volume rendering and can be met by exploiting parallelism among a set of dedicated processors [5, 16, 15, 10] [9, Chapter 6]. Sub-second rendering times for a 128^3 dataset have been reported by Schröder and Stoll on a Princeton Engine of 1024 processors [18] and by Vézina et al. on a 16k-PE MasPar MP-1 computer [22]. For higher resolution datasets, however, the number of processors and their interconnection bandwidths must increase, placing hard-to-meet requirements on existing general-purpose workstations or supercomputers.

The main objective of the Cube-3 architecture is to develop a special-purpose real-time volume visualization system for high-resolution 512^3 16-bit per voxel datasets that achieves frame rates over 20 Hz. This rendering performance is orders of magnitude higher than that of previously reported systems, while requiring only moderate hardware complexity. Furthermore, Cube-3 allows interactive control over a wide variety of rendering and segmentation parameters. The availability of such a system will revolutionize the way scientists and engineers conduct their studies.

2. System Overview

Figure 1 shows the overall organization of two real-time volume visualization environments. The host computer controls the entire environment and runs non time-critical parts of the Cube-3 software. It also contains a frame buffer for the final image display on a color monitor. The acquisition device is a sampling device such as a confocal microscope, microtomograph, ultrasound, MRI, or CT scanner. Alternatively, the acquisition device is 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). The sampled, simulated, or modeled dataset is either a sequence of cross sections, a regular 3D reconstructed volume, or an irregular data that has been converted into a regular volume. All these

Authors' Address: Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400
email pfister@cs.sunysb.edu, ari@cs.sunysb.edu, chiueh@cs.sunysb.edu

formats can be maintained and archived by the filing module of the host software.

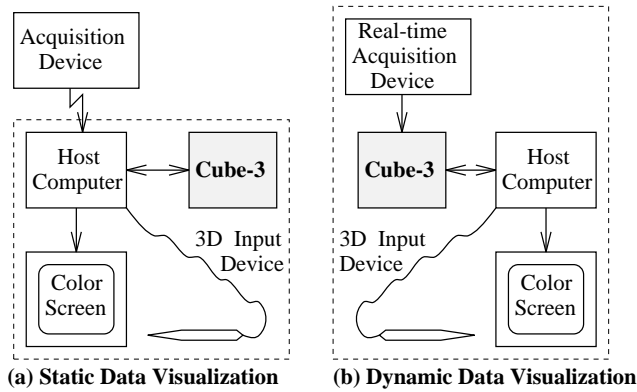


Figure 1: Real-Time Volume Visualization Systems.

Figure 1 (a) shows an environment in which the acquisition and reconstruction are performed in several seconds to several minutes on the acquisition device, while the visualization and manipulation are running on Cube-3 in real-time. Such a system fulfills the needs for *static visualization*, where the nature of the acquired data is static and Cube-3 allows for interactive control of viewing parameters, clipping planes, shading parameters, and data segmentation. 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. The need for this *dynamic visualization* clearly will arise, since the data rates of emerging acquisition devices such as microtomographs will reach several complete 3D high-resolution datasets per second well before the end of the decade. In addition to controlling the Cube-3 volume visualization engine on a device driver level, the host also runs a volume visualization software system and user interface called VolVis [2], which has been developed at SUNY Stony Brook and complements the Cube-3 hardware.

The next section describes aspects of the Cube-3 system in detail. Sections 4 and 5 provide estimated performance and hardware real-estate.

3. Cube-3 Architecture

Cube-3 implements a ray-casting algorithm, a flexible and frequently used technique for direct volume rendering [14]. Figure 2 shows a block diagram of the overall dataflow. In order to meet the high performance requirements of real-time ray-casting, the Cube-3 architecture is highly-parallel and pipelined. The *Cubic Frame Buffer (CFB)* is a 3D memory organized in n dual-access memory modules, each storing n^2 voxels. 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). All the rays belonging to the same scan line of a parallel or perspective projection reside on a slanted plane inside the CFB, termed the *Projection Ray Plane (PRP)*. In order to support arbitrary viewing, each PRP is fetched as a sequence of voxel beams and stored in consecutive *2D Skewed Buffers (2DSB)* (see Section 3.2). A high-bandwidth interconnection network, the *Fast Bus*, allows the alignment of the discrete rays on the PRP parallel to a main axis in the 2DSB modules (see Section 3.3).

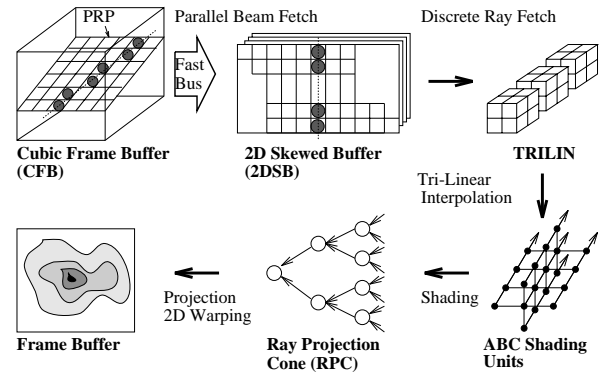


Figure 2: Cube-3 System Overview.

Several 2D Skewed Buffers are used in a pipelined fashion to support tri-linear interpolation and gray-level shading. Aligned discrete rays from 2DSBs are fetched conflict-free and placed into special purpose *Tri-Linear Interpolation (TRILIN)* units (see Section 3.4). The resulting continuous projection rays are placed onto *ABC Shading Units*, where each ray sample is converted into both an intensity and an associated opacity value according to lighting and data segmentation parameters (see Section 3.5). These intensity/opacity ray samples are fetched into the leaves of the *Ray Projection Cone (RPC)*. The RPC is a folded binary tree that generates in parallel and in a pipelined fashion the final ray-pixel value using a variety of projection schemes on the cone nodes (see Section 3.6). The resulting pixel is post-processed (e.g., post-shaded or splatted), 2D transformed, and stored in the 2D frame-buffer.

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 [10]. 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 (x, y, z) is being mapped onto the k -th module by:

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

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: $i = x, j = y$.

This skewing scheme has successfully been employed in Cube-1 [10] and Cube-2 [3], first and second generation 16^3 prototype architectures built at SUNY Stony Brook. They use a sequence of n processing units, 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 [6, 10]. 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 only from a finite number of predetermined directions [4]. The next section describes the enhanced architecture used in Cube-3 for arbitrary parallel and perspective viewing.

3.2. Architecture for Arbitrary Viewing

All the rays belonging to the same scan line of the 2D frame-buffer reside on the same plane, the PRP (see Figure 3). 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 can be reversed after projection in the 2D frame-buffer. 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, the 2DSB.

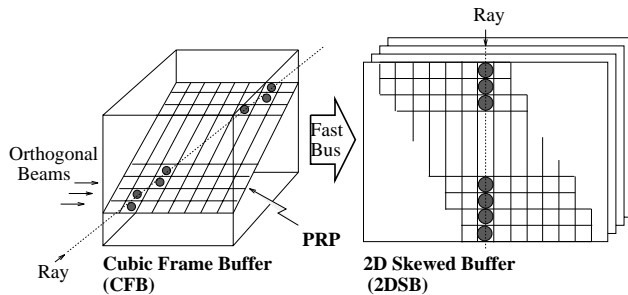


Figure 3: Arbitrary Viewing Mechanism.

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, 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). This step is a de-skewing step that is accomplished by the high-bandwidth Fast Bus that interconnects the CFB and 2DSB memory modules. Once the discrete viewing rays are aligned vertically within the 2D memory, they can be individually fetched and placed at the leaves of the ray projection mechanism for interpolation and shad-

ing. Since there may be up to $2n - 1$ parallel viewing rays entering the PRP, the 2DSB size is $2n \times n$ voxels.

The 2DSB thus supports conflict-free storage of *horizontal* beams coming from the CFB and conflict-free retrieval of *vertical* discrete rays. The 2DSB is divided into n memory modules, each storing $2n$ 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 $(i + j) \bmod n$ in location $i + j$ (see also [13], which is a hardware solution for 90° rotation and mirroring of bitmaps).

As an example, assume that a parallel projection is performed without loss of generality approximately from the $+z$ direction; namely, for each projection ray, z grows faster than x and y in absolute value. The 26-connected discrete ray representation is pre-generated on the host computer using a 3D variation [11] [9, pp. 280–301] of Bresenham's algorithm modified for non-integer endpoints [7]. First, the representation of the projection of the ray along the fetch axis is generated. This representation determines which beams to fetch from the CFB for each PRP. These viewing parameters are broadcast to the CFB addressing units. Second, the ray parameters within the PRP are calculated, determining how much to shear each beam. For parallel projections all rays have the same slope and thus the generalized Bresenham's steps in both directions have to be calculated only once (cf. [23]). These pre-calculated slope templates are down-loaded from the host into the Fast Bus control units (see Section 3.3). Each PRP contributes to all pixels of one scanline in the final image up to a 2D transformation.

A perspective projection can be generated in a way similar to the parallel projection. However, for perspective projections several beams are averaged into a single beam that is stored in the 2DSB. The number of beams averaged depends on the divergence of the perspective rays. Voxels in a given beam are not only averaged but also scaled and sheared between the Fast Bus and 2DSB. This is equivalent to casting a fan of rays, where larger portions of the volume are sampled as the fan diverges. Note that the assumption that for each projection ray z grows faster than x and y in absolute value is not always true in perspective projection, which may require the separate processing of z -major, x -major, and y -major rays.

3.3. Modular Fast Bus

The Fast Bus is an interconnection network that allows the high-bandwidth transfer of data from the n CFB modules to the n 2DSB modules. It enables the arbitrary shearing and averaging necessary for parallel and perspective projections. A set of fast multiplexers and transceivers with associated control units and a multi-channel bus are used to accomplish the data transfer speeds necessary for real-time rendering.

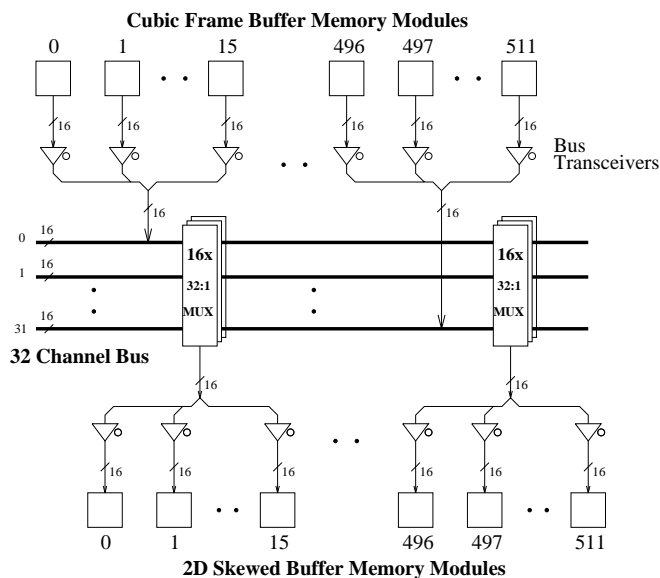


Figure 4: Fast Bus Configuration for $n = 512$.

Figure 4 shows the Fast Bus configuration with $n = 512$ CFB memory modules and 32 bus channels. The CFB modules are first divided into 32 groups of 16 memory modules each. The data from the 16 modules of each group are time-multiplexed onto the designated 16-bit Fast Bus channel for that group. The data now appears on the Fast Bus as illustrated in Table 1. The multiplexing is achieved with the bus transceivers associated with each memory module.

Channel	Time Slice				
	00	01	...	14	15
00	000	001	...	014	015
01	016	017	...	030	031
⋮	⋮	⋮	⋮	⋮	⋮
30	480	481	...	494	495
31	496	497	...	510	511

Table 1: Memory Module Data Time-Multiplexed on the Fast Bus.

The 2DSB modules are likewise divided into 32 groups of 16 modules each. For each group of 2DSB modules, data from the 32 channels of the Fast Bus are multiplexed into the group of 16 2DSB modules. The multiplexers are placed on the backplane of the Fast Bus, and de-multiplexing is implemented with the associated bus transceivers. Hardware necessary for the averaging operation between beams for perspective projections can be incorporated between the bus receivers and the 2DSB modules.

Operation of the multiplexers/transceivers is controlled by lookup tables, called bus channel maps. The maps are pre-computed for each projection and downloaded from the host. A change of viewing parameters requires re-computation of these maps, but their limited

size allows for real-time update rates. A requirement of the system is that the data intended to reach the 16 2DSB modules of a channel group are not transmitted during the same time step. Note that this is trivially satisfied, as all voxels of the beam are kept in a sequence and are moving the same amount either left or right.

We investigated the use of alternative structures such as multistage cube/shuffle-exchange networks [20] or expanded delta networks with packet routing [1]. Although these networks typically require less hardware, their routing overhead severely limits their performance. Furthermore, the Fast Bus requires only readily available, off-the-shelf hardware components.

3.4. Fast 3D Interpolation

When sampling in non-grid locations along the ray for compositing [14], the sampled value is tri-linearly 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. In Cube-3 this interpolation is performed at the leaves of the Ray Projection Cone with data from the 2DSB.

Instead of fetching the eight-neighborhood of each resampling location, four discrete rays are fetched from the 2DSBs, two from each of two consecutive buffer planes. The four rays are subdivided into voxel cubes and fed into the tri-linear interpolation units. Because of the skewing of the 2DSB, neighboring rays reside in adjacent memory modules, requiring a local shift operation of voxels between neighboring units. The pipelined operation of the tri-linear interpolation accounts for this additional latency.

Due to the discrete line-stepping algorithm and the hardware organization into n memory modules, exactly n voxels per discrete ray are fetched, independent of the viewing direction. Since the maximum viewing angle difference along a major axis is not more than 45 degrees up to a 2D rotation, the volume sample rate is defined by the diagonal through the cube and is by a factor of $\sqrt{3}$ higher for orthographic viewing. For ray-compositing, this is of no consideration due to the averaging nature of the compositing operator. High-quality surface rendering, however, requires the adaptation of the stepping distance along the continuous ray according to the view direction.

TRILIN, the 3D interpolation unit, computes the interpolated data values of the samples on the projection ray as 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 (a, b, c) 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:

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

A brute-force implementation of this formula requires 13 multiplications and 20 additions for *each* sample 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 (see Figure 5).

From the above equation it is clear that the only part that allows pre-computation is the intermediate values involving a , b , and c . With a 16-bit data path and $n = 512$, the number of bits left for fractionals, i.e., relative coordinates within a cube, is seven. With a seven-bit resolution, the number of possible combinations for $\langle a b c \rangle$ triples becomes $2^7 2^7 2^7 = 2^{21}$. For each triple, there are eight intermediate products, each being 8-bit wide. Thus, the total size of the look-up table of partial products would be 16 MBytes. Such a look-up table is needed for the simultaneous computation of each interpolated data value. Therefore, it cannot be shared and needs to be replicated n times. Simply because of the required memory size, this design is clearly too expensive and potentially slow.

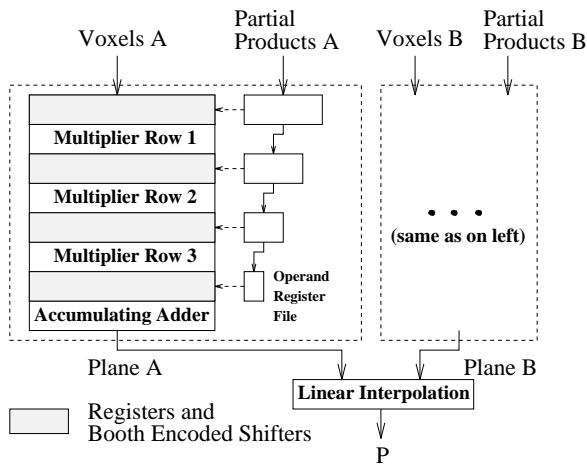


Figure 5: TRILIN: Tri-Linear Interpolation Unit.

By substituting two bi-linear interpolations followed by a linear interpolation for a tri-linear interpolation, the look-up table size is only 64 KBytes. The price we pay for this design decision is that two more multiplications are needed than in the above equation. Fortunately, the performance overhead associated with these additional multiplications can be minimized by exploiting parallelism and pipelining.

The second key idea in the fast 3D interpolation unit design is to exploit the internal structure of a parallel multiplier. To a first approximation, a parallel multiplier is actually a 2D 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, to reduce the hardware cost, one can pipeline multiple multiply-add operations through such an augmented parallel multiplier. Consequently, it becomes feasible to implement the entire 3D-interpolation function in one chip.

3.5. Volumetric Shading Mechanisms

A prominent object-based volumetric shading method is gray-level gradient shading [8]. It uses the gradient of the data values as a measure of surface inclination. The gradient is approximated by the differences between the values of the current sample and its immediate neighbors.

In Cube-3 we use the tri-linearly interpolated values of neighboring rays to evaluate the gradient field inside the dataset. In order to evaluate the gradient at a particular point, we need the rays on the immediate left, right, above and below, as well as the values in the current ray. The left and right point sample values can be fetched from neighboring shading units, and the above and below samples arrive from the consecutive processing of PRPs. Since we need to store complete rays, we call the shading units *ABC Shaders* for their above, below, and current ray sample buffers.

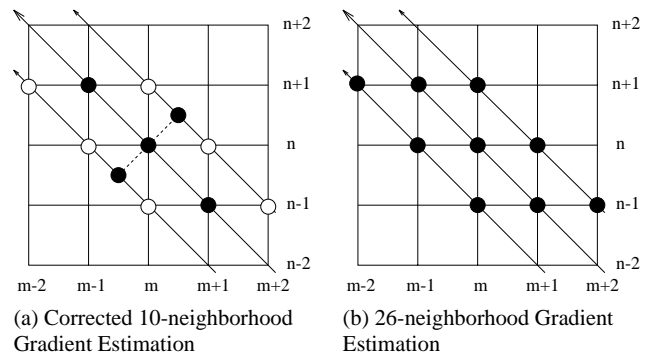


Figure 6: Gradient Estimation Schemes.

Figure 6 illustrates the different gradient estimation schemes (using a 2D drawing). The simplest approach is the 6-neighborhood gradient, which uses the differences of neighboring sample values along the ray, $P_{(n,m+1)} - P_{(n,m-1)}$ in x and $P_{(n+1,m+1)} - P_{(n-1,m-1)}$ in the ray direction (Figure 6 (a)). Although the left, right, above and below ray samples are in the same plane and orthogonal to each other, the samples in the ray direction are not. More importantly, when a change in the viewing direction causes a change in the major axis from m to n , the values of $P_{(n+1,m)} - P_{(n-1,m)}$ are

used to calculate the gradient in the x direction. This leads to noticeable motion aliasing.

To circumvent this problem we use an additional linear interpolation step to resample the rays on correct orthogonal positions (Figure 6 (a), black samples). We call this approach the *10-neighborhood gradient estimation*, and it adequately solves the problem of switching the major axis during object rotations. The linear interpolation weights are constant along a ray and correspond to a constant shift of the complete ray in the viewing direction.

We also simulated the use of a 26-neighborhood gradient (Figure 6 (b)). Instead of fetching sample values from four neighboring rays, 26 interpolated samples from 8 neighboring rays are fetched and the gradient is estimated by taking weighted sums inside and differences between adjacent planes. This method leads to better overall image quality, but the switching of major axis is still noticeable, although less than with the 6-neighborhood gradient.

In the case of perspective projections, the front of each PRP is uniformly sampled with n rays one unit apart. As the rays diverge towards the back of the volume, the distance between rays increases, and the averaged values described above are used instead.

With the gradient estimation and a light vector lookup table, the sample intensity is generated using a variety of shading methods (e.g., using an integrated Phong Shader [12]). Opacity values for compositing are generated using a transfer function represented as a 2D lookup table indexed by sample density.

3.6. Ray Projection Mechanism

The pipelined hardware mechanism for ray projection is the RPC, which can generate one projected pixel value per clock cycle 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 (see Figure 7). 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 for de-skewing of the 2DSB. The leaves of the cone contain the TRILIN interpolation and the ABC Shading units.

The cone accepts as input a set of n samples 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 consecutive intensity/opacity pairs as input and combines them into an output value. At any given snapshot the cone is processing $\log n$ rays simultaneously in a pipelined fashion,

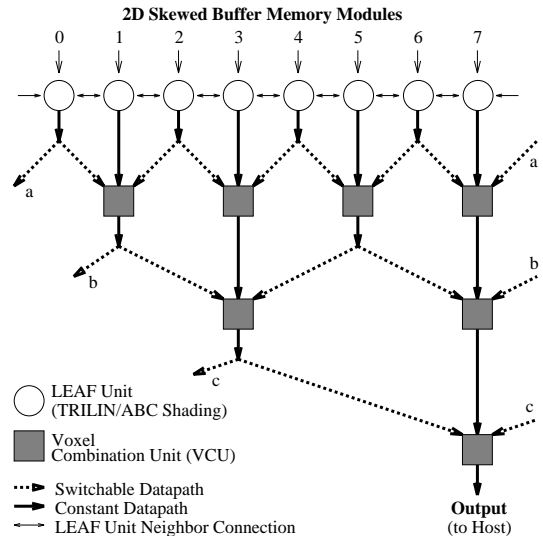


Figure 7: Folded Binary Cone Tree ($n = 8$ Leaf Nodes).

producing a new pixel color every clock cycle. Sectioning and clipping can be implemented on the RPC by discarding voxels according to user specified clip-planes.

Each VCU is capable of combining its two input samples 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 α -compositing. Accordingly, each VCU selects as input the left and center or center and right datapaths, each one consisting of color C and opacity α of the current ray sample.

The opacity of the voxel is either pre-stored with every voxel or provided through a look-up table of a transfer function inside the ABC Shading Unit at the leaves of the cone. The VCU produces an output voxel V' by performing one of the following operations:

First opaque:	if (α_L is opaque)	$V' = V_L$
	else	$V' = V_R$
Maximum value:	if ($C_L < C_R$)	$V' = V_R$
	else	$V' = V_L$
Weighted sum:	$C' = C_L + W_k C_R$	

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

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

where the first level VCUs compute $C_I = C_I \alpha_I$, assuming the values are gray-levels or RGB. This is actually a parallel implementation of the front-to-back (or back-to-front) compositing. The pixel output is transmitted,

for example, to the host, where post-processing, such as post-shading, splatting, and 2D transformation or warping, is performed. A frame buffer outputs the final image to a color display.

4. Performance Estimation

The parallel conflict-free memory architecture of Cube-3 reduces the memory access bottleneck from $O(n^3)$ per projection to $O(n^2)$ and allows for very high data throughput. Due to the highly pipelined architecture, the frame rate is limited only by the data-transfer rate on the Fast Bus. If we assume a dataset size of n^3 , one resample location per volume element, and a final screen resolution of n^2 pixels, we need to transfer a discrete ray of n voxels on the Fast Bus in $\frac{1}{n^2 f}$ seconds. f is the frame rate in updates per seconds. Since the Fast Bus operates in a time-multiplexed fashion with m time-slices, we need $\frac{1}{n^2 f m}$ seconds per transfer or a clock speed on the bus of $n^2 f m$.

Dataset $n \times n \times n$	Frame Rate f	Bus Frequency
$128 \times 128 \times 128$	30 Hz	8 MHz
$256 \times 256 \times 256$	30 Hz	33 MHz
$512 \times 512 \times 512$	15 Hz	66 MHz
$512 \times 512 \times 512$	30 Hz	125 MHz

Table 2: Fast-Bus Performance Requirements ($m = 16$).

Table 2 gives some examples of required bus performance for a multiplexing rate of $m = 16$. High-bandwidth buses have been used by other researchers [16], and technologies and driving chip sets are readily available for most bus speed requirements [19, 21]. We believe that a high-resolution compositing projection of a 512^3 dataset can be generated in Cube-3 with about 30 frames per second.

5. Hardware Estimation

Figure 8 shows the overall hardware structure of Cube-3. It is a modular design that is scalable to higher resolution datasets. The CFB boards contain several CFB modules, each consisting of a memory module, an addressing and bus control unit, and a bus transceiver. Using off-the-shelf components such as SIMMs (Single Inline Memory Modules) and FPGAs (Field Programmable Gate Arrays), it is possible to fit up to 128 CFB modules together with I/O hardware and I/O bus access logic on a single board. The CFB modules on each board can be connected to the acquisition device by high-speed parallel input channels.

Each 2DSB consists of a Fast Bus transceiver, a memory module, and a special purpose LEAF chip. This chip contains the addressing and bus control units, the TRILIN interpolator, and the ABC shading unit. A special purpose VCU chip contains several VCUs of the

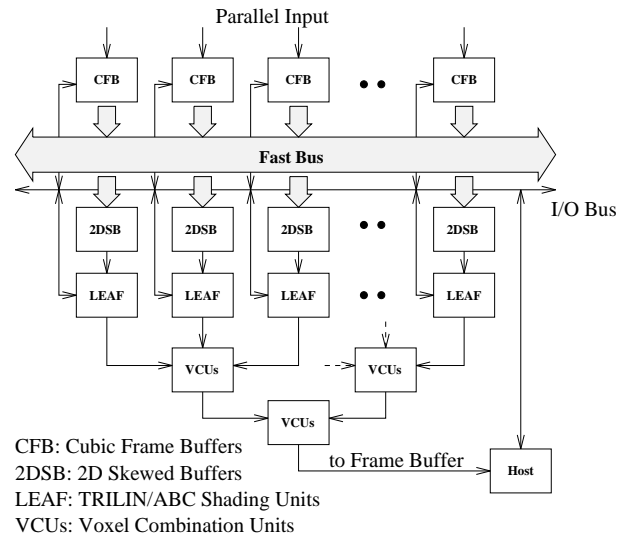


Figure 8: Cube-3 Hardware Architecture.

RPC. Each individual VCU has only modest complexity, so that the number of VCUs per chip is determined by the width of the I/O datapath. Assuming an I/O pin count of 260 pins and 16-bit datapaths, it is feasible to put 8 VCU units per chip. Sixteen VCU chips together with 128 2DSB/LEAF units fit on a single board.

The CFB and 2DSB boards are connected through the high-bandwidth Fast Bus on the backplane, which is the main technological challenge in Cube-3. Assuming a voxel resolution of 16-bit and a 512^3 dataset, the backplane contains a 512-bit wide bus at clock-speeds possibly over 100MHz. Furthermore, the backplane contains a separate I/O bus for LEAF node and host connections.

With the above board estimations, a Cube-3 system for 512^3 16-bit per voxel datasets would require 8 boards and a custom fabricated backplane. This board layout and chip count may change depending on off-the-shelf chip availability, pin count, package size, and bus interface technology.

6. Conclusions

Cube-3 is a scalable, high-resolution volume visualization architecture that exploits parallelism and pipelining to achieve real-time performance. It will provide the following capabilities to the scientist and researcher: viewing from any parallel and perspective direction, control of shading and projection parameters (e.g., first opaque, max value, x-ray, compositing), programmable color segmentation and thresholding, and control over translucency, sectioning, and slicing.

We have simulated the Cube-3 architecture in C and in Verilog, and are designing the general layout of a 512^3 16-bit per voxel prototype implementation. We are currently simulating the effects of the 10-neighborhood gradient estimation for perspective projections. Future

work includes the full design of the LEAF and VCU nodes, optimizing the pipelining between the different units, enhanced support for perspective projections, the design of high-speed parallel input channels, and the hardware implementation of a prototype.

7. Acknowledgments

This work has been supported by the National Science Foundation under grants MIP 88-05130 and CCR-9205047. The authors would like to thank all the members of the Cube-3 team that contributed to this research, especially Reuven Bakalash, Robert Pacheco, Adam Xuejun, and Frank Wessels. A preliminary short version of this paper was presented at the Eurographics Hardware Workshop 1993 [17].

References

1. B. Alleyne and I. Scherson. Expanded delta networks for very large parallel computers. In *ICPP '92 Conference Proceedings*, August 1992.
2. 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.
3. R. Bakalash, A. Kaufman, R. Pacheco, and H. Pfister. An extended volume visualization system for arbitrary parallel projection. In *Proceedings of the 1992 Eurographics Workshop on Graphics Hardware*, Cambridge, UK, September 1992.
4. D. Cohen and A. Kaufman. A 3D skewing and deskewing scheme for conflict-free access to rays in volume rendering. To appear in *IEEE Transactions on Computers*, 1994.
5. H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel. Pixel-Planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. *Computer Graphics*, 23, No. 3:79–88, July 1989.
6. R. Gemballa and R. Lindner. The multiple-write bus technique. *IEEE Computer Graphics & Applications*, 2(7):33–41, September 1982.
7. R.D. Hersch. Real scan-conversion of shape contours. In T.L. Kunii, editor, *Proceedings Computer Graphics International 87*, Karuizawa, Japan, 1987. Springer Verlag.
8. K. H. Höhne and R. Bernstein. Shading 3D-images from CT using gray-level gradients. *IEEE Transactions on Medical Imaging*, MI-5(1):45–47, March 1986.
9. A. Kaufman. *Volume Visualization*. IEEE CS Press Tutorial, Los Alamitos, CA, 1991.
10. A. Kaufman and R. Bakalash. Memory and processing architecture for 3D voxel-based imagery. *IEEE Computer Graphics & Applications*, 8(6):10–23, November 1988.
11. A. Kaufman and E. Shimony. 3D scan-conversion algorithms for voxel-based graphics. In *Proceedings ACM Workshop on Interactive 3D Graphics*, pages 45–76, Chapel Hill, NC, October 1986.
12. G. Knittel. VERVE: Voxel engine for real-time visualization and examination. In *Computer Graphics Forum*, volume 12, No. 3, pages C–37 – C–48, September 1993.
13. C. Korenfeld. The Image Prism: A device for rotating and mirroring bitmap images. *IEEE Computer Graphics & Applications*, 7(5):21–30, May 1987.
14. M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.
15. M. Levoy. Design for real-time high-quality volume rendering workstation. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, pages 85–90, Chapel Hill, NC, May 1989.
16. S. Molnar, J. Eyles, and J. Poulton. Pixelflow: High-speed rendering using image composition. *Computer Graphics*, 26(2):231–240, July 1992.
17. H. Pfister and A. Kaufman. Real-time architecture for high-resolution volume visualization. In *Proceedings of the 8th Eurographics Hardware Workshop*, volume 12, No. 3, pages 72–80, Barcelona, Spain, September 1993.
18. P. Schröder and G. Stoll. Data parallel volume rendering as line drawing. In *1992 Workshop on Volume Visualization, Boston*, pages 25–31, October 1992.
19. National Semiconductor. Transceiver chip set gets BTL treatment. *Electronic Engineering*, 63(4):11–12, April 1991.
20. H. J. Siegel. *Interconnection Networks for Large-Scale Parallel Processing*. McGraw-Hill Publishing Company, 1990.
21. D.M. Taub. Clockless synchronization of distributed concurrent processes. *IEEE Proceedings*, 139:88–91, January 1992.
22. G. Vézina, P. Fletcher, and P. Robertson. Volume rendering on the MasPar MP-1. In *1992 Workshop on Volume Visualization*, pages 3–8, Boston, MA, October 1992.
23. R. Yagel and A. Kaufman. Template-based volume viewing. *Computer Graphics Forum*, 11(3):153–167, September 1992.