ボリュームレンダリング専用並列計算機 *ReVolver*/C40 の レイキャスティングステージの設計

本多 俊之

平成8年2月15日

内容梗概

ボリュームレンダリングは物体の内部を可視化する手法として、従来から医療用画像の生成手法として重要な地位を占めてきた。更に近年では、構造解析や流体力学の分野においても、スーパコンピュータなど高速な計算機を利用した科学技術計算の結果の解析手法として、ボリュームレンダリングは注目を集めている。しかし、この手法は膨大な記憶容量と計算量を必要とするため、高速な画像生成は困難であった。

そこで、我々はボリュームレンダリング専用並列計算機 *ReVolver*/C40 を現在開発している。*ReVolver*/C40 は三つのステージからなり、各ステージはそれぞれ視線生成処理 / ピクセル値(色や透明度)の計算 / シェーディング処理を行う。これら三つのステージのパイプライン動作によって高速な画像生成が可能となる。また、各ステージについても並列処理やパイプライン処理により画像生成の高速化を図っている。

ReVolver/C40 においてはレイキャスティング法を用いており、視点とスクリーンのピクセルとを結ぶ視線上のボクセル群をサンプリングすることにより、ピクセルの輝度計算が行われる。ここで、サンプリング時に必要となる、視線に関するデータを生成する処理を視線生成処理と呼び、本稿では、この視線生成処理を行うレイキャスティングステージの設計について述べる。

レイキャスティングステージの設計では、(1) 視線生成処理の高速化 (2) 視線生成処理と通信との独立化 (3) 他ステージの負荷の軽減 (4) 柔軟性 を設計の方針とした。

各ピクセルの視線生成処理には本質的な依存関係が無く、並列処理により高速化可能である。そこで、レイキャスティングステージを8台のプロセッサ (PE) からなるマルチプロセッサシステムとした。これらの PE には、スクリーンをスキャンライン単位で分割した領域をサイクリックに割り当てる。各 PE は、担当領域内のピクセルに対する視線生成処理を、他の PE の処理とは独立に実行する。

レイキャスティングステージでは、視点およびスクリーンのパラメータをホストから受信する場合と、生成したデータを次のピクセル値計算ステージに送信する場合に通信が起こる。双方のステージで準備ができてから通信を行う同期通信では、通信が視線生成処理の大きな遅れを起こす可能性がある。そこで、ステージ間通信は共有メモリを介して非同期に行う。ここで、ホストとの共有メモリを View Buffer、ピクセル値計算ステージとの共有メモリを Ray FIFO Buffer と呼ぶ。

View Buffer、Ray FIFO Buffer はいずれも PE 間の共有メモリでもある。このため PE 間での調停制御が必要となるが、これには仮想トークンリング方式を採用した。更に、共有メモリに対する排他制御も可能にした。

PE には汎用の高性能 DSP を使用し、柔軟性を高めた。DSP は CPU とは独立にメモリ間転送を行える DMA を内蔵している。ここで、生成したデータの Ray FIFO Buffer への転送に DMA を使用することにより、 CPU による視線生成処理と DMA によるデータ転送が独立に実行可能となる。

また、 Ray FIFO Buffer への転送を行う際には、生成したデータを圧縮し、一定数蓄積した後に転送することで、バスの使用効率の向上を図っている。

ReVolver/C40 では、ピクセル値計算ステージが全体を律速している。そこで、このステージの負荷を軽減するため、レイキャスティングステージの各 PE は、視線が可視化の対象となるボリューム空間を通過するか否かを判断し、通過しない場合にはデータを送信しない設計とした。

PE にはそれぞれローカルメモリを設け、更にメモリから Ray FIFO Buffer への直接の転送が可能となるような機構も設計した。これにより、種々のアルゴリズムの実行が可能である。

以上のような設計方針で詳細設計を行った結果、レイキャスティングステージでは、秒間約 4.6 フレームのデータ生成が可能であることが判明した。

Abstract

Volume rendering is a technique to visualize the inner structure of various objects. It has been a important method in the field of medical image generation. Nowadays, it is gradually regarded also as a powerful analysis tool for scientific computations such as structual analyses and fluid dynamics computation on high-performance supercomputers. Since volume rendering requires an enormous amount of storage and computational power, it is difficult to realize high-speed image generation.

We are developing a special purpose parallel machine, *ReVolover*/C40, for volume rendering. This machine can generate images fast by pipelining the processes of three stages : the Ray Casting Stage, the Pixel Calculation Stage, and the Shading Stage. Parallel processing or pipelining are used on each stage for high-speed image generation.

Based on the ray-casting algorithm, the values (color and opacity) of a pixel are based on the values of sampled voxels on the ray which starts from the viewing point and goes through the pixel and the volume. Several data concerning the ray are generated in the ray generation process.

This paper describes a design for the Ray Casting Stage which executes ray generation processes in *ReVolver*/C40.

In designing for the Ray Casting Stage, there are four elements to be considered: (1) acceleration of the ray generation process, (2) parallelization between the ray generation process and the data communication, (3) lightening of loads of the other stages, and (4) flexibility.

Because ray generation processes for a pixel is inherently independent of those for other pixels, they can be speeded up by parallel processing. Therefore, I designed the Ray Casting Stage as a multiprocessor system composed of 8 Processing Elements (PEs). For parallel processing a screen is devided by scanlines into 8 areas and each area is assigned cyclically to a PE. Each PE executes ray generation processes for the area assigned to that PE, independently of other PEs.

Data communications are made when the Ray Casting Stage receives the viewing points and screens and when it transmits generated data to the next Pixel Calculation Stage. When the data communication does not start until both stages are ready, the data communications make the delay of the ray generation larger. In order to avoid this delay, memories shared between the stages are used for the asynchronous data communication. The memory shared with the host machine is called the View Buffer, and the memory shared with the Pixel Calculation Stage is called the Ray FIFO Buffer.

Since both the View Buffer and the Ray FIFO Buffer are also shared among the PEs, arbitration among PEs is essential. Virtual token rings are used for the arbitration. Furthermore, I designed a system where each PE can access the shared memory exclusively.

High-performance general purpose DSPs are used as PEs for flexibility. A DSP has a DMA which transmitts data from memory to memory independently of CPU. If the DMA is used for transmission

of generated data to the Ray FIFO Buffer, the ray generation process by CPU and the transmission by DMA can be executed in parallel.

Generated data are compressed before the transmission to the Ray FIFO Buffers and transmitted together without any idle cycle during the transmission, leading to improvement in efficiency of bus usage.

The speed of *ReVolver*/C40 depends on that of the Pixel Calculation Stage. In order to lighten the loads of this stage, in the Ray Casting Stage each PE checks if the ray passes through the volume space to be visualized. If it does not pass through that space, the PE transmitts no data to the stage.

Each PE has an external local memory and I designed a system which trasmitts data from the memory to the Ray FIFO Buffer directly, which enables execution of various algorithms in each PE.

As a result of the detailed design based on the above-mentioned method, the Ray Casting Stage proved to be capable of generating data at 4.6 frames/sec.

1　はじめに

ボリュームレンダリングは、物体内部の可視化が可能であるという特徴を持っており、従来、医療画像分野において重要な画像生成法であった。更に近年では、スーパーコンピュータなどの高速な計算機を利用した構造解析や流体力学をはじめとする科学技術計算結果の解析 理解の手段としても、ボリュームレンダリングは注目を集めている。この手法は、膨大な記憶容量と計算量を必要とするため、並列アルゴリズムの開発とともに専用計算機の開発が行われてきた。しかし、既存の専用計算機では、平行投影が主流であり、物体の大きさを強調した表示などはできなかった。

現在我々は、医療画像生成ならびに科学技術計算の結果の可視化を目的とし、遠近法を用いて半透明ボリュームが高速に表示できるボリュームレンダリング専用並列計算機 *ReVolver*/C40 を開発中である。*ReVolver*/C40 では、高速表示のために、サンプリング方法を単純化し、コンフリクトの無いメモリ構成を採用する。更に全体を (1) 視線ベクトルの生成、(2) ピクセル値の計算、(3) シェーディング処理の三つに分けパイプライン化することで高速化を図っている。

本稿では、*ReVolver*/C40 において (1) の視線ベクトルの生成を行うレイキャスティングステージの処理と構成について述べる。以下、第 2 章においては背景として *ReVolver*/C40 の概要について述べ、第 3 章においてレイキャスティングステージの主な処理を述べる。第 4 章ではレイキャスティングステージの構成について述べ、第 5 章で評価を行い、第 6 章でまとめを述べる。

2　*ReVolver*/C40 の概要

2.1　ボリュームレンダリング

2.1.1　ボリュームレンダリングの概要

ボリュームレンダリングにおいて、可視化の対象となる空間をボリューム空間と呼ぶ。ボリューム空間には、科学技術計算の結果や自然現象などのデータセットが色や透明度などに変換されて置かれる。ボリュームレンダリングは、ボリューム空間からサーフェイスを算出してレンダリングを行うサーフェイスレンダリングとは異なり、ボリュームの色や透明度を直接スクリーンに射影する手法である [1, 2]。

ReVolver/C40 では、ボリューム空間は立方体であり、ある頂点を含む 3 辺がワールド座標系における各座標軸（x, y, z 軸）の正の方向にあるように設定される。ボリュームを立方体格子に分割し、離散的にデータを与えたものをボクセルと呼び、各ボクセルには色や透明度の値（ボクセル値）が与えられる。

ボクセル値を格納したメモリをボリュームメモリと呼ぶが、*ReVolver*/C40 では、メモリ容量を抑えるために、そのボクセルの色や透明度を格納した属性テーブルのインデックスをボリュームメモリに格納する。

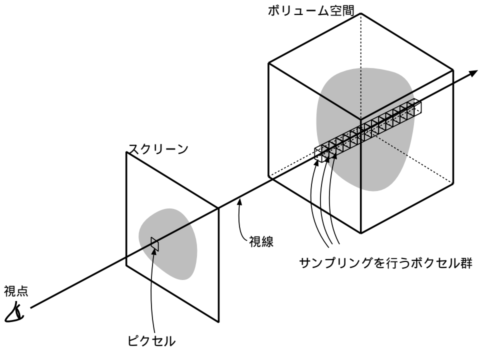

図 1: レイキャスティング法

2.1.2　レイキャスティング法

ReVolver/C40 では、投影法としてレイキャスティング法を用いている。

ワールド座標系において、視点からスクリーン上のあるピクセルの中心へと向かうベクトルのことを視線ベクトルと呼び、視点を一端としピクセルの中心を通る半直線を視線と呼ぶ。

図 1 のように、レイキャスティング法では、ピクセルごとに視線ベクトルを生成し、その視線上のボクセル群のボクセル値をスクリーンに投影する。

2.2　*ReVolver*/C40 の特徴

ReVolver/C40 は、最大 512^3 ボクセルのボリューム空間を持ち、512^2 ピクセルのスクリーンへの表示を行うボリュームレンダリング専用並列計算機であ

る [3]。*ReVolver*/C40 では、可視化の対象となる物体の内部構造も表示できるように、図 2 のようにボリュームを半透明なものとして扱う [4]。このため、

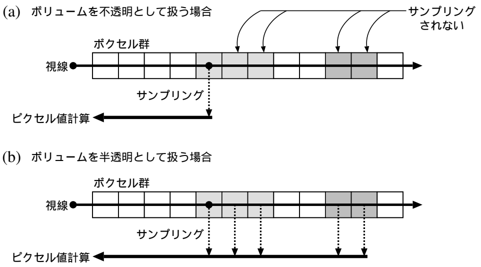

図 2: 半透明ボリュームと不透明ボリュームとの違い

視線上の無色・透明でない全ボクセルが順にアクセスされる必要がある。そして読み出されたボクセル値はすべてピクセルの色や透明度（以下、ピクセル値と呼ぶ）に反映される。

また、視点の位置や視線の方向に制限が無く、遠近法による奥行きのある画像表示が可能である。

このような画像表示を高速に行うために、*ReVolver*/C40 が有している特徴を以下に示す。

1. サンプリング方法の単純化

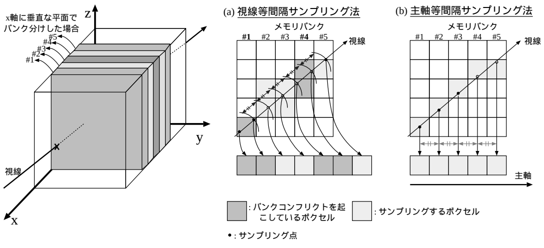

図 3: ボクセルのサンプリング方法

従来のサンプリング方法は、視線の方向に対して等間隔にボクセルをサンプリングするものであった。しかし、図 3 (a) のようなバンク構成でこのサンプリング法を用いると、コンフリクトを生じてしまう。そこで、*ReVolver*/C40 では主軸等間隔サンプリング法と呼ぶ手法を用いる。この手法においては、視線ベクトルの成分の中で絶対値が最大であるものを選び、その座標軸を主軸と定義する。そして、図 3 (b) のように、この主軸の方向に対して等間隔にサンプリングを行う。

2. バンクコンフリクトフリーなメモリ構成

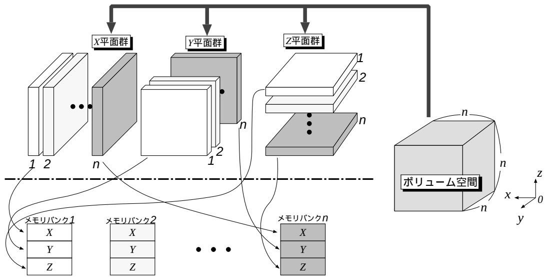

図 4: バンクコンフリクトフリーなメモリ構成

主軸等間隔サンプリング法では、主軸に垂直な平面でボクセル空間をバンク分けすれば、その主軸に対してバンクコンフリクトフリーとすることができる。しかし、主軸は一つには定まらないので、図 4のように、ボリュームデータを 3 重化して、主軸が x, y, z 軸のそれぞれの場合に対してバンクを設ける。すなわち、座標（p, q, r）にあるボクセルは、メモリバンク p の X 領域、メモリバンク q の Y 領域、メモリバンク r の Z 領域の 3 箇所に格納される。この構成により、視線が任意方向であっても、その主軸に対応したメモリバンクからボクセル値を読み出すことでバンクコンフリクトフリーとなる [5]。

3. 並列処理による高速化

2.3 節で述べるように、*ReVolver*/C40 においては、3 ステージのマクロパイプラインと各ステージ内の並列処理によって高速化を図っている。

2.3 *ReVolver*/C40 の構成

図 5 に示すように、*ReVolver*/C40 は、三つのステージと、全体の制御を行う System Controll Unit（以下、SCU と呼ぶ）から構成される。

1. System Controll Unit（SCU）

   ReVolver/C40 全体の制御を行う。ホストマシンからコマンド、プログラム、ボクセル値を受

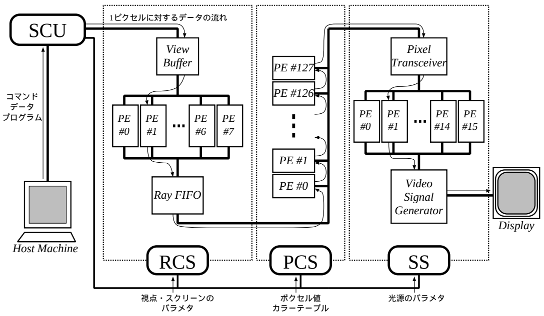

図 5: *ReVolver*/C40 の全体構成

け取り、受信したデータを各ステージに転送する。

2. レイキャスティングステージ（Ray Casting Stage : RCS）

   スクリーン上のすべてのピクセルについて、視点からピクセルへ向かう視線ベクトルを生成し、ボリュームと交差する視線に関するデータをピクセル値計算ステージへ転送する。視線生成処理はスクリーンの分割を行って 8 台のプロセッシングエレメント（PE）に割り当て、並列実行させる。

3. ピクセル値計算ステージ（Pixel Calculation Stage : PCS）

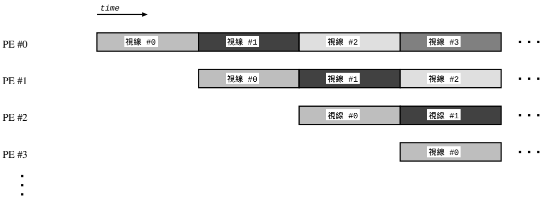

図 6: PCS における視線のパイプライン処理

   RCS から投入された視線に対し、視線上のボクセルをサンプリングしてボクセル値を読み出し、その値から属性テーブルを参照してピクセル値の計算を行う。PCS では、128 台の PE を用い、図 6 のように、視線ごとにパイプラインでボクセルのサンプリングを行う [6]。また、視線上のボクセルで、無色透明でなく視点に最も近いものを視線と物体表面との交点と定める。交点は次のシェーディング処理において必要である。交点の座標はピクセル値とともに隣接 PE へ転送され、最終的には次のシェーディングステージへ転送される。

4. シェーディングステージ（Shading Stage : SS）

   PCS からピクセル値と交点の座標を受け取り、交点とスクリーンの距離（以下、深さと呼ぶ）を求める。各ピクセルに対する深さ情報から物体表面の法線を近似計算して輝度値を求める depth griadient shading を行う [7]。シェーディング処理は、スクリーン分割を行い 16 台の PE で並列実行させる [8, 9]。

3 レイキャスティングステージの視線生成処理

3.1 RCS の入出力データ

入力データ SCU から受け取る視点とスクリーンに関するパラメータがビューデータである。このビューデータの構成を表 1 に示す。RCS は、これらのデータとピクセル座標から視線ベクトルを求める（図 7 参照）。

表 1: 視点、スクリーンのパラメータ（ビューデータ）

視点 $\vec{E}$	32-bit float x 3	ワールド座標系における視点の座標。
スクリーン		
$\vec{S}_v$	32-bit float x 3	スクリーンの垂直方向の単位ベクトル
$\vec{S}_h$	32-bit float x 3	スクリーンの水平方向の単位ベクトル
$\vec{S}_b$	32-bit float x 3	視点からスクリーンの右上頂点へのベクトル

出力データ PCS におけるボクセルサンプリング、および SS におけるシェーディングに必要なデータが視線データである。視線データの構成を表 2 に示し、これらの関係を図 8 に示す。

3.2 視線生成処理の内容

RCS は、ビューデータをもとに視点からスクリーンへ向かう視線ベクトルを生成し、その視線ベクトルの中で、ボリューム空間を通過するものについて視線データを生成する。この処理を視線生成処理と呼ぶ。視線生成処理を順に以下の節で述べる。

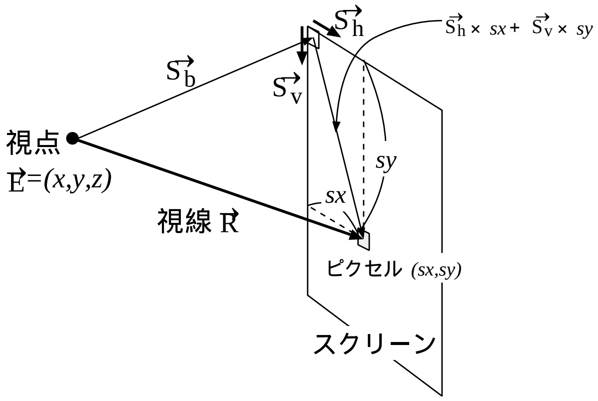

図 7: ビューデータの相互関係

表 2: 視線データ

AXIS (主軸名)	2 bits	主軸が x, y, z に対し、 それぞれ 2 進で 01、10、00 となる。 PCS はこれによりボクセルを 読み出すメモリ領域を選択する。
DIR (主軸方向)	1 bit	PCS におけるピクセル値計算 および SS におけ る深さ計算に必要。
BV1、BV2 (基本ベクトル)	10 bits χ 2	主軸成分が 1 である視線ベクトルの 他成分を BV1、BV2 (注) とする。 符号部 1bit、小数点以下 9bit。 PCS で読み出すボクセル のアドレス計算に必要。
BLEN (基本距離)	32-bit float	基本ベクトルの長さ。 SS で必要。
IN1、IN2 (初期座標)	20 bits χ 2	PCS で最初にボクセル サンプリングが行われる 際の平面上アドレス。 符号部 1 bit、整数部 10 bits、 小数点以下 9 bits
ILEN (初期距離)	32-bit float	視点から初期座標 までの距離。SS で必要。
PXL.x、PXL.y (スクリーン座標)	9 bits χ 2	ピクセルのスクリーン座標。 SS で必要。
MASK (サンプリング開始位置)	9 bits	視点がボリュームの内部に あるときなど、ボクセル 読み出しが一部不要になる 部分を PCS はこの値により 判断する。

(注) (主軸 ,BV1, BV2) の組み合わせは、(x, y, z), (y, z, x), (z, x, y) とする。IN1、IN2 も同様である。

3.2.1 視線ベクトルの生成

スクリーン座標における水平および垂直方向の単位ベクトルを、それぞれワールド座標系で表したものを $\vec{S}_h$、$\vec{S}_v$ とし、視点からスクリーンの左上 (スクリーン座標における原点) に向かうベクトルを $\vec{S}_b$ とする。ここで、スクリーン座標が $(\mathrm{s_x}, \mathrm{s_y})$ であるピクセルに対する視線ベクトルを $\boldsymbol{\vec{R}}$ とすると、$\vec{R}$ は次式 (1) で表される。

$$\vec{R} = \vec{S}_b + \mathrm{s_x}\vec{S}_h + \mathrm{s_y}\vec{S}_v \qquad (1)$$

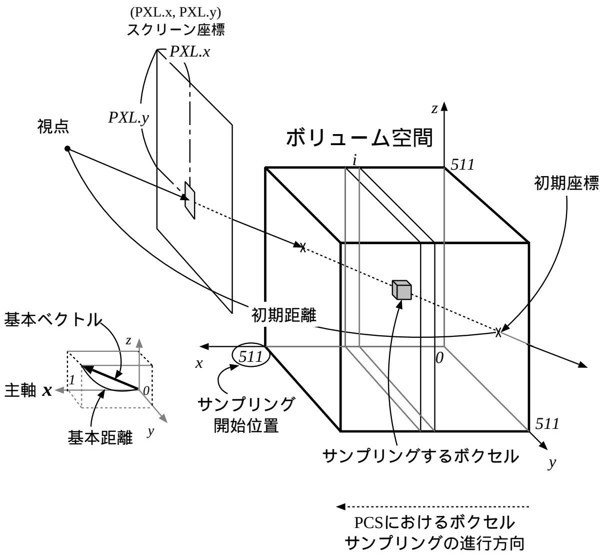

図 8: 視線データの相互関係

3.2.2 主軸及び主軸方向の判定

PCS で保持しているボリュームデータは、主軸が x, y, z 軸のいずれであっても対応できるように 3 重化され、それぞれの軸に対してバンク分けされている。そのため、ボクセルの読み出しにおいては、主軸名 **AXIS** が必要となる。また、視線ベクトルの符号によりピクセル値計算の方法が異なるので、主軸方向 **DIR** を求める。$\vec{R}$ の成分の中で絶対値が最大のものが主軸となり、その符号が主軸方向となる。

3.2.3 基本ベクトルの計算

主軸成分が 1 である視線ベクトルを、基本ベクトル $\boldsymbol{\vec{B}}_v$ と呼び、$\vec{R}$ の主軸成分が $\mathrm{R_i}$ ($\mathrm{i} \in \{\mathrm{x, y, z}\}$) であるとき、$\vec{B}_v$ は次式 (2) で表される。

$$\vec{B}_v = \frac{1}{\mathrm{R_i}}\vec{R} \qquad (2)$$

視線上の点で視線の主軸成分が 0 となる点 (初期座標) を $\vec{R}_0$ とすると、PCS において m 番目 ($\mathrm{m} \in \{0, 1, ..., 511\}$) のサンプリングで読み出されるボクセルのアドレスは、$\vec{B}_v$ を用いて

$$\vec{R}_0 + m \times \vec{B}_v \qquad (3)$$

によって求めることができる。また、ここで基本ベクトルの大きさ **BLEN** も求める。これは、後に求

める視点から $\vec{R}_0$ までの距離（以下、初期距離と呼ぶ）とともに、SS におけるシェーディングに用いられる。BLEN は、$\vec{B}_v$ の主軸でない成分を **BV1**, **BV2** とすると

$$\mathrm{BLEN} = \sqrt{1 + \mathrm{BV1}^2 + \mathrm{BV2}^2} \tag{4}$$

により求まる。

3.2.4 視線とボリューム空間の交差判定

ボリューム空間を全く通過しない視線については、後のステージへ視線データを転送しない。そこで、視線がボリューム空間を通過するか否かの判定が必要となる。この判定は、以下の二つの点について行う。

1. 視線方向による判定

   主軸成分に着目し、視点の成分が負で主軸方向が負の場合、および視点の成分が 511 より大きく主軸方向が正の場合は、視線がボリューム空間へ向かわないのでこのピクセルに対する視線生成処理を終了する。

2. ボリューム空間の主軸成分における端点の座標による判定

   まず、視線上の点で主軸成分の座標がボリューム空間の両端である 0, 511 に等しい点 $\vec{R}_0$, $\vec{R}_{511}$ の座標をそれぞれ求める。ここでは、例として主軸が x である場合を考えて、$\vec{R}_0 = (0, \mathrm{R_{y0}}, \mathrm{R_{z0}})$, $\vec{\mathrm{R}}_{511} = (511, \mathrm{R_{y511}}, \mathrm{R_{z511}})$ とする。次に、図 9 (a) のように、視線とボリューム空間の xy 平面及び zx 平面への投影図を考え、それぞれの投影図において視線の投影とボリューム空間の投影が交差を判定する。その結果、いずれの投影図においても視線とボリューム空間の投影が交差する場合のみ、視線がボリューム空間を通過すると考えられる。

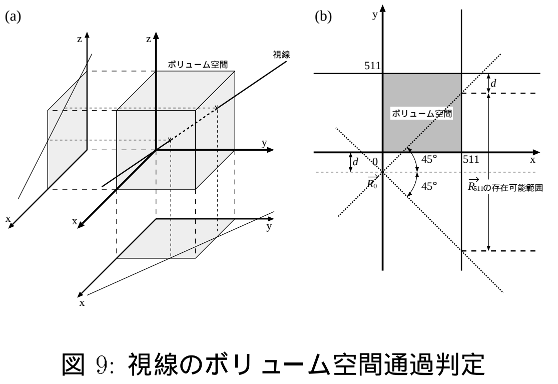

図 9: 視線のボリューム空間通過判定

   そこで、まず $\vec{R}_0$, $\vec{R}_{511}$ の y 成分に着目すると、視線がボリューム空間を通過するためには、以下の条件のいずれかが成り立たなければならない。

   (i) $\vec{R}_0$、または $\vec{R}_{511}$ がボリューム空間の領域内にある

   $$0 \le \mathrm{R_{y0}} \le 511 \text{ または } 0 \le \mathrm{R_{y511}} \le 511 \tag{5}$$

   (ii) 視線がボリューム空間の領域を貫く

   $$\mathrm{R_{y0}} > 511 \text{ かつ } \mathrm{R_{y511}} < 0 \tag{6}$$

   または

   $$\mathrm{R_{y0}} < 0 \text{ かつ } \mathrm{R_{y511}} > 511 \tag{7}$$

   しかし、図 9 (b) に示すような xy 平面への投影図で見ると、視線の絶対値の最大成分を持つものを主軸としているため、視線の方向と主軸がなす角度は 45 度以下であり、ボリューム空間が正方形であることから、条件 (ii) の場合が存在しないことがわかる。

   z 成分についても同様のことが言えるので、ボリューム空間の通過の判定には、主軸でない他の成分について、それぞれ条件 (i) が成立するかどうかを調べる。条件 (i) がともに成立する場合は視線がボリューム空間を通過するものとし、そうでない場合は、視線がボリューム空間を通過しないと判断することができる。

3.2.5 初期座標、初期距離の計算

PCS においてアドレス計算の初期値となる $\vec{R}_0$ の座標を初期座標と呼び、その主軸成分でない成分を **IN1**, **IN2** とする。入射距離 **ILEN** は、視点の位置ベクトルを $\vec{E}$ とし、その主軸成分を $\mathrm{E_i}$ とすると次式 (8) で表される。

$$\mathrm{ILEN} = |\vec{E} - \vec{R}_0| = |\mathrm{E_i}| \times \mathrm{BLEN} \tag{8}$$

入射距離および基本距離は、SS におけるシェーディング処理において、ピクセルの深さを求めるのに必要となる。

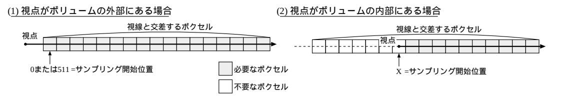

図 10: サンプリング開始位置

3.2.6 サンプリング開始位置の計算

視点が、ボリュームの内部にある場合には、視点より手前のボクセルがピクセルに投影されないようにするために、そのボクセルを透明とみなすマスク処理を行う必要がある。そこで、図 10 のように、読み出すボクセルがどこから輝度計算に必要となるかを、サンプリング開始位置 **MASK** により PCS に知らせる。

3.3 視線生成処理の並列化

個々のピクセルに対する視線データの生成には、本質的な依存関係は無い。すなわち、別のピクセルに対する視線生成処理は並列に実行可能である。そこで、スクリーンを均等に分割し、各 PE がそれぞれ分割された領域上のピクセルに対する視線生成処理を行うことで処理を並列化する。

処理を行う PE の台数と画面生成速度[1]との関係を表 3 に示す。*ReVolver/C40* のプロトタイプにおいて要求される速度は約 2.5 frames/sec であるので、PE 台数は 8 台とする。

表 3: PE 台数と画面生成速度との関係

PE 台数	2	4	8	16
画面生成速度 (frame/sec)	1.15	2.29	4.58	9.17

次に、スクリーンの分割方法であるが、まずここで、ピクセル間のコヒーレンシについて考える。

3.2.1 節では、視線ベクトル $\vec{R}$ は 式 (1) の計算により求めると述べた。しかし、同じスキャンライン上の隣接ピクセルに対する処理を連続して行う場合には、$\vec{R}$ は前のピクセルに対して求めた $\vec{R}$ に増分 $\vec{S}_h$ を加算するだけで求めることができる。

ゆえに、同一スキャンライン上にあるピクセルに対する処理を全て同じ PE が行うことにすれば、計算は単純化され、高速化が図れる。この点を考慮して、スクリーンはスキャンライン単位で分割する[2]。

分割した領域の PE への割り当て方として、図 11 のようにサイクリックに分割する方法が考えられる。すなわち、スキャンラインに画面の上から順に 0 から 511 までの番号を与え、第 n 番のスキャンラインに対して、$n \bmod 8$ 番の PE が処理を行う。この分割により、負荷がほぼ均等に分散されると思われる。

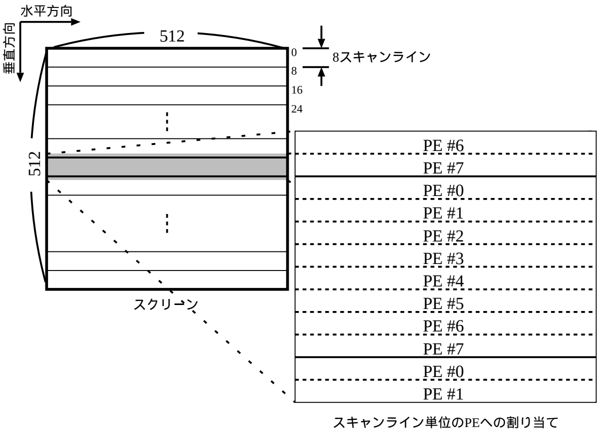

図 11: RCS のスクリーン分割による並列化

4 レイキャスティングステージの構成

4.1 RCS の全体構成

RCS の構成とデータの流れの概略を図 12 に示す。RCS の構成要素は主に以下の三つである。

- **View Buffer**

  全 PE で共有されるメモリで、SCU からのビューデータ、プログラムなどを保持する。全 PE の共有バスと接続される。View Buffer は 容量が 64 Kwords の SRAM である。

- **PE**

  PE には DSP (Digital Signal Processor) を使用する。DSP は 種々のアルゴリズムのプログ

---

[1] 画面生成速度は、PE のプログラムから推定したものである。

[2] 同様に、スクリーンの垂直方向のラインを 1 単位として分割することも考えられるが、ここでは、スキャンラインを単位として分割する。

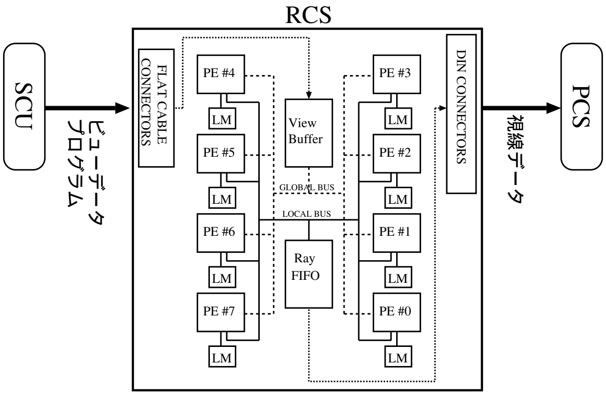

図 12: RCS の構成図

ラミングが可能であり、積和算や平方根の計算が高速に行えるという特徴を持つ。

ReVolver/C40 では、DSP に TMS320C40 を使用している。TMS320C40 は 25 MHz で高速に動作し、GLOBAL BUS および LOCAL BUS と名付けられた二つの入出力ポートを持つ（付録 図 ?? 参照）[10]。通常、これらのポートを使用したリード命令は 1 サイクルで実行され、ライト命令は 2 サイクルで実行される。

DSP 内には、容量 1 Kwords の RAM BLOCK という二つのメモリ（以下、内部ローカルメモリと呼ぶ）と 容量 128 words の命令キャッシュが存在する。更に、独自のバスを所有し、メモリマップ上の転送を CPU と並列に実行できる DMA を内蔵する。DMA の起動は CPU が行い、指定したブロックの転送が終了したときには、DMA から CPU に割り込みをかけることが可能である。

GLOBAL BUS には View Buffer が接続され、LOCAL BUS には Ray FIFO Buffer と 64 Kwords の外付のローカルメモリ（以下、外部ローカルメモリと呼ぶ）が接続されている。

- **Ray FIFO Buffer**

  PE の生成した視線データを受け取る FIFO（first-in first-out）メモリで、2 ポートのうち 1 ポートから RCS が視線データを書き込み、もう一方のポートから PCS がデータを読み出す。PCS では、サンプリングするボクセルのアドレス計算をアドレスジェネレータ（AG）が行い、ピクセル値計算を DSP が行うが、それぞれで必要となるデータが異なるので AG-FIFO Buffer、DSP-FIFO Buffer の 2 種類の Ray FIFO Buffer を設け、視線データの転送は個別に行う。いずれの Ray FIFO Buffer も全 PE の共有バス（View Buffer が接続されているバスとは異なる）と接続される。

  Ray FIFO Buffer には 深さ 64、幅 36 bit の FIFO メモリを使用した。

RCS の処理の流れを図 13 に示す。各 PE 内では、視線生成処理を CPU が行い、視線転送を DMA が行う。これらは並列に実行可能である[3]が、視線転送の遅れにより CPU の視線生成処理が停止する場合もある（4.5.3 節参照）。また、ビューデータを読み出すときや同期操作（4.5.4 節参照）の際の View Buffer での競合でアクセス権を得られない場合にも、CPU は停止する。DMA は、Ray FIFO Buffer での競合によってアクセス権が得られない場合は、停止してアクセス権が得られるのを待つ。

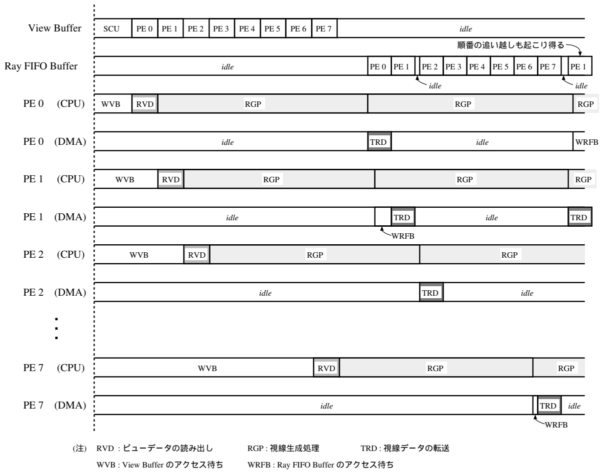

図 13: RCS の処理の流れ

4.2 共有資源の調停方針

共有メモリである View Buffer、Ray FIFO Buffer には 8 台の PE からのアクセスが行われ、ここで調

---

[3] ただし、DMA 転送の開始と終了には、CPU での処理が必要となる。

停が必要となる。

- 調停回路の設計方式

  各 PE の視線生成処理に要する時間は様々な要因により一定とはならない。視線がボリューム空間を通過しない場合の処理中断もその要因の一つである。このため、8 PE が共有資源に対して行うアクセスの順番なども予測できない。そこで、アクセスがどのような順番で行われても速やかにアクセス権を渡すことができ、かつ各 PE の優先度が均等となるように、仮想トークンリング方式を用いる。

  仮想トークンリング方式において、8 台の PE は仮想的に図 14 に示すような番号順のトークンリングを形成し、トークンを所有する PE のみが共有資源にアクセスできる。PE はアクセスを終えるとトークンを解放し、トークンはそのとき資源の要求を出している PE に渡される。複数の DSP が資源の要求を出しているときは、リング上で現在トークンを持っている PE から $0 \rightarrow 1 \rightarrow \cdots \rightarrow 7 \rightarrow 0$ の方向に見て最初に要求のあった PE にトークンを渡す。

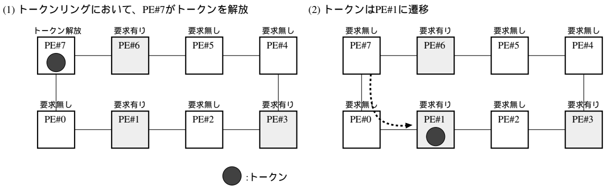

図 14: PE 間トークンリングにおけるトークンの遷移

  物理的にはトークンを所有する PE 番号を保持するレジスタが存在し、このレジスタの値の変化によってバス制御が行われる。なお、実装の結果、トークンの受け渡しはどのような場合でも 1 サイクル[4]で可能となった。

- 共有資源使用のプロトコル

  RCS における PE から View Buffer および Ray FIFO Buffer に対して行われるアクセスには、それぞれ以下の問題がある。

  - View Buffer からのビューデータを読み出しでは、ビューデータ 12 words すべてを読み出さなければ PE は視線生成処理に移ることができない。ここで、1 word ごとに調停すると、オーバヘッドが大きくなる。
  - PE が Ray FIFO Buffer に対して 1 視線に対する視線データを書き込む際に、視線データ 10 words が一定の順序で書き込まれていなければ、PCS においてデータを読み出す時にデータは正しく認識されない。そこで、PE の 1 視線に対する視線データの書き込みには排他制御が必要となる。

いずれの場合においても、PE が共有資源を占有排他とできるようなロック機構が有用となる。

そこで、共有資源の要求時には PE は必ずロックをかけるようにプロトコルを定める。使用する DSP の TMS320C40 においては、インタロック命令という特殊なロード / ストア命令により、物理的にロック信号を生成することが可能である。ロック信号はインタロックロード命令の開始とともにアクティブとなり、インタロックストア命令の終了とともに非アクティブとなる信号である。

この信号を用いた共有資源へのアクセスのプロトコルは以下のようになる。

1. PE は、共有資源へのアクセスの開始時にインタロックロード命令によりロック信号を出す。、
2. アービタは、この信号を共有資源の要求と考え、調停を行う。その結果トークンを得た PE は、次のアクセスを行うことができる。ロック信号を出してもトークンを得られなかった PE に対する制御は、4.3.1 節および 4.5.3 節で述べる。
3. トークンを得た PE は、共有資源使用の終了時にインタロックストア命令を用いてロック信号を解除する。
4. アービタは、ロック信号の解除を知ると、次に資源を要求している PE に対して調

---

[4] ただし、ハードの制約などから調停制御全体にかかる時間は View Buffer では 2 サイクル、Ray Buffer では 4 サイクルとなる。

停を行う。

4.3 View Buffer の動作

4.3.1 View Buffer の調停

View Buffer は、8 台の PE および SCU からアクセスされる。SCU と PE の調停においては、SCU が RCS の制御を行うこと、また SCU のビューデータの転送が遅れると RCS の計算がその間停止することから、SCU のアクセスが PE のアクセスよりも優先されるべきである。そこで、View Buffer の調停には 2 段階調停方式を採用する。この調停方式では、SCU と PE から同時に要求が生じたときには SCU を優先し、PE 間には仮想トークンリング方式を用いて調停を行う。

PE が調停の結果アクセス権を得られなかった場合には、CPU はアクセス権が得られるのを待つ。この間視線生成処理は停止するが、View Buffer へのアクセスは、通常 1 画面に 2 回しか起きず、停止する時間は 1 画面の生成にかかる時間に比べると微少である。

4.3.2 View Buffer へのリード / ライトサイクル

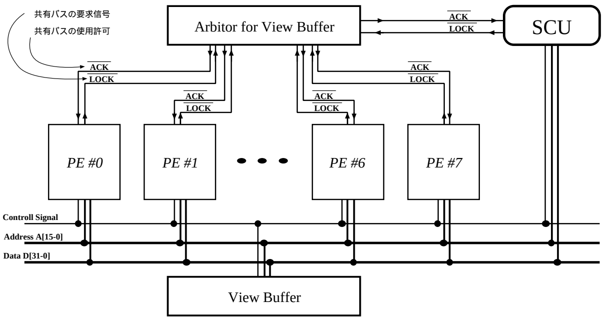

図 15: View Buffer の制御機構

View Buffer の制御機構を図 15 に示す。アービタには PLD (Programmable Logic Device) を使用して実装を行った。

実装の結果、View Buffer へのリードサイクルは 2 サイクル、ライトサイクルは 2 サイクルとなった。ただし、最初に View Buffer にアクセスするときのみ、アクセスの競合が無い場合でも調停のため更に 2 サイクルが必要となる。

4.4 Ray FIFO Buffer の動作

4.4.1 Ray FIFO Buffer の調停

Ray FIFO Buffer には AG-FIFO Buffer と DSP FIFO Buffer が存在するが、同一のバスを使用するので、調停は一括して行う。調停は仮想トークンリング方式を用いる。

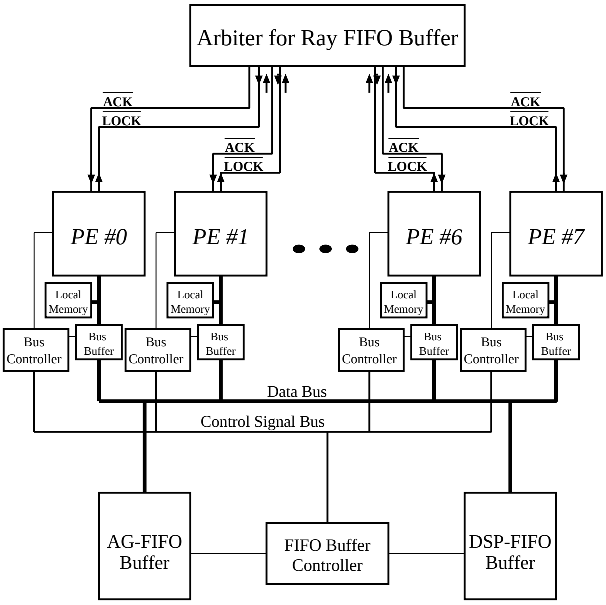

図 16: Ray FIFO Buffer の制御機構

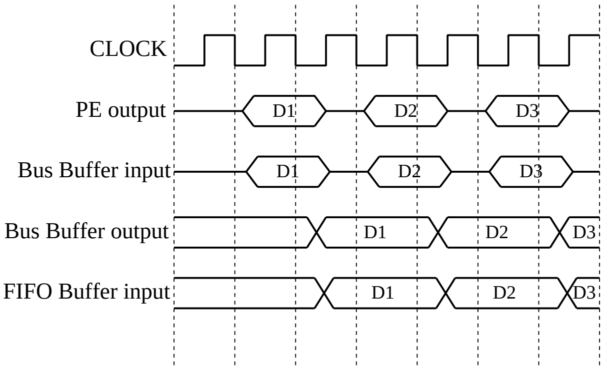

図 17: 視線データの流れ

4.4.2 Ray FIFO Buffer へのライトサイクル

Ray FIFO Buffer の制御機構を図 16 に示す。アービタなどの制御機構は PAL を使用して実装を行った。

Ray FIFO Buffer は TMS320C40 の LOCAL BUS ポートに接続されているが、この LOCAL BUS は 2 系統の制御信号を持ち、そのうちの一方をローカルメモリに使用し、他方を Ray FIFO Buffer に使用する。LOCAL BUS へ出力された視線データ 1 word は、PE ごとに設置した Bus Controller により一時的に Bus Buffer に取り込まれる。Bus Buffer の出力は各 PE の共有バスへと接続されており、そこから Ray FIFO Buffer への転送は Bus Controller が行う。図 17 に示すように、PE から Bus Buffer への転送と Bus Buffer から Ray FIFO Buffer への転送はオーバーラップされ、その結果、視線データの Ray FIFO Buffer への書き込みはノーウェイトの 2 サイクルで行える。

ただし、最初に視線データを転送するときには、アクセスの競合が無い場合でも、調停のため更に 4 サイクルが必要となる。

4.5 PE の動作

以下の節では PE の各動作について説明する。

4.5.1 ビューデータの読み出し

前述の手順に従い、インタロック命令を使用して 1 画面分のビューデータ (12 words) を連続して読み出す。

4.5.2 視線生成処理

視線生成処理としては、まず視線ベクトルの生成、主軸及び主軸方向の判定、基本ベクトルの計算、視線とボリューム空間の交差判定、初期座標と初期距離の計算、サンプリング開始位置の計算を行う。

次に、生成したこれらのデータを PCS に必要な精度で丸め、更に図 18 に示すようにデータフォーマットの変換を行い、複数のデータを 1 word の中に組み込む。この処理によって通信量が削減でき、PCS における処理も容易となる。

ここで、DSP-FIFO Buffer へ送るデータ INTS は PCS における交点座標 (2.3 節参照) 計算の初期値である。また、AG-FIFO Buffer へ送るデータのうち、FRAME は 1 画面のデータ生成の終了を SS が知るための信号である。

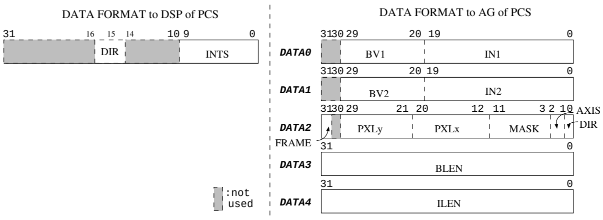

図 18: 視線データ転送時のフォーマット

4.5.3 視線データの転送

4.2 節で述べたように、視線データの転送においては、1 ピクセルの視線データ転送中は共有バスを占有しておく必要がある。そのため、視線データ転送中には、その他の処理に費やす時間は共有バスを無駄に使用している時間となる。この時間をなくすため、視線データは、生成後に、あるメモリ領域 (以下、**Local Ray Buffer** と呼ぶ) に一時的に保存しておき、それをまとめて転送するという方法をとる。

また、視線データ転送には TMS320C40 の内蔵 DMA を使用して視線生成処理とは並列に行う。

DMA を使用した視線データ転送の手順を以下に示す。

(1) フォーマットの変換の終了した視線データを、Local Ray Buffer に一時的に保存しておく。

(2) Local Ray Buffer にデータを保存しておく視線の数を定めておき、それに等しい量のデータが蓄積されれば、視線データ転送を行っているかどうかを示すフラグ TRANSFLAG を調べる。

    TRANSFLAG が 1 にセットされていれば、まだ前回の転送が終了していないので、その終了を待つ。

    TRANSFLAG が 0 であれば、視線データ転送は現在行われていないので、インタロックロード命令によって、ロック信号をアクティブにし、転送を開始する。

(3) TRANSFLAG を 1 にセットし、DMA チャネルを起動してこれまでに蓄積された視線デー

タの転送を開始する。しかし、このとき PE がトークンを所有していない場合は、DMA による転送は PE がトークンを得るまで延期される。ただし、CPU はトークンを所有していない場合でも停止することなく視線生成処理を行うことができる[5]。また、Local Ray Buffer にはダブルバッファリングを施す。

(4) DMA は、まず AG-FIFO Buffer に対する転送を行い、それが終了すると DSP-FIFO Buffer への転送を行う。そして、DSP-FIFO Buffer への転送が終了すると CPU に対して割り込みをかける。

(5) 割り込みを受けた CPU は、インタロックストア命令を用いてロックを解除し、TRANSFLAG を 0 にリセットする。

4.5.4 1 画面分の視線データ転送の終了時における同期動作

SS では、1 画面に対する処理の終了を知る必要があり、RCS において 1 画面に対する視線データの転送が終了したとき、そのことをフラグ FRAME によって PCS, SS に伝える。そこで、各 PE は、1 画面に対するデータの転送が終了すると、View Buffer 内のバリア変数 Sync（初期値は PE 台数 8）をデクリメントし、各自のローカルメモリの TRANSFLAG を −1 にセットする。

- このとき、Sync をデクリメントした値が 0 となれば、その PE が最後に視線データ転送を行ったことになるので、FRAME フラグをセットして PCS へ転送する。
- そうでないときは、各 PE は次の画面に対する視線データの生成を開始する。そして、視線データの転送において TRANSFLAG を調べる際に、TRANSFLAG が −1 であれば、View Buffer にアクセスし、前画面の Sync を調べる。
    * Sync が 0 のとき：全 PE が前画面に対する転送を終了しているので、TRANSFLAG を 1 として転送を開始する。
    * Sync が正のとき：前画面の転送が終了していない PE があるので、一度 View Buffer を解放し、その後再びアクセスして Sync を調べる。Sync が 0 となるまでは、転送は行わない。

4.6 メモリ領域の割り当て

PE には以下の三つのローカルメモリが存在する。

1. 内部ローカルメモリ RAM BLOCK 0
2. 内部ローカルメモリ RAM BLOCK 1

   これらは、容量が 1 Kwords でリード / ライトサイクルはともに 1 サイクルである。

3. 外部ローカルメモリ：容量は 64 Kwords でリード / ライトサイクルはともに 2 サイクルである。

以上のメモリに割り当てるものとして、コード領域、ローカル変数領域、Local Ray Buffer が存在する。以下では、これらの割り当てについて考察する。

4.6.1 コード領域の割り当て

まず、コード領域の設置場所であるが、これは次の二つが考えられる。

1. 内部ローカルメモリ（RAM BLOCK 0 または RAM BLOCK 1）
2. 外部ローカルメモリ

外部ローカルメモリにコード領域を置いた場合は、命令は PE の命令キャッシュに読み込まれるので、命令フェッチはキャッシュに対して起こることになる。ただし、キャッシュのサイズは 128 words であり、それを越えるようなコード数の場合は、キャッシュミスヒットによる外部ローカルメモリへのアクセスが起き、同一ポートに接続されている Ray FIFO Buffer へのアクセスとコンフリクトを起こす可能性がある。

視線生成処理のプログラムは最内ループでも 128 命令を越えるので、1. を採用してコード領域は PE の内部メモリ RAM BLOCK 1 [6]に置くものとする。

---

[5] これは、ソフトウェアによりインタロックロード命令を強制終了させるためである。詳しくは付録参照。

[6] RAM BLOCK 0 でもよい。

4.6.2 ローカル変数領域の割り当て

ローカル変数領域の設置場所としては、この領域へのアクセスが頻繁に起きることから、内部メモリがよいと思われ、以下の選択が生じる。

1. RAM BLOCK 0
2. コード領域と同じ RAM BLOCK 1

コード領域とローカル変数領域を同じ RAM BLOCK においた場合、PE の同一 RAM BLOCK へのアクセスは 1 サイクルに二つまでしか行えないので、命令フェッチと 2 変数へのアクセスが同一サイクルで起こるような命令では、コンフリクトが生じ、その結果実行サイクルが余分にかかってしまう。よって、RAM BLOCK 0 にローカル変数領域を設ける。

4.6.3 Local Ray Buffer の割り当て

視線データ転送の速度は、Local Ray Buffer の設置場所に大きく依存する。設置場所の議論を行うため、まず三つの用語を定義する。

コンフリクト CPU と DMA は並列に動作するが、同じ RAM BLOCK へのアクセスが生じた場合、および同じ入出力ポートを使用した場合にはコンフリクトが生じる。コンフリクトが生じた場合は、DMA による転送を優先して、CPU の命令実行を遅らせる。DMA を優先する理由は、DMA の転送中は 1 PE の DMA が共有バスを占有しているからである。

なお、表 4 中のコンフリクト頻度は、作成したプログラム（付録参照）から算出した相対値である。

並列命令 DSP は 1 命令で 2 words のロードや、2 words のストアなどが行える並列命令を持つ。視線データのストアにこの命令を使用すると、Local Ray Buffer を RAM BLOCK 0 においた場合には、1 サイクルで 2 words の保存が可能である。 RAM BLOCK 0 以外の場合では、DSP 内部のコンフリクトなどによりアクセス時間に遅れが生じる。

バイパス機構 外部ローカルメモリから Ray FIFO Buffer への転送は、図 19 (a) のように 外部ローカルメモリから PE 内部にデータを読み込むためにまず 2 サイクルかかり、次にそこから Ray FIFO Buffer へ転送するのに 2 サイクルかかるので、転送サイクルが 4 サイクルとなる欠点がある。そこで、外部ローカルメモリと Ray FIFO Buffer が同じ LOCAL BUS のポートに接続されていることを利用して、図 19 (b) のように、Local Ray Buffer からデータを読み込むときに、Ray FIFO Buffer のバスへもデータが流れるようにハードウェアを設計する。

CPU が、この手法を用いて外部ローカルメモリから Ray FIFO Buffer へ直接データを転送する場合は、CPU が連続にロード命令を実行すればよく、2 cycles/word の速度で転送可能である。しかし、DMA がこの手法を用いる場合、DMA はリードの後に必ずストアを実行するという仕様であるため、転送速度は、冗長なストア命令により 3 cycles/word となる。

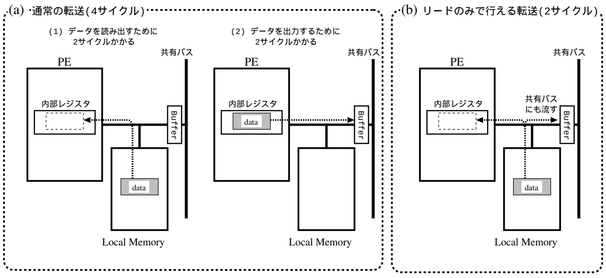

図 19: 外部ローカルメモリから Ray FIFO Buffer への転送

表 4 は Local Ray Buffer の設置場所と視線データ転送速度の関係を示したものである。

表 4: Local Ray Buffer の領域割り当ての選択

設置場所	RAM BLOCK 1	RAM BLOCK 0	外部ローカルメモリ
他に置くデータ	コード領域	ローカル変数領域	
コンフリクト要因	CPU の命令フェッチと DMA のデータロード	CPU の 2 変数アクセスと DMA のデータロード	CPU のデータ保存と DMA の転送動作
コンフリクト頻度	1	0.1 程度	0.05 程度
視線データ保存速度	1 cycle/word	0.5 cycles/word（並列命令使用）	2 cycles/word
視線データ転送速度	2 cycles/word	2 cycles/word	2 cycles/word（バイパス機構使用）

表 4 から Ray Buffer の三つの設置場所について以下のことがわかる。

- RAM BLOCK 1 :Local Ray Buffer のサイズがかなり小さくなり、コンフリクトの回数も他に比べかなり多いと思われる。
- RAM BLOCK 0 :Local Ray Buffer のサイズは最大で約 80 視線分となる。コンフリクトはあまり起きないと思われ、保存や転送の速度は最も速い。
- 外部ローカルメモリ :Local Ray Buffer のサイズは最大で約 5000 視線分となる。コンフリクトはほとんど発生しないと思われるが、保存の速度は他の半分となる。DMA を使用した場合は転送速度も 3 cycles/word となり効率が悪くなる。

ここで、Local Ray Buffer のサイズについて検討する。Local Ray Buffer のサイズが大きくなれば、調停 および DMA 起動などのオーバーヘッドが減少する。ここで、このオーバーヘッドが視線データ生成速度に及ぼす影響を作成したプログラム (付録参照) から算出した結果を表 5 に示す。表 5 から Local Ray Buffer のサイズが 32 以上では、バッファサイズの増加に対する速度向上率が小さいことがわかる。これらの結果を総合評価した結果、Local Ray Buffer の設置場所は RAM BLOCK 0 とした。

表 5: Local Ray Buffer のサイズと画面生成速度の関係

Local Ray Buffer のサイズ (保存する視線数)	画面生成速度 (frames/sec)
0	3.60
2	4.05
4	4.31
8	4.46
16	4.54
32	4.58
64	4.60
512	4.62

5 レイキャスティングステージの評価

5.1 RCS のパフォーマンスの評価

5.1.1 View Buffer

View Buffer へのリード / ライトサイクルはともに 2 サイクルであり、最大メモリバンド幅は 50 Mbytes/sec である。1 画面の生成においては、1 PE につきビューデータ 12 words の読み出しと、同期変数 1 words の読み出しと書き込みが行われ、アービトレーションによるオーバーヘッドが最小である場合で、32 サイクルかかることになる。このサイクル数のみで画面生成速度を見積もると、最大約 97.7 Mframes/sec となり、View Buffer のメモリバンド幅には十分な余裕がある。

5.1.2 Ray FIFO Buffer

Ray FIFO Buffer へのライトサイクルはいずれの転送方法でも 2 サイクルとなり、最大メモリバンド幅は 50 Mbytes/sec である。視線データは 1 ピクセルあたり 6 words であり、1 画面は 512^2 ピクセルで Local Ray Buffer のサイズが 32 であるとすると、最大 7.87 frames/sec の書き込みが可能である。

5.1.3 PE

各動作におけるサイクル数は表 6 のようになる。表 6 に示すサイクル数から、PE の視線データの生成速度は約 4.58 frames/sec となる。

表 6: PE の各処理にかかるサイクル数

PE の処理		サイクル数
1 画面に対するビューデータの読み出し		45
1 ピクセルに対する処理	視線生成処理	計 158
	視線ベクトルの生成	11
	主軸及び主軸方向の判定	14
	基本ベクトルの計算	26
	交差判定と初期座標 ·初期距離の計算	44
	サンプリング開始位置の計算	9
	フォーマットの変換	54
1 ピクセルに対するデータ転送		20
1 ブロック (32 ピクセル) ごとに行う DMA の起動		41
1 画面に対する同期操作		17

5.1.4 RCS の実装結果について

View Buffer、Ray FIFO Buffer、PE について、それぞれ性能を見積もった結果、RCS 全体が PE の視線生成処理に律速されていることがわかった。PE の処理速度は、単純に台数を増やすことでも処理速度を上げることができる。しかし、PE 台数が 14 以上になると、Ray FIFO Buffer のバスのバンド幅に律速される。そこで、このバンド幅を上げるには、以下のような方法がある。

- FIFO の個数を増やし、バスを多重化する。この場合には、PCS の仕様も変更しなければならない。
- 現在の仕様では、AG-FIFO と DSP-FIFO には同じバスを使用しているが、バスを別々に割り当てることで、PCS の仕様を変えることなく速度を上げることができる。しかし、AG-FIFO に i 番目に書き込まれたデータと、DSP-FIFO に i 番目に書き込まれたデータは同じ視線に対するデータでなければならない。この制約を満たすには、図 20 のように、FIFO 間で書き込みを行う PE の順序を一致させることが必要である。

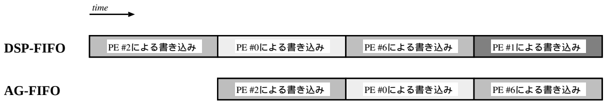

図 20: AG-FIFO と DSP-FIFO のアービトレーションにおける制約

5.2 視線生成アルゴリズムの評価

本稿で述べた視線生成アルゴリズムでは、ピクセルごとに主軸の判定やボリューム空間の通過判定を行った。しかし、視点とスクリーンのパラメータから、あらかじめスクリーンを主軸やボリューム空間の通過・不通過によって領域分割しておく方法も考えられる。

図 21 にスクリーンを主軸によって領域分割した場合の例を示す。一般にスキャンラインを走査していく上で、主軸が変化する回数はせいぜい数回であると思われる。そこで、スキャンラインごとに主軸が変化する境界をあらかじめ求めておき、ピクセルごとの処理においてはその境界に達しているかどうかのみを判断するようにすれば、ピクセルごとに主軸の判定を行う手間を省くことができる。

また、ボリューム空間を一つの立方体のオブジェクトとしてスクリーンへ投影すれば、スクリーンをボリューム空間を通過する領域と通過しない領域に分けることができる。そこで、同様にその境界の情報を使用することで、ピクセルごとにボリューム空間の通過・不通過の判定を行う手間を省くことができる。

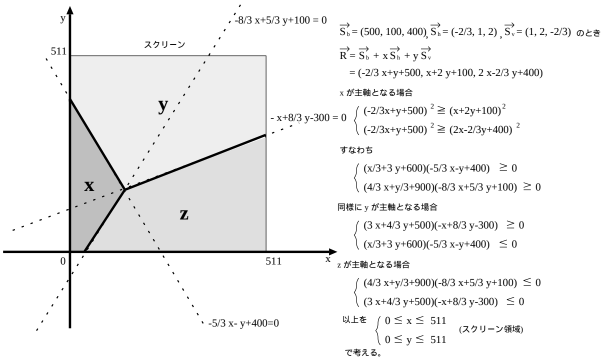

図 21: 主軸によるスクリーンの分割

これらの手法はピクセル数が大きくなる程有効であると思われる。しかし、前処理として境界の情報をどのように保持するかが問題となる。

5.3 他のステージとの関連における評価

プロトタイプの実装においては、PCS の 1 PE がボクセルのサンプリングを 4 回行うため、PCS が *ReVolver*/C40 全体を律速することが予想された。そこで、PCS の負荷を軽減するために、RCS において不要な視線データの判定や、視線データのフォーマットの変換を行っている。しかし、PCS の PE 台数を増やし、1 PE に 1 平面分だけの処理を担当させる場合には、RCS のさらなる高速化が必要となる。

高速化の手法については、既に述べたものの他に、個々の PE の処理の高速化として、フォーマット変換の高速化が可能である。表 6 に示すように、フォーマット変換にはかなりの時間を費やしているが、これは論理演算やシフト演算をソフトウェア上で行っているためであり、専用ハードウェアを設けることで高速化可能であると思われる。

6 おわりに

本稿では、*ReVolver*/C40 におけるレイキャスティングステージ（RCS）の構成と視線生成処理について述べた。

RCS において視線生成処理は 8 台の PE により並列に実行され、更に、生成したデータの転送がデー

タの生成とは並列に実行される。詳細設計を行った結果、約 4.6 frames/sec のデータ生成が可能であることが判明した。また、視点やスクリーンに制限を与えることなく、遠近法による画像表示か可能である。

PE 間の共有資源としては、View Buffer と書き込みのみが行える Ray FIFO Buffer が存在し、このアービトレーションは仮想トークンリング方式で行った。また、データの圧縮などによるバスの使用効率の向上も図っている。

実装は既に終了し、主にデバッグ用として RCS の制御をホストマシンから行うことのできるインタフェースも作成した。これにより、RCS を通常の共有メモリ型マルチプロセッサシステムとして用いることも可能である。

今後の課題としては、さらなる高速化へ向け、ハードウェア面ではバスの多重化など、ソフトウェア面ではピクセルのコヒーレンシを利用した視線生成アルゴリズムの改良などを考えていく必要がある。また、平行視線やスクリーンが主軸に垂直である場合の単純なものから、レイトレーシングなどの複雑なものまでの様々なアルゴリズムを試してみることも重要である。

本研究の機会を与えてくださり、適切な御指導を賜わった富田眞治教授に深く感謝の意を表します。

数々の有用な御意見を戴いた中島浩助教授、森眞一郎助教授に心から感謝致します。

本研究の共同研究者である、金喜都氏、中山明則氏、木綿芳行氏をはじめ、計算機工学講座計算機構成分野の諸氏に感謝致します。

参考文献

[1] Arie Kaufmann(訳 :藤代一成): ボリュームビジュアライゼーション概論, *PIXEL*, No. 116.

[2] Drebin, R.A. et al.: Volume Rendering, *ACM Computer Graphics*, Vol. 22, No. 4, pp. 64–75 (1988).

[3] 對馬雄次: ボリュームレンダリング専用並列計算機 — ReVolver/C40 —, 修士論文, 京都大学 (1995).

[4] Levoy, M.: Display of Surfaces from Volume Data, *IEEE Computer Graphics and Applications*, Vol. 8, No. 5, pp. 29–37 (1988).

[5] Hanspeter Pfister, A. K. 他.: Cube-3: A Real-Time Architecture forHigh-Resolution Volume Visualization, *IEEE Computer Graphics and Applications*, pp. 75–82 (1995).

[6] 中山明則: ボリュームレンダリング専用並列計算機 ReVolver/C40 のピクセル値計算ステージの設計, 修士論文, 京都大学 (1996).

[7] Gordon, D. et al.: Image Space Shading of 3-Dimensional Objects, *IEEE Computer Graphics and Applications*, Vol. 29, No. 3, pp. 42–55 (1985).

[8] 荻野友隆: ボリュームレンダリング専用並列計算機 — ReVolver/C40 — のシェーディングステージ, 京都大学特別研究報告書 (1995).

[9] 木綿芳行: ボリュームレンダリング専用並列計算機 ReVolver/C40 のシェーディングステージの設計, 京都大学特別研究報告書 (1996).

[10] Texas Instruments: *TMS320C4x User's Guide* (1993).