PAPER

A Parallel Pipeline Convolution for Perspective Projection in Real-Time Volume Rendering

Masato OGATA[†], *Member*, **Takahide OHKAMI**[††], **Hanspeter Pfister**[††], **Hugh C. Lauer**[††*], *Nonmembers*, *and* **Yasunori DOHI**[†††], *Member*

SUMMARY This paper proposes a convolution with systolic array sturucture for perspective projection in real-time volume graphics based on the shear-warp method. In the original method, the further the ray proceeds, the more voxels are required for the calculation of convolution. The increase of required voxels makes difficult to implement the method in VLSI oriented architecture. 1) We use several sets of resolution of voxels associated with depth, in order that convolution can be done with constant number of voxels independent of depth. 2) We implement 3D convolution by three serial 1D convolutions along X, Y and Z axes, which reduces the calculation steps from M^3 to $3M$, where the convolution is calculated for M^3 area. For V^3 voxels dataset, the number of pipelines for rays is V^2 and their pipeline stage is $3M$. If the hardware of a single pipeline has the capability of calculating V rays, each of the implemented pipelines is assigned to V theoretical pipelines (for V^2 rays). In actual implementation, a number of hardware pipelines should be much smaller than the V theoretical pipelines. We fold the theoretical pipelines and reduce them to the certain number of hardware pipelines. Regarding this folding, we show the relation between folding process and its necessary time delay. The architecture can generate image of 256^3 voxles dataset($V = 256$) at 30Hz with 4 pipelines. In addition, the architecture can be extended easily for $512^3 (V = 512)$ and $1024^3 (V = 1024)$ dataset with 32 pipelines and 256 pipelines respectively.

key words: Volume Graphics, Volume Rendering, Graphics Architecture, Real-Time, Perspective Projection, Scientific Visualization, Computer Graphics, Systolic Array.

1. Introduction

Fast direct volume rendering systems are in high demands. This is due to an increasing number of scientific data generated by a variety of computer simulations, medical data obtained by MRI and CT scanners, and geological, oceanographic, and meteorological data collected from various sensors. One of the notable characteristics shared by these volume data is the sheer amount of data elements to be processed in rendering. This requires a huge amount of computing resources for animated visualization, which is essential to observe some physical phenomena. Another characteristic of the data is that they can not be represented by surfaces as in the conventional polygon-based graphics; the volume data may include complicated internal structures and shapeless features.

Although there are many algorithms for volume rendering, the ray-casting algorithm is the precisest algorithm based on a physical model. It casts rays from the center of projection into a volume to calculate each pixel value on a screen. Let suppose $I(a, b)$ be a intensity from a ray through the volume between points a and b, $s(r)$ be a light added per unit length at distant r along the ray, and $\alpha(r)$ be an absorption coefficient corresponding to the attenuation of the light per unit length. The following Equation (1) calculates the effects of the light, and it has been used as volume rendering equation [1] [5] [10].

$$I(a,b) = \int_a^b s(r) e^{-\int_a^r a(t)dt} dr \qquad (1)$$

As a simplified implementation of Equation (1), each sample is computed from voxels surrounding the sample point by interpolation, then being accumulated along the ray to calculate the intensity of the pixel. Each resampling operation is relatively simple, but the total number of resampling operations is very large, and the time spent for the operations is dominant in the rendering time. Because of this, a ray-casting-based volume rendering system could be considered a resampling machine. Therefore in real-time implementation[†], how to organize a parallel processing for the resampling is one of major issues.

In this paper, we propose a new VLSI oriented architecture of resampling for both parallel and perspective projections in real-time volume rendering based on the shear-warp method[6][5]. In particular, we show implementation of 3D convolution for resampling with systolic array structure.

2. Previous works

From architectural stand point, the ray-casting algorithm is categorized into two schemes for implementation: sample-order and voxel-order scheme. As the term implied, the sample-order scheme uses sample point as a starting point of processing then get memory address of voxel. On the other hand, the voxel-order

Manuscript received June 21, 1999.
Manuscript revised

[†]Mitsubishi Precision Co, Ltd., 345 Kamimachiya, Kamakura, Japan
[††]A Mitsubishi Electric Research Laboratory, 201 Broadway Cambridge, M.A, U.S.A
[†††]Division of Elec. & Comp. Eng., Yokohama National University,156 Tokiwadai, Hodogaya, Yokohama, Japan

[†]generating more than 30 images in a second

scheme uses memory address of voxel directly as starting point for processing. Each scheme has advantages and disadvantages for structuring real-time volume rendering architectures.

2.1 Sample-order scheme

The sample-order scheme is straightforward implementation of ray-casting algorithm. The ray-parallel method in the scheme parallelizes rendering operations on a ray basis[4]. The number of rays to cast is determined by the screen size and resolution. It can use some optimization techniques available for this scheme. Early ray termination and coherence encoding are two examples to reduce the number of memory accesses [7]. The major disadvantage of this scheme is that one voxel is simultaneously accessed by multiple rays for resampling, which increases the total number of memory accesses time. In addition, it requests complicated memory address calculation from the sample point. These are disadvantages for hardware implementations. VIRIM[2] is one of typical real-time parallel rendering system in this scheme. It is a parallel rendering system for both parallel and perspective projections, but not organized in a complete parallel-pipeline structure for VLSI implementation. In addition, it can not generate images for large data more than 64^3 grids in real-time.

2.2 Voxel-order scheme

The voxel-order scheme on the other hand, uses voxel address directly so that memory address calculation is simple and suitable for VLSI implementation. The major advantages of this scheme are the reduction of the number of memory accesses, and the fixed resampling structure. The voxel-parallel method[11][10] in the scheme reads voxeles once and retains them until all the samples requiring the voxels. The disadvantage is that there are no major optimization techniques available for this scheme. The optimization techniques for the ray-parallel method can not be used for this voxel-order scheme, because they break the fixed resampling structure. Cube-4[11] and its VLSI implementation: EM-Cube[9](product name is VolumePro[12]) are rendering system in this scheme. The architectures are organized in systolic array structure. However, they support only parallel projections. The memory access for parallel projections are very regular and amenable to parallel pipelined processing; a rendering pipeline is directly corresponded to a ray. The memory access for perspective projections, however, are not regular due to diverging perspective rays. This processing variability adversely affects the parallel-pipeline structure for perspective projections and it has been a major obstacle for hardware implementation in this scheme[12].

The shear-warp[6][5] is another method in voxel-order scheme. It has been used as software oriented method for both parallel and perspective projections. Although it has the significant advantage, i.e. both projections can be treated with unified way, hardware implementations of convolution for variavle size is hard due to diversing rays so that it has been used for software implementations.

3. Issues of Perspective Projections

The shear-warp method produces a distorted base plane† image with a shearing matrix H[6][5] as an intermediate image, then the image is warped to produce a correct image on a screen. Fig. 1 shows the perspective rays parallelized at the base plane by the shearing matrix H. They are parallel to the principal viewing axis and perpendicular to the base plane. The shearing transformation shears and progressively scales the voxel grid as shown in Fig. 1, where the voxel grid becomes finer as the distance from the base plane increases.

Starting from a position in the first slice, i.e. base plane, a parallelized perspective ray proceeds in the progressively scaled grid to compute a sample at each slice. The computed samples are accumulated to produce the final pixel value in the base plane image. A resampling operation is a convolution over the voxels in a resampling area.

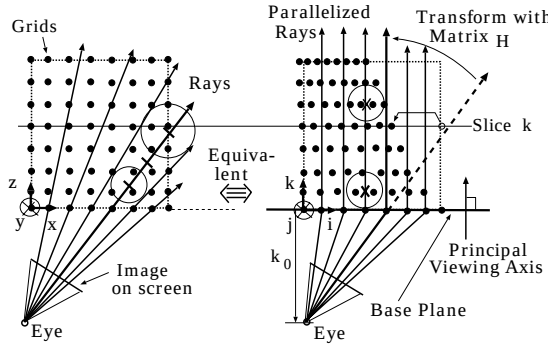


Fig. 1 Shearing and scaling in perspective projections.

There are two important issues for a parallel pipelined implementation, i.e. systolic array, of the method: **(1) how to bound a number of voxles to be convoluted** and **(2) how to organize a convolution**. A number of voxels to be convoluted in one dimension, M, is computed by:

$$M = 1 + k/k_0, \tag{2}$$

where k †† is the slice number or the distance from the base plane and k_0 the distance between the eye position (the center of projection) and the base plane as

† A plane the most perpendicular to the viewing vector, which includes the front face of the volume.

†† We use notation (x, y, z) for the dataset description and (i, j, k) for that of transformed as shown in Fig. 1

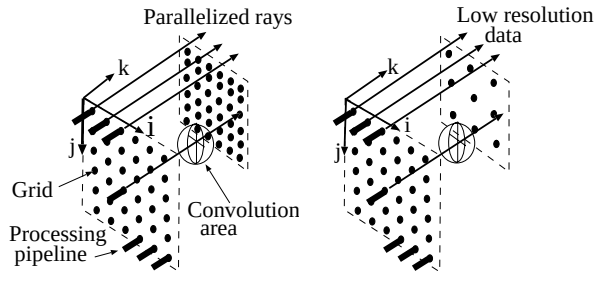
shown in the Fig. 1. It can be arbitrarily large with large values of k and/or small values of k_0. Since the hardware implementation can only use a fixed amount of resources for convolution, it has to ignore the voxels outside the convolution area, producing a low-quality or aliased base plane image.

The convolution structure is another issue. The underlying architecture pairs a memory module and a rendering pipeline so that each pipeline can take one voxel along with neighboring voxels through sideway communications and produces one sample computed from the voxels.In this systolic array structure, the number of inputs (voxels) is equal to the number of outputs (computed samples). This holds for parallel projections, but not for perspective projections.

These issues make hard to implement the parallel-pipeline structure for perspective projections and has been a major obstacle for hardware implementation in this method.

4. Basic Ideas

The parallel projection is amenable to parallel processing by assigning a set of rendering pipelines to a set of rays with one to one corresponding as illustrated in Fig. 2(a). In order to organaize rarallel pipelined convolution, the number of voxels to be convoluted has to be limited. It is reasonable to use low resolution data for distant samples so that selecting appropriate resolution data from multiple resolution dataset, prepared in advance, limits the number of voxels to be convoluted within a limit as shown in Fig. 2(b).



(a) Sheared without multi-resolution.

(b) Sheared with multi-resolution.

Fig. 2 Bounding the number of voxels for convolution with multi-resolution dataset.

4.1 Sample-Parallel Architecture

We shift a focus from voxels to samples and reorganize the rendering architecture as a *sample-parallel* architecture to provide a unified parallel-pipeline structure for both parallel and perspective projections.

As shown in Fig. 3, the proposed architecture places a resampling module between the voxel memory and rendering pipelines. This module is specialized for resampling with a variable number of voxels in the resampling area. All the complicated communication and its control between other pipelines are moved from the rendering pipelines to this resampling module in order to organize the rendering pipelines in a fixed parallel-pipeline structure.

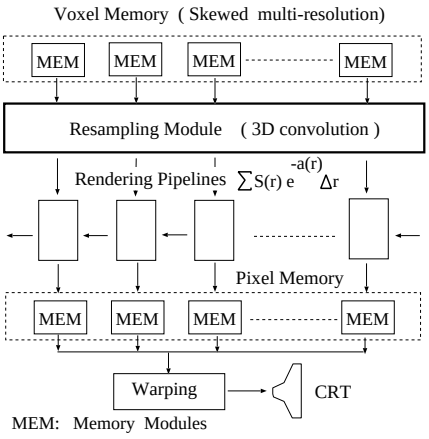


Fig. 3 Proposed sample-parallel architecture.

4.2 Transformation

To estimate a sample, we use four coordinate systems and transformations between them. The transformed coordinates are used to calculate weights for convolution.

The four coordinate systems are the normalized, shear-shrink, scale-up, and compositing coordinate systems, as illustrated in Fig. 4. The normalized coordinate system defines the original voxel grid at slice 0. It is equivalent to the base plane coordinate system. The shear-shrink coordinate system defines the voxel grid sheared and scaled by a shear-shrink matrix. The scale-up coordinate system defines a grid scaled up by a scaling factor D. This is the effect of using multi-resolution datasets; a dataset covering a larger area effectively scales up the grid. The compositing coordinate system defines a pixel grid which coincides with the original voxel grid in the normalized coordinate system.

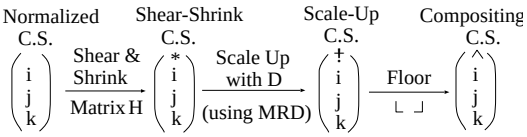


Fig. 4 Four coordinate systems.

The sequence of transformations from the normalized coordinate system to the compositing coordinate

system entails the sequence of grid changes as follows;

1) transform a grid point at (i, j, k) in the normalized coordinate system into a grid point at (i^*, j^*, k^*) by the shear-shrink matrix;

2) scale up the grid point at (i^*, j^*, k^*) using the scaling factor D to a grid point at $(i^\dagger, j^\dagger, k^\dagger)$;

3) apply the floor operation to the grid position $(i^\dagger, j^\dagger, k^\dagger)$ to get the final position $(\hat{i}, \hat{j}, \hat{k})$.

The scaling factor D is defined by $D = 2^L$, where $L = \lfloor \log_2 M \rfloor$, and M is the convolution area size in Equation (2). The resulting position $(\hat{i}, \hat{j}, \hat{k})$ and $(i^\dagger, j^\dagger, k^\dagger)$ are used for calculation of convolution weight.

4.3 Resampling with convolution

Let suppose convolution area be $M \times M \times M$ for sample estimation. A sample value at point $(\hat{i}, \hat{j}, \hat{k})$ along a parallelized ray is given by following 3D convolution.

$$s_{\hat{i}\hat{j}\hat{k}} = \sum_{l=0}^{M-1} \sum_{m=0}^{M-1} \sum_{n=0}^{M-1} w_{lmn} \cdot v_{i_c+l,\, j_c+m,\, k_c+n}, \quad (3)$$

where i_c, j_c, k_c are given by:

$$i_c = i - \lfloor \tfrac{M}{2} \rfloor, \quad j_c = j - \lfloor \tfrac{M}{2} \rfloor, \quad k_c = k - \lfloor \tfrac{M}{2} \rfloor,$$

and v_{i_c+l, j_c+m, k_c+n} is a voxel addressed by $(i_c + l, j_c + m, k_c + n)$ and w_{lmn} is a convolution weight given by:

$$w_{lmn} = W_{lmn}(\, \hat{i} - (i_c + l)^\dagger,\ \hat{j} - (j_c + m)^\dagger,\ \hat{k} - (k_c + n)^\dagger \,).$$

If we assume separable weights w_{lmn} , i.e. $w_{lmn} = w_n w_m w_l$, then Equation (3) can be transformed into Equation (4).

$$s_{\hat{i}\hat{j}\hat{k}} = \sum_{n=0}^{M-1} w_n \sum_{m=0}^{M-1} w_m \sum_{l=0}^{M-1} w_l \cdot v_{i_c+l,\, j_c+m,\, k_c+n}, \quad (4)$$

where w_l, w_m, and w_n are weights given by:

$$w_l = W_l(\hat{i} - (i_c + l)^\dagger), \quad w_m = W_m(\hat{j} - (j_c + m)^\dagger), \quad w_n = W_n(\hat{k} - (k_c + n)^\dagger). \quad (5)$$

This assumption is reasonable and useful for the implementation of 3D convolution. For example, *3D Lagrange interpolation* formula and a *3D Sinc* function are separable and belongs to this category. By using separable weight, the 3D convolution can be implemented using series of 1D convolution. This implementation uses $3M$ calculation units, i.e. multiplier and adder, instead of M^3 for ordinary 3D convolution.

4.4 Parallel Pipelined Convolution for a special case

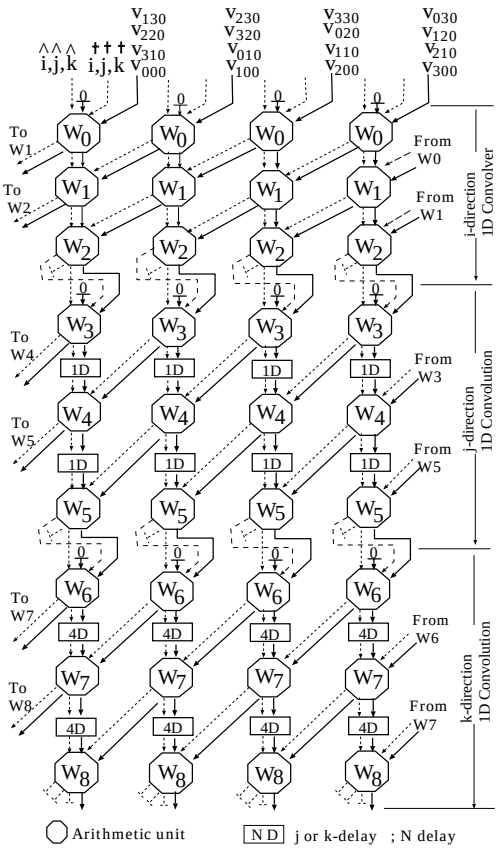
Fig. 5 illustrates an implementation of parallel pipelined convolution with $3 \times 3 \times 3$ area, based on Equations (4) and (5). The induction of the structure is described in Appendix A. The figure shows a special case: the dataset size in one dimension is equal to the number of processing pipelines. In the figure (a), the convolution has two types of data path; the solid line shows data path of voxels and dotted line shows that of sheared position. In each data path, operations are divided into three groups: one group for the i, the next for the j, and the last for the k-direction. In j-direction 1D convolution, it has *j-delay* to adjust timing of next scanline voxels. In k-direction 1D convolution, it has *k-delay* to adjust timing of next slice of voxles. In Fig. 5 (b), how to access all the voxels with a set of pipelines in sequential manner is shown. Fig. 5 (c) illustrates geometrical relation between sheared positions and original positions on a slice. The diffrence between original position and sheared position, δ_i is used for generating convolution weight for each direction. Fig. 5 (d) illustrates data flows of an arithmetic unit for i-direction 1D convolution in Fig. 5 (a). For j and k-derection 1D convolution, $(\hat{j} - m^\dagger)$ and $(\hat{k} - n^\dagger)$ are used to calculate weights respectively.

Table 1 shows several snapshots of the pipeline operations of the 1D convolution for the i-direction in Fig. 5 (a). In this example, one slice of 4×4 voxels is used as input for each pipline, see Fig. 5(b). The table shows that the 4 results of 1D convolution are generated simultaneously, see time-5 at location $W_2 - c$ in the table. Snapshots for whole pipeline operations can be induced from the snapshots in Table 1. Because the 1D convolution structure for the j and k-direction is similar to that of i-direction, but having different time delays due to the different timing of neighboring voxels in the j and k-directions.

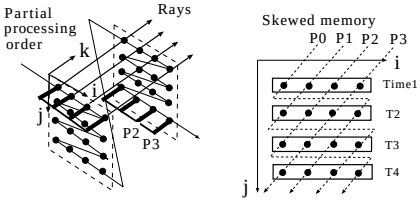
4.5 Selecting Multi-Resolution Datasets

The use of multi-resolution datasets can reduce the number of voxels required for convolution. By selecting an appropriate multi-resolution dataset depending on the resolution of the scaled voxel grid, the architecture can always use a bounded number of voxels for resampling regardless of the depth. It is a 3D version of the mip-mapping scheme for texture mapping [14]. The memory overhead to store multi-resolution datasets for a volume of size V^3 is less than $V^3/7$, which is not considered a very large overhead.

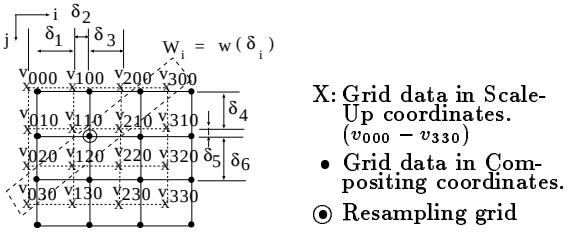
The resolution level is an indicator of the convolution area size at each slice in the progressively scaled voxel grid to choose an appropriate multi-resolution dataset. The power-of-2-based resolution level L is given by:



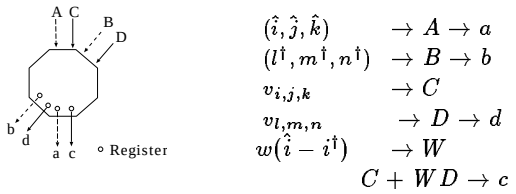
(a) A parallel pipelined convolution.



(b) Sequential access of voxels to be used for each pipeline.



(c) One slice data of volume for convolution input.



(d) An arithmetic unit in the pipelined convolution.

Fig. 5 3 × 3 × 3 pipelined 3D convolution with four pipelines for a special case: $V = N_P = 4$.

Table 1 Snapshots of data flow in 1D convolutions. The data flow for i-direction is shown.

Time	Location	Pipe 0	Pipe 1	Pipe 2	Pipe 3
1	$W_0 - D$	v_{000}	v_{100}	v_{200}	v_{300}
	$W_0 - c$	(− , − , −)	(− , − , −)	(− , − , −)	(− , − , −)
2	$W_0 - D$	v_{310}	v_{010}	v_{110}	v_{210}
	$W_0 - c$	$(v_{000}, 0, 0)$	$(v_{100}, 0, 0)$	$(v_{200}, 0, 0)$	$(v_{300}, 0, 0)$
	$W_1 - c$	(− , − , −)	(− , − , −)	(− , − , −)	(− , − , −)
3	$W_0 - D$	v_{220}	v_{320}	v_{020}	v_{120}
	$W_0 - c$	$(v_{310}, 0, 0)$	$(v_{010}, 0, 0)$	$(v_{110}, 0, 0)$	$(v_{210}, 0, 0)$
	$W_1 - c$	$(v_{000}, v_{100}, 0)$	$(v_{100}, v_{200}, 0)$	$(v_{200}, v_{300}, 0)$	$(v_{300}, v_{000}, 0)$
	$W_2 - c$	(− , − , −)	(− , − , −)	(− , − , −)	(− , − , −)
4	$W_0 - D$	v_{130}	v_{230}	v_{330}	v_{030}
	$W_0 - c$	$(v_{220}, 0, 0)$	$(v_{320}, 0, 0)$	$(v_{020}, 0, 0)$	$(v_{120}, 0, 0)$
	$W_1 - c$	$(v_{310}, v_{010}, 0)$	$(v_{010}, v_{110}, 0)$	$(v_{110}, v_{210}, 0)$	$(v_{210}, v_{310}, 0)$
	$W_2 - c$	$(v_{000}, v_{100}, v_{200})$	$(v_{100}, v_{200}, v_{300})$	$(v_{200}, v_{300}, v_{000})$	$(v_{300}, v_{000}, v_{100})$
5	$W_0 - D$	v_{001}	v_{101}	v_{201}	v_{301}
	$W_0 - c$	$(v_{130}, 0, 0)$	$(v_{230}, 0, 0)$	$(v_{330}, 0, 0)$	$(v_{030}, 0, 0)$
	$W_1 - c$	$(v_{220}, v_{320}, 0)$	$(v_{320}, v_{020}, 0)$	$(v_{020}, v_{120}, 0)$	$(v_{120}, v_{220}, 0)$
	$W_2 - c$	$(v_{310}, v_{010}, v_{110})$	$(v_{010}, v_{110}, v_{210})$	$(v_{110}, v_{210}, v_{310})$	$(v_{210}, v_{310}, v_{010})$

$$(p, q, r) = pW_1 + qW_2 + rW_3$$

$$L = \lfloor \log_2 M \rfloor, \tag{6}$$

where M is the convolution area size in Equation (2).

The scaling factor D is defined by $D = 2^L$.The multi-resolution data, $v_{i,j,k}$, for convolution are specified by the logical address $(L, \lfloor i/D \rfloor, \lfloor j/D \rfloor, \lfloor k/D \rfloor)$.

Note that their geometrical positions are given by $(i^\dagger, j^\dagger, k^\dagger)$.

4.6 Multi-Resolution Skewed Memory Organization

The skewed memory organization is a technique to store voxels in separate memory modules so that voxels in a slice can be accessed in parallel without any memory conflict regardless of the viewing direction [3]. It does not require multiple volume copies. The proposed architecture uses it to store multi-resolution datasets.

Consider a system with N_p rendering pipelines for a volume of size V^3. A logical memory address for the multi-resolution skewed memory is specified by (L, i, j, k), where L is the resolution level. Each multi-resolution dataset can be stored in the skewed memory as if it were an original volume dataset.

Let n_p be a memory module number, L be a resolution level in the module, and i_p be an index in the module, the multi-resolution data are accessed with the physical address (n_p, L, i_p) given by the following addressing scheme:

$$n_p = m \bmod N_p, \tag{7}$$

$$i_p = \lfloor m/N_p \rfloor + j'V'/N_p + k'V'^2/N_p, \tag{8}$$

where

$$\begin{aligned} &i' = \lfloor \tfrac{i}{D} \rfloor,\ j' = \lfloor \tfrac{j}{D} \rfloor,\ k' = \lfloor \tfrac{k}{D} \rfloor, \\ &V' = \lfloor \tfrac{V}{D} \rfloor, \\ &m = (i' + j' + k') \bmod V'. \end{aligned} \tag{9}$$

5. Proposed Architecture

5.1 Pipelined Convolution for a general case.

Fig. 5 in the previous section shows the 3D convolution for a special case $V = N_p = 4$, where V is a size of dataset in one dimension and N_p is a number of processing pipelines. In general, V is equal to or greater than N_p. By folding a string of voxel with N_P voxels, the set of pipelines can access a string of voxels repeatedly. Fig. 6 shows folding the string of voxels with 4 pipelines for V^3 dataset.

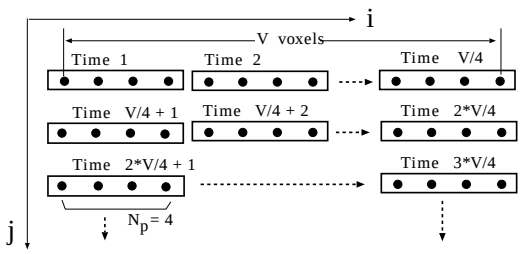


Fig. 6 Folding a string of voxels with $N_P = 4$ for a data dimension V which is greater than 4.

The folding scheme requests another delay elements in the architecture. Those are: *folding-delays*, *left-folding-delays* and *selectors*. Fig. 7 illustrates derivation of these elements and their control. In the figure, derivation of time delay for the case of $V = 8$ and $N_P = 4$ is shown. Fig. 7 (a) shows a naive implementation, i.e. no folding. Fig. 7 (b) illustrates dummy time-slots generated for folding. The input timing of the left half voxels is one unit time early to the right one. To compensate this timing mismatch, delay units are used. Fig 7 (c) shows that the dummy time slots are filled by folding. By this folding, there is no wasted time slots. As a result of the derivation, we can see that the time delay for *folding-delays* is always 1 unit time regardless of V, on the contrary that of *left-folding-delays* is V/N_P. Using these additional delay elements and controls, Fig. 5 can be extended to 3D convolution for a general case.

Fig. 8 shows a block diagram of 3D convolution for the general case: $V \geq N_p$ and $N_p = 4$. The structure is a direct extension from 3D convolution in Fig. 5. In the structure, the *j-delay* of $V/4$ is used to adjust time delay for voxels in a next scanline, and the *k-delay* of $V^2/4$ for a next slice of voxels. In addition, the structure has *folding-delays* and *left-folding-delays* to compensate time delay caused by folding as described in Fig. 7.

In summary, the number of delay time for general case with N_P pipelines is shown in Table 2. It is actually configured as a variable delay element for different resolution levels in adaptive processing. For a given scaling factor D, representing a resolution level, the delay element causes a delay of $(V/D)/4$ for *j-delay* and $(V/D)^2/4$ for *k-delay* respectively as shown in Table 2. The variable delay element can be implemented using a FIFO memory addressed in a circular manner by a single pointer for both read and write operations. The cycle time from one location to the same location determines the delay time, which can easily be changed by changing the maximum address value.

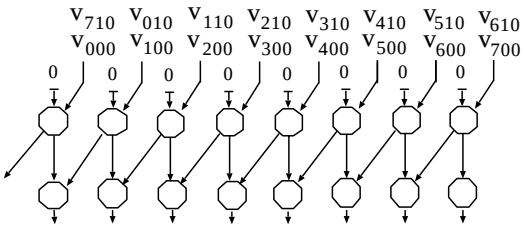
In the figure, the shearings $(i, j, k) \rightarrow (i^\dagger, j^\dagger, k^\dagger)$ and $(i, j, k) \rightarrow (\hat{i}, \hat{j}, \hat{k})$ are carried out on the fly at *s-hear* block in the figure. This shearing is carried out using DDA instead of matrix multiplication.

Table 2 Number of time delay versus data dimension V and the number of processing pipelines N_P.

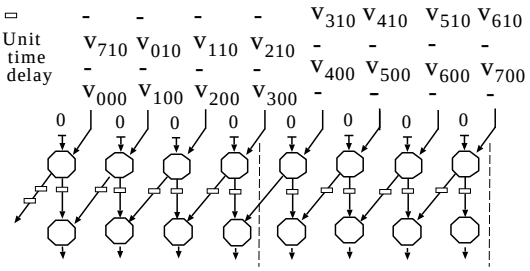
Location of delay	Number of delay	
	Non adaptive	Adaptive
j-delay	V/N_P	$(V/D)/N_P$
k-delay	V^2/N_P	$(V^2/D)/N_P$
left folding delay	V/N_P	$(V/D)/N_P$
folding delay	1	1

5.2 Rendering Timings

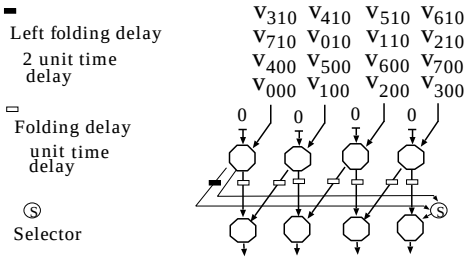
The rendering time is directly related to the number of resampling operations to perform, which can be re-



(a) No folding.



(b) Timing shift of input data with delay.



(c) Folding a string of voxels with 4 pipelines.

Fig. 7 A derivation chart of folding delays and their control for the folding process; $V = 8$ and $N_P = 4$.

duced by the use of multi-resolution data. It is upper-bounded by the total number of resampling operations without multi-resolution datasets, that is, only with original voxels.

Since the resampling and other rendering operations can fully be pipelined, the pipeline cycle time can be equal to the memory access time T_m. This implies that the processing bottleneck is the memory access time. For a given set of rendering parameters, accesses to the voxel memory are regular and deterministic. A double buffering technique can be used to provide continuous data streams from the voxel memory to the resampling module and rendering pipelines. The pixel memory does not require much bandwidth on average, because the pixel write operations are bursty but intermittent. A simple pixel buffering technique with a FIFO memory will be enough for pixel write operations.

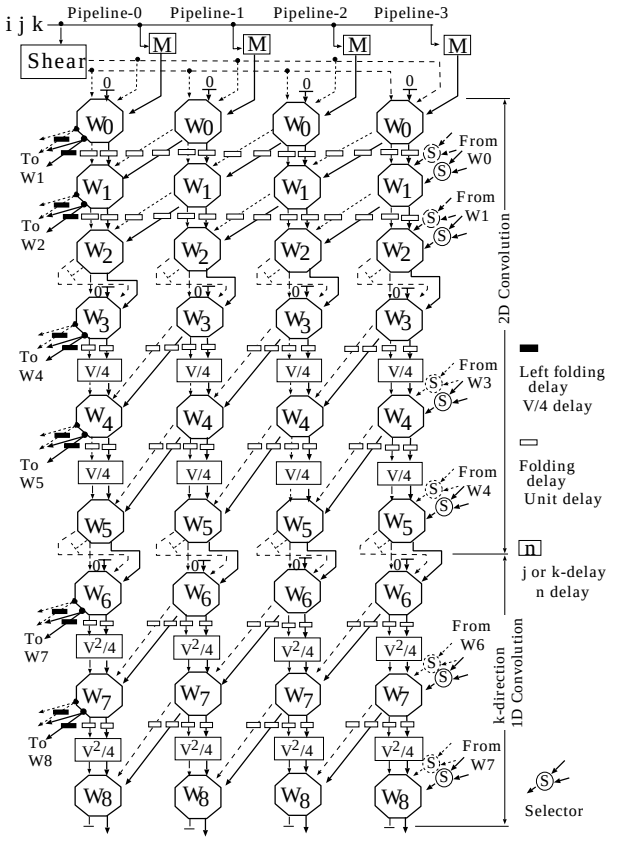


Fig. 8 $3 \times 3 \times 3$ pipelined 3D convolution for general case: $V \gg N_P$, $N_p = 4$.

The voxel and pixel memories can be implemented by SDRAM (Synchronous DRAM) chips to exploit their burst access mode.

Let V^3, N_p, and N_f be the volume size, the number of rendering pipelines, and the number of image frames generated per second. The total number of samples N_s to compute in each pipeline for one second is given by:

$$N_s = V^3 N_f / N_p. \tag{10}$$

For each second,

$$N_s T_m \leq 1. \tag{11}$$

For a given set of parameters T_m, N_f, and N_p, the maximum dimension of volume that can be rendered is given by:

$$V \leq \sqrt[3]{N_p/(N_f T_m)}. \tag{12}$$

Assuming that $T_m = 8$ ns as in a 125-MHz SDRAM chip and $N_f = 30$ frames/second, the volume dimensions computed for several values of N_f and N_p are shown in Fig.9. The volume dimensions computed for several typical values of N_f and N_p are also shown in Table 3. These values verify that the proposed ar-

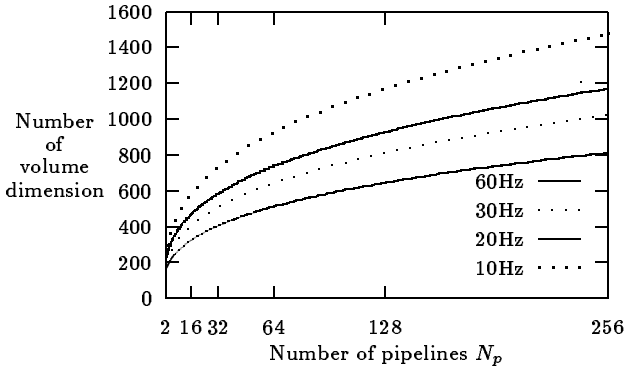


Fig. 9 Maximum volume dimensions.

Table 3 Maximum volume dimensions for typical frame rate.

N_f (frames/sec)	N_p (# pipelines)					
	2	4	8	16	32	64
30	203	255	322	405	511	644
20	232	292	368	464	585	737
10	292	368	464	585	737	928

chitecture can render volumes of practical sizes in real-time.

Multi-resolution datasets are generated before a volume dataset is loaded into the voxel memory. They can be generated off-line by software. The set of operations to compute a single data from eight data at a lower level includes seven additions, one division (shift), and eight memory reads, and one memory write; $V^2(V-1)^2/4$ sets of operations are required for a volume of V^3. Although the number of operations can be reduced by using some optimization techniques, such as buffering and pipelining, it seems still difficult to perform these operations on the fly with the current technologies. It may be reasonable, however, to perform these operations using several frame periods for practical applications.

6. Experiments

We built a software simulator to simulate the pipeline data flow of the proposed architecture and verify the concept for both parallel and perspective projections. To compare images, we also built a sample-order ray-casting renderer that computes slices of samples perpendicular to the viewing vector and accumulates them to produce the final image. We conducted several rendering experiments with these simulators.

Fig. 6 (a) shows a perspective image rendered from an opaque cube of size 64^3 to verify the perspective projection. The filter kernel based on the $2\times2\times2$ Lagrange formula is used in resampling.

Figs. 6 (b) and (c) show two perspective images rendered from an opaque checker-board cube of size 128^3 (spatial frequency of 64 Hz) to explore the aliasing problem; a fully opaque dataset gives the worst case for aliasing. The image in Fig. 6 (b) is generated by using the nearest neighbor voxel values in resampling, showing the aliasing problem clearly. The image in Fig. 6 (c) is generated by using a $3\times3\times3$ box filter kernel in resampling, showing the antialiasing effect by convolution using multi-resolution datasets.

Figs. 6 (d), (e), and (f) show the images rendered from the engine block of size 256^3 used in Lacroute's rendering experiments [6] with a manually adjusted opacity table. Fig. 6 (d) shows a perspective projection image, and Fig. 6 (e) shows a parallel projection image for a comparison purpose. These two images are generated by using a kernel based on the $3\times3\times3$ Lagrange formula.

Fig. 6 (f) is a perspective projection image generated by the sample-order ray-casting renderer with interpolations using multi-resolution datasets. The number of the slices taken for this image is 258, about the same number of slices (256) used in Fig. 6 (d). The two images in Figs. 6 (d) and (f) look comparable in quality.

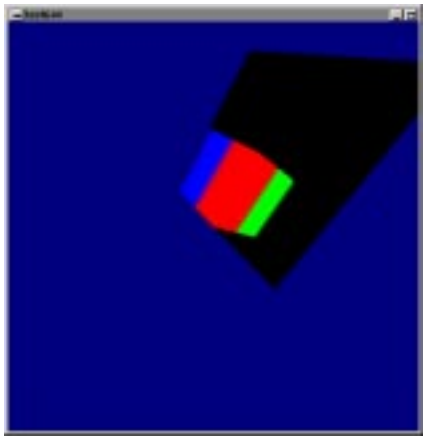
7. Future Work

The analysis for fixed-point arithmetic is an essential work for hardware implementation. And the other future work is the theoretical analysis for convolution on re-sampled data from irregular volumes.

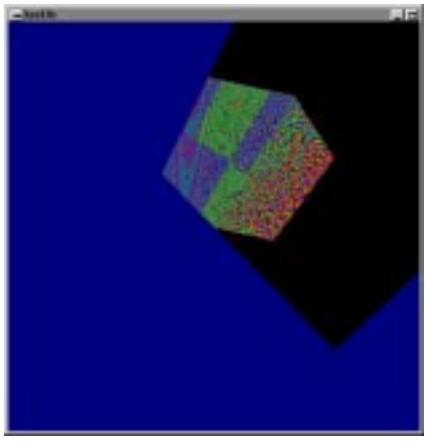
8. Conclusion

We have proposed a new VLSI architecture of real-time volume graphics using 3D convolution for both parallel and perspective projections based on the shear-warp algorithm. In the original algorithm, the further the ray proceeds, the more voxels are required for the calculation of convolution. The increase of required voxels makes difficult to implement the algorithm in hardware. 1) We prepare several sets of resolution of voxels associated with depth, in order that convolution can be done with constant number of voxels independent of depth. 2) We implement 3D convolution by three serial 1D convolutions along X, Y and Z axes, which reduces the calculation steps from M^3 to $3M$, where the convolution is calculated for M^3 area.

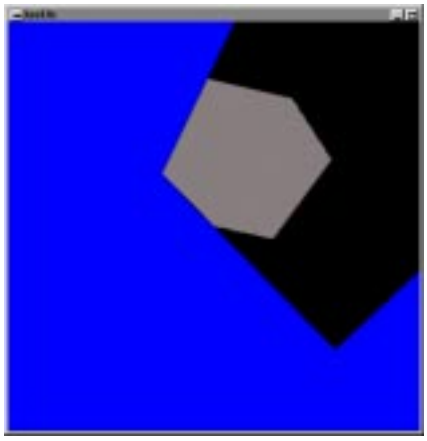
For V^3 voxels dataset, the number of pipelines for rays is V^2 and their pipeline stage is $3M$. If the hardware of a single pipeline has the capability of calculating V rays, each of the implemented pipelines is assigned to V theoretical pipelines (for V^2 rays). In actual implementation, a number of hardware pipelines should be much smaller than the V theoretical pipelines. We fold the theoretical pipelines and reduce them to the certain number of hardware pipelines. Regarding this folding, we show the relation between folding process and its necessary delay.



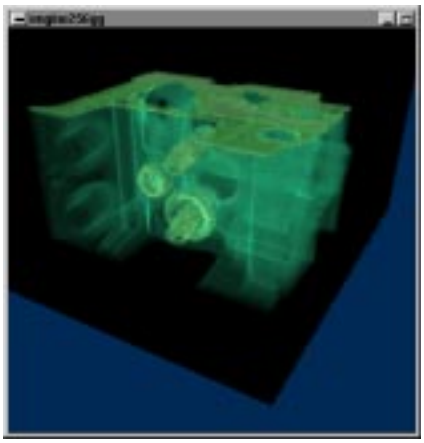
(a) Perspective projection with $2\times 2\times 2$ convolution.



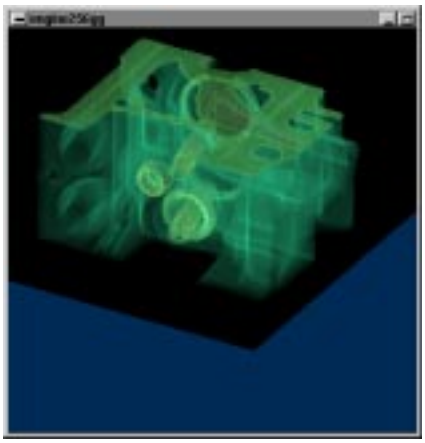
(b) Perspective projection by resampling the nearest neighbor voxels.



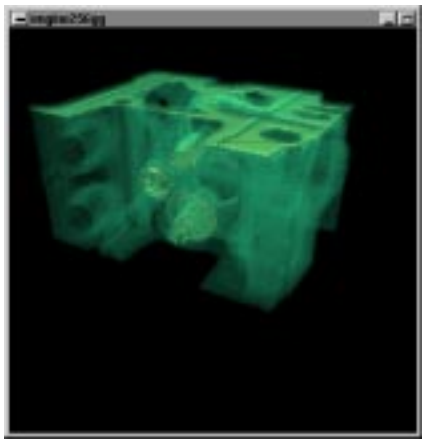
(c) Perspective projection with $3\times 3\times 3$ convolution.



(d) Perspective projection with $3\times 3\times 3$ convolution.



(e) Parallel projection with $3\times 3\times 3$ convolution.



(f) Screen-to-Object perspective projection by interpolations using multi-resolution datasets.

Fig. 10 Generated images with simulators.

The architecture can generate image of 256^3 voxles dataset($V = 256$) at 30Hz with 4 pipelines. In addition, the architecture can be extended easily for $512^3(V = 512)$ and $1024^3(V = 1024)$ dataset with 32 pipelines and 256 pipelines respectively. Our architecture has processing expandability.

Acknowledgments

We would like to thank all the members of the volume graphics project at Mitsubishi Electric Information Technology Center America for their support. Especially, many thanks to Beverly Schultz for her help. Also many thanks to Mr. Kagenori Kajihara of Mitshubishi Precision Co., for his valuable comments for the paper.

References

[1] A. S. Glassner. Principles of Digital Image Synthesis. *Morgan& Kaufman*, 1995.

[2] T. Günther, C. Poliwoda, C. Reinhart, J. Hesser, R. Männer, H.-P. Meinzer, and H.-J. Baur. Virim: A massively parallel processor for real-time volume visualization in medicine. *Computers & Graphics*, 19(5):705–710, 1995.

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

[4] G. Knittel and W. Strasser. A compact volume rendering accelerator. In *Proceedings of the IEEE Symposium on Volume Visualization*, pages 67–74, October 1994.

[5] P. G. Lacroute. Fast volume rendering using a shear-warp facrorization of the viewing transformation. Technical Report CSL-TR-95-678 (Ph.D. Dissertation), Computer Sys-

tems Laboratory, Stanford University, September 1995.

[6] P. G. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the ACM SIGGRAPH '94 Conference*, pages 451–457, July 1994.

[7] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[8] M. Ogata, H. Ohkami, H.C. Lauer and H. Pfister, A Real-Time Volume Rendering Architecture with Resampling Scheme for Parallel and Perspective Projections. *Proceedings of the ACM/IEEE Symposium on Volume Visualization*, pages 20–29, October 1998.

[9] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt and T. Ohkami. Em-cube: An architecture for low-cost real-time volume rendering. In *Proceedings of the 1997 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 131–138, August 1997.

[10] H. Pfister. Architecture for real-time volume rendering. Ph.d. dissertation, Department of Computer Science, State University of New York at Stony Brook, 1996.

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

[12] H. Pfister, J. H. Hardenburg, H. Lauer and S. Sailor. The VolumePro Real-Time Ray-Casting System. In *Proceedings of the ACM SIGGRAPH '99 Conference*, pages 131–138, August 1999.

[13] L. Westover. Footprint evaluation for volume rendering. In *Proceedings of the ACM SIGGRAPH '94*, pages 144–153, October 1994.

[14] L. Williams. Pyramidal parametrics. In *Proceedings of the ACM SIGGRAPH '83 Conference*, pages 1–11, July 1983.

Appendix A: Induction of systolic array structure in a special case

. The access timing difference T_d between two voxels is defined by Equation (A· 1) using voxel dimension V and the number of pipelines N_p, refer to Fig.6.

$$T_d(v_{ijk}\,;\,v_{lmn}) = \lfloor \frac{(i-l)+V(j-m)+V^2(k-n)}{N_p} \rfloor \qquad \text{(A·1)}$$

The 3D convolution, Equation (4), is separated into to the following series of equations.

$$\begin{aligned} a_{i_c,*,*} &= \textstyle\sum_{l=0}^{M-1} w_l \cdot v_{i_c+l,*,*} \quad , \\ b_{*,j_c,*} &= \textstyle\sum_{m=0}^{M-1} w_m \cdot a_{*,j_c+m,*} \quad , \\ s_{*,*,k_c} &= \textstyle\sum_{n=0}^{M-1} w_n \cdot b_{*,*,k_c+n} \quad . \end{aligned} \qquad \text{(A·2)}$$

In this induction, the real sheared positions indicated in the Equation (4) are not necessary so that we do not used it for simplicity.

Fig.A· 1 shows 4 $a_{i_c,*,*}$ $(i_c = 0, \cdots, 2)$ with standing up form. In the figure, same voxels are indicated with arrows. These same voxels need not to be accessed each time from memory, but with data passing through. The delay time for these passing thorough can be calculated using Equation (A· 1). In this case, $T_{d1}(v_{i+1,*,*}\,;\,v_{i,*,*}) = 0$. Therefore delay units for timing adjustment are not necessary for the data passing.

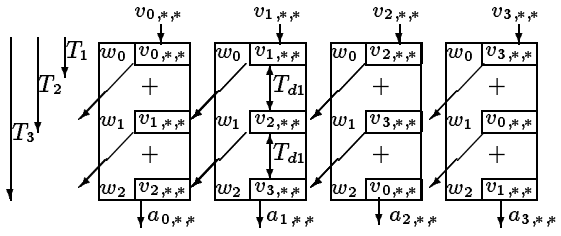


Fig. A· 1 Induction of 4 parallel pipelined 1D convolution for i-direction.

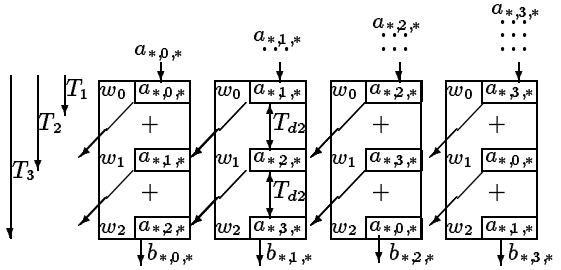


Fig. A· 2 Induction of 4 parallel pipelined 1D convolution for j-direction.

Fig A· 2 illustrates 4 $b_{*j_c,*}$ with standing up form. There are same results of i-direction 1D convolution value $a_{i,*,*}$ as indicated with arrows. In this case, $T_{d2}(a_{*j_c+1,*}\,;\,a_{*j_c,*}) = 4/4$. Therefore 1 unit time delays are necessary to adjust timing for data passing.

Same discussion can be applied for k-direction. In this case, $T_{d3}(b_{*,*,k+1}\,;\,b_{*,*,k}) = 16/4$. This means 4 unit times delay is necessary for data passing. From these discussions, Fig.5 is induced. Without loosing generality, these discussions can be applied for the induction of the parallel pipelined convolution for special case, i.e. $V = N_p$.

Masato OGATA is an Assistant General Manager of research and development department at Mitsubishi Precision Co., Ltd. in Japan. His research background has involved mostly computer systems, and real-time computer graphics and related topics. He developed several real-time image generators for flight simulators. His current research interests are volume graphics, scientific visualization, and real-time volume graphics architectures. He graduated Oita National College of Technology in 1969, majored Electrical Engineering. He received M.S. from Yokohama National University in 1995. Currently, he is a Ph.D. student in Yokohama National University. He is a member of IEEE.

Takahide OHKAMI is a Technical Staff member of the Volume Graphics Organization at Mitsubishi Electric Information Technology Center of America, Cambridge, M.A, U.S.A. His Research background has involved mostly in computer architecture , operating systems , compiler, parallel and distributed computing systems, and 3D computer graphics. His current research interests are parallel processor architectures and 3D graphics rendering architectures. He received B.E from University of Tokyo in 1977, M.S from University Rochester in 1987, Ph.D from University Tokyo in 1992. He is a member of ACM,IEEE, and IPSJ.

Hanspeter Pfister is a Research Scientist at MERL. - A Mitsubishi Electric Research Laboratory in Cambridge, MA., U.S.A. He is the chief architect of VolumePro, Mitsubishi Electric's real-time volume rendering system for PC-class computers. His research interests include computer graphics, scientific visualization, computer architecture and VLSI design. He received his Dipl.-Ing. degree in electrical engineering from Swiss Federal Institute of Technology in 1991, Ph.D. in Computer Science from the State University of New York in 1996. In his doctoral research he developed Cube-4, a scalable architecture for real-time volume rendering. He is a member of the ACM, IEEE, the IEEE Computer Society, and the Eurographics Association.

Hugh C. Lauer is a Senior Research Scientist and Chief Technical officer of the Volume Graphics at MERL. His research background has involved mostly computer architecture, operating systems, distributed computing and related topics. His current research interests are real-time volume graphics and visualization. He received B.S in Mathematics form Antioch College in 1965, M.S in mathematics from Carnegie Institute of Technology in 1967, Ph.D. in Computer Science from Carnegie-Mellon University in 1973. He is a member of the ACM, IEEE, the IEEE Computer Society.

Yasunori DOHI is a Professor of Electrical Engineering and Computer Science at Yokohama National University. His research background has involved mostly computer architecture, VLSI algorithms, electrical circuits design and related topics. His current research interests are VLSI oriented architectures, and real-time computer graphics architecture. He received B.S in 1962, M.S in 1964, Ph.D in 1967 from Tokyo Institute of Technology. He is a member of IEEE, IEC, and IPSJ.