PAPER *Special Issue on Development of Advanced Computer Systems*

ReVolver/C40: A Scalable Parallel Computer for Volume Rendering—Design and Implementation—

Shin-ichiro MORI[†], ***Regular Member***, **Tomoaki TSUMURA**[††], **Masahiro GOSHIMA**[†], **Yasuhiko NAKASHIMA**[††], **Hiroshi NAKASHIMA**[†*], ***Nonmembers***, ***and*** **Shinji TOMITA**[†], ***Fellow***

SUMMARY This paper describes the architecture of *ReVolver/C40* a scalable parallel machine for volume rendering and its prototype implementation. The most important feature of *ReVolver/C40* is view-independent real time rendering of translucent 3D object by using perspective projection. In order to realize this feature, the authors propose a parallel volume memory architecture based on the principal axis oriented sampling method and parallel treble volume memory. This paper also discusses the implementation issues of *ReVolver/C40* where various kinds of parallelism extracted to achieve high-perfromance rendering are explained. The prototype systems had been developed and their performance evaluation results are explained. As the results of the evaluation of the prototype systems, *ReVolver/C40* with 32 parallel volume memory is estimated to achieve more than 10 frame per second for 256^3 volume data on 256^2 screen by using perspective projection. The authors also review the development of *ReVolver/C40* from several view points.

key words: *volume rendering, parallel processing, scalable architecture, visualization*

1. Introduction

Volume rendering is a visualization technique for a three dimensional object called *volume data* which has color and transparency on its grid points called *voxels*. Such a volume data is conventionally obtained by medical scanning equipments such as CT and MRI scanner, but recently it is also produced by scientific computation such as for structural analysis and fluid dynamics. Since volume rendering requires a huge amount of computational power and resources, it is necessary to exploit parallel processing techniques for high speed rendering.

In order to realize real-time volume rendering for large volume data, we had proposed a parallel hardware architecture [6], [29] for volume rendering and we have designed *ReVolver/C40* [22], [23], [30] as its prototype implementation (Fig. 1).

In this paper, we introduce the architecture of *ReVolver/C40* in Sect. 2. Then we discuss issues related to the implementation of *ReVolver/C40* in Sect. 3, followed by its performance evaluation in Sect. 4. After that, we review the development of *ReVolver/C40* from several view points in Sect. 5 and conclude in Sect. 6.

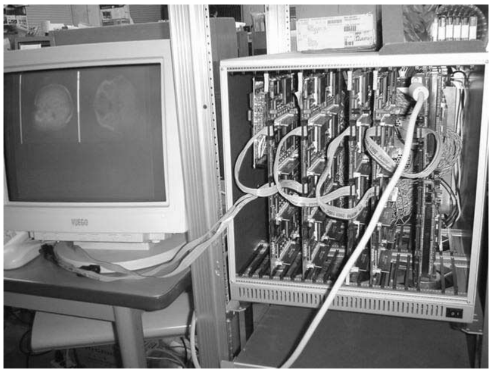


Fig. 1 ReVolver/C40-demo.

2. Characteristics of ReVolver/C40

2.1 Fundamental Features in ReVolver/C40

ReVolver/C40 adopts ray-casting style volume rendering algorithm which is widely used in image-space approach. This algorithm traces rays through the volume, accumulating color along each ray (Fig. 2). Let $v_0, v_1, \ldots, v_n$ be voxels along a ray spatially ordered from view point, c_i and t_i be color and transparency of v_i respectively. The pixel value (or color) C for this ray is

$$C = \sum_{i=0}^{n-1} c_i(1-t_i) \prod_{j=0}^{i-1} t_j. \tag{1}$$

This calculation is referred to as *composition operation* [1] and it can be computed recursively using the following equation.

$$\begin{aligned} C_{out} &= C_{in} + T_{in} \times t_i \times c_i \\ T_{out} &= T_{in} \times t_i \end{aligned} \tag{2}$$

where C_{out} and T_{out} are the total accumulated color and transparency just after the ray hits i-th voxel, and C_{in} and T_{in} are the total accumulated color and transparency just before the ray hit the voxel. This calculation is referred to as *over operation* [1]. It is because C_{out} and T_{out} can be thought as color and opacity of the combined object in which an object that has color

Manuscript received January 24, 2003.

[†]The authors are with Graduate School of Informatics, Kyoto University, Kyoto-shi, 606–8501 Japan.

[††]The authors are with Graduate School of Economy, Kyoto University, Kyoto-shi, 606–8501 Japan.

[*]Presently, with Toyohashi University of Technology.

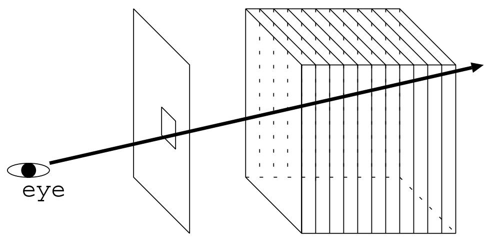


Fig. 2 Volume rendering (Ray casting algorithm).

C_{in} and transparency T_{in} is placed over voxel v_i.

These equations show that volume rendering belongs to a class of memory intensive computation. Thus the memory performance is crucial to the rendering speed. If we are able to have enough memory performance, we can benefit too much from fruitful sources of parallelisms exist in this rendering alogrithm. The ReVolver/C40 was designed by compiling all these parallelisms into this machine.

Since the target of *ReVolver/C40* is real-time visualizaion of not only medical volume data but also scientific data, *ReVolver/C40* also has the following features.

1. *Discrete and continuous models*
   In *discrete* model, it is assumed that a voxel is the center of a unit cube and any point in the cube has the same voxel value. On the other hand, voxels in *continuous* model are at vertices of the cube and the value of a point in the cube is obtained by linear interpolation of the voxel values. Both models should be applicable depending on the objective of rendering.
2. *Translucent volumes*
   Volumes to be rendered should be *translucent* so that the interior of objects is seen. The translucent rendering also copes with an *opaque* object by giving zero transparency to the voxels for its surface.
3. *Perspective and parallel projection*
   For the visualization of scientific computation results, such as analysis of seismic waves, *perspective* projection is often suitable for showing wide range of space. For small objects, such as human brain, *parallel* projection is also available as in conventional systems.
4. *Real-time rendering*
   Quick response following the movement of view point is essentially important to help the analysis and recognition of complicated volume data. This *real-time* rendering requires high processing speed over 10 frames per second.

To include these features into *ReVolver/C40*, we have devised a simplified voxel sampling method. We have also contrived a three dimensional volume memory fit for this algorithm in order to read all volume data on a ray of any direction in parallel.

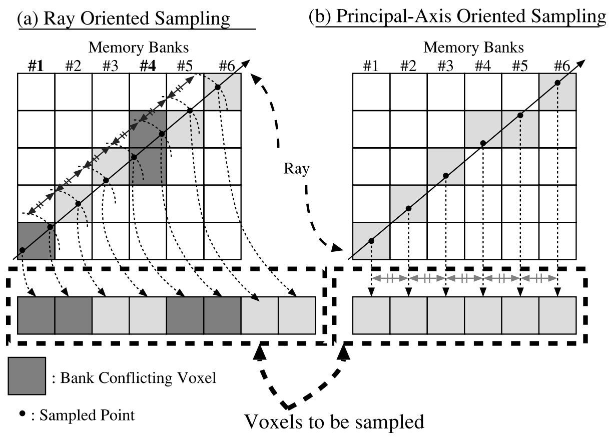


Fig. 3 Simplified sampling method.

2.2 Architectural Features of ReVolver/C40

2.2.1 Sampling Method and Parallel Volume Memory

Rendering one image (or frame) of $N \times N$ pixels from N^3 volume data requires the memory throughput proportional to N^3. Thus, to satisfy the real-time requirement, we had to develop a conflict free highly parallel multi-bank memory which can simultaneously read out all voxels along a ray of any direction. Since *ReVolver/C40* supports perspective projection method as well as parallel projection method, requirements to the memory system are much more severe than the systems which don't support perspective projection. In order to alleviate this severity, we had proposed a simplified voxel sampling method [6], [29]. It is called object-based (or slice-aligned) sampling recently [1], [2].

In this method, instead of sampling voxels on a ray at regular intervals of the ray itself (Fig. 3 (a)), we sample voxels on the ray at regular intervals of the ray's principal-axis. The principal-axis is one of x, y, or z-axis in which the projection of the ray vector to the corresponding axis is the biggest among all (Fig. 3 (b)). Therefore, if the sampling interval is equal to the unit of the volume coordinate system, the coordinates of all sampling points with respect to the principal-axis are different from each other. This makes it possible to construct a bank-conflict free parallel volume memory for both perspective and parallel projections (Fig. 4). Here, the principal-axis of each ray varies among x-, y-, and z-axis according to the view point and we want to avoid coordinate transformation in the memory to keep real-time follow-up to view point movement. So, we slice the volume space into planes perpendicular to x-, y-, and z-axis and store a set of i-th planes from Xplanes, Yplanes and Zplanes into i-th Parallel Treble Volume Memory node. As it is obvious from this memory structure, the effective memory throughput scales with the number of the memory nodes, irrespective of

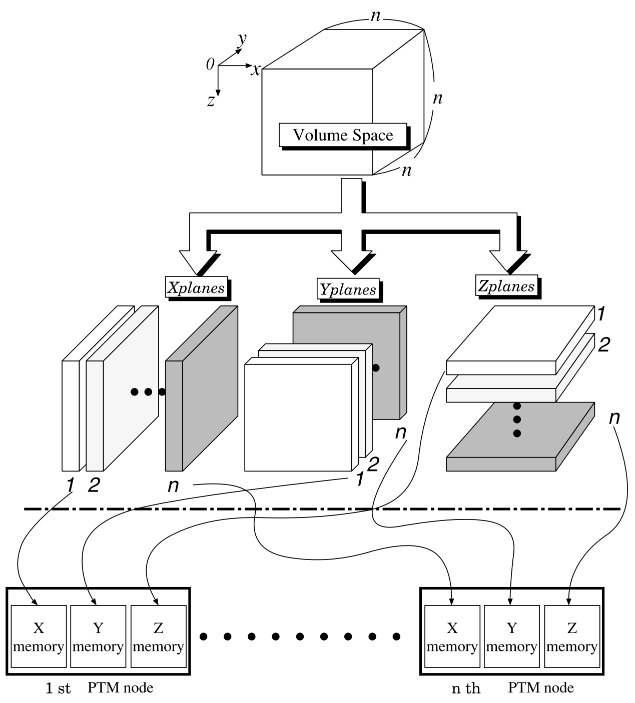
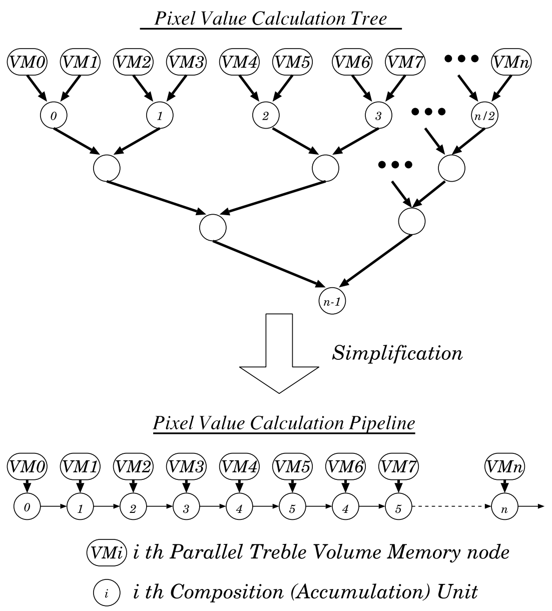


Fig. 4 Parallel treble volume memory.



Pixel Value Calculation Tree
VM0
VM1
VM2
VM3
VM4
VM5
VM6
VM7
VMn
n/2
n-1
Simplification
Pixel Value Calculation Pipeline
VMi i th Parallel Treble Volume Memory node
i i th Composition (Accumulation) Unit

Fig. 5 Composition network.

the view point and the projection method.

2.2.2 Composition Network

The next to be considered in the architectural design is how to organize the composition network in which the composition operation (Eq. 1) is performed.

Since a pixel value can be calculated by applying the composition operation recursively, as we have mentioned before, it is relatively easy to organize a N-stages linear pipeline of over operations (Eq. 2) [14], [30].

On the other hand, since this composition operation can be transformed also into a pipeline of tree reduction style [6], [29], there exists parallel volume rendering systems which adopt tree-based composition network [8], [25]. In our initial proposal [6], [29], we also used a tree-based composition network.

In *ReVolver/C40*, we organize the composition network as one directional link structure (Fig. 5)[†]. The primary advantages of this link network are good scalability and ease of implementation; the cost of wiring becomes half while the logical throughputs of both networks are the same at the output-end. Though the latency increase from $O(logN)$ to $O(N)$, it is negligible if each created image has the order of N^2 pixels. This argument is not always applicable to most parallel volume rendering systems [12], [25]. It is because they use frame-based composition to hide their large communication overhead, whereas in *ReVolver/C40* we can use pixel-based composition since *ReVolver/C40* has the low latency composition network embedded into its N-stages linear pipeline of over operations.

Table 1 Preliminary specification.

Parallel Volume Memory	
Number of Nodes (N)	128
Memory Size (total)	512MB
Memory Size (node)	4MB
Rendering Speed (perspective projection,discrete model)	
128^3 Volume on 128^2 Screen	30>FPS
256^3 Volume on 256^2 Screen	30>FPS
512^3 Volume on 512^2 Screen	10 FPS

3. Prototype Implementation

3.1 Preliminary Specification

As a prototype implementation of our parallel volume rendering architecture, we have decided the specification of *ReVolver/C40* as shown in Table 1 based on the technology available in early '90s.

3.2 Hardware Configuration of *ReVolver/C40*

Figure 6 shows the overview of the hardware configuration of *ReVolver/C40*. The system consists of a control unit and a three-stage macro pipeline as follows. In *ReVolver/C40*, we adopt TI's Digital Signal Processor (DSP) TMS320C40 [31] operating at 50 MHz since it has useful internal configuration for volume rendering calculation.

System Control Unit (SCU) This unit gives other units various data/commands, such as volume

[†]The latest version of PCB boards of *ReVolver/C40* has bi-directional ring structure [16] (see Sect. 5) but they are still waiting for parts assembly.

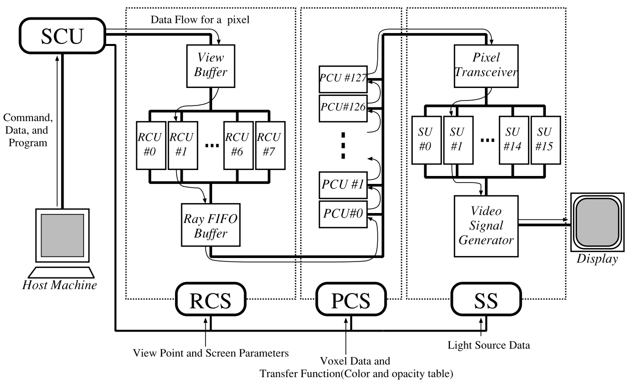


Fig. 6 Overview of the prototype system: ReVolver/C40.

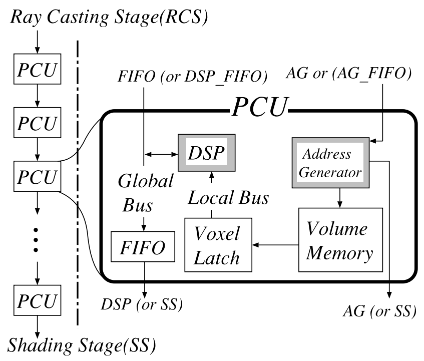


Fig. 7 Pixel value calculation unit (PCU).

data, viewing parameters (for example positions of view point and screen), and rendering mode.

Ray Casting Stage (RCS) This stage casts rays which start from the view point and penetrate into pixels on the screen, and sends them to Pixel Calculation Stage. In order to generate a ray data in every machine cycle, this stage consists of 8 Ray Casting Units (RCUs) where each RCU is configured with one DSP and small working memory. The unit of load distribution is one scanline and the scanlines are statically assigned to each RCUs in cyclic fashion. Generated ray data is stored into a FIFO memory common to the 8 PCUs. The output of the FIFO memory is directly connected to the address generator mentioned below to feed ray data directly into Pixel Value Calculation Stage.

Pixel Value Calculation Stage (PCS)
This stage organizes a 128-stages linear pipeline of over operations. We call the unit correspond to one stage of this pipeline Pixel value Calculation Unit (PCUs). Each PCU is configured with Volume Memory (VM), Address Generator (AG), *C&T* Look-Up-Table (LUT), and one DSP.

Volume Memory (VM): The VM corresponds to the PTM node in Fig. 4 and may contains upto 3 sets[†] of four 512^2 planes out of 512^3 volume data.

Address Generator (AG): The Address Generator computes Volume Memory addresses of voxels to be sampled using viewing parameters, current pixel coordinate and PCU's relative position in PCS. Since this kind of computation includes a lot of bit-based operations, we have implemented AG with FPGA (Xilinx XC4010D, 10,000 gates). All information necessary to compute the address is transferred through the direct link between neighboring AGs without any intervention by the DSP. The AG also has an important function of voxel preloading to hide the latency of the VM accesses, as follows: Voxels to be sampled are read out from the VM using the addresses generated by AG and are stored into the Voxel Latch (see Fig. 7) while the DSP is doing the over operation for the previous pixel.

C&T **Look-Up-Table (LUT)**: Before the over operation, the sampled pixel value has to be translated into color and transparency. *C&T* Look-Up-Table (LUT) is provided with each PCU to do this translation. In our *ReVolver/C40* implementation, this table is allocated into the embedded memory located at local bus inside the DSP so that it could be accessed in one cycle.

Digital Singal Processor (DSP): As we have metioned before, we have adopted TI's DSP TMS320C40, we call it C40 in the rest of this paper, as our processing unit. It is because, 1) C40 has two physically separated 32-bit buses, called *Global Bus* and *Local Bus*, which support simultaneous access to the two different external/internal devices if they are located at different busses; 2) C40 has two embedded SRAMs which can be used as *C&T* LUT and storage for program memory and runtime environments; 3) C40 has six embedded bi-directional communication ports with 20 MB/s peak transfer speed for each which can be used at initial program loading and debugging in *ReVolver/C40*.

A FIFO buffer (32 bits × 64 entries) is placed on Global Bus as shown in Fig. 7. The output of this FIFO buffer is directly connected to the Global Bus of the C40 in the next PCU. The maximum throughput of this FIFO is 100 MB/s. During volume rendering operation, the C40 picks up intermediate color and transparency data from the FIFO of its previous PCU, then it performs the over operations using preloaded voxels and stores the result into its local FIFO.

Shading Stage (SS) This stage consists of a 4 × 4 two-dimensional array of Shading Units (SUs). The SU consists of one C40 and two kinds of double-buffered memories: Receive Buffer (RB) and frame memory. The pixel value from the pixel calculation stage is distributed among 16 RBs ac-

[†]These 3sets correspond to X-memory, Y-memory, and Z-memory in Fig. 4, respectively

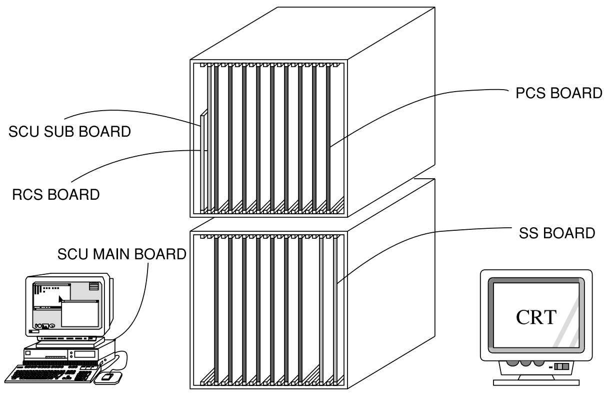


Fig. 8 An exterior of the prototype system: ReVolver/C40.

cording to the pixel's coordinate. When all pixels belonged to an image get ready in RBs, this image is shaded using depth gradient shading, by default, if the user want to have better visual effect. The final image is stored into the double-buffered video memories distributed among 16 C40s. A video signal generation circuit collects sub-images generated by each C40s, converts them into analog video signal and sends it to a Disply.

3.3 PCB-Level System Configuration

We have designed four kinds of printed circuit board: SCU boards, RCS boards, PCS board and SS board. Each board except for SCU board is VME standard triple-height board and each of these contains 8 C40s. So, 16 PCS boards for the PCS and 2 SS boards for the SS have to be mounted onto *ReVolver/C40* prototype. Figure 8 shows an exterior of the prototype system.

3.4 Overview of Rendering Pipeline in *ReVolver/C40*

Figure 9 shows how the rendering pipeline of *ReVolver/C40* works. From the highest level, there is the frame-based pipeline where image generation, shading, and image output organize a 3-stages pipeline. The last two stages are implemented by the well-known double buffering of the frame memory in the SS. The second level is pixel-based pipeline that exists in the image generation stage. This pipeline consists of the ray casting stage and the N-stages linear pipeline of over operations. Though the RCS itself utilizes scanline-based parallelism, it is not visible to the image generation pipeline as long as there exists ray information in RCS's output FIFO.

3.5 Development of *ReVolver/C40*

After 5 years from the initial proposal of the ReVolver architecture [6], [29], a part of RCS board with 3 RCUs, one PCS board with 8 PCUs, and one SS board with

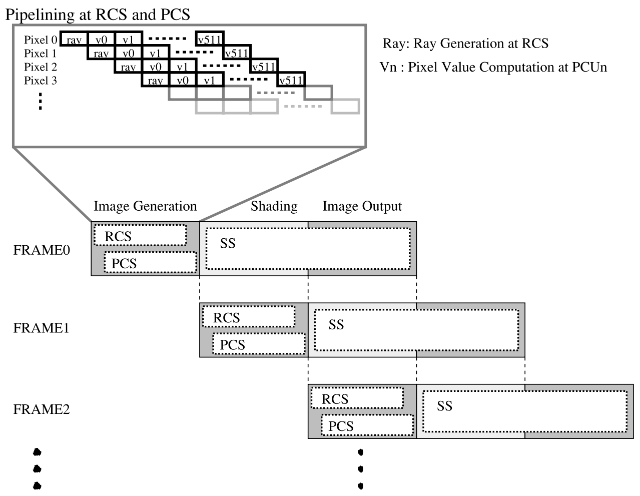


Fig. 9 Rendering pipeline.

8 SUs became operational in late 1998. Using these parts, we have constructed a evaluation subsystem which we call *ReVolver/C40-mini*. And then, in early 1999, we have reorganized it into a little bit larger system which we call *ReVolver/C40-demo*. *ReVolver/C40-demo* has been configured with four PCS boards (32 PCUs) and one SS board (8 SUs) but it has neither voxel preload function of PCS boards[†] nor RCS boards. In *Revolver/C40-demo*, all the communication between neighboring DSPs is performed through the embedded communication ports of TMS320C40 and host computer (PentiumIII 800 MHz) also functions as RCS. Due to the memory size restriction, *ReVolver/C40-demo* can render upto 256^3 volume at the price of declining frame rate.

4. Performance Evaluation

We have made detailed performance evaluation based on *ReVolver/C40-mini* and the results are summarized in [22], [23]. In this section, we have focused on the evaluation of PCS since the performance of PCS is the most dominant part in the overall performance. In the following discussion, we assume the image size of 256×256 unless noticed especially. We also assume that each PCUs is responsible for 3-sets (Xplanes, Yplanes, and Zplanes) of P-planes (256×256 in size) out of 256^3 volume data where P is the number of planes per PCU and P may vary among 1,2,4 and 8.

4.1 Rendering Performance

The voxel preloading is one of the most important is-

[†]Due to some miss implementaions in the circuits for interface between two PCS boards, we have bypassed the circuits in *ReVolver/C40-demo* configuration. This bypass disables the function of AG and enables the communication ports of C40s.

Table 2 Performance of AG in ReVolver/C40-mini.

# of planes per PCU (P)	Voxel Preload Time (μs)	Frame Rate (FPS)
1	2.52	6.05
2	2.80	5.45
4	3.36	4.54

Table 3 Effect of AG in ReVolver/C40-mini.

# of planes	Rendering Time (μs/pixel)/Frame Rate (FPS)	
per PCU (P)	with AG	without AG
1	3.28/4.65	8.31/1.84
2	4.80/3.18	11.96/1.28
4	7.24/2.11	19.11/0.80

sues in the implementation, we first examine the performance of the address generator (AG). Table 2 shows the time spent for preloading P voxels along a ray. From this table, we could figure out that the AG by itself has the performance of 6 FPS (Frame Per Second) if P = 1[†].

The next thing to examine is the time spent for over operation which determines the pipeline pitch of the rendering pipeline. Table 3 shows rendering time per pixel and frame rate for 256×256 image. In order to examine the effects of AG, this table contains the result of two different configurations of PCU: with and without AG. In case of PCU without AG, not only the results of over operation but also ray-data have to be transferred between C40s and also C40 has to substitute for AG. This table confirms the great impact of AG on the rendering performance. Indeed, the PCU with AG performs roughly 2.5 times faster than the PCU without AG. However, comparing with the results in Table 2, the rendering performance has declined. It is due to the communication overhead of C40. Because of the implementation dependent critical timing of FIFO access and the restriction of C40 on the consecutive write accesses, roughly 2 μs are spent for communication between C40 and FIFO memories. As a result, effective throughput decreased to only 20 MB/s which is comparable to the embedded communication port of C40.

In the above discussion we assumed *ReVolver/C40-mini*. From now, we continue our discussion based on the performance of *ReVolver/C40-demo*. Table 4 shows the rendering performance of *ReVolver/C40-demo*. In this table, the performance of simulated configuration that assumes a perfect AG (which always supplies voxel data in one cycle) is shown as well as the normal configuration without AG. The normal configuration without AG has better performance in *ReVolver/C40-demo* than in *ReVolver/C40-mini*. It is due to the very high internal I/O throughput (100 MB/s) of communication ports in C40. When we assume the perfect AG, the performance reaches to 11.6 FPS and seems to be saturated when $P < 4$. From this result, we can consider the effective throughput of communica-

Table 4 Rendering speed in ReVolver/C40-demo.

# of planes	Rendering Time (s/frame)/Frame Rate (FPS)	
per PCU	without AG (real system)	with perfect AG (simulation)
1	0.31/3.2	0.086/11.6
2	0.56/1.8	0.086/11.6
4	1.0 /1.0	0.094/10.7
8	1.8 /0.55	0.26/3.8

tion port between two C40s in *ReVolver/C40-demo* to be 12.2 MB/s. According to the hand-coded assembly program which assume the perfect AG, the peak performance of *ReVolver/C40-demo* could achieve about 20 FPS for P=1 if there is a sufficient bandwidth between two C40s.

In the above discussions, we always assume the image size of 256^2. Since the image size does not affect the rendering time for a pixel, the overall rendering performance is inversely proportional to the number of rays in the image. On the other hand, if there is an enough capacity to store volume data in each PCUs, the overall rendering performance decrease with the number of planes (P) per PCU but it is much better than 1/P because the increase of the computational time for over operation reduces the communication overhead relative to the computational time.

4.2 Volume Data Initialization

This section discusses the issue of volume data initialization. In advance to any rendering operation, volume data should be stored into volume memories of PCUs. In *ReVolver/C40-demo*, volume data stored in the hard disk of host computer is downloaded sequentially as a set of four consecutive voxels. This is done through the communication link between SCU and PCU. Every time a set of four voxels is arrived at a PCU, the PCU performs the following operations: 1) It examines if any of them should be stored inside its volume memory according to the voxel's (x,y,z)-coordinate and relative location of the PCU. 2) It performs memory accesses if necessary. 3) It forwards them to the next PCU. The parallel trebled volume memory is initialized in this way. During the initialization, $9/4N^3$ memory accesses are performed in PCS. It takes about 500 ns for each of this memory accesses including miscellaneous overhead in PCU. Indeed, for 256^3 volume data, we have observed that it takes 19.8 sec on the host machine including disk access time. Recent graphics cards for volume rendering [2], [9] also takes similar latency,

[†]We also have designed another version of the AG where 1) ray-data read-out from previous FIFO, 2) Address Calculation, 3) Volume Memory Access, and 4) ray-data store into its FIFO are performed in pipelined fashion. The pipelined version of AG could achieve 29.3 FPS for P=1, but it was little bit unstable. So we have used the non-pipelined version of AG in this evaluation.

though it is not negligible if the user wants to change the volume data frequently. One solution to this problem is to have multiple paths to download the data [3], [11].

5. Considerations

5.1 Advances in Technology

As we mentioned before, the design of *ReVolver/C40* is based on the technology roughly 10 years ago. So we are going to review *ReVolver/C40* from the view point of technology advances.

1. Processor Technology

We adopt TI's DSP TMS320C40 as our processing element. The C40's features which benefit *ReVolver/C40* are 1) hardware support for inverse and inverse square root which are very useful in the RCS, 2) parallel instructions which realize multiply-and-add like operations useful in all stages, 3) repeat block operation which uses single cycle branch that is very useful in PCS, for P=1 in particular, 4) embedded communication ports for parallel processing, 5) embedded small local SRAM block, and so forth.

Indeed, we didn't have any plan to use the embedded communication during rendering operation at the time of initial design, *ReVolver/C40-demo* exactly benefits from the low latency read/write operations to the communication ports. If they have much more throughput, like Alpha 21364 [26], it would be very useful for fine grain communication like we need in pixel-based rendering pipeline.

Embedded memory of small-capacity, but fast, is quite useful in the applications which does not necessarily require high precision calculation. It is because some function calls can be replaced by a single cycle memory lookup, like *C&T* LUT in PCU.

A drawback in C40 is that it does not have multimedia extension instruction like quadruplet instructions which can be used to compute color value. If C40 would have such an instruction, performance of over operation in PCS would be tripled. Since the clock frequency of the latest processor is more than 50 times faster than C40, it is possible to achieve the 150 times total speedup if we are able to use all these available technologies.

2. Memory Technology

The volume memory in PCU is 4 MB DRAM of 32 bit width. It's RAS latency is 160 ns and has throughput of 66.6 MB/s. Now, we can purchase 512 MB DDR-SDRAM module of 64-bits width, 2666 MB/s, at the similar price. This memory is large enough for a volume data of 512^3 and has an acceptable bandwidth for a screen of 512^2. However, the ras access latency for random accesses is still far beneath the required speed. So, still there is need for parallel volume memory organization even with the latest memory technology in order to realize view-independent real-time volume rendering by using perspective projection. However, if we allow view-restricted or view-dependent rendering, there are some possibilities to reduce the randomness in memory access patterns: cache blocking, or tiling, technique is one of such possibility.

3. FPGA and Dynamic Reconfigurable Tech.

The most powerful FPGA, we used in *ReVolver/C40*, is Xilinx XC4010D which has 10,000 equivalent gates, operating at 25 MHz in our system. It was a biggest FPGA reasonably available at that time. Since it was impossible to implement both AG and circuits for over operation in one chip, we had decided to employ DSP for over operation. Now, we have various choices of versatile FPGAs. Some FPGAs have embedded processor (or DSP)-cores running at 300 MHz and up, some have embedded synchronous memory block accessible at 150 MHz and up, and some have more than 10 million gates. If we can use such an FPGA device, we can implement a few tens of PCU including AG in one FPGA with a few channels of DDR-SDRAM Memories. Indeed, we are currently designing such a system in our laboratory [18]. In addition to that, if we can utilize the latest technique on dynamic reconfigurability [7], it is possible to consider a graphics processor which can dynamically adapt itself to meet the requirement of rendering conditions, like projection modes, volume data structures [15], lighting and shading parameters and so on.

5.2 Volume Rendering Algorithm

The fundamental algorithm, we used in *ReVolver/C40*, is the ray casting algorithm. Due to its simplicity and its image quality, almost all of the hardware accelerated volume rendering systems [5], [8], [9], [14], [21], [25] adopt this algorithm. But none of these system support parallel volume rendering by using perspective projection, except *ReVolver/C40*.

Recent advances in PC-based commodity graphics cards which support *α-blending* for rendering non-opaque polygon realize the texture-based approach to the volume rendering [2]. If a whole volume data can be stored inside video memory of such a graphics card, it can perform volume rendering with acceptable frame rate. However, such a commodity graphics card does not have enough video memory to store large volume data, so it significantly slows down for large volume data due to the slicing effect. Some systems have proposed to alleviate this effect by parallel processing [17], [20].

There are some other algorithms [10] for parallel volume rendering, like splatting and cell projection [13]. These algorithms have a capability of volume rendering with unstructured grid volume data, though they are difficult to implement in hardware due to some kind of complicated list processing to keep the order of over operation to intermediate sub-images.

5.3 Human Visual Perception

So far, we have intended not to consider the approximation techniques in image generation because we think that it is not acceptable in medical applications which are the biggest market of volume rendering. However, there are a lot of possibilities to increase rendering speed if we take the human visual perception into account. Here we consider three types of approximation.

1. Early Ray Termination (ERT) [19], [24]

During the composition computaion, if the accumulated transparency becomes very low, say ϵ, the further accumulations do not affect the image too much. The early ray termination is a technique which cuts off the accumulation for the rest of sampling points on a ray if the accumulated transparency $< \epsilon$. This technique is known to be very useful. But it has no effect in current *ReVolver/C40* because, even if ERT applies, intermediate results have to be propagated to the end of rendering pipeline. If *ReVolver/C40* is configured with a feedback loop from the tail of rendering pipeline to the top, say from the last PCU to the first PCU, and we apply cyclic data distribution, we could benefit from the ERT [16].

2. Level of Detail

If the view point moves very quickly, it is impossible to recognize the detail of each image. So it is better to generate less detailed image to keep track of the view point movement in real time, rather than to generate full detailed image but less frame rate. Since this technique is easy to implement in *ReVolver/C40*, we have already examined in some cases and obtained very nice results.

3. Pseudo-Perspective Projection

If the screen size S gets wider, parallel projection becomes no more acceptable. However, real-time rendering which uses perspective projection is fairly expensive to support, in general. So, we have been investigating a technique which we call *Pseudo-Perspective Projection*. Let's consider that the screen S has been divided into sub-screens $s_{ij}(i,j : [0 : N-1])$ of equal sizes. For a sub-screen s_{ij}, calculate a ray R_{ij} which goes through the center of s_{ij}. In *Pseudo-Perspective Projection*, for the pixels inside $s_{i,j}$, we cast rays parallel to $R_{i,j}$. If N=1 it is equivalent to the parallel projection. As N gets bigger, generated image gets closer to the image using perspective projection. Because of the increased ray coherency inside s_{ij} in *Pseudo-Perspective Projection* mode, it can increase the regularity in memory access patterns. Though there exists trade off between rendering time and image quality, we believe it is a feasible approach to the real-time rendering using perspective projection.

6. Conclusion

We have described the architecture of *ReVolver/C40* a scalable parallel machine for volume rendering and its prototype implementation. The most important feature of *ReVolver/C40* is view-independent real time rendering of translucent 3D object using perspective projection. In order to realize this feature, we have proposed a parallel volume memory architecture based on the principal axis oriented sampling method and parallel treble volume memory. According to the evaluation of the prototype systems, *ReVolver/C40* with 32 parallel volume memory is estimated to achieve more than 10 frames per second for 256^3 volume data on 256^2 screen. We have also confirmed the scalability of this system though we could not describe its detail in this paper.

Currently, we are continuing the work on the parallel volume rendering for large-data visualization [27] and interactive visualization of real-time simulation data [3], [4], [17], [18]. As a part of these works, we have been developing a new graphics card for parallel volume rendering [18]. In order to deal with the time-varying volume data, we are also developing a new rendering algorithm based-on the work in [28] which does not require any replication of volume data.

Acknowledgment

Development of *ReVolver/C40* can not be accomplished without contribution of the following students who graduated from Tomita Laboratory; H. Akashi, T. Kitasuga, T. Kuroda, M. Susukita, Y. Tsushima, T. Ogino, X. Jin, A. Nakayama, Y. Kiwata, T. Honda, I. Kinaka, M. Fujiwara, N. Yoshitani, D. Shigeta, F. Harase and S. Yamauchi. We wish to express our appreciation for thier great work. We also thank to M. Ikumo, M. Takayama, Y. Maruyama, S. Nakata, and the other members of Tomita Laboratory.

A part of this research was supported by the Grant-in-Aid for Scientific Research (A)(1)#06508001, (B)(2)#10558045, (B)(2)#13480083 from Japan Society for the Promotion of Science and also by the Grant-in-Aid for Scientific Research on Priority Areas (C)#13224050 from the Ministry of Education, Culture, Sports, Science, and Technology of Japan.

References

[1] B. Lichtenbelt, R. Crane, and S. Naqvi, Introduction to volume rendering, Prentice-Hall, Upper Saddle River, 1998.
[2] C. Rezk-Salama, et al., "Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization," Proc. SIGGRAPH/EUROGRAPHICS Graphics Hardware Workshop 2000, pp.109–118, 2000.
[3] F. Harase, et al., "Real-time volume visualization on ReVolver/C40," IPSJ SIG Notes, 2002-ARC-148, pp.7–12, May 2002.

[4] F. Harase, et al., "A hardware assist for scientific visualization," Proc. 30th Symp. on Visualization, pp.119–122, July 2002.

[5] G. Knittel and W. Strasser, "VIZARD — Visualization accelerator for real-time display," Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware, pp.139–146, 1997.

[6] H. Akashi, A parallel computer architecture for volume rendering, Master Thesis, Dep. of Computer Science, Kyoto University, Feb. 1993.

[7] H. Amano, "A dynamically adaptive hardware using a multi-context reconfigurable processor," IPSJ SIG Notes, 2002-ARC-150, pp.59–64, Nov. 2002.

[8] H. Pfister, et al., "Cube-3: A real-tme architecture for high-resolution volume visualization," Proc. 1994 Symp. on Volume Visualization, pp.75–82, 1994.

[9] H. Pfister, et al., "The VolumePro real-time ray-casting system," SIGGRAPH '99 Conference Proceedings, pp.251–260, 1999.

[10] K. Brodlie and J. Wood, "Recent advances in volume visualization," EUROGRAPHICS Computer Graphics Forum, vol.20, no.2, pp.125–148, June 2001.

[11] K. Itakura, et al., "Parallel visualization for parallel data stream," Proc. JSPP2001, pp.189–196, June 2001.

[12] K.L. Ma, et al., "Parallel volume rendering using binary-swap image composition," IEEE Trans. Computer Graphics and Applications, vol.14, no.4, pp.59–68, 1994.

[13] K.L. Ma, et al., "A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data," Proc. Parallel Rendering Symposium (PRS'97), pp.95–104, 1997.

[14] L. Moll, et al., "Sepia: Scalabel 3D compositing using PCI Pammete," Proc. 7th IEEE Symp. on Field-Programmable Custom Computing Machines, April 1999.

[15] M. Fujihara, et al., "Realtime visualization method for hierarchical grid volume data," IPSJ SIG Notes, 98-ARC-128, pp.7–12, March 1998.

[16] M. Ikumo, et al., "The effects of early ray termination in parallel volume rendering with cyclic data distribution," Proc. Forum on Information Technology, vol.3, no.J-96, pp.233–234, Sept. 2002.

[17] M. Ikumo, et al., "Parallel volume rendering environment for interactive real-time simulation," Proc. IPSJ Kansai-branch Forum, pp.121–124, Nov. 2002.

[18] M. Ikumo, Development of a graphics card VisA for real-time visualization of time-varying volume data, Master Thesis, Grad. School of Informatics, Kyoto University, Feb. 2003.

[19] M. Levoy, "Efficient ray tracing of volume data," ACM Trans. Graphics, vol.9, no.3, pp.245–261, July 1990.

[20] M. Magallon, et al., "Parallel volume rendering using PC graphics hardware," Proc. Pacific Graphics, 2001.

[21] M. Meissner, et al., "VIZARD II: A reconfigurable interactive volume rendering systems," Proc. Graphics Hardware, pp.137–146, Sept. 2002.

[22] N. Yoshitani, et al., "Performance evaluation of parallel volume rendering machine ReVolver/C40," IPSJ SIG Notes, 99-ARC-132, pp.79–84, 1999.

[23] N. Yoshitani, Implementation and performance evaluation of parallel volume rendering machine ReVolver/C40, Master Thesis, Dep of Computer Science, Kyoto University, Feb. 1999.

[24] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," Proc. SIGGRAPH'94, pp.451–458, July 1994.

[25] S. Muraki, et al., "VG cluster: Large scale visual computing system for volumetric simulations," SC2000 Research Gems (Poster), Nov. 2000.

[26] S. Shubhendu, et al., "The Alpha 21364 network architecture," IEEE Micro, pp.26–35, Jan. 2002

[27] S. Yamauchi, et al., "Active volume rendering with simulation steering," IEICE Technical Report, CPSY2001-35, July 2001.

[28] X.D. Jin, et al., "Super-high speed volume rendering architecture based on pixel parallelism with the restrained visual angle," J. Inf. Process. Soc. Jpn., vol.38, no.9, pp.1668–1680, 1997.

[29] Y. Tsushima, et al., "A parallel computer architecture for volume rendering: ReVolver," Trans. Inf. Process. Soc. Jpn., vol.36, no.7, pp.1709–1718, 1995.

[30] Y. Tsushima, et al., "The parallel computer for volume rendering —ReVolver/C40," Proc. Joint Symp. on Parallel Processing 1995, pp.11–18, 1995.

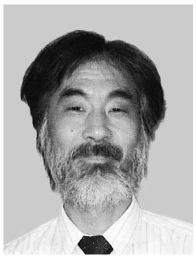
[31] Texas Instruments: TMS320C4x User's Guide, 1993.

Shin-ichiro Mori was born in 1963. He received his B.E. in electronics from Kumamoto Univ. in 1987 and M.E. and Ph.D. in computer science from Kyushu Univ. in 1989 and 1995, respectively. From 1992 to 1995, he was a research associate in the Faculty of Engineering, Kyoto University. Since 1995, he has been an associate professor in the Department of Computer Science, Kyoto University. His research interests include computer architecture, parallel processing and visualization. He is a member of IEEE, ACM, EUROGRAPHICS and IPSJ.

Tomoaki Tsumura was born in 1973. He received his M.E. degree from Kyoto Univ. in 1998. Since 2001 he has been an research associate in Graduate School of Economics, Kyoto Univ. His research interests include computer architecture for brain-like computing and parallel processing. He is a member of IPSJ and JSAI.

Masahiro Goshima was born in 1968. He received his M.E. degree from Kyoto Univ. in 1994. He was a research fellow of the Japan Society for the Promotion of Science from 1994. Since 1996 he has been in Kyoto Univ. as an assistant professor. He has been engaging in the research area of high-performance computing systems. He received IPSJ Yamashita SIG research award and IPSJ best paper award in 2001 and 2002, respectively. He is a member of IPSJ and IEEE.

Yasuhiko Nakashima was born in 1963. He received the B.E., M.E. and Ph.D. degree in Computers Engineering from Kyoto University, Japan in 1986, 1988, and 1998, respectively. He was a computer architect at Computer and System Architecture Department, FUJITSU Limited in 1988-1999. Since 1999 he has been an Associate Professor in Graduate School of Economics, Kyoto University. His research interests include processor architecture, emulation, CMOS circuit design, and evolutionary computation. He is a member of IEEE CS, ACM and IPSJ.



Hiroshi Nakashima was born in 1956. He received his B.E., M.E. and Ph.D. degree in Computer Science from Kyoto Univ. in 1979, 1981 and 1991 respectively. He had been in Mitsubishi Electric Corporation since 1981 until 1992 and had deeply involved in the Fifth Generation Computer Systems Project as the chief architect of inference machines. Then he joined Faculty of Engineering, Kyoto Univ. as an associated professor. Since 1997, he has been in Faculty of Engineering, Toyohashi Univ. of Tech. as a professor. His research interests include parallel/distributed processing. He received Motooka Memorial Award and Sakai Memorial Award. He is a member of IEEE-CS, ACM, IPSJ, ALP and TUG. He has been the Editor-in-Chief of IPSJ Trans. Advanced Computing Systems since 2002.

Shinji Tomita was born in 1945. He received the B.E., M.E. and Ph.D. degree in Electronics from Kyoto University in 1968, 1970, and 1973, respectively. He was a research associate from 1973 to 1978 and an associate professor from 1978 to 1986 both in the Department of Computer Science, Kyoto University. From 1986 to 1991, he was a professor in the Department of Information Systems, Kyushu University. From 1991 to 1998, he was a professor in the Faculty of Engineering, Kyoto University. Since 1998, he has been a professor in the Graduate School of Informatics, Kyoto University. His current interests include computer architecture and parallel processing. He is a member of IEEE, ACM and IPSJ. He was a board member of IPSJ directors in 1995, 1996, 1998 and 1999. He is a fellow of IPSJ.