Implementations of Cube-4 on the Teramac Custom Computing Machine

U. Kanus, M. Meißner, W. Straßer

WSI/GRIS, University of Tübingen, Germany

H. Pfister, A. Kaufman

Center for Visual Computing, Computer Science Department, SUNY at Stony Brook, NY, USA

R. Amerson, R.J. Carter, B. Culbertson, P. Kuekes, G. Snider

Hewlett-Packard Research Laboratories, Palo Alto, CA, USA

Abstract

We present two implementations of the Cube-4 volume rendering architecture, developed at SUNY Stony Brook, on the Teramac custom computing machine. Cube-4 uses a slice-parallel ray-casting algorithm that allows for a parallel and pipelined implementation of ray-casting. Tri-linear interpolation, surface normal estimation from interpolated samples, shading, classification, and compositing are part of the rendering pipeline. Using the partitioning schemes introduced in this paper, Cube-4 is capable of rendering in real-time large datasets (e.g., 1024^3) with a limited number of rendering pipelines. Teramac is a hardware simulator developed at Hewlett-Packard Research Laboratories. Teramac belongs to the new class of custom computing machines, which combine the speed of special-purpose hardware with the flexibility of general-purpose computers. Using Teramac as a development tool, we implemented two working Cube-4 prototypes capable of rendering 128^3 datasets in 0.65 seconds at a very low 0.96 MHz processing frequency. The results from these implementations indicate scalable performance with the number of rendering pipelines and real-time frame-rates for high-resolution datasets.

1 Introduction

Volume rendering is a key technology with increasing importance for the visualization of 3D sampled, computed, or modeled datasets. 3D volumetric data is delivered by acquisition devices such as biomedical scanners (MRI, CT) or

acoustic wave devices for geophysical explorations, as well as super-computer simulations and scientific experiments, including aerodynamics, weather simulations, material tests, and many more. Volume rendering provides a powerful technique to reveal the information contained in these datasets. Volume rendering is also used in volume graphics for rendering geometry-based models represented as volume datasets [KCY93].

The computational cost for volume rendering is very high and becomes worse for the visualization of dynamically changing datasets in real-time, a process that is called 4D (spatio-temporal) visualization. Numerous software approaches for interactive volume rendering, mainly based on algorithmic optimizations and large-scale parallelism, have been introduced. The highest performance for rendering of a 256^3 dataset at over 10 frames per second was achieved on a 16 processor SGI Challenge using the shear-warp algorithm [Lac96]. This impressive achievement is only possible by using lengthy precalculations, storage of large auxiliary data structures, approximations, 2D instead of 3D interpolation, and expensive multi-processor machines.

Providing real-time volume rendering at a reasonable cost with high image quality is the goal of special-purpose volume rendering hardware. The Cube project [KB88,PKC94,PK96] for hardware accelerated volume rendering pioneered several volume rendering architectures using parallel rendering processors and a special interleaved memory organization to provide high processing performance and memory bandwidth.

Cube-4, the most recent approach, is a parallel and scalable architecture with modular rendering pipelines using only local and fixed bandwidth interconnections [PK96]. Cube-4 is estimated to achieve real-time performance (30 frames per second) for high-resolution (e.g., 1024^3) datasets. Cube-4 uses 3D interpolation and high-quality surface normal estimation without any precomputations or additional data storage. The performance of Cube-4 grows proportionally with increasing number of rendering pipelines, ultimately limited only by memory speed. The cost-performance ratio of Cube-4 is significantly better than existing solutions.

This paper describes two prototype implementations of the Cube-4 architecture on the *Teramac* hardware simulator at Hewlett-Packard research laboratories, Palo Alto, CA. Teramac belongs to a new class of machines called *custom computing machines* (CCM) which provide the user with a huge amount of programmable logic, thus combining the speed of special-purpose hardware with the flexibility of general-purpose computers.

In Section 2 we describe the Cube-4 rendering pipeline which implements slice-parallel ray-casting, an efficient parallel algorithm for volume rendering. We discuss two architectural partitioning schemes for rendering large volumes

with a small number of rendering pipelines. Section 3 gives an overview of the Teramac hardware and software system. In Section 4 we discuss our two Cube-4 implementations on the Teramac and present results in the form of performance numbers and images.

2 Cube-4

Cube-4 implements *ray-casting*, the most commonly used image-space volume rendering method [Lev88]. Rays are cast from the viewpoint into the volume. At evenly spaced locations along each ray, a sample value is computed using surrounding voxels. A surface normal approximation for a sample point is obtained by computing the gray-level gradient [HB86]. The so computed surface normal together with the computed sample value is used to assign each sample a color based on a local shading model. Using the density value and gradient magnitude each sample is classified by assigning an opacity. Shaded and classified sample values are composited along the rays into pixel values of the final image.

To achieve real-time performance we need to remove several bottlenecks of the ray-casting algorithm, the most important being the frequent and mostly random accesses to memory. Voxels may be addressed multiple times due to the non-uniform mapping of samples along the rays and due to the overlap of voxel neighborhoods during independent calculations, namely interpolation and gradient estimation. To get a one-to-one mapping of ray-samples onto voxels we use a template-based ray-casting technique first introduced by Yagel and Kaufman [YK92], and shown in Figure 1.

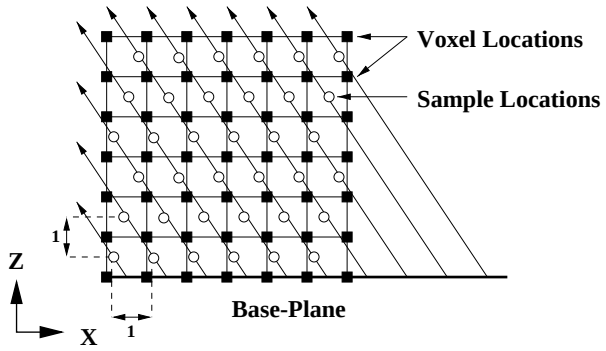


Fig. 1. Template-based ray-casting.

Discrete voxel rays with a constant stepping of one in the major viewing direction are sent into the volume from each pixel on the base-plane, which is the face of the volumetric dataset that is most perpendicular to the viewing direction. After the volume has been rendered, the base-plane contains a distorted image which has to be warped onto the view-plane [LL94].

For real-time performance this template-based ray-casting algorithm needs to be parallelized. In Cube-4 we implement a form of parallelism called *slice-parallel* processing [PK96]. During ray-casting, the volume is traversed along consecutive slices parallel to the base-plane. The conceptual dataflow of slice-parallel ray-casting is shown in Figure 2.

Two consecutive slices are required for tri-linear interpolation. To reduce the number of memory accesses, the previously fetched slice is stored in a plane buffer (FIFO) so that it can be retrieved without further access to the voxel memory. The gradient is computed using samples from three slices of interpolated samples [PKC94]. The two previously calculated slices of interpolated sample are stored in FIFO plane-buffers, delaying them by n and $2n$ cycles, respectively. After shading and classification each slice is composited onto the intermediate results of the previous slices, yielding the final base-plane image after n^2 cycles.

The slice-parallel approach discussed so far operates on beams of n voxels, thus requiring n memory modules and n rendering pipelines, where n is the resolution of the dataset. This leads to an undesirable amount of hardware and limits the maximum dataset size that can be rendered. To render datasets of size n^3 with $p < n$ rendering pipelines, we developed two different architectural partitioning approaches, called *sub-volume partitioning* and *beam partitioning*.

In sub-volume partitioning, a volumetric dataset of size n^3 is divided into smaller sub-volumes of resolution p, each being processed by p pipelines. The images of each sub-volume are combined to yield the final image. Our first prototype implementation on Teramac, described in Section 4, uses sub-volume partitioning.

However, this first prototype revealed two main problems with this approach. First, the voxel neighborhood required for tri-linear interpolation and gradient

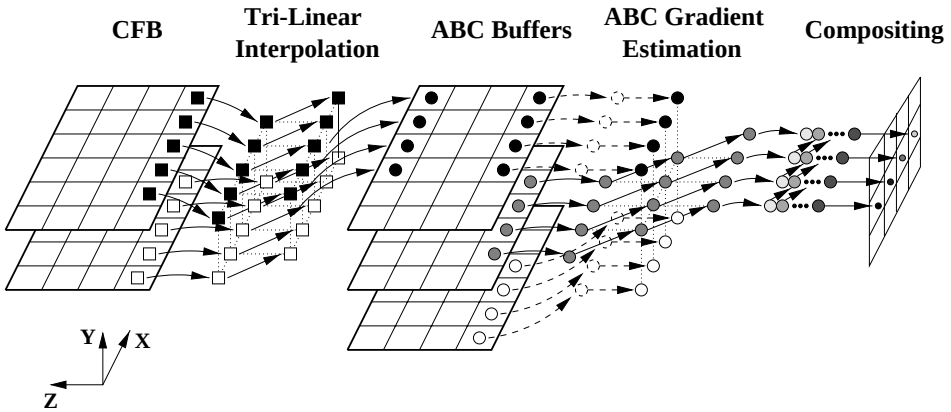


Fig. 2. Dataflow of slice-parallel ray-casting. (CFB = Cubic Frame Buffer, ABC = Above Below Current)

estimation at sub-volume boundaries can only be provided by overlap of sub-volumes. As Table 1 shows, this results in substantial memory overhead, which leads to higher execution time (see Section 5).

Rendering Pipelines p	Memory Overhead (in percent) Sub-volume Size is $p \times p \times 128$
8	61%
16	34%
32	18%
64	10%

Table 1
Memory overhead in percent due to boundary-voxel overlap for sub-volume partitioning of a 128^3 dataset.

The second problem is that rays can traverse multiple sub-volumes for non-orthogonal viewing directions, as illustrated in Figure 3. The intermediate compositing results for rays that cross the sub-volume boundary have to be stored in a buffer so that they can be accessed during processing of the next sub-volume. The order in which the sub-volumes have to be processed depends on the viewing direction and the compositing order (front-to-back or back-to-front). To access the buffer of intermediate compositing results requires global connectivity between processing pipelines.

These problems with sub-volume partitioning lead to the development of beam partitioning. A *beam* is a vector of voxels which is parallel to one of the main dataset axes. The parallel skewed memory organization used in all Cube architectures allows conflict free access to any beam in one memory access cycle [KB88]. Instead of subdividing the volume into sub-volumes, the size of beams

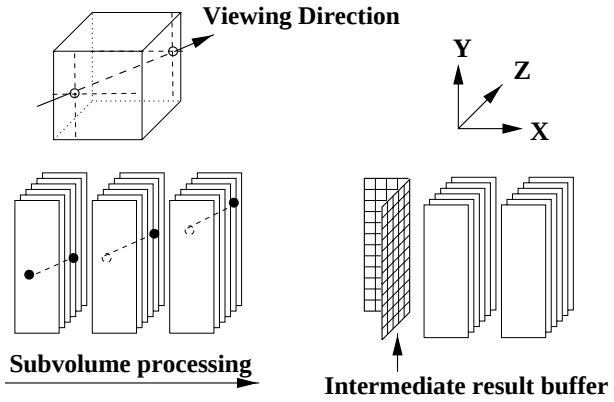


Fig. 3. Sub-volume processing order for front-to-back compositing and a given viewing direction. Intermediate results at sub-volume boundaries have to be stored for subsequent processing.

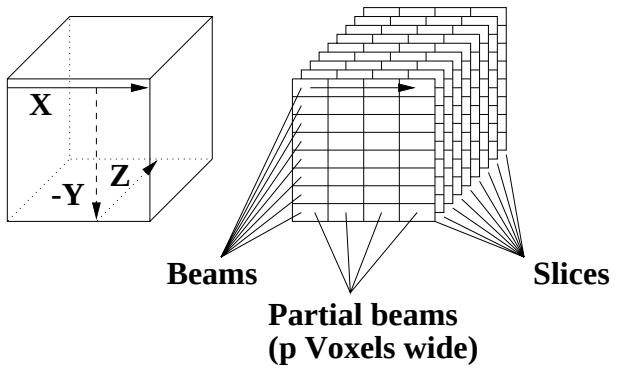


Fig. 4. Volume traversal for beam partitioned slice-parallel ray-casting.

is adjusted to the number of processing elements (see Figure 4). With p processing units, beams are partitioned into b *partial beams* of width p, which are subsequently processed. In our Cube-4 implementations on Teramac, processing proceeds along partial beams in $+X$, inside slices in $-Y$, and across slices in $+Z$ direction.

Similar to sub-volume partitioning, the voxel-neighborhoods required for tri-linear interpolation and gradient estimation need to overlap at the border of partial beams. For example, tri-linear interpolation at the rightmost position of a partial beam requires voxels from the partial beam which will be fetched in the next cycle. Using a technique called *beam extension*, these border cases can be handled without the overhead in computation and storage of sub-volume partitioning. Partial beam i at time t is delayed by one cycle so that the necessary extension for partial beam i can be transferred from partial beam $i+1$ at time $t+1$ (see Figure 5).

The next section gives an overview of the Teramac system. In Section 4 we describe the sub-volume partitioned prototype implementation of Cube-4 on Teramac, and Section 4.2 describes our beam partitioned Cube-4 prototype on Teramac.

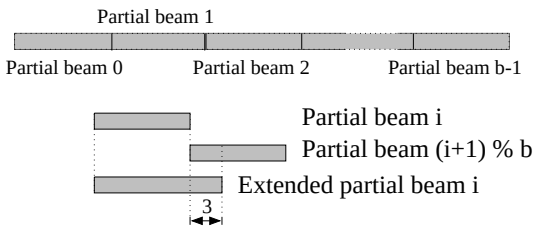


Fig. 5. Beam extension provides the necessary data on partial beam boundaries.

3 Teramac - a CCM

The merits of general-purpose versus special-purpose computers have long been debated by computer architects. The configurable custom machine (CCM) [BRV89,ABD92] is a new class of machine that falls between these extremes. Teramac [ACC+95], the largest such machine built to date, achieves the massive parallelism of special-purpose computers and the re-usability of general-purpose computers. Teramac provides large numbers of programmable gates, wires, and memories that can be configured to implement user designs. When special-purpose hardware is built, its correctness and usability can be verified first with a custom computer. The high speed of custom computing, relative to conventional software simulations, makes much more exhaustive testing possible.

General-purpose computers have many virtues: they are ubiquitous, inexpensive, and easy to program. They typically also have significantly higher clock speeds than custom computers. However, because general-purpose computers execute at most a handful of instructions per clock cycle, while custom computers perform hundreds, custom computers are potentially much faster. On many applications, Teramac has out-performed high-performance workstations by a factor of a hundred or more.

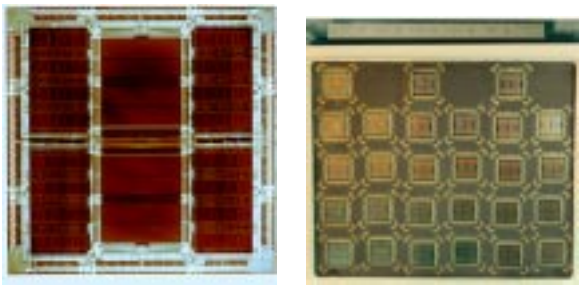
3.1 Teramac Hardware

Teramac is scalable, with systems comprising one to sixteen boards. Figure 6 shows four Teramac boards with the attached controller boards and the board to board connections. A full sixteen-board system is capable of running user designs with one million gates at speeds typically in the range of one megahertz.

A custom field-programmable gate array (FPGA), called Plasma [ACC+96], supplies the majority of Teramac's programmable resources: gates, crossbars, and multi-ported register files. Groups of twenty-seven FPGAs are assembled into large multi-chip modules (MCMs) [AK94] (see Figure 7). Each board contains four MCMs. Each board also contains four dual-ported two-megaword by 32-bit RAM's; thus, Teramac's memory resources are very ample in both capacity and bandwidth.

The Teramac routing resources, consisting of crossbars in the FPGAs and wires on the MCMs and boards, are sufficient for implementing almost any circuit topology. In particular, user circuits are not limited to systolic arrays, as they were in earlier custom computers. Users control Teramac from a host workstation, which connects to Teramac via a SCSI bus. The host also provides

Fig. 6. Four Teramac boards, connected to each other with ribbon cable, and to a controller board. The pins of one multi-chip module (MCM) can be seen in the middle.



(a) (b)

Fig. 7. The Teramac hardware. a) A PLASMA FPGA chip, configurable in three seconds. b) MCM with 27 PLASMA chips on it. The interconnections are routed in 39 layers. Each MCM has over 3,000 pins.

configurations and I/O.

3.2 Teramac Software

Configurable computers are of limited usefulness unless they include software to map designs onto them. Teramac was designed with the goal that user designs would be mapped onto it quickly and completely automatically. To insure that this goal was achieved, the Teramac hardware and mapping software were created in tandem. Large designs that fill our eight-board Teramac system typically are mapped onto the system in about half an hour, making design iterations reasonably painless.

Users enter their designs into software tools that transform them in two steps into configurations that are ready to run on Teramac. For design entry and the first step of the transformation process, we use general-purpose digital

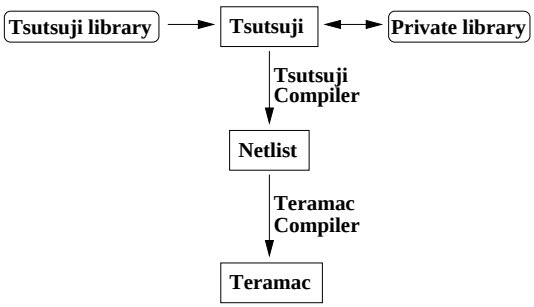


Fig. 8. Design-flow for Teramac.

hardware design tools. To maximize user productivity, we have chosen tools that permit the user to express their designs at a high level of abstraction. These tools use logic synthesis to automatically convert the highlevel designs into netlists of simple gates.

The Cube-4 design was created with the Tsutsuji design system [COO+93]. Tsutsuji accommodates large designs particularly well and synthesizes them into gates within minutes. Tsutsuji designs are hierarchies of block diagrams. The blocks represent one of three things: sub-designs which are themselves block diagrams; data path elements (adders, multipliers, multiplexors, etc.) for which Tsutsuji provides an extensive library of sophisticated module generators; and sub-designs whose behavior is described in Tsutsuji's textual Logic Description Format (LDF). LDF is intended for describing state machines, random logic, and truth tables. We have found that LDF is also useful for creating parameterized designs. Parameterized designs are ideal for parallel applications because they allow the degree of parallelism in the design to be scaled to fill the available hardware.

The second step of the process of creating configurations is called mapping. It is performed by the Teramac compiler, which was written expressly for Teramac. It reads the netlists, merges the simple gates into FPGA-specific gates, performs placement and routing, and ultimately creates configuration bitstreams. Figure 8 shows the design-flow for Teramac.

In the following section we introduce the implementation of two Cube-4 prototype designs using the Teramac system and highlight the achieved results.

4 Cube-4 Prototypes on Teramac

Two prototype designs of Cube-4 were implemented on the Teramac custom computing system. The first design is based on the sub-volume approach, while the second uses beam partitioning.

4.1 Sub-Volume Partitioned Design

The sub-volume partitioned approach has been implemented with eight parallel pipelines, shown in Figure 9. Each pipeline includes the Cubic Frame Buffer (CFB) volume memory, the CFB address generator, tri-linear interpolation unit (TRI), and gradient estimation unit (GRA). Shading, classification and compositing have been implemented in software.

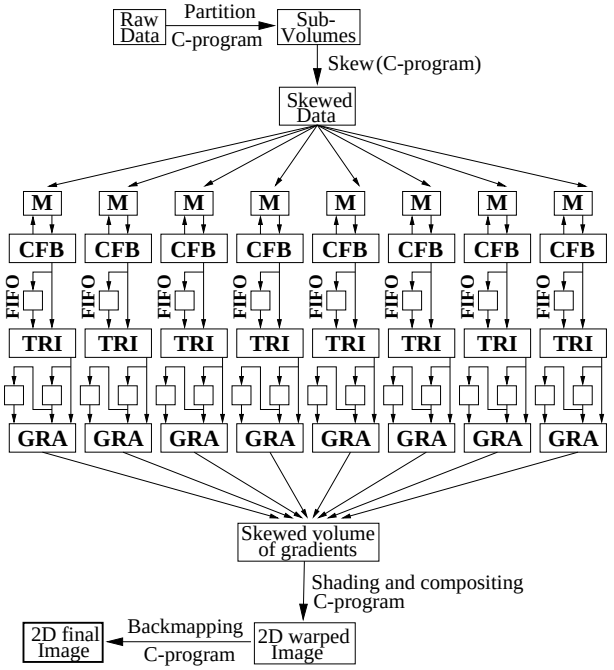


Fig. 9. Block diagram of the sub-volume partitioned Cube-4 implementation on the Teramac with eight rendering pipelines. (M = DRAM Memory Module, CFB = Cubic Frame Buffer Address Generation, TRI = Tri-linear Interpolation Unit, GRA = ABC Gradient Estimation Unit.)

To provide the original volume data in a skewed and partitioned format we use a software front-end written in C. A dataset is transformed into a file containing the skewed data of all sub-volumes in sequential order, for down-loading to the Teramac memory. Our implementation on Teramac performs memory access for arbitrary viewing directions, tri-linear interpolation between data slices, and ABC gradient estimation around sample points. The resulting sample values and gradient vectors are transferred from the Teramac memory onto the host computer for post-processing (shading, classification and compositing) with the software back-end.

Our slice-parallel sub-volume partitioned Cube-4 design on Teramac is capable of rendering datasets of 128^3 Voxels. Our implementation contains eight rendering pipelines, although available logic gates on Teramac would allow implementing a design with 16 pipelines. The timing results of this design

(see Section 5) indicate high performance. However, the global connectivity required for the partial result buffers in the compositing units is a major drawback of the sub-volume partitioned design. Consequently, no further effort was put into this implementation.

4.2 Beam Partitioned Design

Our second prototype design on Teramac uses beam partitioning and implements the complete rendering pipelines, including shading (SHA) and compositing (COM) (see Figure 10). The back-end software performs the 2D image warp, while all other rendering operations are implemented in hardware.

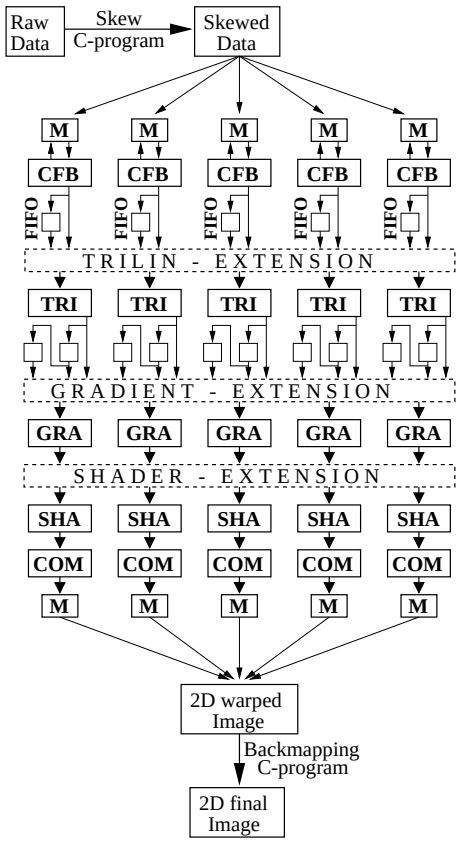


Fig. 10. Block diagram of the beam partitioned Cube-4 implementation on the Teramac with five rendering pipelines. (M = DRAM Memory Module, CFB = Cubic Frame Buffer Address Generation, TRI = Tri-linear Interpolation Unit, GRA = ABC Gradient Estimation Unit, SHA = Shading Unit, COM = Compositing Unit.) The interconnections provided inside the extension units are only local, not global.

We implemented a Cube-4 configuration with five parallel rendering pipelines. The limitation to five pipelines was given due to the structure of the Teramac memory system. A total of 256 MBytes of memory, distributed across several memory banks, is available on Teramac. We use memory banks to re-

alize the plane-buffers, the look-up tables for opacity, color transfer-functions, and shading parameters, as well as the intermediate image buffers in the compositing units. Five Cube-4 rendering pipelines used up all available Teramac memory banks.

Our beam partitioned Teramac prototype is able to process datasets of 125^3 voxels. A dataset is down-loaded into Teramac memory, processed, and the final base-plane pixels are stored in memory modules at the end of each rendering pipeline. A software program uploads the pixel values and performs the 2D image warp from the base-plane to the image plane. In the following section we describe the design of the different pipeline stages in more detail.

4.3 Rendering Pipeline Hardware

The address of a voxel in volume space can be described in terms of a *slice index* (*S_INDEX* or S) in major viewing direction, a *beam index* (*B_INDEX* or B) in scanline direction, a *partial beam index* (*PB_INDEX* or PB) and a (*PIPELINE_INDEX*) for the location inside a partial beam. For $p = 5$ memory banks, we obtain the memory address A using the following formula:

$$A = S * \frac{125^2}{5} + B * \frac{125}{5} + PB \tag{1}$$

This formula is used in the CFB to address the memory banks. The CFB is the main control unit of each pipeline. It is split up into four sub-units as shown in Figure 11. The first is the *TRAVERSAL_UNIT* which keeps track of the position of the currently fetched voxel inside the volume. It consists of three cascaded counters, one for PB_INDEX, one for B_INDEX, and one for S_INDEX (see Figure 4). The values of the three counters are provided to the other sub-units of the CFB unit.

The *ADDRESS_UNIT* is connected to the voxel memory of each pipeline, one 8 MBytes bank of Teramac memory. The *TEMPLATE_UNIT* generates the resampling weights for the tri-linear interpolation which are forwarded to the TRI unit. To reduce the amount of logic, weights are updated incrementally every time the S_INDEX changes. The current resampling weights in X and Y are updated by simply adding the components of the viewing vector *VIEW_X* and *VIEW_Y*, respectively, modulo 256 (we use 8 bits for resampling weights).

The *CONTROL_UNIT* provides the control information (13 bits, shown in Table 2) forwarded with data, allowing the other stages of the pipeline to correctly align the data. *Start* and *End* indicate the beginning and the end of a volume. *Forget* marks invalid intermediate values. *X-wrap* and *Y-wrap*

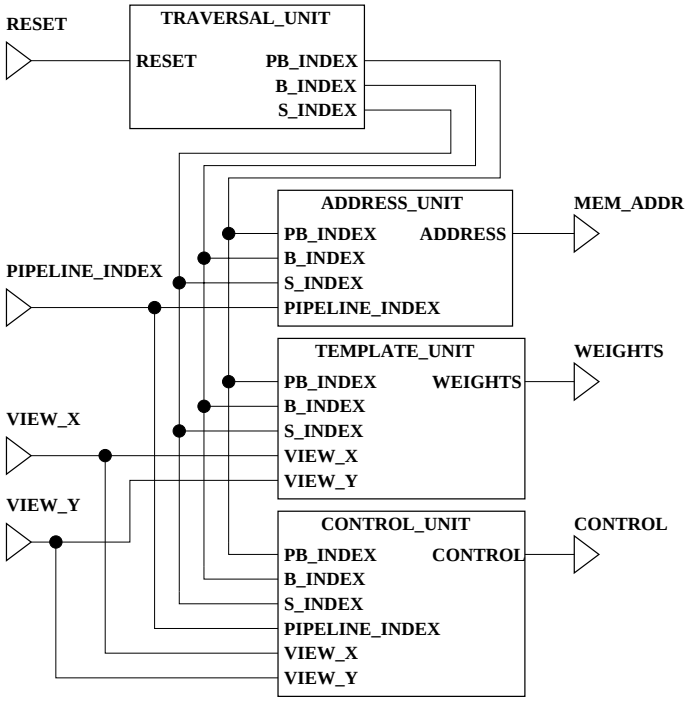


Fig. 11. CFB address unit block diagram. PB_INDEX indicates the index of the current partial beam, while B_INDEX and S_INDEX indicate the current beam- and slice-index.

indicate that a sample is the last one along a ray. *old-X-step*, *old-Y-step*, *X-step* and *Y-step* mark discrete steps along rays between slices. This information is required to reconstruct the rays for compositing.

In the tri-linear interpolation unit (TRI) the interpolation of the samples is performed using the weights calculated in the CFB. Seven linear interpolators are able to calculate one sample per cycle [Kni93]. The gradient unit (GRA) aligns samples out of three consecutive slices to compute the gray-level gradient [HB86]. This unit also performs a correction of the values to generate a gradient parallel to the Z-axis and to prevent aliasing [PWK94].

Bit-No.	0	1	2
Signal	Start	End	Forget
Bit-No.	3/4	5,6/7,8	9,10/11,12
Signal	X/Y-wrap	old-X/Y-step	X/Y-step

Table 2
Control signals for the beam partitioned approach.

The shading unit (SHA) uses the three components of the gray-level gradient for a lookup-table based implementation of Phong shading [BvS95]. The lookup-table requires only 1.5 kBytes of memory and four memory accesses per computation cycle. We used a four times wider implementation with 6 kBytes lookup-tables because the Teramac memory banks limit memory access to one read and one write per cycle. The resulting intensity value from the shading unit is then used as an index into three color lookup tables, resulting in red, green, and blue color components. Sample values are used to assign 32-bit opacity values for compositing. The tables for the classification of the samples are 32 bits wide and 256 entries deep, corresponding to the 8-bit representation of voxel values.

In the compositing unit (COM), the color samples delivered by the shading unit are blended into final pixels. The slice-by-slice order requires a base-plane buffer for one slice of intermediate compositing results, which has 125 entries of 25 bits each per pipeline. Incoming shaded samples are directly composed with the corresponding previous values from the base-plane buffer. Compositing is performed in front-to-back order [Lev88], and the base-plane buffer is implemented using Teramac memory banks. After a ray is finished, its final pixel value is output into Teramac memory together with its base-plane x and y address.

5 Results

The sub-volume partitioned design with eight rendering pipelines is capable of rendering 128^3 datasets. Using multiple register stages in the rendering pipeline allowed us to optimize the design from an initial processing frequency of 0.37 MHz to a final frequency of 0.96 MHz. At 0.96 MHz we achieved a frame-rate of 1.5 Hz using eight parallel rendering pipelines. The design of the eight rendering pipelines uses 162,816 logic gates, where one CFB unit requires 5,578 gates, one tri-linear unit (TRI) requires 8,557 gates, and one gradient estimation unit (GRA) requires 6,142 gates. The tri-linear unit requires more gates than any of the other units due to the multipliers for the seven linear interpolators used for tri-linear interpolation. Figure 12 shows volume rendered images of a CT scanned lobster with different transfer functions and different light sources rendered with the sub-volume partitioned Cube-4 design.

The beam partitioned Cube-4 implementation with five pipelines has not been optimized for speed. A SPICE-estimated maximum clock-rate of 0.2 MHz was achieved for 125^3 datasets. The resulting frame rate of 0.5 Hz could be increased to 2.5 Hz by pipelining the design further to a clock frequency of 0.96 MHz. In that case the beam partitioned design with five pipelines would be faster than the sub-volume partitioned design with eight pipelines.

The complete design uses 380,341 logic gates, where one CFB unit requires 3,918 gates, one tri-linear unit requires 11,037 gates, one gradient estimation unit requires 18,030 gates, one shading unit requires 13,858 gates, and one compositing unit requires 12,861. The logic needed to implement the beam extensions requires 80,932 gates. Tri-linear and gradient estimation units have a larger size due to the necessary partial-beam buffers. Many gates can be saved if the partial-beam buffers are implemented with Teramac memory or hardware FIFOs instead of using the expensive Teramac registers.

Assuming perfect pipelining of interpolation, shading, and compositing, the theoretical performance of Cube-4 is dependent on the number of rendering pipelines p and the processing frequency f_p. If n is the dimension of the dataset, and f_r the rendering rate in frames per second, we can calculate the necessary processing frequency f_p in Hz as:

$$f_p = \lceil \frac{n^3 f_r}{p} \rceil .$$

Figure 12 shows the processing frequency f_p as a function of the number of rendering pipelines p for different dataset resolutions and 32 projections per second ($f_r = 32$).

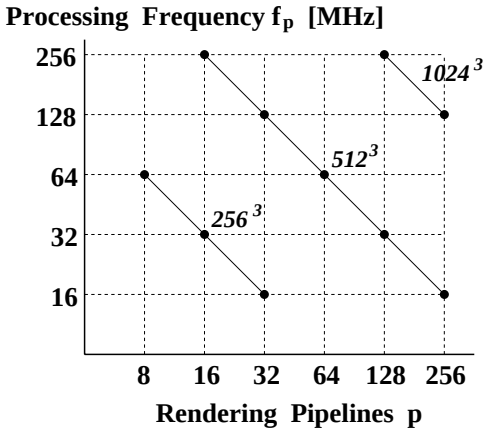


Fig. 12. Processing frequency, f_p, vs. the number of rendering pipelines p for three dataset resolutions. Rendering performance is $f_r = 32$ frames per second.

It can be seen from Figure 12 that 8 rendering pipelines achieve 32 frames per second projection rates for 256^3 datasets at 64 MHz processing frequency. At 16 bits per voxel, such a dataset requires 32 MBytes of DRAM. Using two 16 Mbits synchronous DRAM (SDRAM) modules per rendering pipeline requires only 16 SDRAMs. Given the gate count for logic from our Teramac implementation, it is fair to assume that we can fit four rendering pipelines onto one application specific integrated circuit (ASIC). Such a Cube-4 ASIC would require less than 500,000 logic gates and about 40 kBytes SRAM for the

internal data buffers. Two ASICs (with 8 rendering pipelines), 16 SDRAMs (with 32 MBytes total capacity), and a PCI host interface can fit onto a PCI card for cost-effective, 30 frames per second visualization of 256^3 datasets. Practical implementations for higher resolution datasets require more Cube-4 ASICs and higher processing frequencies.

Figure 14 shows parallel projections of several datasets. Those images were rendered completely on Teramac. Additionally, we implemented a protocol for automatically generating all the frames for an animation on Teramac.

6 Conclusions

We presented two scalable and modular partitioning schemes for the Cube-4 slice-parallel ray-casting architecture and proved their feasibility by implementing them on the Teramac system. Simulating architectures of this size is not a trivial task. Teramac was a valuable tool that allowed us to efficiently implement those designs in a very limited time-frame. An important future extension to the Teramac system is a frame-buffer to display graphics without uploading results to a host. Furthermore, porting designs to Teramac will be easier in the future when the software is able to directly compile a VHDL description.

Implementing Cube-4 on the Teramac system was a major step towards a full-fledged real-time volume rendering system. We were able to prove the feasibility of the scalable and modular Cube-4 design and obtained a first impression of its image quality. The next logical step is to use this experience to develop an improved VLSI implementation of Cube-4 which will then provide real-time performance for datasets of up to 1024^3 voxels. These are our near future goals.

Acknowledgments

Cube-4 has been developed at the Visualization Lab of the Center for Visual Computing, State University of New York at Stony Brook, NY, and has been supported by the National Science Foundation under grant MIP-9527694, Japan Radio Corporation, and Hewlett Packard. Datasets for Figures 15 and 17 are courtesy of Siemens, Scripps Clinic, AVS, UNC, Howard Hughes Medical Institute, and the Visualization Laboratory at Stony Brook. The authors would like to thank all the members of the Cube-4 team that contributed to this research, especially Frank Wessels, Ingmar Bitter, and Pat Tonra. Urs

Kanus and Michael Meißner performed this work as part of their MS thesis at Stony Brook, NY, and at Hewlett-Packard Research Laboratories, Palo Alto, CA. We would like to thank Fred Kitson and Tom Malzbender at Hewlett-Packard for their support that made this collaboration possible.

References

[ABD92] J. M. Arnold, D. A. Bell, and E. G. Davis. Splash 2. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 316–322, 1992.

[ACC+95] R. Amerson, R. J. Carter, W. B. Culbertson, P. Kuekes, and G. Snider. Teramac – configurable custom computing. In *Proceedings of the 1995 IEEE Symposium on FPGA's for Custom Computing Machines*, pages 32–38, Napa, CA, April 1995.

[ACC+96] R. Amerson, R. Carter, W. Culbertson, P. Kuekes, and G. Snider. Plasma: An fpga for million gate systems. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 10–16, 1996.

[AK94] R. Amerson and P. Kuekes. The design of an extremely large mcm-c – a case study. In *The International Journal of Microcircuits and Electronic Packaging*, pages 337–382, 1994.

[BRV89] P. Bertin, D. Roncin, and J. Vuillemin. Introduction to programmable active memories. In *Systolic Array Processors*, pages 301–309, Killarney, Ireland, May 1989.

[BvS95] M. Bosma and J. van Scheltinga. Efficient super resolution volume rendering. Master's thesis, University of Twente, Faculty of Electrical Engineering, Enschede, The Netherlands, August 1995. TR EL-BSC-079N95.

[COO+93] W.B. Culbertson, T. Osame, Y. Otsuru, J.B. Shackleford, and M. Tanaka. The hp tsutsuji logic synthesis system. *Hewlett Packard Journal*, pages 38–51, 1993.

[HB86] 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.

[KB88] A. Kaufman and R. Bakalash. Memory and processing architecture for 3D voxel-based imagery. *IEEE Computer Graphics & Applications*, 8(6):10–23, November 1988.

[KCY93] A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *Computer*, 26(7):51–64, July 1993.

[Kni93] G. Knittel. VERVE: Voxel engine for real-time visualization and examination. In *Computer Graphics Forum*, volume 12, No. 3, pages 37–48, September 1993.

[Lac96] P. Lacroute. Analysis of a parallel volume rendering system based on the shear-warp factorization. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):218–231, September 1996.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics & Applications*, 8(5):29–37, May 1988.

[LL94] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transform. In *Computer Graphics*, Proceedings of SIGGRAPH 94, pages 451–457, July 1994.

[PK96] H. Pfister and A. Kaufman. Cube-4 – a scalable architecture for real-time volume rendering. In *1996 ACM/IEEE Symposium on Volume Visualization*, pages 47–54, San Francisco, CA, October 1996.

[PKC94] H. Pfister, A. Kaufman, and T. Chiueh. Cube-3: A real-time architecture for high-resolution volume visualization. In *1994 ACM/IEEE Workshop on Volume Visualization*, pages 75–83, Washington, DC, October 1994.

[PWK94] H. Pfister, F. Wessels, and A. Kaufman. Sheared interpolation and gradient estimation for real-time volume rendering. In *Proceedings of the 9th Eurographics Hardware Workshop*, pages 70–79, Oslo, Norway, September 1994.

[YK92] R. Yagel and A. Kaufman. Template-based volume viewing. *Computer Graphics Forum, Proceedings Eurographics*, 11(3):153–167, September 1992.