A Parallel Computer Architecture for Volume Rendering

Yuji Tsushima
(Hitach Ltd., Japan)
Akinori Nakayama
(Kyoto University, Japan)

Hideya Akashi
(Hitachi Ltd., Japan)
Shin-ichiro Mori
(Kyoto University, Japan)
Shinji Tomita
(Kyoto University, Japan)

Jin Xidu
(Kyoto University, Japan)
Hiroshi Nakashima
(Kyoto University, Japan)

Abstract

This paper proposes the architecture of a high-speed volume rendering machine ReVolver. This machine has the capability of real-time perspective projection of translucent volumes with a new simplified voxel sampling method and 3-D memory fit for it. It also has a mechanism for ray-traced visualization of volume data in order to visualize obscure boundary objects which are hardly expressed in surface model.

1 Introduction

Volume rendering is a visualization technique for a three dimensional volume 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 computation power and resources, it is natural and necessary to exploit parallel processing techniques for high speed rendering. In fact, there have been various proposals of rendering architectures of special purpose parallel machines[1][2][3][4][5]. These machines, however, only support parallel projection and opaque volume data, because they mainly target medical use.

In contrast with those conventional machines, our machine ReVolver has the capability of perspective projection of translucent volumes, which is required to visualize scientific computation results. To make ReVolver have this capability, we devised a simplified voxel sampling method. We 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. Moreover, we introduced a sophisticated method for ray-traced rendering of volume data. This makes it possible to visualize obscure boundary objects, such as cloud, fog and smoke, which are hardly represented in surface model.

In the following sections, we will show the architecture of ReVolver as follows: In Section 2, various parallel volume rendering methods are briefly introduced: Section 3 and 4 describe the architecture and hardware configuration of ReVolver, respectively: Section 5 shows the mechanism for ray tracing: In Section 6, related works are surveyed contrasting functionality and performance of ReVolver with machines and algorithms proposed in them: Section 7 gives conclusions and future work plans.

2 Parallel Volume Rendering

Volume rendering requires a huge amount of computation power and memory space. To cope with it, various parallel methods are proposed and implemented for parallel volume rendering machines as shown below.

1. *Parallel backward mapping* In backward mapping, each ray is cast from the view point into the volume data through each pixel on the screen as shown in Figure 1(a) and (b). Parallelization of this algorithm is classified into the following two methods.

   Pixel parallel method The screen is subdivided and each sub-screen is allocated to a processing element (PE) as shown in Figure 1(a). Each PE casts rays through its own sub-screen and calculates the values of pixels in the sub-screen. This method is used in PARCUM II[1].

   Voxel parallel method The volume space is sliced and each plane or set of planes is allocated to a PE as shown in Figure 1(b). To calculate the pixel value of a ray, each PE takes charge of a voxel or a set of voxels through which the ray is cast. This method is used in CUBE[2].

   In addition, the combination of these methods is naturally conceivable.

2. *Parallel forward mapping* In forward mapping, the volume space is subdivided and sub-

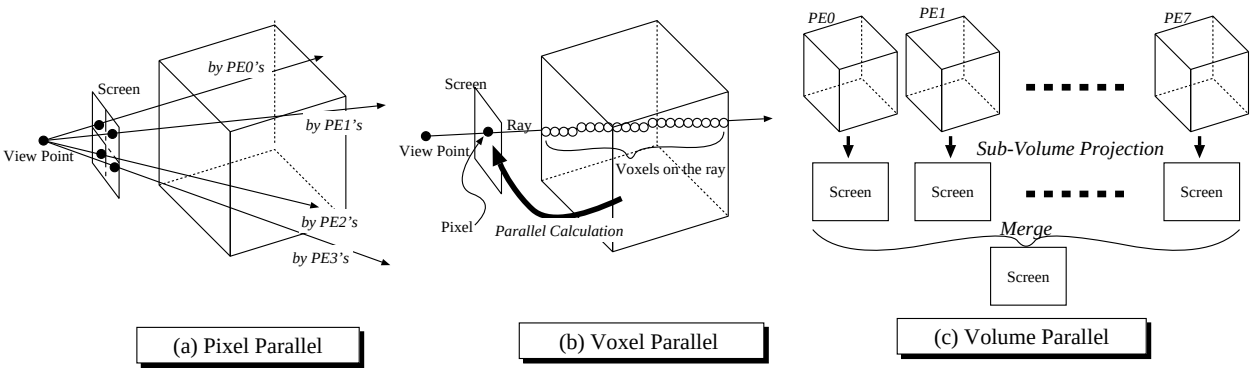

Figure 1: Parallel Volume Rendering Methods

volumes is allocated to a PE as shown in Figure 1(c). After each PE projects its own sub-volume onto the screen, projected images are merged taking account of spatial order of sub-volumes. This method is used in GODPA[3], SCOPE[4] and 3DP^4[5].

3 Architecture of ReVolver

3.1 Requirements

The target of ReVolver is not only visualization of medical volume data which most of conventional machines is suitable to, but also that of scientific data. Therefore, ReVolver should have following capability.

1. *Discrete and continuous models*
   In *discrete* model, it is assumed that a voxel is at the center of a unit cube and any points in the cube have 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 can be seen. The translucent rendering also copes with an *opaque* object, which conventional systems can visualize only, 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 to showing wide range of space. For small objects, such as human brains, *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 20 frames per second.

5. *Ray tracing*
   Cloud, fog and smoke are hardly represented in surface model because of their obscure boundaries. Realistic image of these objects will be given by modeling them as volumes and applying *ray tracing* method which detects their surfaces and generates reflected/refracted rays.

3.2 Key Design Issues

The requirements discussed above lead us to important architectural decisions as shown below.

1. *Parallel backward mapping*
   Translucent rendering requires that the spatial order of voxels which contribute a pixel must be exactly kept in the computation of the pixel value. Since forward mapping usually ignores the order in the computation of sub-volume, it cannot be suitable for ReVolver. Therefore, we must adopt *backward mapping* and parallelize it in order to achieve high performance for real-time rendering.

2. *Voxel parallel method*
   Since pixel parallel method generates many rays at a time, memory access conflict to the same voxel memory bank will frequently occur and will degrade performance. Therefore, we devised *pipelined voxel parallel* method in which all voxels on a ray are accessed at a time. The pixel value for the ray is calculated by a tree structured pipeline whose pitch is as fast as the volume memory cycle time. Therefore, a pixel value will be obtained in every memory cycle time so that the system follows the view point movement in real-time.

3. *Conflict free volume memory*
   The efficiency of the pipelined voxel parallel

method relies upon it that all voxel values on any ray can be read at a time. This requires us to devise a new voxel sampling method called *principal-axis oriented sampling* and a volume memory structure called *Parallel Treble Memory*. This memory structure does not only assure fully parallel access of the voxels on a ray, but also is efficient in the volume surface detection for ray tracing.

4 Hardware Configuration of ReVolver

4.1 System Configuration

Figure 2 shows the overview of the hardware configuration of ReVolver. The system consists of a control unit and a four-stage macro pipeline as follows.

1. *System Control Unit (SCU)*
   This unit gives other units various data/ commands, such as volume data, the positions of view point and screen, and rendering mode.

2. *Ray Casting Stage*
   This stage casts rays which start from the view point and penetrate pixels on the screen, and broadcast it to all banks of the Parallel Treble Memory. In order to generates a ray data in every machine cycle, the stage consists of 16 DSPs. In the ray tracing mode, this stage is given a set of parameters for reflected/refracted rays by the shading stage and converts it to two rays.

3. *Voxel Access Stage*
   This stage consists of two pipelined subunits, *Parallel Treble Memory* and *c/α lookup table*. Each subunit has n parallel nodes for n^3 volume space. A ray data given by the ray casting stage is converted to the addresses of voxels on the ray, and values of the voxels are read in parallel from the Parallel Treble Memory. Each value is converted to color, transparency and refractive index, by looking up them in the c/α look up table. Detailed description of these subunits is given in Section 4.2.

4. *Pixel Calculation Stage*
   This stage calculates the pixel value of a ray from the color and transparency of voxels obtained by the voxel access stage. In the usual rendering, the pixel value for a ray is calculated as follows. Let $v_0, v_1, \ldots, v_n$ be voxels on the ray spatially ordered from view point, c_i and α_i be color and transparency of v_i respectively. The pixel value P for the ray is

$$P = \sum_{i=0}^{n-1} c_i(1-\alpha_i)\prod_{j=0}^{i-1}\alpha_j . \qquad (1)$$

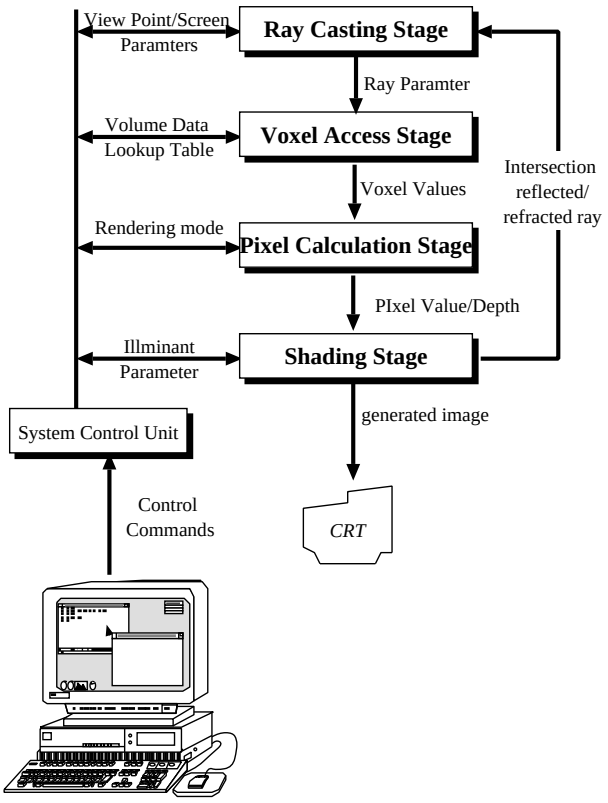

Figure 2: System Configuration of ReVolver

In Section 4.3, we will discuss the hardware configuration of the tree structured pipeline to calculated the equation (1).

In the ray-traced rendering, this stage detects the surface intersecting the ray and calculates the pixel value by equation (1) with respect to the voxels on view point side of the intersection point. The method of surface detection will be discussed in Section 5.1.

5. *Shading Stage*
   In the usual rendering, the pixel value from the pixel calculation stage is modified by depth gradient shading in this stage[9]. In the ray tracing, on the other hand, this stage calculates the normal of the surface at the intersection and the directions of reflected/refracted rays. This stage consists of two dimensional array of 64 DSPs.

4.2 Voxel Access Stage

4.2.1 Principal-Axis Oriented Sampling

Ordinarily, voxels on a ray are sampled at regular intervals without respect to the direction of the ray. This *ray oriented sampling* makes it hard to form a conflict free multi-bank memory as shown in Figure 3(a), if the direction of the ray is not restricted. Since ReVolver must cope with perspective projection in which rays have any directions, some other

sampling method is required.

Therefore we devised a new simplified sampling method, *principal-axis oriented sampling*, which is shown in Figure 3(b). This method doesn't samples voxels at regular intervals of ray itself, but of its *principal-axis* which is the axis corresponding to the ray vector element having largest absolute value. Therefore, if the sampling interval is equal to the unit of the volume coordinate system, the coordinates of any sampling points with respect to the principal-axis are different each other. This makes it possible to construct a bank-conflict free volume memory shown in Section 4.2.2.

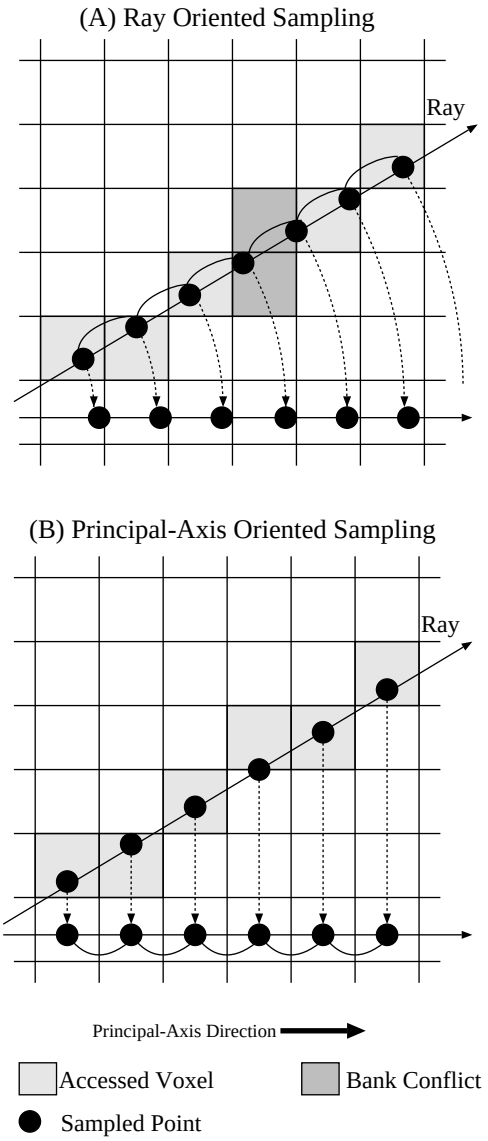

Figure 3: Ray and Principal-Axis Oriented Sampling

The problem of our method is that the interval with respect to the ray depends on the ray direction and varies from 1 to $\sqrt{3}$. This error is approximately compensated by modifying color c and transparency α as

$$\begin{aligned} c' &= c\alpha^{\Delta d-1} \\ \alpha' &= \alpha^{\Delta d} \end{aligned} \tag{2}$$

where Δd is the interval with respect to the ray. In ReVolver, this equation is not directly evaluated but simulated by the lookup table as described in Section 4.2.3.

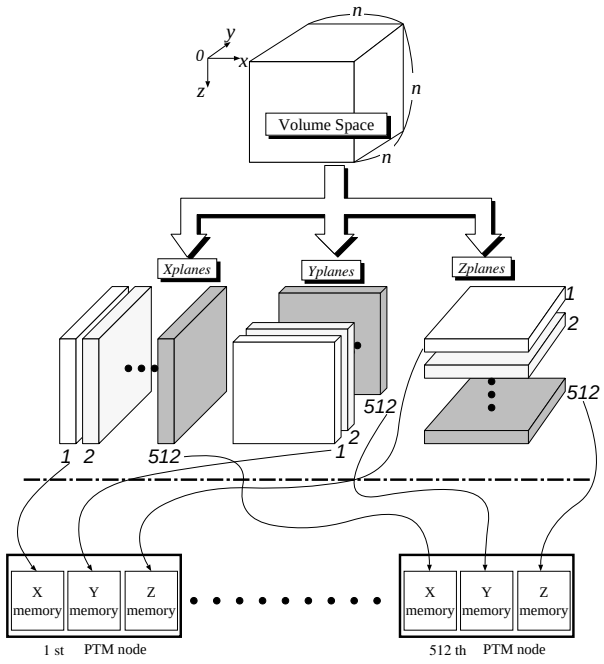

Figure 4: Parallel Treble Memory

4.2.2 Parallel Treble Memory

Suppose the volume space is a cube containing n^3 voxels, and it is sliced into n planes perpendicular to an axis of coordinates, say x-axis. If the volume memory is subdivided into n parallel accessible banks corresponding to the planes, the principal-axis oriented sampling ensures that all voxels on a ray can be read in fully parallel as far as ray's principal-axis is x.

However, the principal-axis may be y or z, and we want to avoid coordinate transformation in the memory to keep real-time followup to view point movement. Therefore, we sliced the volume space into planes not only perpendicular to x-axis but also to y and z, as shown in Figure 4. Each plane is numbered from 1 to n and three i-th planes, perpendicular to x, y and z-axes, are allocated on the i-th bank of the Parallel Treble Memory (PTM). Thus a voxel having the integer coordinates (i, j, k) is in i-th, j-th and k-th bank as a member of the planes perpendicular to x, y and z-axes respectively. That is, volume data is *trebly* stored in the PTM.

Now we have three sets of planes perpendicular to three axes and planes in a set are allocated separately.

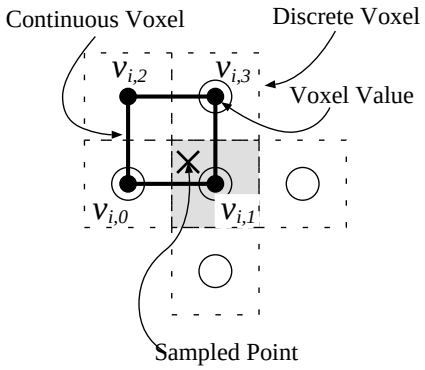

Figure 5: Access Methods of PTM

Thus the principal-axis oriented sampling works well on any ray direction, because we can choose a set of planes according to the principal-axis of the ray.

When a ray with principal-axis z is processed, voxels for sampling point (x, y, k) are accessed as follows, depending on the rendering mode.

1. Discrete Mode
   The voxel having integer coordinates $(\lfloor x + 1/2 \rfloor, \lfloor y+1/2 \rfloor, k)$ is accessed. That is, integer coordinate (i, j, k) represent points in the hatched area of Figure 5 and in the set P being

$$P = \{(x, y, k) \mid \quad i - \tfrac{1}{2} \le x < i + \tfrac{1}{2}, \quad j - \tfrac{1}{2} \le y < j + \tfrac{1}{2}\} . \tag{3}$$

2. Continuous Model
   The voxels indicated as filled circles in Figure 5 are accessed. That is, four voxels having the following coordinates are accessed and used for linear interpolation as described in Section 4.2.3.

$$\begin{array}{lll} v_{k,0} & : & (\lfloor x \rfloor, \lfloor y \rfloor, k), \\ v_{k,1} & : & (\lfloor x \rfloor + 1, \lfloor y \rfloor, k), \\ v_{k,2} & : & (\lfloor x \rfloor, \lfloor y \rfloor + 1, k), \\ v_{k,3} & : & (\lfloor x \rfloor + 1, \lfloor y \rfloor + 1, k) \end{array} \tag{4}$$

3. Ray Tracing Mode
   Let $i = \lfloor x + 1/2 \rfloor$ and $j = \lfloor y + 1/2 \rfloor$. Accessed voxels have coordinates (i, j, k) and $(i \pm 1, j \pm 1, k)$. That is, the voxel at (i, j, k) and its four neighbors indicated as empty circles in Figure 5 are accessed. The four neighbors are on the rays which are parallel to the ray passing through the center voxel. They are used for the calculation of normals of surfaces.

4.2.3 c/α Lookup Table

The value of a voxel read from the PTM does not represent color, transparency nor refractive index directly, but is used to make the index for the lookup table containing them. ReVolver has the following three modes to generate the table index as shown in Figure 6.

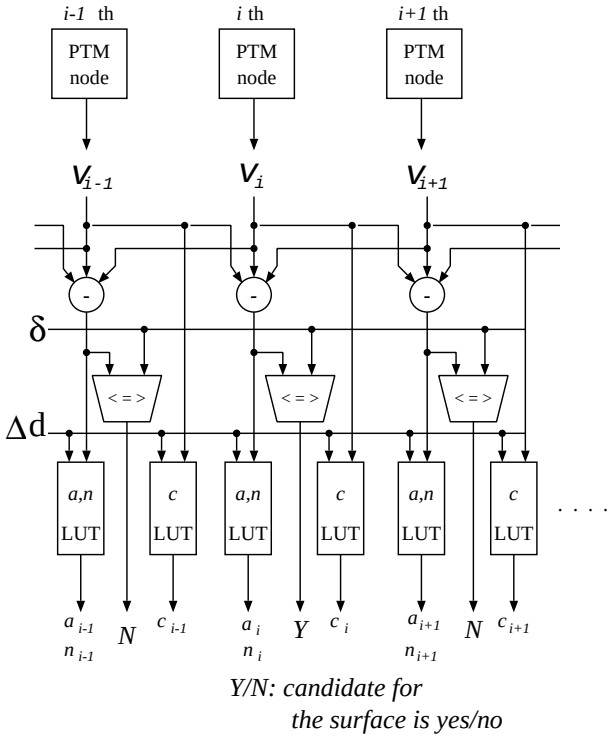

Figure 6: c/α Lookup Table

1. Discrete Mode
   The voxel value directly gives the index.

2. Discrete Differential Mode
   The index is the difference of voxel values on two adjacent planes. This mode will give a *differential* view of the volume emphasizing contrast of voxel values.

3. Continuous Mode
   The value of four voxels described in Section 4.2.2 is used to obtain the index by simplified tri-linear interpolation[6] as follows. Let (x, y, k) be a sampling point of a ray with principal axis z, $v_{k,0}$, $v_{k,1}$, $v_{k,2}$ and $v_{k,3}$ be the values of voxels defined in equation (4), and Δx and Δy be fraction part of x and y. The index value v_k for the sampling point is given by the equation

$$\begin{aligned} v_k = \quad & \Delta x \Delta y v_{k,0} + \quad (1 - \Delta x)\Delta y v_{k,1} + \\ & \Delta x (1 - \Delta y) v_{k,2} + \quad (1 - \Delta x)(1 - \Delta y) v_{k,3}. \end{aligned} \tag{5}$$

   Since the evaluation is performed by additional sub-stages between PTM and the lookup table, the throughput of the system is not degraded.

   Note that ordinary interpolation especially for the ray oriented sampling requires eight points, while four in our method owing to the principal-axis oriented sampling. This simplification re-

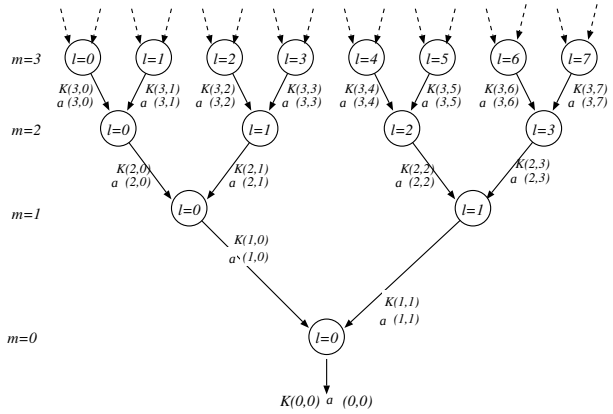

Figure 7: Pixel Calculation Tree

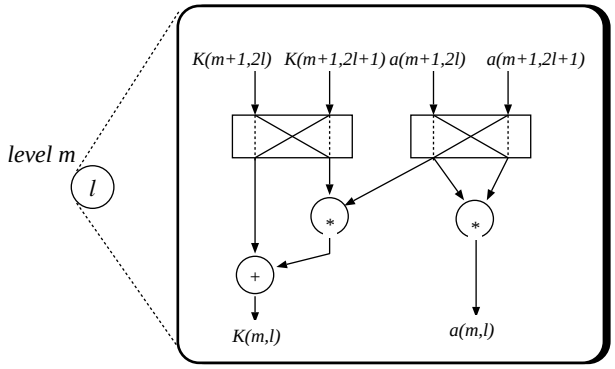

Figure 8: Pixel Calculation Node

duces not only computation cost but also communication cost between parallel nodes.

In addition, the sampling interval with respect of the ray, Δd, is combined with the index given above to obtain the real index of the table in order to compensate the error caused by the simplification of sampling.

For the ray tracing, each parallel node has a comparator to detect the surface. That is, the absolute value of the difference between two adjacent voxel values are compared with a predefined threshold δ as described in Section 5.1.

4.3 Pixel Calculation Stage

This stage consists of a tree structured pipeline, Pixel Calculation Tree (PCT) shown in Figure 7, which evaluates the equation (1). For the evaluation by the tree, we transform the equation (1) as follows.

First, we define $K(m,l)$ and $\alpha(m,l)$ as

$$\begin{aligned} K(m,l) &= \sum_{i=nl/2^m}^{n(l+1)/2^m-1} c_i(1-\alpha_i)\prod_{j=nl/2^m}^{i-1}\alpha_j \\ \alpha(m,l) &= \prod_{i=nl/2^m}^{n(l+1)/2^m-1}\alpha_i \end{aligned} \tag{6}$$

where $m = 0, 1, \ldots, \log_2(n-1)$ and $l = 0, 1, \ldots, 2^m - 1$.

From the equation (6), we have the following recursive definition of $K(m,l)$ and $\alpha(m,l)$.

$$\begin{aligned} K(m,l) &= K(m-1,2l) + \\ &\quad \alpha(m+1,2l)K(m+1,2l+1) \\ \alpha(m,l) &= \alpha(m+1,2l)\alpha(m+1,2l+1) \end{aligned} \tag{7}$$

On the other hand, $K(m,l)$ and $\alpha(m,l)$ have the following relationship with c_i, α_i and P in equation (1).

$$\begin{aligned} K(\log_2 n, i) &= c_i(1-\alpha_i) \\ \alpha(\log_2 n, i) &= \alpha_i \\ K(0,0) &= P \end{aligned} \tag{8}$$

Therefore, we will obtain P by starting from the initial value shown in (8) and evaluating (7) recursively.

Thus the pipelined tree PCT shown in Figure 7 evaluates P with nodes which perform one recursion step of (7). The PCT node is shown in Figure 8. The switches in the node pass through or exchange the partial results from lefthand and righthand predecessors according as the ray direction is left-to-right or right-to-left.

Since the tree has pipelined levels of $\log_2 n$ and nodes in a level work in parallel, we will obtain the pixel value of a ray in every machine cycle with $\log_2 n$ cycle delay.

4.4 Rendering Performance Estimation

The key factor limiting the system performance is the cycle and access time of the DRAM for PTM. It should be reliable estimation even for low cost ordinary DRAM that its cycle time for random access is 160 ns. Since the throughput of other parts of the system can be at least as high as PTM with appropriate parallelization and pipelining, we can reasonably expect that machine cycle will be 160 ns.

In the discrete mode rendering, PTM is accessed once for a ray and the pixel value is output in every machine cycle. Therefore, it takes $512^2 \times 160\,\text{ns} = 42\,\text{ms}$ to fill the 512^2 screen. This means the rendering performance is 24 frames per second which is enough to be called as *real-time* performance.

On the other hand, the continuous mode rendering takes more time because it requires to read four voxels in each PTM. However, performance degradation factor is less than four, because the pair of voxels can be read by the fast page mode in which the cycle time for the second access will be 80 ns or less. Therefore, it will take $(160 + 80) \times 2 = 480\,\text{ns}$ to access four voxels, and $512^2 \times 480\,\text{ns} = 125\,\text{ms}$ to fill the screen. This performance, 8 frames per second, should be still tolerable for most of real-time use.

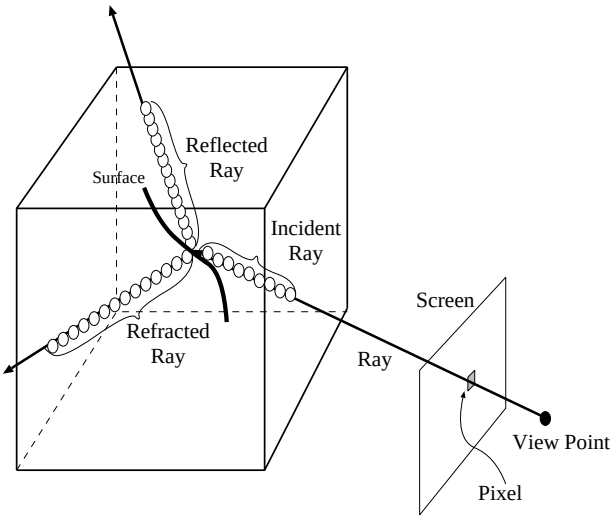

Figure 9: Volume Ray Tracing

5 Ray Tracing

In ray-traced rendering, a ray proliferates by reflection and refraction when it intersects the surface of an object as shown in Figure 9. Since the proliferation is repeated recursively until reflected/refracted rays escape from volume space or are attenuated, rays to be processed are much more than those in usual rendering. The volume memory architecture of Re-Volver is effective to keep ray-traced rendering time reasonably short, because it can process a reflected/refracted ray with any direction in a machine cycle.

The access of voxels on secondary and higher order rays, however, is a part of problem in ray tracing, because there are more difficult problems, *surface detection* and *normal calculation*. The following sections show our solutions of these problems.

5.1 Surface Detection

Since an object represented as volume data does not have a definite surface, it is not easy to find the point where a ray is reflected and refracted. We introduced a surface detection rule for volume data as follows. For each adjacent pair of voxels on the ray, the absolute difference of their value is calculated. If the difference is larger than a predefined threshold δ, the successor of the pair is regarded as the intersection of the ray and a surface.

For example, there are two potential intersections of the ray shown in Figure 10, $(i-1)$th and $(i+2)$th voxels. The $(i-1)$th voxel, however, is not an intersection because the absolute difference between its value and its predecessor's is less than δ, while $(i+2)$th has a large difference enough to be an intersection. If there are other candidate intersections, the voxel nearest to the view point is chosen.

The subtraction and comparison for the surface detection is performed in the c/α lookup table stage in parallel to the table lookup. The final intersection is chosen from the candidates by the Pixel Calculation Tree also in parallel to the evaluation of the recursive equation. Therefore, these operation will not bring any overheads into the ray processing.

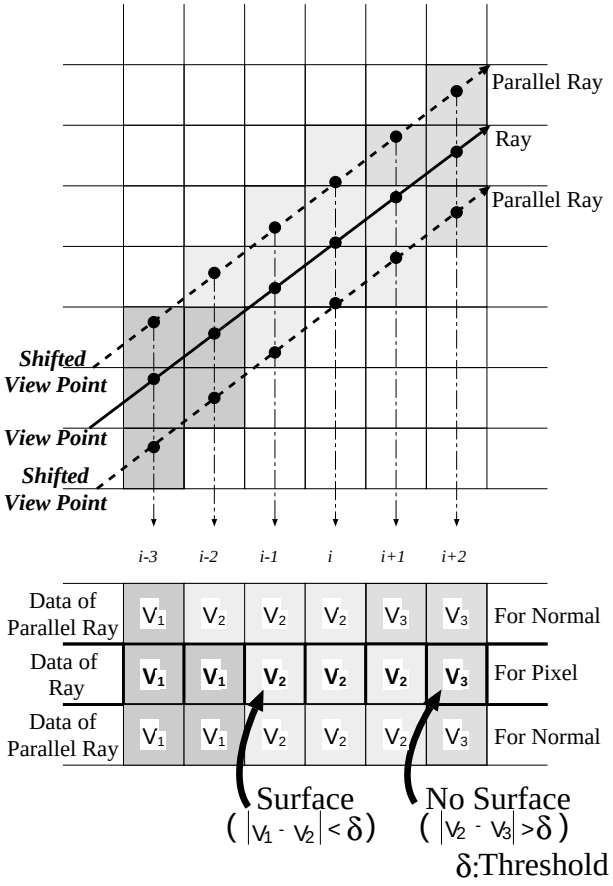

Figure 10: Surface Detection

5.2 Normal Calculation

Another difficulty lies in the calculation of reflected/refracted ray direction. This requires the normal of the surface at the intersection, which cannot be obtained only from the intersection. A few methods for the normal calculation, for example in [7], are proposed, but they bring irregular volume memory access which prevents high performance rendering.

Thus, we devised a simple method with *parallel rays* shown in Figure 10. A parallel ray has the same direction as the ray to be reflected/refracted but penetrates one of the four voxel adjacent to the intersection on the plane perpendicular to its principal-axis. Therefore, for a ray intersecting the surface at the point A, we have four parallel rays which have intersect N, W, S and E.

The normal vector $\vec{n}$ of the surface at the point A is defined as the mean of the normals of triangles

Table 1: Overview of Parallel Volume Rendering Machines

machine	mapping	parallel method	volume size	translucent/ perspective	performance (frame/sec)
GODPA[3]	forward	volume	256^3	No/No	20–30
SCOPE[4]	backward	volume	—	—/No	—
3DP^4[5]	forward	volume	$64^3 \times 256$	No/Yes	10
PARCUM II[1]	backward	pixel	512^3	No/No	—
CUBE[2]	backward	voxel	512^3	$*1/*1$	$15^{*2}/1.7^{*3}$
ReVolver	backward	voxel	512^3	Yes/Yes	24

*1 : Available only for non-real-time use. *2 : Without rotation. *3 : With rotation.

ANW, AWS, ASE and AEN. That is

$$\begin{aligned} \vec{n} \quad &= \overrightarrow{AN} \times \overrightarrow{AW} + \overrightarrow{AW} \times \overrightarrow{AS} + \\ &\quad \overrightarrow{AS} \times \overrightarrow{AE} + \overrightarrow{AE} \times \overrightarrow{AN} \end{aligned} \tag{9}$$

where $\vec{a} \times \vec{b}$ is the outer product of the vector $\vec{a}$ and $\vec{b}$.

With the vector $\vec{n}$ and the ray vector $\vec{r}$, we have vectors $\vec{u}$ and $\vec{v}$ representing reflected and refracted rays respectively as follows.

$$\begin{aligned} \vec{u} \quad &= 2\vec{n}' + \frac{\vec{r}'}{|\vec{r}' \cdot \vec{n}'|} \\ \vec{v} \quad &= k_f(\vec{n}' + \frac{\vec{r}'}{|\vec{r}' \cdot \vec{n}'|}) - \vec{n}' \end{aligned} \tag{10}$$

where $\vec{n}' = \vec{n}/|\vec{n}|$, $\vec{r}' = \vec{r}/|\vec{r}|$, and k_f is defined with refractive indices n_1 and n_2 of incidence and penetration as

$$k_f = \left(\left(\frac{n_1}{n_2} \right) \left| \frac{\vec{r}'}{\vec{r}' \cdot \vec{n}'} \right|^2 - \left| \frac{\vec{r}'}{\vec{r}' \cdot \vec{n}'} + \vec{n}' \right|^2 \right)^{-1/2}.$$

While the surface detection for parallel rays is performed in the volume access stage and pixel calculation stage, the the normal and reflected and refracted ray vectors are calculated by the shading stage.

5.3 Ray Tracing Performance Estimation

The performance of ray tracing strongly depends on the number of intersections of a ray and a surface. Here we assume a ray from the view point intersects surfaces at most three times until attenuation or escape from the volume space. Therefore, the number of rays reproduced from a ray from the view point is $2^0 + 2^1 + 2^2 + 2^3 = 15$.

For each ray, a PTM bank is accessed five times, one for the ray and four for its parallel rays. Since three voxels in a memory column can be accessed in the fast page mode while other two require random accesses, the total time of five accesses should be $(160 \times 3 + 80 \times 2) = 640$ ns with the parameters used in Section 4.4. The delay time for surface detection at the c/α lookup table and each PCT stage can be less than 80 ns so that the shading stage will get parameters of an intersection in every 640 ns. This cycle should be long enough for 64 DSPs of the shading stage to calculate the normal, reflection and refraction vectors without throughput degradation.

Thus the rendering performance will be $512^2 \times 15 \times 640 = 2520$ ms, or 0.4 frames per second. Although it is hard to say this performance is sufficient for real-time use, a user will not be irritated with the delay to obtain a quite fine and realistic volume image.

6 Related Works

Table 1 shows the overview of special purpose parallel volume rendering machines which have already been proposed. The last row for our ReVolver shows that it excels in both capability and performance. There have also been various research on parallel rendering with general purpose machines.

From those research works, this section shows two related works, one for special purpose and the other for general purpose machine, and compare their results with ours.

6.1 Example of Parallel Rendering Machine

The architecture of the parallel rendering machine CUBE[2] is relatively similar to ReVolver, because parallel backward mapping with volume parallel is adopted for it. However, the projection mechanism of CUBE is much more restricted than ReVolver. That is, CUBE deals with only parallel projection from six fixed view direction.

This restriction is caused by its volume memory architecture. CUBE has n memory banks for n^3 volume space and k-th bank contains voxels whose coordinates (x, y, z) satisfies

$$k = (x + y + z) \bmod n.$$

Therefore, if a ray is parallel to x, y or z axis, all voxels on the ray can be read without bank conflict so that CUBE achieves relatively high performance 64 ms per frame. However, this architecture naturally fits parallel projection only. Moreover, if a user

Table 2: Ray Tracing Peformance in [8]

# of PE	64	128	256	512
time (sec)	24.4	12.7	6.3	3.2

wants to have a view point with rays not parallel to axes, the contents of the volume memory should be rotated before the rendering paying a penalty more than 500 ms.

In contrast with CUBE, our ReVolver generates a rendered image in 42 ms with both parallel and perspective projection from any view points. This capability for real-time interactive use gives much advantage to ReVolver over CUBE.

6.2 Parallel Volume Ray Tracing on General Purpose Machine

Since the authors could not find any special purpose parallel machines which support volume ray tracing, here we show a work[8] with a general purpose parallel machine, CM-5. In this work, an image is generated by the volume parallel forward mapping method.

Table 2 shows the performance given in [8]. From this table we can conclude that ReVolver is faster than CM-5 with 512 processors, which is much more expensive than ReVolver.

7 Conclusions

This paper described the architecture of a high speed parallel volume rendering machine, ReVolver. The principal-axis oriented sampling method, the Parallel Treble Memory and tree structured pipeline are devised so that ReVolver makes an image in every 42 ms with perspective projection of translucent volumes. This real-time performance of the highly sophisticated rendering function makes ReVolver superior to other conventional machines.

Moreover, ReVolver has fine and realistic rendering modes, continuous model rendering and ray tracing. The principal-axis oriented sampling also acts an important role in continuous model rendering, and contributes to achieving high performance of 8 frames per second. As for the ray tracing, the parallel pipeline for usual rendering is sophisticatedly utilized for the surface detection and normal calculation with parallel rays. Owing to these methods, ReVolver gains high performance of ray tracing, 2.5 second per frame, and overcomes CM-5 with 512 processors.

We are now constructing a prototype of ReVolver[9]. Through the design of the prototype, we will evaluate the performance more precisely, not only about rendering speed but also about the quality of rendered images.

Acknowledgments

We would like express our thanks to all members in Tomita Laboratory of Kyoto University for their contribution through daily discussion. A part of this work was supported by the Grant-in-Aid #06402057 from the Ministry of Education, Science, Sports and Culture.

References

[1] D. Jackel and W. Strasser, Reconstructing Solids from Tomographic Scans —The PARCUM II System—, in *Advances in Computer Graphics Hardware II*, pp. 209–227, 1988.

[2] A. Kaufman and R. Bakalash, Memory and Processing Architecture for 3D Based Imagery, in *IEEE Computer Graphics & Applications*, Vol. 8, No. 6, pp.10–23, Nov., 1988.

[3] S. M. Goldwasser, A Generalized Object Display Processor Architecture, in *IEEE Computer Graphics & Applications*, Vol. 4, No. 10, pp.10–23, Oct., 1984.

[4] T. Uchiki and M. Tokoro, Solid and Colored Object Projection Environment —SCOPE—, in *Trans. IEICE*, Vol. J68-D, No. 4, pp.741–748, Apr., 1985. (In Japanese)

[5] T. Ohash, T. Uchiki and M. Tokoro, A Three-Dimensional Shaded Display Method for Voxel-Based Representation, in *Proc. EUROGRAPHICS 85*, pp. 221–232, 1985.

[6] M. Levoy, Display of Surfaces from Volume Data, in *IEEE Computer Graphics & Applications*, Vol. 8, No. 5, pp. 29–37, May, 1988.

[7] N. Tayama, N. Shimizu and F. Uruma, A Speedy Voxel-Tracing Method by Use of a Variable Stepping DDA in 3D Memory Space, in *Trans. IPSJ*, Vol. 32, No. 5, pp. 581–589, May, 1991. (In Japanese)

[8] K. L. Ma, J. S. Painter, C. D. Hansen and M. F. Krogh, A Data Distributed Parallel Algorithm for Ray-Traced Volume Rendering, in *Proc. Parallel Rendering Symp.*, pp. 15–22, 1993.

[9] Y. Tsushima, The Parallel Computer for Volume Rendering -*ReVolver*/C40-, in *Proc. Joint Symp. On Parallel Processing*, pp. 11–18, 1995.