Paper

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

(実時間ボリュームレンダリングにおける透視投影のための並列パイプライン・コンボリューション)

Masato Ogata (member)†, Hanspeter Pfister††, Hugh C. Lauer††, Yasunori Dohi†††

Abstract This paper describes a convolution with a systolic array structure 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 to calculate the convolution. The increase in required voxels makes it difficult to implement the method in a VLSI-oriented architecture. We implement a 3D convolution using three serial 1D convolutions along the X, Y, and Z axes, which reduces the number of calculation units from M^3 to $3M$, where the convolution is calculated for the M^3 area. The number of pipelines for the rays is V^2 for V^3 voxel datasets. If the hardware of a single pipeline can calculate the V rays, then each of the implemented pipelines is assigned to V theoretical pipelines (for V^2 rays). The number of hardware pipelines should be much smaller than V theoretical pipelines in actual implementation. We folded the theoretical pipelines and reduced them to a certain number of hardware pipelines. We examined the relation between the folding process and its necessary time delay. The architecture can generate an image of a 256^3 voxel dataset ($V = 256$) at 30 Hz with four pipelines. In addition, the architecture can be extended easily for 512^3 ($V = 512$) and 1024^3 ($V = 1024$) datasets, with 32 pipelines and 256 pipelines. Our architecture has processing scalability.

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 demand due to the increasing amount 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 amount of data elements to be processed in rendering. This requires a substantial amount of computing resources for animated visualization, which is essential to observe some physical phenomena.

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

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

As a simplified implementation of Equation (1), each sample is computed from the voxels surrounding the sample point by interpolation, then 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 on the operations is the greatest portion of the rendering time. This time requirement has made arranging parallel processing for resampling one of the

Received January 19, 2000; Revised May 30, 2000; Accepted July 3, 2000

† 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)

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

This paper is organized as follows, section 2 presents related work, section 3 presents some issues of the shear-warp method, section 4 presents key ideas for implementation, section 5 shows the proposed architecture, section 6 evaluates the architecture, section 7 describes future work, and section 8 concludes the paper.

2. Related Work

From an architectural standpoint, the ray-casting algorithm is categorized into two schemes for implementation: sample-order and voxel-order schemes. As the term implies, the sample-order scheme uses the sample point as the processing start point, then obtains the memory address of the voxel. In contrast, the voxel-order scheme uses the memory address of the voxel directly as a 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 a straightforward implementation of the ray-casting algorithm[2)4)]. It can utilize some available optimization techniques. Early ray termination and coherence encoding are two methods 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. In addition, it does not access the volume data in storage order so it requires a complicated memory address calculation. These are disadvantages for VLSI-oriented implementations of real-time volume rendering systems. There is no rendering system in the scheme that can generate images for large data more than 64^3 grids in real-time[2)].

2.2 Voxel-order scheme

The voxel-order scheme uses the voxel address directly so that it can access the volume data in storage order. This makes the scheme suitable for VLSI-oriented implementation. The voxel-parallel method[10)] in the scheme reads voxels once and retains them until all the samples that require the voxels are computed. Cube-4[10)] and its VLSI implementation, EM-Cube[9)11)], are the rendering systems in this method. The architectures are organized in a systolic array structure. However, they support only parallel projections.

Shear-warp[5)6)] is another method in the voxel-order scheme. It can treat both parallel and perspective projections in a unified way.

3. Issues of the Shear-warp Method for Hardware Implementation

The shear-warp method produces a distorted base plane* image with the shearing matrix H[5)6)] as an intermediate image; the image is then warped to produce the correct image on a screen. **Fig. 1** shows the perspective rays parallelized at the base plane by the shearing matrix H. Starting from a position in the first slice, i.e. the 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. The resampling operation is a convolution over the voxels in a resampling area.

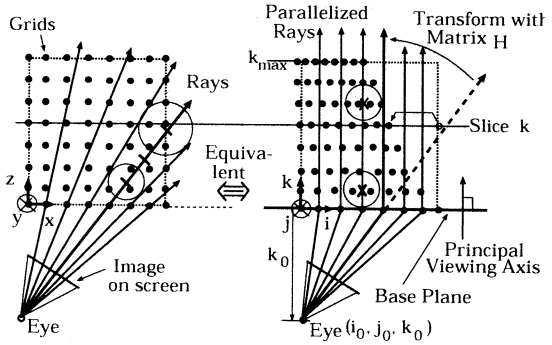


Fig. 1 Shearing and scaling in perspective projections.

This method has a significant advantage in that both projections can be treated in a unified way. However, there are two important issues for parallel pipelined implementation, i.e. the systolic array, of the method: **(1) reducing the number of calculation units for a convolution** and **(2) organizing a parallel pipeline convolution.**

The number of voxels to be convoluted in one dimension, M, in the worst case is computed by:

$$M = 1 - k_{max}/k_0 \ , \qquad (2)$$

where k_{max} **, is the most distant slice number or the

* generating more than 30 images per second

* The plane 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 the transformed, as shown in **Fig. 1**

distance from the base plane, and k_0 is the distance between the eye position (the center of the projection) and the base plane, as shown in **Fig. 1**. It can be large with high values for k_{max} and/or small values for k_0. A large area is required for convolution since hardware implementation can only use a fixed amount of resources for the convolution. Without a large area, low quality or aliased base plane images are produced. Although in perspective projection using M voxels for convolution ensures covering the convolution area at all samples, a naive 3D convolution for M voxels area requires M^3 calculation units, so it is almost impossible to implement with hardware for a large M.

The convolution structure is another issue. We intended to make a systolic array structure for the 3D convolution to get real-time speed. These issues make it difficult to implement the perspective projections in real-time and have been major obstacles for hardware implementation in this method.

4. Basic Ideas

4.1 Sample-parallel method

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

All the complicated communications and controls between other pipelines are moved from the rendering pipelines to a resampling module in order to organize the rendering pipelines in a fixed parallel-pipeline structure.

As shown in **Fig. 2**, the architecture places the resampling module between the voxel memory and rendering pipelines. This module is specialized for resampling using the voxels in the resampling area.

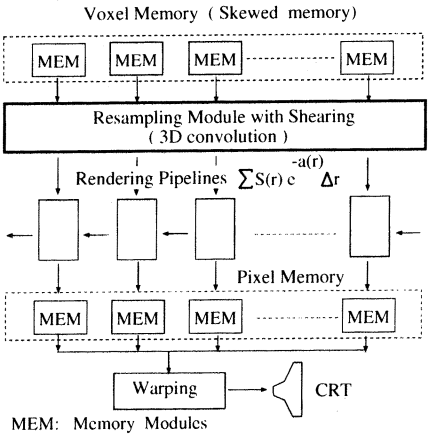


Fig. 2 Sample-parallel architecture.

4.2 3D convolution by series of 1D convolutions

The sheared voxel position $(i^\dagger, j^\dagger, k^\dagger)$, and the resampling position $(\hat{i}, \hat{j}, \hat{k})$ that correspond to voxel address (i, j, k) are given by the following equations.

$$\begin{aligned}(I, J, K, W)^T &= H \cdot (i, j, k, 1)^T \\ (i^\dagger, j^\dagger, k^\dagger) &= (I/W, J/W, K/W) \\ (\hat{i}, \hat{j}, \hat{k}) &= (\lfloor i^\dagger \rfloor, \lfloor j^\dagger \rfloor, \lfloor k^\dagger \rfloor)\end{aligned} \tag{3}$$

where suffix T indicates a transposition and matrix H is given by[5]:

$$\begin{pmatrix} 1 & 0 & -\frac{i_0}{k_0} & 0 \\ 0 & 1 & -\frac{j_0}{k_0} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{k_0} & 1 \end{pmatrix} , \tag{4}$$

where (i_0, j_0, k_o) is the eye position as illustrated in **Fig. 1**.

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

$$s_{\hat{i}\hat{j}\hat{k}} = \sum_{l=0}^{M-1} \sum_{m=0}^{M-1} \sum_{n=0}^{M-1} w_{lmn} \cdot v_{i_c+l,\, j_c+m,\, k_c+n} , \tag{5}$$

where i_c, j_c, k_c are given by:

$$\begin{aligned} i_c &= i - \lfloor \tfrac{M}{2} \rfloor , \quad j_c = j - \lfloor \tfrac{M}{2} \rfloor , \\ k_c &= k - \lfloor \tfrac{M}{2} \rfloor . \end{aligned}$$

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 normalized convolution weight given by:

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

$$\sum_{l} \sum_{m} \sum_{n} w_{lmn} = 1 .$$

If we assume separable weights w_{lmn} , i.e. $w_{lmn} = w_n w_m w_l$, then Equation (5) can be transformed into Equation (6).

$$s_{\hat{i}\hat{j}\hat{k}} = \sum_{n=0}^{M-1} w_n \sum_{m=0}^{M-1} w_m \sum_{l=0}^{M-1} w_l \cdot v_{i_c+l,\, j_c+m,\, k_c+n}, \tag{6}$$

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

$$\begin{aligned} w_l &= W_l(\hat{i} - (i_c+l)^\dagger), \\ w_m &= W_m(\hat{j} - (j_c+m)^\dagger), \\ w_n &= W_n(\hat{k} - (k_c+n)^\dagger). \end{aligned} \tag{7}$$

This assumption is reasonable in many practical cases. For example, a 3D*Lagrange interpolation* formula and a 3D*Sinc* function are separable and belong

to this category. By using a separable weight, the 3D convolution can be implemented using a series of 1D convolutions. This implementation uses $3M$ calculation units, i.e. a multiplier and adder, instead of M^3 for an ordinary 3D convolution. This greatly reduces the calculation cost.

4.3 Induction of a parallel pipeline structure for the 3D convolution

We first define the access timing difference T_d between two voxels. **Fig. 3** illustrates the voxel accessing order in a slice with four pipelines. All voxels in the volume are accessed in this manner.

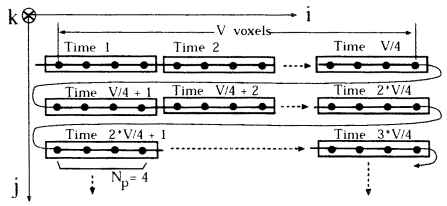


Fig. 3 Voxel-accessing-order in a slice with four pipelines for a data dimension V greater than four.

The access timing difference T_d, caused by the accessing order, is defined by Equation (8) using voxel dimension V and the number of pipelines N_p, as shown in **Fig. 3**.

$$T_d(v_{ijk}\,;\,v_{r,s,t}) = \left\lfloor \frac{(i-r)+V(j-s)+V^2(k-t)}{N_p} \right\rfloor \qquad (8)$$

The 3D convolution, Equation (6), is separated into the following series of 1D convolutions.

$$\begin{aligned} a_{\hat{i},*,*} &= \textstyle\sum_{l=0}^{M-1} w_l \cdot v_{i_c+l,*,*} \quad , \\ b_{\hat{i}\hat{j},*} &= \textstyle\sum_{m=0}^{M-1} w_m \cdot a_{\hat{i},j_c+m,*} \quad , \\ s_{\hat{i}\hat{j}\hat{k}} &= \textstyle\sum_{n=0}^{M-1} w_n \cdot b_{i,\hat{j},k_c+n} \quad . \end{aligned} \qquad (9)$$

In this induction, we assume $M = 3$ and $N_p = 4$ for simplicity. The following discussion can be generally applied for any M and N_p. **Fig. 4** shows four $a_{\hat{i},*,*} = \sum_{l=0}^{M-1} w_l \cdot v_{i_c+l,*,*}$ in a standing form. Equivalent voxels are indicated by arrows in the figure. These same voxels need not be accessed each time from memory, but with data passing through. The delay time for those passing through can be calculated using Equation (8). In this case, $T_{d1}(v_{i_c+1,*,*}\,;\,v_{i_c,*,*}) = 0$. Therefore, delay-units for timing adjustment are not necessary for the data passing.

Fig. 5 illustrates four $b_{\hat{i}\hat{j},*}$. The results for an i-direction 1D convolution, $a_{\hat{i},*,*}$, are the same, as indicated by arrows. In this case, $T_{d2}(a_{\hat{i}j_c+1,*}\,;\,a_{\hat{i}j_c,*}) = 4/4$. Therefore, delays of one time unit are necessary to adjust the timing for the data passing.

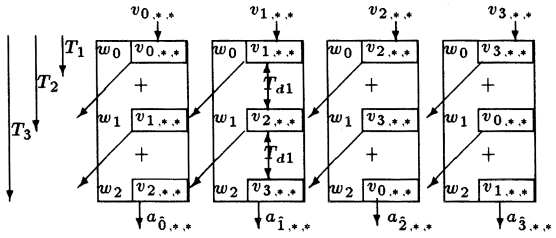


Fig. 4 Induction of four parallel pipeline 1D convolutions for i-direction.

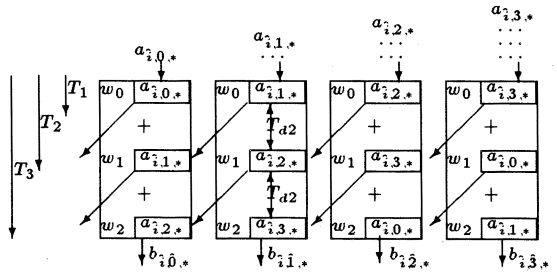


Fig. 5 Induction of four parallel pipeline 1D convolutions for j-direction.

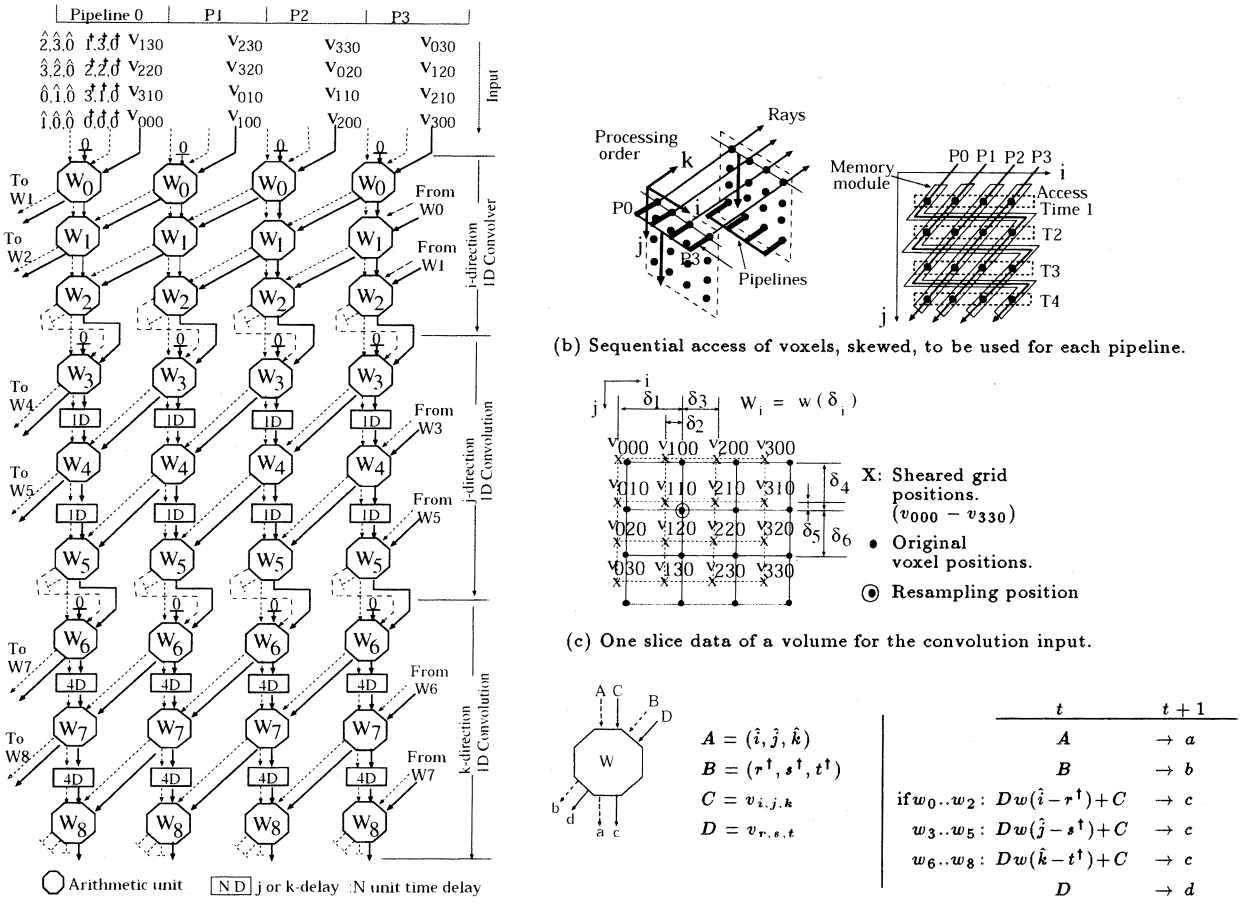
The same discussion can be applied to the k-direction. In this case, $T_{d3}(b_{\hat{i}\hat{j},k_c+1}\,;\,b_{\hat{i}\hat{j},k_C}) = 16/4$. This indicates that delays of four time units are necessary. We can obtain a parallel pipeline 3D convolution by serially connecting the above three 1D convolutions.

4.4 Parallel pipeline convolution for a special case

Fig. 6 illustrates an implementation of a parallel pipeline convolution with a $3 \times 3 \times 3$ area, based on the previous discussion. The figure shows a special case in which the dataset size in one dimension is equal to the number of processing pipelines. In **Fig. 6**(a), the convolution has two types of data paths; the solid line indicates the data path for voxels, and the dotted line indicates that for the sheared and resampling positions. Operations are divided into three groups in each data path: one group for the i, another for the j, and the third for the k-direction. The j-direction 1D convolution has a *j-delay*; i.e. , T_{d2}, to adjust the timing of the next scanline-voxels. The k-direction 1D convolution has a *k-delay*; i.e., T_{d3}, to adjust the timing of the next slice of voxels.

The technique for accessing all the voxels with a set of pipelines in a sequential manner is shown in **Fig. 6** (b).

Fig. 6 (c) illustrates the geometrical relation between the sheared positions and original positions on a slice. The difference between the resampling position and the sheared position, δ_i, is used to generate the convolution weight for each direction. **Fig. 6** (d) illustrates the data flows of an arithmetic unit in **Fig. 6** (a).



(a) Parallel pipeline convolution.

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

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

(d) Arithmetic unit in the pipelined convolution.

Fig. 6 $3 \times 3 \times 3$ parallel pipeline 3D convolutions with four pipelines for a special case: $V = N_P = 4$.

Table 1 shows several snapshots of data flow of the pipeline operations of the 1D convolution for the i-direction in **Fig. 6** (a). In this example, one slice of 4×4 voxels is used as input for each pipeline; see **Fig. 6** (b). The table indicates that the four results of the 1D convolution are generated simultaneously, as shown in time-5 at location $W_2 - c$ in the table. The data flow for resampling position $(\hat{i}, \hat{j}, \hat{k})$ and sheared position $(i^\dagger, j^\dagger, k^\dagger)$ is shown in **Table 2**. The overall pipeline operations can be deduced from the snapshots in **Table 1** and bf Table 2 since the 1D convolution structure for the j and k-directions is similar to that of the i-direction, but with different time delays, T_{d2} and T_{d3}, as described in section 4.3.

4.5 Convolution area

The underlining architecture pairs a memory module and a convolution pipeline so that each pipeline can take one voxel along with a neighboring voxel through a lateral communication and produces one sample. M voxels guarantee to wrap the convolution area at any slices for perspective projections.

The order of the voxels to be read cannot be controlled because each memory is connected to each pipeline. However, the position of resampling $(\hat{i}, \hat{j}, \hat{k})$ for each pipeline can be controlled. **Fig. 7** illustrates the relation between the pipelines for voxels to be read and the pipelines for their outputs. An example with $N_p = 4$ and $M = 3$ is shown in the figure. The pipeline-outputs that do not contribute to resampling, although they contribute to the simultaneous reading of voxels for resampling, are neglected when compositing in the rendering pipelines using the enable/disable flags.

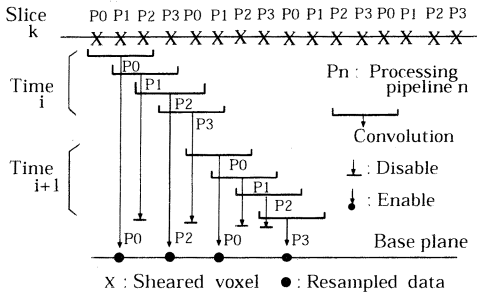


Fig. 7 Relation between the pipelines for voxels to be read and pipelines for their outputs.

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}
	$W_0 - c$	(−, −, −)	(−, −, −)	(−, −, −)	(−, −, −)
	$W_1 - c$	(−, −, −)	(−, −, −)	(−, −, −)	(−, −, −)
	$W_2 - c$	(−, −, −)	(−, −, −)	(−, −, −)	(−, −, −)
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)
	$W_1 - c$	(−, −, −)	(−, −, −)	(−, −, −)	(−, −, −)
	$W_2 - c$	(−, −, −)	(−, −, −)	(−, −, −)	(−, −, −)
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})

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

Table 2 Snapshots of data flow for resampling and sheared position in 1D convolutions. The data flow for i-direction is shown.

Time	Location	Pipe 0	Pipe 1	Pipe 2	Pipe 3
		$(\hat{i}\hat{j}\hat{k})$, $(r^\dagger s^\dagger t^\dagger)$	$(\hat{i}\hat{j}\hat{k})$, $(r^\dagger s^\dagger t^\dagger)$	$(\hat{i}\hat{j}\hat{k})$, $(r^\dagger s^\dagger t^\dagger)$	$(\hat{i}\hat{j}\hat{k})$, $(r^\dagger s^\dagger t^\dagger)$
1	$W_0 - A,\ B$	$(\hat{1}\hat{0}\hat{0})$, $(0^\dagger 0^\dagger 0^\dagger)$	$(\hat{2}\hat{0}\hat{0})$, $(1^\dagger 0^\dagger 0^\dagger)$	$(\hat{3}\hat{0}\hat{0})$, $(2^\dagger 0^\dagger 0^\dagger)$	$(\hat{0}\hat{0}\hat{0})$, $(3^\dagger 0^\dagger 0^\dagger)$
	$W_0 - a,\ b$	−, −	−, −	−, −	−, −
2	$W_0 - A,\ B$	$(\hat{0}\hat{1}\hat{0})$, $(3^\dagger 1^\dagger 0^\dagger)$	$(\hat{1}\hat{1}\hat{0})$, $(0^\dagger 1^\dagger 0^\dagger)$	$(\hat{2}\hat{1}\hat{0})$, $(1^\dagger 1^\dagger 0^\dagger)$	$(\hat{3}\hat{1}\hat{0})$, $(2^\dagger 1^\dagger 0^\dagger)$
	$W_0 - a,\ b$	$(\hat{1}\hat{0}\hat{0})$, $(1^\dagger 0^\dagger 0^\dagger)$	$(\hat{2}\hat{0}\hat{0})$, $(2^\dagger 0^\dagger 0^\dagger)$	$(\hat{3}\hat{0}\hat{0})$, $(3^\dagger 0^\dagger 0^\dagger)$	$(\hat{0}\hat{0}\hat{0})$, $(0^\dagger 0^\dagger 0^\dagger)$
	$W_1 - a,\ b$	−, −	−, −	−, −	−, −
3	$W_0 - A,\ B$	$(\hat{3}\hat{2}\hat{0})$, $(2^\dagger 2^\dagger 0^\dagger)$	$(\hat{0}\hat{2}\hat{0})$, $(3^\dagger 2^\dagger 0^\dagger)$	$(\hat{1}\hat{2}\hat{0})$, $(0^\dagger 2^\dagger 0^\dagger)$	$(\hat{2}\hat{2}\hat{0})$, $(1^\dagger 2^\dagger 0^\dagger)$
	$W_0 - a,\ b$	$(\hat{0}\hat{1}\hat{0})$, $(0^\dagger 1^\dagger 0^\dagger)$	$(\hat{1}\hat{1}\hat{0})$, $(1^\dagger 1^\dagger 0^\dagger)$	$(\hat{2}\hat{1}\hat{0})$, $(2^\dagger 1^\dagger 0^\dagger)$	$(\hat{3}\hat{1}\hat{0})$, $(3^\dagger 1^\dagger 0^\dagger)$
	$W_1 - a,\ b$	$(\hat{1}\hat{0}\hat{0})$, $(2^\dagger 0^\dagger 0^\dagger)$	$(\hat{2}\hat{0}\hat{0})$, $(3^\dagger 0^\dagger 0^\dagger)$	$(\hat{3}\hat{0}\hat{0})$, $(0^\dagger 0^\dagger 0^\dagger)$	$(\hat{0}\hat{0}\hat{0})$, $(1^\dagger 0^\dagger 0^\dagger)$
	$W_2 - a,\ b$	−, −	−, −	−, −	−, −
4	$W_0 - A,\ B$	$(\hat{2}\hat{3}\hat{0})$, $(1^\dagger 3^\dagger 0^\dagger)$	$(\hat{3}\hat{3}\hat{0})$, $(2^\dagger 3^\dagger 0^\dagger)$	$(\hat{0}\hat{3}\hat{0})$, $(3^\dagger 3^\dagger 0^\dagger)$	$(\hat{1}\hat{3}\hat{0})$, $(0^\dagger 3^\dagger 0^\dagger)$
	$W_0 - a,\ b$	$(\hat{3}\hat{2}\hat{0})$, $(3^\dagger 2^\dagger 0^\dagger)$	$(\hat{0}\hat{2}\hat{0})$, $(0^\dagger 2^\dagger 0^\dagger)$	$(\hat{1}\hat{2}\hat{0})$, $(1^\dagger 2^\dagger 0^\dagger)$	$(\hat{2}\hat{2}\hat{0})$, $(2^\dagger 2^\dagger 0^\dagger)$
	$W_1 - a,\ b$	$(\hat{0}\hat{1}\hat{0})$, $(1^\dagger 1^\dagger 0^\dagger)$	$(\hat{1}\hat{1}\hat{0})$, $(2^\dagger 1^\dagger 0^\dagger)$	$(\hat{2}\hat{1}\hat{0})$, $(3^\dagger 1^\dagger 0^\dagger)$	$(\hat{3}\hat{1}\hat{0})$, $(0^\dagger 1^\dagger 0^\dagger)$
	$W_2 - a,\ b$	$(\hat{1}\hat{0}\hat{0})$, $(3^\dagger 0^\dagger 0^\dagger)$	$(\hat{2}\hat{0}\hat{0})$, $(0^\dagger 0^\dagger 0^\dagger)$	$(\hat{3}\hat{0}\hat{0})$, $(1^\dagger 0^\dagger 0^\dagger)$	$(\hat{0}\hat{0}\hat{0})$, $(2^\dagger 0^\dagger 0^\dagger)$

5. Proposed Architecture

5.1 Skewed memory organization

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. We used skewed memory organization for volume data.

Consider a system with N_p rendering pipelines for a volume with a size V^3. A logical memory address for the skewed memory is specified by (i, j, k). Let n_p be a memory module number, and i_p be the index in the module; the physical address (n_p, i_p) is given by the following addressing scheme using the logical address:

$$n_p = m \bmod N_p, \tag{10}$$

$$i_p = \lfloor m/N_p \rfloor + jV/N_p + kV^2/N_p, \tag{11}$$

where

$$m = (i + j + k) \bmod V\ . \tag{12}$$

Fig. 6 (b)-right illustrates skewed memory, with one slice of volume data.

5.2 Parallel pipelined convolution for a general case

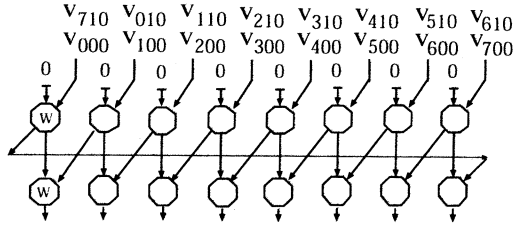
We have shown the architecture of 3D convolution for the special case of $V = N_p$. V is generally equal to or greater than N_p. By folding a string of voxels with N_P voxels, the set of N_p pipelines can access the entire string of voxels repeatedly, as shown in **Fig. 3**.

The folding scheme requires other types of delays in the architecture, which are *folding-delays*, *left-folding-delays*, and *selectors*. **Fig. 8** shows the derivations of these delays and their controls. The derivation of the time delay for the case of $V = 8$ and $N_P = 4$ is shown in the figure. **Fig. 8** (a) shows a naive implementation with no folding. **Fig. 8** (b) shows the dummy time-slots generated for the folding. The input timing of the left half of the voxels is one unit-time sooner than the right one. Delays are used to compensate for this timing mismatch. **Fig. 8** (c) shows that the dummy time slots are filled by folding. There are no wasted time-slots because of this folding. This derivation enables us to see that the time delay for *folding-delays* is always one unit time, regardless of V; in contrast, that of *left-folding-delays* is V/N_P.

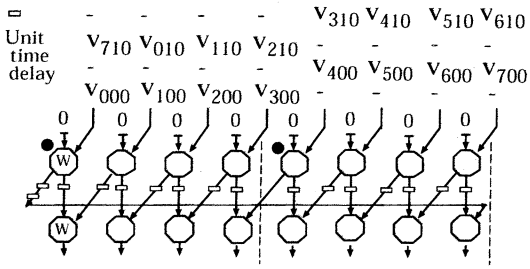
Fig. 9 shows a block diagram of the 3D convolution for the general cases of $V \geq N_p$ and $N_p = 4$. This structure is a direct extension from **Fig. 6**. In the structure, the *j-delay* of $V/4$ is used to adjust the time delay for voxels in the next scanline, and the *k-delay* of $V^2/4$ for the next slice of voxels. In addition, the structure has *folding-delays* and *left-folding-delays* to compensate for the time delay caused by folding. In summary, the number of time-delays for general cases of N_P pipelines is shown in **Table 3**.

The following steps describe the operation of the parallel pipeline 3D convolution in the figure. *Address Gen* generates voxel address (i, j, k) with incremental step N_p for i, starting from $(0, 0, 0)$. It then increments j and k until the address reaches $(V-1, V-1, V-1)$. The internal address of each memory is calculated with this (i, j, k), using Equation (11) to fetch voxels. At the same time, the shearing $(i, j, k) \rightarrow (i^\dagger, j^\dagger, k^\dagger)$ is carried out on the fly in the *shear* bock. $(\hat{i}, \hat{j}, \hat{k})$ is generated in the same block using $(i^\dagger, j^\dagger, k^\dagger)$. This shearing is carried out using DDA instead of matrix multiplication. A set of pipelines reads voxels $V_{i,j,k}, V_{i+1,j,k} \cdots V_{i+N_p-1,j,k}$ simultaneously from each memory connected to the pipeline using the base address (i, j, k). The fetched voxels, generated resampling positions $(\hat{i}, \hat{j}, \hat{k}) \cdots ((i+\hat{N_p}-1), \hat{j}, \hat{k})$, and sheared positions $(i^\dagger, j^\dagger, k^\dagger) \cdots ((i+N_p-1)^\dagger, j^\dagger, k^\dagger)$, are put into the convolution pipelines to get resampled data.

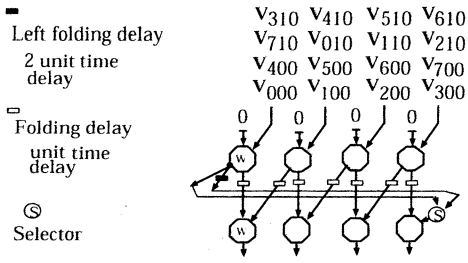
The resampled data are skewed so that *deskew* block deskews the resampled outputs by a multiplexer using address $(\hat{i}, \hat{j}\hat{k})$. The outputs of the deskewing are put into the rendering pipelines to carry out compositing along the parallelized rays.



(a) No folding.



(b) Timing shift of input data with delay.



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

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

Table 3 Number of time delays versus data dimension V and the number of processing pipelines N_P.

Location of delay	The number of unit-time
j-delay	V/N_P
k-delay	V^2/N_P
left folding delay	V/N_P
folding delay	1

6. Analysis of Proposed Architecture

6.1 Rendering timing

We estimated the timings for rendering volumes of practical sizes. The rendering time is directly related to the number of resampling operations to be performed. Since the resampling and other rendering operations can be fully pipelined, the pipeline cycle time can be equal to the memory access time T_m. This implies that the

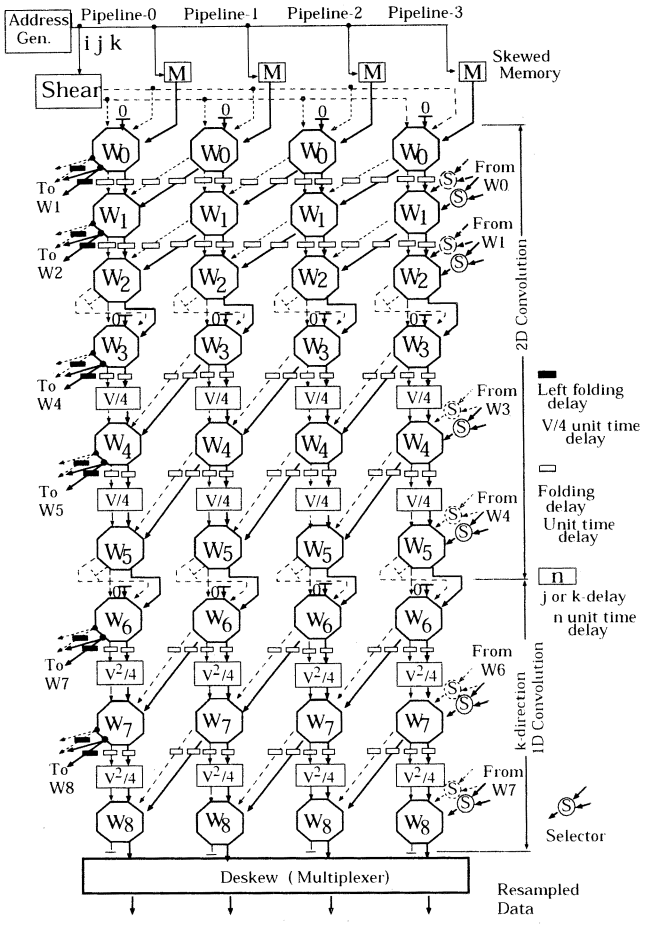


Fig. 9 3 × 3 × 3 parallel pipelined 3D convolution for general case: $V \gg N_P$, $N_p = 4$.

processing bottleneck is the memory access time. Accesses to the voxel memory are regular and deterministic for a given set of rendering parameters.

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:

$$N_s = V^3 N_f / N_p. \tag{13}$$

For each second,

$$N_s T_m \leqq 1. \tag{14}$$

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

$$V \leq \sqrt[3]{N_p/(N_f T_m)}. \tag{15}$$

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. 10**. The volume dimensions computed for several typical values of N_f and N_p are also shown

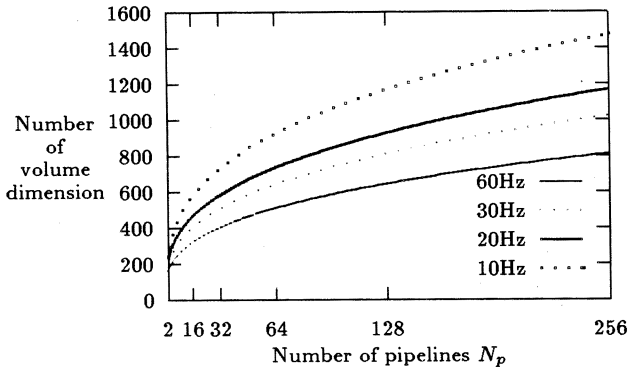


Fig. 10 Maximum volume dimensions.

in **Table 4**. These values verify that the proposed architecture can render volumes of practical sizes, i.e., more than 256^3 voxels, in real-time.

Table 4 Maximum volume dimensions for typical frame rate.

N_f	N_p (# pipelines)					
(frames/sec)	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

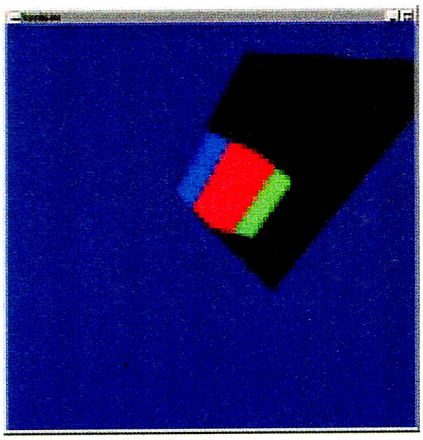
6.2 Processing scalability and expandability of the convolution area

Adding rendering pipelines increases the volume size for a fixed frame rate or the frame rate for a volume of fixed size. There are no architectural problems in adding rendering pipelines, because 1) in the voxel memory interface, each memory module in the voxel memory is connected one-to-one to a pipeline in the resampling module; 2) the resampling pipelines communicate only with the left and right pipelines in the resampling module; 3) in the interface between the resampling module and the rendering pipelines, each rendering pipeline is connected to one resampling pipeline; and 4) each pipeline communicates only with the left and right pipelines in inter-pipeline communications. Therefore, the proposed architecture has processing scalability.

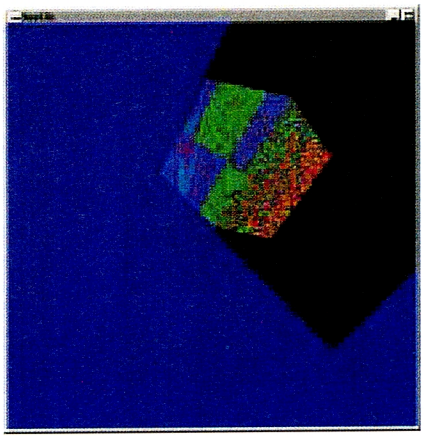
In addition, it is easy to increase the convolution area M in the architecture by just adding calculation units along with pipelines.

6.3 Image quality

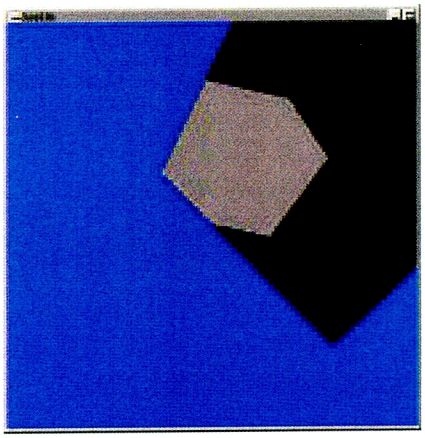
We built a software simulator to simulate the pipeline data flow of the proposed architecture, and verified the concepts of both parallel and perspective projections. To compare images, we built a sample-order ray-casting renderer that computes slices of samples perpendicular to the viewing vector and accumulates them to produce



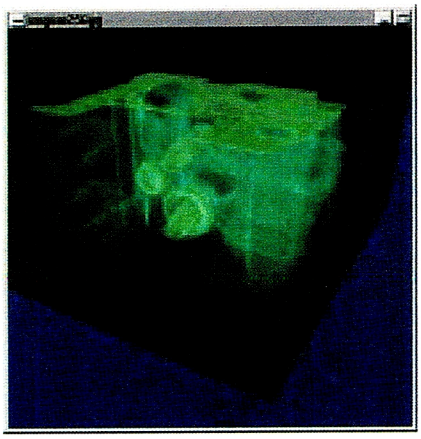
(a) Perspective projection with $2 \times 2 \times 2$ convolution.



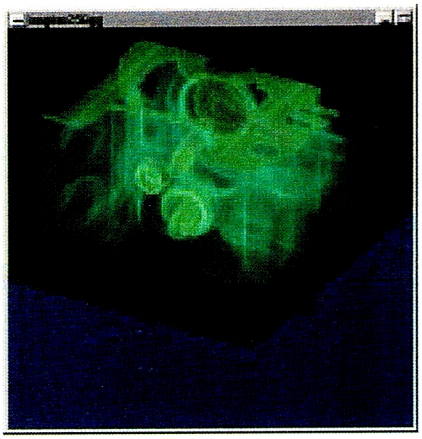
(b) Perspective projection by resampling the nearest neighbor voxels.



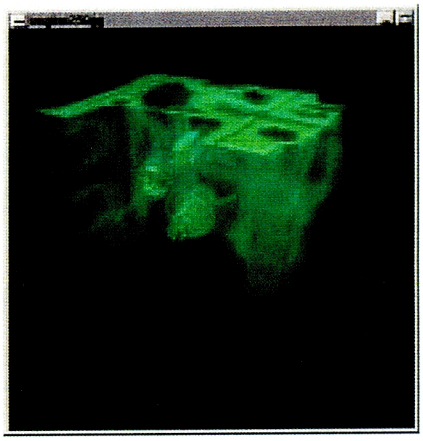
(c) Perspective projection with $3 \times 3 \times 3$ convolution.



(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. 11 Images generated with simulators.

the final image. We conducted several rendering experiments with these simulators.

Fig. 11 (a) shows a perspective image rendered from an opaque cube sized 64^3 to verify the perspective projection. We used a filter kernel based on the $2 \times 2 \times 2$ Lagrange formula in the resampling.

Figs.11 (b) and (c) show two perspective images rendered from an opaque checker-board cube sized 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. 11** (b) was generated by using the nearest neighbor voxel values in resampling and clearly shows the aliasing problem. The image in **Fig. 11** (c) was generated by using a $3 \times 3 \times 3$ box filter kernel in resampling, showing the antialiasing effect by a convolution.

Figs.11 (d), (e), and (f) show the images rendered from the engine block sized 256^3 used in Lacroute's rendering experiments[6] with a manually adjusted opacity table. **Fig. 11** (d) shows a perspective projection image, and **Fig. 11** (e) shows a parallel projection image for comparison. These two images were generated using a kernel based on the $3 \times 3 \times 3$ Lagrange formula.

Fig. 11 (f) is a perspective projection image generated by the sample-order ray-casting renderer with interpolations using multi-resolution datasets. There were 256 slices taken for this image, about the same number of slices (256) used in **Fig. 11** (d). The two images in **Figs. 11** (d) and (f) look comparable in quality.

7. Future Work

Error analysis is essential for hardware implementation, which is most likely to use fixed-point arithmetic. The application of resampling by convolution to render

a class of irregular volumes is another topic for future study.

8. Conclusion

We have proposed a convolution with a systolic array structure for perspective projection in real-time volume graphics based on the shear-warp method. In the original algorithm, the further the ray proceeds, the more voxels are required to calculation the convolution. This increase in required voxels makes it difficult to implement the algorithm in hardware. By assuming separability of the kernel weight for convolution, we implemented a 3D convolution with three serial 1D convolutions along the i, j and k axes, which reduces the number of calculation units from M^3 to $3M$, where the convolution is calculated for a M^3 area.

The number of pipelines for rays is V^2 for V^3 voxels datasets. If the hardware of a single pipeline can calculate V rays, each of the implemented pipelines is assigned to V theoretical pipelines (for V^2 rays). The number of hardware pipelines should be much smaller than the V theoretical pipelines in actual implementation. We folded the theoretical pipelines and reduced them to a certain number of hardware pipelines. We show the relation between this folding process and its necessary delay.

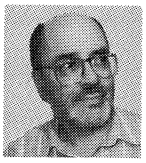
The architecture can generate an image of a 256^3 voxel dataset ($V=256$) at 30 Hz with four pipelines. In addition, the architecture can be easily extended for 512^3 ($V=512$) and 1024^3 ($V=1024$) dataset with 32 pipelines and 256 pipelines. Our architecture has processing scalability.

〔References〕

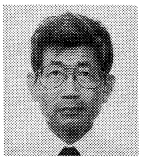
1) A. S. Glassner: "Principles of Digital Image Synthesis", Morgan& Kaufman (1995)
2) T. Günther, C. Poliwoda, C. Reinhart, J. Hesser, R. Männer, H.-P. Meinzer, and H.-J. Baur: "Virim: A massively parallel processor for real-time volume visualization in medicine", Computers & Graphics, Vol.19, No.5, pp.705-710 (1995)
3) A. Kaufman and R. Bakalash: "Memory and processing architecture for 3d voxel-based imagery", IEEE Computer Graphics & Applications, Vol.8, No.6, pp. 10-23 (Nov. 1988)
4) G. Knittel and W. Strasser: "A compact volume rendering accelerator", In Proceedings of the IEEE Symposium on Volume Visualization, pp. 67-74 (Oct. 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 Systems Laboratory, Stanford University (Sep. 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, pp. 451-457 (July 1994)
7) M. Levoy: "Efficient ray tracing of volume data", ACM Transactions on Graphics, Vol.9, No.3, pp. 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, pp. 20-29 (Oct. 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, pp. 131-138 (Aug. 1997)
10) 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 , pp. 47-54, San Fransisco, CA (Oct. 1996)
11) 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, pp. 131-138 (Aug. 1999)
12) L. Westover: "Footprint evaluation for volume rendering", In Proceedings of the ACM SIGGRAPH '94, pp. 144-153 (Oct. 1994)
13) L. Williams: "Pyramidal parametrics", In Proceedings of the ACM SIGGRAPH '83 Conference, pp. 1-11 (July 1983)

Masato Ogata is an assistant general manager of the research and development department at Mitsubishi Precision Co., Ltd. in Japan. His research background has involved 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 from Ohita National College of Technology in 1970, where he majored in Electrical Engineering. He received his M.S. from Yokohama National University in 1995. He is currently a Ph.D. student at Yokohama National University. He is a member of IEEE.

Hanspeter Pfister is a research scientist at MERL -A Mitsubishi Electric Research Laboratory- in Cambridge, MA, USA. 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, and VLSI design. He received his Dipl.-Ing. degree in electrical engineering from the Swiss Federal Institute of Technology in 1991, and a Ph.D. in Computer Science from the State University of New York in 1996. He is a member of ACM, IEEE, the IEEE Computer Society, and the Eurographics Association.



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



Yasunori Dohi is a Professor of Electrical Engineering and Computer Science at Yokohama National University. His research background has involved 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 his B.S. in 1962, an M.S. 1964, and a Ph.D. in Computer Science from Tokyo Institute of Technology in 1967 . He is a member of IEEE, and IEC.