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

中山 明則

平成 8 年 2 月 15 日

内容梗概

ボリュームレンダリングは 3 次元空間を直接 2 次元画像に投影する手法であり、従来から、医療用の画像生成手法として重要な地位を占めてきた。さらに近年では、スーパコンピュータなど高速な計算機を利用した構造解析や流体力学など、科学技術計算の結果の解析手法としても、ボリュームレンダリングは注目を集めている。しかし、ボリュームレンダリングは膨大な計算資源を必要とするため、高速な画像生成は困難であった。

そこで現在我々は、ボリュームレンダリングを高速に行う専用並列計算機 ReVolver/C40 の開発を行っている。ReVolver/C40 では、従来の医療用の画像だけでなく、科学技術計算の結果の可視化をサポートする。このため、高速な画像生成とともに、半透明ボリュームの表示、遠近法による画像生成、を目標としている。

これらの目標を達成するために、ReVolver/C40 では、視点から各ピクセルへの視線を生成する視線生成ステージ、視線生成ステージで生成された視線のデータを元にサンプリングするボクセルのアドレスを計算し、読み出したボクセルの値からピクセルの輝度を計算するピクセル値計算ステージ、生成した画像にシェーディング処理を施し画像の出力を行うシェーディングステージ、からなる 3 ステージのマクロパイプライン構成を採用している。また、各ステージ内においても種々の並列処理により画像生成の高速化を図っている。また、各ステージの主要な構成要素として汎用の高性能 DSP を使用しており、柔軟性の非常に高い構成となっている。

本論文では、主に ReVolver/C40 のピクセル値計算ステージについて述べる。ピクセル値計算ステージは視線生成ステージから送信される視線に関するデータから、ピクセルの輝度計算を行う。またシェーディング処理に必要な、視線とボリューム空間内のオブジェクトとの交点を求める。

ピクセル値計算ステージはピクセル値計算ユニットを 128 台、一次元に接続した構成をとり、これらをパイプライン動作させることでピクセル値計算を行う。このパイプライン処理速度が ReVolver/C40 の処理速度に大きく影響を与えるため、パイプラインが停滞しないことが非常に重要である。これには、バンク・コンフリクト・フリーなメモリ構成の実現が要求される。そこで、ReVolver/C40 では、単純化されたサンプリング方法とデータの 3 重化によりバンク・コンフリクトが起こらないメモリ構成を実現した。

本研究ではピクセル計算ステージの設計を行った。その際に、処理の高速化を第 1 の目標とし、この他に、コストの抑制、拡張性、様々な処理への柔軟な対応、を設計の方針とした。

処理の高速化を達成するために、ボクセル値の読み出し時間の隠蔽、ピクセル値計算と通信の高速化を行った。

ピクセル値計算ユニットで行う処理は、1) 視線に関するデータから読み出すボクセルのアドレスを計算する処理、2) メモリアクセスの処理、3) 読み出したボクセルの値からピクセル値の計算を行う処理、の 3 つである。これらの処理を、アドレス計算とメモリアクセスを行うステージ、ピクセル値の計算を行うステージ、の 2 つのステージに分け、2 つのステージをパイプライン動作させ

る。この構成により、ボクセル値のプリロードが実現でき、ボクセル値の読み出し時間の隠蔽を実現した。

ピクセル値計算にはDSPを用いることで、柔軟性を高めた。このDSPの特徴として並列命令をサポートしていることが挙げられる。DSPの並列命令は、2命令の計算を1サイクルで行うことができ、また1命令の計算と1命令のデータ通信も1サイクルで行うことができる。ピクセル値計算の高速化を達成するために、DSPの特徴である並列命令を十分利用することが重要である。

計算とデータ通信の並列命令を効果的に利用するためには、ウェイトなしの通信の実現が重要である。そこで、DSP間の通信の実現にFIFOメモリを採用し、処理の遅延の伝搬を防いだ。また、FIFOメモリからのデータの先読みを行うことで、ウェイトなしでDSP間通信を可能とするよう設計を行った。

また、今回設計したピクセル値計算ステージは128台構成であるが、512台構成への拡張も可能とし、また規模の縮小も可能な設計となっている。

実装の結果、128台構成のピクセル値計算ステージの性能は、ボリューム空間が半透明で大きさが512^3voxelの時には、約$398 ms/frame$ (約$2.5 frame/sec$) となり、ボリューム空間の大きさが128^3voxelの時には、約$178 ms/frame$ (約$5.6 frame/sec$) となった。

A design for the Pixel Calculation Stage of Parallel Volume Rendering Computer: *ReVolver*/C40

Abstract

Volume rendering is a direct visualization method of three dimensional space. It has been a very 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 structural analyses and fluid dynamic computations on high-performance supercomputers. Since volume rendering requires an enormous amount of computational resources, it is dificult to realize high-speed image generation.

Hence we are developing a high-speed volume rendering machine, the *ReVolver*/C40. This machine visualizes not only medical data but also scientific data as computation results. Therefore, the goals of this machine are high-speed image generation and visualization of translucent volume both with parallel and with perspective projections.

In order to achieve these goals, *Revolver*/C40 adopts the macro-pipeline of the following three stages: the Ray Casting Stage which generates rays, the Pixel Calculation Stage which calculates pixel values, and the Shading Stage which shades in images. Parallel processing is used at each stage for high-speed image generation. Each stage is configured with high-performance general purpose DSPs in order to deal with various patterns of computations.

This paper describes mainly about the Pixel Calculation Stage. This stage receives data concerning to the rays from the Ray Casting Stage and uses it to calculate pixel values and intersection points which are needed for the Shading Stage.

This stage consists of 128 Pixel Calculation Units which are connected in one dimension. Pixel values are calculated through the pipeline organized with these units. Since the performance of this pipeline-processing directly affects the speed of *ReVolver*/C40, it is very important not to block pipeline-processing by supplying data continuously. Therefore, a memory organization that can be accessed without bank conflict is required. *ReVolver*/C40 adopts a simplified voxel sampling

method and a volume memory structure fitted for the pipeline.

The main goal of the Pixel Calculation Stage design is high processing speed. It also aimed for cutting expenses, getting scalability, and coping with various computations.

For these purposes, it is required processing to hide the time of reading voxel values from DRAM and to speed up pixel calculation and data communication.

The processes of the Pixel Calculation Unit includes 1)the address calculation 2)the memory access for voxel data , and 3)the pixel value calculation. These three processes are proceeded with two stage pipeline: one stage for the memory access and address calculation and the other for the pixel value calculation. This pipeline enables the preloading of voxel values and thus it hides the time to read them.

For the flexibility of computations, Digital Signal Processors (DSPs) are adopted to calculate pixel values. One of the features of DSPs is to support parallel instructions. It is possible to execute two computational instructions or one computational and one data communication instructions in one cycle. In order to calculate pixel values in a certain period, it is important to make good use of them.

For the efficient use of parallel instructions of the later case, it is important to realize no wait memory access. Hence, fifo memories are adopted to transfer data between DSPs to prevent the processing delay from being propagated. We have realized the mechanism to read data from fifo memories in advance to the accesses to them. This makes it possible to transfer data between DSPs without any wait time.

Currently, the Pixel Calculation Stage is designed for 128 PEs, but it can be scaled up to 512 PEs or down to smaller the number of PEs.

The performance of this stage with 128 processing elements is $398ms/frame$ (about $2.5frame/sec$) with translucent volume of 512^3voxels and $178ms/frame$ (about $5.6frame/sec$) with that of 128^3voxels.

1　はじめに

近年、計測やシュミレーションなどから得られる内部情報を持った 3 次元データから画像データを生成するボリューム・ビジュアライゼーションが注目されている。

旧来の 3 次元コンピュータ・グラフィックスが 3 次元オブジェクトを 0 次元、1 次元、あるいは 2 次元の幾何図形 (プリミティブ) を組み合わせて表現し、これらを表示対象にするのに対し、ボリューム・ビジュアライゼーションは、3 次元オブジェクトを内部情報を持った 3 次元データ (ボリュームデータ) で表現し、このボリューム・データを表示対象とする。

ボリューム・ビジュアライゼーションにより、物体の内部を探索し、解析することができる。ボリューム・データは、ボクセルと呼ばれる部分空間の集合として表現され、ボリューム・データは普通、立方体のボクセルからなる等間隔 3 次元メッシュ構造で表現される。

ボリューム・データから画像を生成する手法として、ボリューム・データを構成する各ボクセルを直接画面に投影することにより画像を生成する、ボリュームレンダリングと呼ばれる手法があげられる。

ボリュームレンダリングは、ボリューム・データの内部構造の可視化を可能にし、複雑なボクセル集合の可視化にも対応できる。

一方、ボクセル集合のデータをそのまま保持しなければならず、おおむねボクセル数に比例する計算時間が必要となるのが、ボリュームレンダリングの問題点である。

そこで現在我々は、ボリュームレンダリングを高速に行う専用並列計算機 ReVolver/C40 を提案し、現在 ReVolver/C40 を開発している [1][2][3][4][5]。

ReVolver/C40 は、視線生成ステージ、ピクセル値計算ステージ、シェーディングステージ、からなる 3 ステージのマクロパイプライン処理を行っている。3 ステージのうち、ピクセル値計算ステージの処理量が最も多いため、*ReVolver*/C40 の処理速度はピクセル値計算ステージの処理速度で決定される。

そこで本稿では以下、2章で *ReVolver*/C40 の概要を述べ、3章でピクセル値計算ステージの処理および設計方針を述べ、4章で本研究で行った、ピクセル値計算ステージを構成しているピクセル値計算ユニットの設計について述べ、5章でまとめる。

2　*ReVolver*/C40 の特徴

2.1　*ReVolver*/C40 の目標

ボリュームレンダリングは、従来から、医療用の画像生成手法として重要な地位を占めてきた。さらに近年では、スーパーコンピュータなどの高速な計算機を利用して、構造解析や流体力学など、科学技術計算の結果の解析手法としても、ボリュームレンダリングは注目を集めている。そこで、*ReVolver*/C40 では医療用としてだけでなく、科学技術計算の結果の解析も行えるよう、以下の 3 項目の実現を目標としている。

1. 高速描画
2. 半透明ボリュームの表示
3. 遠近法による画像生成

以下に、これらの目標の有意点と、目標達成のために解決すべき問題点を述べる。

1. 高速描画　:ボリューム空間の断面の表示や視点の移動に対する再描画を高速に行えることは、可視化対象を理解、解析する上で非常に重要である。しかし、ボリュームレンダリングはおよそボクセル数に比例する計算量が必要であるところが問題である。そこで、並列処理により、処理の高速化をはかる。詳しくは、2.2節で述べる。
2. 半透明ボリュームの表示　:ボリュームを半透明な物体として扱うことで、ボリューム全体の様子を把握することが可能になる。これは、科学技術計算の結果を、可視化という解析手法に利用するのに非常に重要である。

   しかし、ボリュームを半透明で扱うためには、図 1に示すように、ピクセル値を計算する際に、視線と交差する全ボクセルのデータをアクセスし、計算する必要があるので、不透明ボリュームを扱う場合に比べ、処理量が増大するという問題点がある。

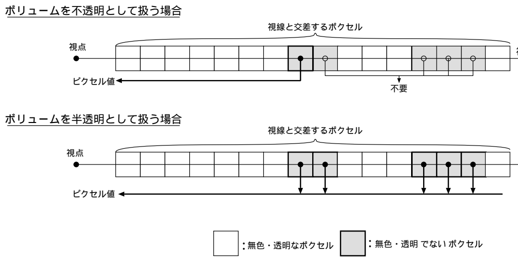

図 1: ボリュームによるアクセスするボクセルの相違

3. 遠近法による画像生成　:遠近法で画像生成することにより、大きな空間を表示する際に、立体感を作り出すのに役立つ。これも、科学技術計算の結果を解析、理解をする上で重要である。

   遠近法をサポートする場合、任意方向の視線が発生する。そのため、メモリアクセスが規則的でなく、メモリバンクへのアクセス競合の可能性が高くなり、並列処理への障害となる。

2.2　*ReVolver*/C40 の並列処理

ボリュームレンダリングは非常に計算量の多い処理であるので、処理の高速化をはかるため従来から様々な並列処理が行われてきてきた。図 2に種々の並列処理の方式を示す。

1. ボクセル並列処理　:1 つのピクセルの輝度 (以下、ピクセル値と呼ぶ) の計算を、複数の Processing Element(以下、PE と呼ぶ) によって処理する方式。ボクセル並列処理は、レイキャスティングと呼ばれるアルゴリズムを並列化したものである。

   レイキャスティングとは、視点からピクセルに向かって視線を発生させ、視線と交差するボクセルからピクセル値を求めるアルゴリズムである。レイキャスティングの概念図を図 3に示す。

   レイキャスティング・アルゴリズムでは、スクリーン上の各ピクセル毎に発生する視線に沿って、視線と交差するボクセルを視点に近い順にサンプリングしていく。

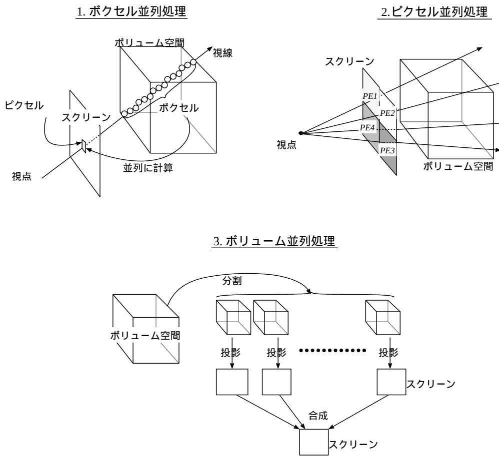

図 2: ボリュームレンダリングの並列処理方式

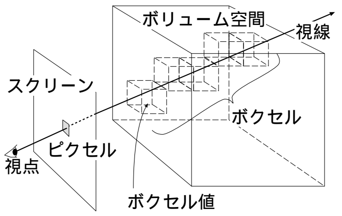

図 3: レイキャスティング

ボクセルのサンプリングは視線上のボクセルが尽きるまで反復される。その時のサンプリングされたボクセル数を Nとすると、対応するピクセルの値は、以下の式 (1) を評価した値となる [6][7]。

$$\text{ピクセル値} = \sum_{i=0}^{N}(1-\alpha_i)c_i\prod_{j=0}^{i-1}\alpha_j \tag{1}$$

ただし、ボクセル値 v_tは、色 c_tと透明度α_tを表すデータであり、サンプリングしたボクセルの値は視点に近い順から、$v_0, v_1, \cdots, v_{N-1}$とする。

2. ピクセル並列処理 :1 つのピクセル値の計算を 1 PE が担当し、複数の視線を並列に処理する方式。ピクセル並列処理も、1. と同様レイキャスティング ·アルゴリズムを並列化したものである。
3. ボリューム並列処理 :ボリュームをいくつかのサブボリュームに分割し、各サブボリュームを PE に割り当てる。各 PE は、担当ボリューム内の各ボクセルがスクリーンに及ぼす影響を計算する。計算結果をマージすることで、最終的な画像を得る。

ReVolver/C40 では、半透明表示を目標としているため、視点に近い順からのボクセルの並びの順序を保証する必要がある。レイキャステング ·アルゴリズムを用いるボクセル並列処理やピクセル並列処理と比べて、ボリューム並列処理では、この順序を保証するのが困難となる。

また、ピクセル並列処理の場合、ボリューム ·メモリを PE 間で共有する必要があるため共有メモリを実現する必要がある。さらに、遠近法による表示を実現する場合、視線が任意方向となるため、複数の視線を同時に処理するピクセル並列処理では、ボリューム ·メモリへのアクセスが競合する可能性が高くなる。

ボクセル並列処理で遠近法による表示を実現する場合、任意方向の視線に対して、視線と交差する全ボクセルをバンク ·コンフリクト ·フリーでアクセスできる機構が必要となる。この機構は、次の 2.3で提案する方法により、実現可能となった。

2.3 バンク ·コンフリクト ·フリーなメモリ

バンク ·コンフリクト ·フリーな 3 次元メモリは次に述べる 1. サンプリング方法の単純化と、2. 単純化したサンプリング方法に適したメモリ構成、によって実現できる。

サンプリング方法の単純化

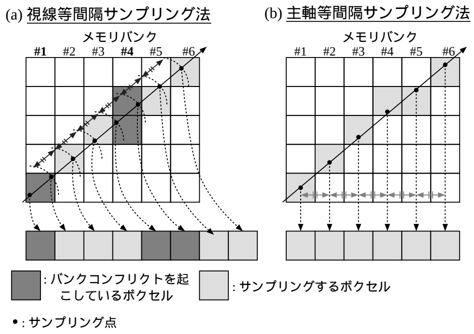
(a) 視線等間隔サンプリング法　(b) 主軸等間隔サンプリング法

図 4: サンプリング法

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してきた (以下、視線等間隔サンプリング法と呼ぶ)。しかし、視線等間隔サンプリング法では、図 4(a) のように、ボリュームデータをメモリバンクに分割して格納すると、同一視線上の複数のボクセルが同一メモリバンクに格納される状況が発生する。これは、ピクセル値計算の並列実行を妨げ、高速化の障害となる。

この問題を回避するために、ReVolver/C40 では、視線ベクトルの 3 つの成分 (x,y,z 成分) の絶対値を比較し、最大値の成分の座標軸を主軸と定め、この主軸方向に対して等間隔にボクセルをサンプリングする方法 (以下、主軸等間隔サンプリング法と呼ぶ) を採用した。この主軸等間隔サンプリング法により、図 4(b) に示すように、バンクコンフリクトが生じなくなる。

バンク ·コンフリクト ·フリーなメモリ構造

主軸等間隔サンプリング法では、ある 1 つの座標軸に対し、これに垂直な平面にボリュームデータを分割してメモリバンクに格納すれば、その座標軸を主軸として持つ視線に関してはバンク ·コンフリクト ·フリーとなる。しかし、任意方向の視線について考慮すると、主軸としても任意の 3 つの座標軸を考えなければならない。

そこで、ボリューム ·データを 3 重化することでこれに対処する [8]。具体的には図 5のようにボリューム空間を x,y,z 軸に垂直な n 枚* の平面に分割し、各平面群に対し、平面に 1,2,···,n と番号をつけ、同じ番号を持つ平面のデータをその番号のメモリバンクの対応する領域に格納する。例えば、ボリューム内の座標 (p,q,r) のボクセルは、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、各平面群の平面はすべて異なったメモリバンクに格納されるので、主軸等間隔サンプリングを行

---

*n はボリューム空間の 1 辺のボクセル数をあらわす。

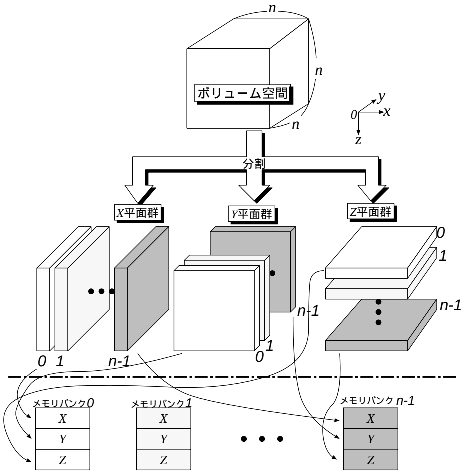

図 5: ボリュームデータの格納方法

う場合、任意方向の視線と交差する全ボクセルを同時に読み出すことが出来る。

2.4 ピクセル値計算のパイプライン化

2.3で述べたメモリ構成により視線と交差する全ボクセルのメモリからの読み出しのバンク ･コンフリクト ･フリーが実現できた。そこで、読み出されたボクセル値からのピクセル値計算をパイプライン処理で行うことで、高速化をはかる。

2.4.1 ピクセル値計算のパイプライン化

ピクセル値は2.2節で述べたように、レイキャスティング ･アルゴリズムを用いると式 (1) を計算することで求められる。ここで、視線上のボクセルの値を視点に近い順に $v_0, v_1, \cdots, v_{m-1}$とすると式 (1) を式 (2) のような漸化式に変形できる。

$$\left\{ \begin{array}{lcl} C_k & = & C_{k-1} + (1-\alpha(v_k)) \cdot c(v_k) \cdot A_{k-1} \\ A_k & = & \alpha(v_k) \cdot A_{k-1} \end{array} \right. \tag{2}$$

ただし、$c(v_i), \alpha(v_i)$ はそれぞれボクセル値 v_iを色、透明度に変換した値であることを示している。

サンプリングするボクセル数が Nの場合、式 (1) によって求められるピクセル値は式 (2) の C_Nと同一の値となる。つまり C_Nを求めることによって、ピクセル値を求めることが出来る。

式 (2) の C_k, A_{k-1}は $C_{k-1}, A_{k-1}, c(v_k), \alpha(v_k)$ から求めることができ、$c(v_k), \alpha(v_k)$ は v_kを変換した値であるから、結局 C_k, A_kは C_{k-1}, A_{k-1}, v_kから求めることができる。

よって、2.3節で述べた主軸等間隔サンプリング法を用い、i 番 $(i = 0, 1, \cdots m)$ の PE に C_k, A_kの計算を行わせると、各 PE が 2.3節で述べたメモリバンクの 1 つをローカルメモリとしてもち、PE を一次元に接続した構成をとることができる。このとき PE 数は、ボリューム ･データが $N^3 voxel$の場合 N台となる。この構成により、PE をパイプライン動作させることで、ピクセル値計算の高速化をはかることが出来る。

2.4.2 パイプラインの単方向化

式 (2) によってパイプライン処理でピクセル値計算ができるが、このパイプライン処理は視点に近いボクセルから順に処理しなければならない。

ところが視線の主軸成分が正 (以下、視線方向が正と呼ぶ) の時と、視線の主軸成分が負 (以下、視線方向が負と呼ぶ) の時とでは、処理の行う方向が逆となる。よって例えば視線方向が正のピクセル値の計算を行っている最中に、視線方向が負のピクセル値計算を行おうとすると、処理の衝突が起こり性能が低下する。

そこで、常に処理の方向を正とすることを、次のようにして実現した。

式 (1) を式 (3) に変形する。

$$C'_{k-1} \quad = \quad C'_k \cdot \alpha(v_{k-1}) + (1 - \alpha(v_{k-1})) \cdot c(v_{k-1}) \tag{3}$$

式 (1) によって求められるピクセル値は、式 (3) の C'_0と同一の値となる。式 (3) の C'_{k-1}の値は $C'_k, c_{k-1}, \alpha_{k-1}$から求めることができるので、結局 C'_k, v_{k-1}から求めることができる。

よって、視線方向が負の場合に i 番の PE に式 (3) の C'_{N-i} の計算を行わせると、ボリューム ･データが $N^3 voxel$とすると、$0, 1, \cdots, N$番の PE の順で処理が行える。

以上より、視線方向が正の時は式 (2) を計算し、視線方向が負の時は式 (3) を計算することで、常に処理の方向を単方向とすることが可能である。

2.5 $ReVolver$/C40 の構成と処理の概要

本節では、$ReVolver$/C40 の全体構成と、各構成要素の処理の概要について述べる。$ReVolver$/C40 の構成を図 6 に示す。

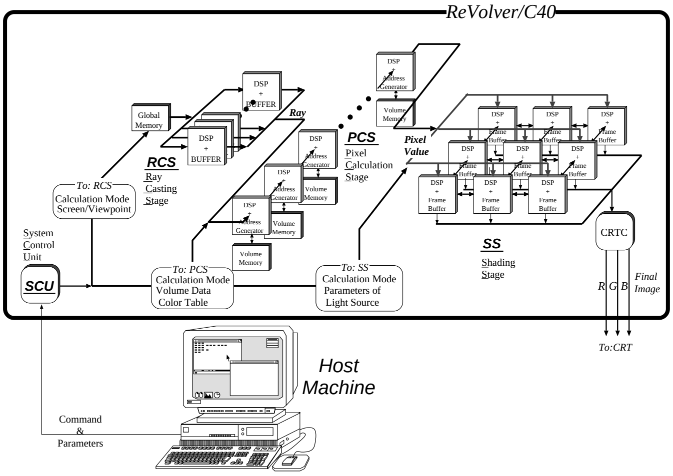

図 6: $ReVolver$/C40 の構成

$ReVolver$/C40 は視線生成ステージ、ピクセル値計算ステージ、シェーディングステージ、の 3 つのステージからなるマクロパイプライン構成をとっている。。各ステージは、それぞれ複数の DSP によって構成される。

- 視線生成ステージ (**RCS**)

  視線生成ステージでは、SCU から送られてくる視点、スクリーンの情報から、視点から各ピクセルを通る視

線ベクトル (以下、視線と呼ぶ) を生成し、視線に関する種々のデータ (以下、視線データと呼ぶ) を計算し、視線データをピクセル値計算ステージに送信する。

視線生成ステージでは、スクリーンを分割して各 PE に処理を割り当る。これにより各 PE は他の PE とは完全に独立に処理を行うことができる。

各 PE は SCU からデータを受けとると、それぞれ独立に処理を開始し、生成されたデータはピクセル値計算ステージに送信する。

- **ピクセル値計算ステージ (PCS)**

  ピクセル値計算ステージでは、視線データに基づきボクセル値を読み出し、レイキャスティング ・アルゴリズムを用いてピクセル値の計算を行う。

  ピクセル値は、前述のとおり、単方向のパイプライン処理によって計算する。そこで、ピクセル値計算ステージは、PE とメモリを 1 つのユニットとしたピクセル値計算ユニット (PCU) を 1 次元に接続する構成を持つ (図 6)。ピクセル値計算ステージで計算したピクセル値はシェーディングステージに送られる。

- **シェーディングステージ (SS)**

  シェーディングステージでは、より現実的な画像を生成するためにシェーディング処理を行う。シェーディング処理の手法として、depth gradient shading[9] を用いる。depth gradient shading では、ピクセルのシェーディング処理を行う際、シェー ディングを行うピクセルの近傍ピクセルの情報が必要となるので、一画面分のピクセル値が全て計算されるのを待って、シェーディング処理を行う。

  シェーディングステージは画像処理に適したメッシュ状に PE が接続されており、スクリーン分割により並列実行する。

  シェーディングステージの各 PE は、シェーディング処理結果をフレームバッファへ送出し、最終的な画像を CRT へ出力する。

これら *ReVolver*/C40 の 3 ステージは、**System Control Unit**(SCU) によって制御される。

ReVolver/C40 では、各ステージの PE の構成要素として汎用の DSP(TI 社: TMS320C40,50MHz[10]) を用いている。これにより、多様な処理に対して柔軟に対処できるようになっている。

3 ピクセル値計算ステージの設計

本章では、ピクセル値計算ステージで行う処理について述べた後、ピクセル値計算ステージの設計方針について述べ、本研究で行ったピクセル値計算ステージの実現方法について述べる。

3.1 ピクセル値計算ステージの行う処理

ピクセル値計算ステージで行う処理は大きく、次の 3 つである。

- 視線生成ステージから視線データを受信する。
- 視線データからサンプリングするボクセルのアドレスを計算し、ボクセル値を読み出し、ピクセル値を計算する。
- 計算したピクセル値をシェーディングステージに送信する。

3.1.1 視線生成ステージから受信するデータ

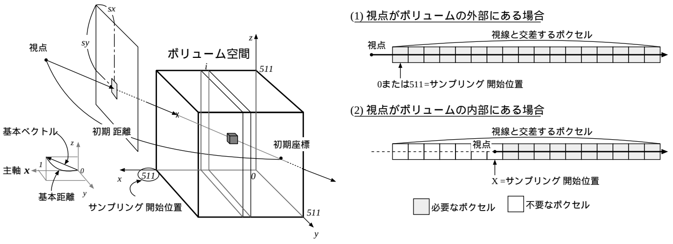

図 7: 視線生成ステージで生成する視線データ

ピクセル値計算ステージは視線生成ステージから表 1と図 7に示すデータを受けとり、これらのデータを基にピクセル値の計算を行う。

以下に、ピクセル値計算ステージに必要なデータについて述べる。

ピクセル値計算ステージのパイプラインは単方向で、ピクセル値計算ステージは視線の向きに関わらず常に 0 番のピクセル値計算ユニットから処理を行う。

ピクセル値計算ステージの i 番の平面を処理するピクセル値計算ユニットがサンプリングするボクセルのアドレスは以下の式 (4) のように表すことが出来る。よって各 PE は、式 (4) の初期座標、基本ベクトルからアドレスを計算する。初期座標は図 7に示すように、例えば主軸が X 軸の場合、平面 $X = 0$ と視線との交点の座標である。また基本ベクトルは視線ベクトルの主軸成分を 1 とした時の視線ベクトルである。

$$\text{初期座標} + \text{基本ベクトル} \times i \qquad (4)$$

さらに、ピクセル値計算ステージで保持しているボリュームデータは三重化されているため、主軸名からボクセルを読み出すメモリ領域の選択を行う。また、ピクセル値計算ユニットでは、視線ベクトルの主軸成分の符号によりピクセル値計算の方法を変更しなければならないので、主軸方向が必要である。

また、視点がボリュームの内部にある場合やボリュームの切断面の表示を行うために、図 7の (1)、(2) のように読み出されたボクセルが不要かどうかを判定しなければならない。そこで、サンプリング開始位置と主軸方向から判定する。サンプリング開始位置は、サンプリングを開始する座標の主軸成分であり、主軸方向は視線ベクトルの主軸成分の符合を表す。

表 2に示すデータは、ピクセル値計算ステージでは用いないが、シェーディングステージで用いる。これらのデータもピクセル値計算ステージが受けとり、そのままシェーディングステージに送る。

3.1.2 ピクセル値計算処理

視線生成ステージからの視線データを基にして、その視線上にあるボクセルの値をボリュームメモリから読み出し、読み出したボクセル値を用いてボクセルの色、透明度を求めて、ピクセル値を計算する。

表 1: ピクセル値計算ステージで用いるデータ

データ名 (記号名)	作成方法	データ・フォーマット
主軸名 $(AXIS)$	$AXIS = \begin{cases} if(\text{主軸} == x)\ 01 \\ if(\text{主軸} == y)\ 10 \\ if(\text{主軸} == z)\ 00 \end{cases}$	$2bit$
主軸方向 (DIR)	$\vec{R} = \vec{E} + \vec{S_b} + sx \times \vec{S_h} + sy \times \vec{S_v};$ $DIR = (R.x > 0)?\ 0 : 1;$	$1bit$
基本ベクトル $(BV1, BV2)$	$BV1 = R.y/R.x;$ $BV2 = R.z/R.x;$	$10bit \times 2$ 符合部 :$1bit$ 小数部 :$9bit$
初期座標 $(IN1, IN2)$	$IN1 = E.y - E.x \times BV1;$ $IN2 = E.z - E.x \times BV2;$	$20bit \times 2$ 符合部 :$1bit$ 整数部 :$10bit$ 小数部 :$9bit$
サンプリング 開始位置 $(MASK)$	$if \quad 0 < E.x < 511$ $then\ MASK = [E.x]$ $else\ MASK = (DIR > 0)?\ 511 : 0$	$9bit\ int$

主軸を x 軸と仮定した場合のデータである

表 2: シェーディングステージで用いるデータ

データ名 (記号名)	作成方法	データ・フォーマット
主軸方向 (DIR)	$\vec{R} = \vec{E} + \vec{S_b} + sx \times \vec{S_h} + sy \times \vec{S_v};$ $DIR = (R.x > 0)?\ 0 : 1;$	$1bit$
基本距離 $(BLEN)$	$BLEN = \sqrt{BV1^2 + BV^2 + 1}$	$32bit\ float$
初期距離 $(ILEN)$	$ILEN$ $= \sqrt{E.x^2 + (E.y - IN1)^2 + (E.z - IN2)^2}$	$32bit\ float$
スクリーン座標 $(PXL.x, PXL.y)$	$PXL.x = sx;$ $PXL.y = sy;$	$9bit\ int \times 2$

主軸を x 軸と仮定した場合のデータである

読み出すボクセルのアドレス計算には、先述したように、視線生成ステージで生成された主軸方向、主軸名、初期座標、基本ベクトル、サンプリング開始位置のデータを用いる。アドレス計算が終るとボクセル値の読み出しを行う。

ボクセル値は色と透明度に関するデータを表現しているが、$ReVolver$/C40 におけるボクセル値は、色と透明度に関する属性テーブルのインデックスである。よって読み出したボクセル値 v_iを使って属性テーブルを参照することにより式 (2) 中の $c(v_i)$ や$\alpha(v_i)$ を求める。

ボクセル値は $1byte$ で表されているため、このテーブルは 256 エントリから構成される。各エントリの構成は $c(v_i) = (R(v_i), G(v_i), B(v_i)), \alpha(v_i)$ からなり、各データは $32bit\ float$ で表わされている。

オプションの処理

以上で述べた処理が、$ReVolver$/C40 が通常行うピクセル値計算であるが、さらに、ピクセル値計算のオプションとして次のような処理も行う [11]。

- 不透明ボリュームの表示 :従来のボリュームレンダリングの利用目的である医療画像の生成にも対処するために、ボリュームを不透明なものとして扱う。
- 連続モデルによって表現されたボリューム空間のレンダリング :ボリューム空間には、離散モデルと連続モデルの 2 つの表現方法がある。$ReVolver$/C40 では通常離散モデルを扱うが、連続モデルも扱うことが出来る。以下に、離散モデルと連続モデルについて簡単に説明する。

  離散モデルでは、ボクセル値を持つ各格子点を中心とした単位立方体をボクセルとし、ボクセル内の任意の点は中心と同じ値を持つ。

  これに対し、連続モデルでは、単位立方体の頂点にボクセル値が存在し、単位立方体の内部の値は、その点を囲む 8 頂点の値から tri-linear 補間により求められる。
- サンプリング間隔の補正 :$ReVolver$/C40 で採用している主軸等間隔サンプリング法は、主軸が座標軸と平行でない視線では、従来の視線等間隔サンプリング法に

比べて、サンプリング点の間隔が大きくなる傾向があり、最大で$\sqrt{3}$倍となる。そこで、サンプリング間隔の違いを、色・透明度を調整することにより補正するオプションを用意する。

- 平行投影法による表示：*ReVolver*/C40は、通常遠近法で表示を行うが、平行投影法による表示も行うことが出来る。平行投影法を行う場合でも、ピクセル値計算ステージの処理は全く変わらない。

3.1.3　シェーディングステージに送信するデータ

先述した通り、ピクセル値計算ステージは表3に示すデータを生成し、シェーディングステージに送信する。

交点は、視線上で視点に最も近い無色透明でないボクセルの、主軸成分を表す。扱うボリューム空間が最大$512^3 voxel$であるので、交点は$9bit$で表すことが出来るが、交点がない場合を識別するためのフラグが必要である。

またこの他に、視線生成ステージで生成したデータ(表2参照)もピクセル値計算ステージがシェーディングステージに送信する。

3.2　設計方針

以上で述べたような処理を行うピクセル値計算ステージの実装に当たって、次のような点を設計方針とした。

- 処理の高速化
  ReVolver/C40の目標でも述べたが、視点の移動に対する高速な描画は、可視化の対象とする3次元データを解析する上で重要である。*ReVolver*/C40の処理速度は、ピクセル値計算ステージの処理速度によって決定されるため、ピクセル値計算ステージの処理の高速化が本研究における設計の最優先事項とする。
- コストの抑制
  実際のシステムを実現する時に、要するコストを抑制することは重要である。しかし、コストを抑制することは、処理の高速化を目指すことと、ある点では相反する。そこでこの両者のトレードオフを考慮する。
- 様々な処理への柔軟な対応
  ReVolver/C40の大きな目標は、科学技術計算の結果の可視化である。よって、半透明ボリュームの表示、遠近法のサポート、が必要となるので、これらを実現するシステムを実装する。しかし、これら以外にも、例えば平行投影法による表示の要求があるのも事実である。

  そこで、半透明ボリュームの表示、遠近法のサポート、の実現を前提として、これらの処理速度がまったく低下しない範囲で、システムに出来得る限りの柔軟性を持たせる。
- 拡張性
  並列計算機を設計する際に、そのPE台数の規模により、実現に対して適切な構成が異なる。よって、今回実装するピクセル値計算ステージのPE台数が128台であり、今後512台構成への規模拡張が容易に実現可能となる設計を行う。また逆に、128台より小規模な構成でも動作するよう設計を行う。

ピクセル値計算ステージの実現に際して様々な選択肢があるが、以上で述べたことを設計方針とした。

DSPの採用

ピクセル値計算ステージでは、他のステージと同様にPEの構成要素として汎用のDSPを用いている。これにより、多様な処理に対して柔軟に対処できるようになっている。

以下に、このDSPの主な特徴を述べる。

- 浮動小数点乗算、加算、減算が1サイクルで行える。
- 乗算&加算(減算)、乗算&ロード(ストア)、加算&ロード(ストア)、ロード(ストア)&ロード(ストア)のような組合せの命令が、1サイクルで並列に実行できる。
- 100MB/secの転送速度のバスを2本(Global Bus,Local Bus)持つ。
- 外部バスへのアクセスは、ロードが1サイクル、ストアが2サイクルで行われる。
- 内部メモリへのアクセスは、ロード、ストアともに1サイクルで行われる。
- 20MB/secの転送速度のCommunication Port (Com Port)を6本持つ。
- CPUと独立にデータ転送を行えるDMAを持つ。Com PortによるはDMAが制御する。

高速な処理を実現するために、DSPのこれらの特徴を最大限に生かせる構成にすることを目標にする。

ピクセル値計算ステージで行うピクセル値計算には、浮動小数点乗算・加算が含まれている。DSPはこの計算を1サイクルで行えるので、ピクセル値計算はDSPに行わせる。

以下では、処理の高速化を柱とし、上で述べた設計方式に基づいたピクセル値計算ステージの実現方法について述べる。

3.3　処理の高速化

3.3.1　ボクセル値の読みだし時間の隠蔽

2.4節で述べたように、一次元に接続したピクセル値計算ユニットのパイプライン処理によってピクセル値計算を行うことができる。各PEは、ボリューム・データを格納したメモリからボクセル値を読みだし、この読み出したボクセル値と前段のPEから受信したピクセル値の途中結果からピクセル値の計算を行う。

ボリューム・データは膨大なデータ量であるため、格納するメモリはDRAMで実現する。ところが、DRAMへのアクセスは比較的時間を要し、ピクセル値計算の処理時間と比較しても無視できないほどの時間である。また、読み出すボクセル値の格納されているアドレスを計算するのも処理を遅くさせる。そこで、ボクセル値の読み出し時間の隠蔽を考える。

読み出すボクセル値のアドレス計算には、ピクセル値計算ステージで処理を行っているピクセル値の途中結果は必要なく、視線データのみから求めることが出来る。

このことを利用して、サンプリングするボクセルのアドレスの計算と、ボクセル値の読み出しを行う専用回路を設けることで、DSPにピクセル値計算に専念させる。以下ではこの専用回路を**Address Generator**として述べる。

Address Generatorを設けたことで処理を分担させることができ、高速化がはかれる。

表 3: ピクセル値計算ステージの生成するデータ

データ名 (記号名)	意味	データ・フォーマット
ピクセル値 (R, G, B)	ピクセルの RGB 値。	$32bit\ float \times 3$
交点 $(INTS)$	視線上で視点に最も近く、 無色透明でないボクセルの主軸の成分。	$10bit$ フラグ :$1bit$ 整数部 :$9bit$

またDSPによるピクセル値の計算結果に関係なく、必要なボクセル値を格納したメモリアドレスが計算できる。これは、先述したようにボクセル値の格納したメモリアドレスは、視線データのみに依存しているからである。

このことを利用して、ボクセル値が実際に必要となる以前に、次に読み出す(であろう)ボクセルの格納されているメモリアドレスを計算し、Address GeneratorがDRAMにアクセスしてボクセル値のプリロードを行い、レジスタ(以下、Voxel Latchと呼ぶ)に保持しておく。そして、実際にボクセル値が必要となった時に、DRAMへアクセスするのではなくVoxel Latchから読み出すことにより、メモリアクセス時間の隠蔽をはかることができる。ピクセル値計算ユニットにおける、処理の流れを図8に示し、データの流れを図9に示す。

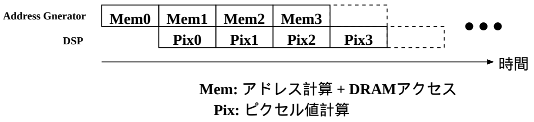

図 8: ボクセル値のプリロードの様子

Address GeneratorのメモリアクセスとDSPのピクセル値計算の処理は、パイプライン実行でき、メモリアドレス計算とメモリアクセス時間の隠蔽がはかれる。

2系統のデータの流れ

先述したように、読み出すボクセル値のアドレス計算には視線データのみが必要でピクセル値は必要ない。また、ピクセル値計算を行うDSPは前段のDSPのピクセル値計算の途中結果と、ボクセル値だけが必要であり、逆に視線データは必要がない。

そこで、アドレス計算用の視線データを流す系統(AGバス)と、ピクセル値計算用のRGB値や透明度を流す系統(DSPバス)の2つの系統を設ける。この構成により、Address GeneratorはAGバスによってデータの通信を行い、DSPはDSPバスによってデータの通信を行う。

よって、ピクセル値計算ユニット間の通信は、図9に示すように、DSP間通信とAddress Genrator間通信の2つの系統で実現できる。

また、0番のピクセル値計算ユニットは、視線生成ステージからデータを受信し、127番のピクセル値計算ユニットは、シェーディングステージにデータを送信する。これらの通信も、ピクセル値計算ユニット間の通信と同様、2つの系統によって実現する。

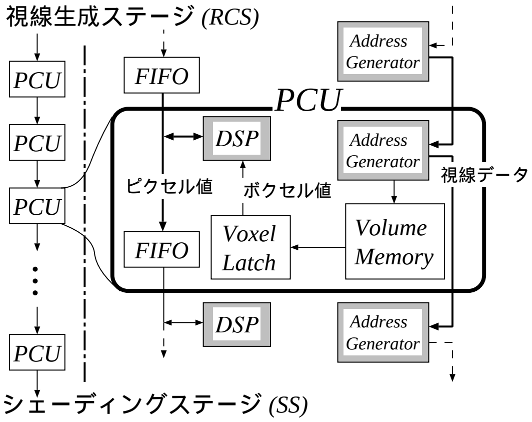

図 9: ピクセル値計算ユニットの構成とデータの流れ

3.3.2 ピクセル値計算とDSP間通信の高速化

先述したように、ピクセル値計算にはDSPを用いる。よって、ピクセル値計算の高速化を達成するためには、DSPの特長を十分に生かした設計を行うことが重要である。

DSPの並列命令

本研究での設計においては、特にDSPの並列命令に注目した。

DSPの並列命令は、乗算&加算(or 減算)、乗算&ロード(or ストア)、加算(or 減算)&ロード(or ストア)、ロード(or ストア)&ロード(or ストア)、といった組合せの2命令を1サイクルで実行することができる。

これらの命令の実行には、様々な資源の制限があるものの、大幅な処理時間の短縮の可能性を秘めている。

また、DSPから外部へのメモリアクセスは、全てDSPのメモリマップへのアクセスで実現されているので、DSP間通信はロード/ストア命令で実現されることになる。よって、並列命令を利用することで、通信時間の隠蔽をはかることも可能となる。

そこで本研究では、ピクセル値計算の高速化を達成するために、並列命令の十分な利用が可能となるよう設計した。

DSP間通信の実現

DSPには100MB/secのバンド幅のバスが2本(Global BusとLocal Bus)と20MB/secのバンド幅のCom Portが6本備わっている。よって、DSP間通信の実現には、バスを用いる、またはCom Portを用いる、の選択肢がある。どちらの通信手段も、DSPのメモリアクセスによって実現できる。

ここで、DSP間通信に要求されるバンド幅について考える。

Com Port は 6 本備わっているので、Com Port の多重化も考えられるが、この場合でも最大 100MB/sec であり、しかもこの性能を引き出すには、DSP の CPU が通信に専念し、ロード&ストアといったメモリアクセスを 1 サイクルに 2 度行う並列命令を使用した場合である。

以上より次のことがいえる。

- DSP 間通信にバスを利用した場合、最大性能が 100MB/sec であり、DSP の備えている外部との通信手段で最大のバンド幅である。
- DSP 間通信に Com Port を利用した場合、最大性能が 20MB/sec である。
- Com Port の多重化により最大 100MB/sec のバンド幅となるが、この性能は、ロード (or ストア) & ロード (or ストア) の組の並列命令を発行した場合である。

ここで、実現に際するコストを考えると、バスを利用した場合、少なくともバスを制御する回路が必要なのに対して、Com Port を利用した場合では、通信を行う DSP の Com Port ピンを直接結ぶだけで良い。よって、Com Port を利用した方がコストをおさえることができる。

そこで、要求されるバンド幅について考察する。

1 視線のピクセル値の計算に、DSP 間には $5words$ のデータ通信が必要である。よって、1 画面が 512^2pixelであると、1 視線の処理に $5words \times 4Byte \times 512^2 = 5.24\text{Byte}$ のデータ通信が起こる。この通信に DSP のバスを用いると、通信能力は約 $105.ms/frame$ となり、Com Port を用いると、通信能力は約 $262ms/frame$ となる。

ここでピクセル値の計算に要する時間を考える。ピクセル値の計算は 1 平面あたり多くても 10 命令であり、128^3voxel のボリューム空間をレンダリングする速度は、計算時間だけで見積もると約 $105ms/frame$ である。

通信能力もこれと同等なだけのバンド幅が要求されるので、Com Port1 本での通信はバンド幅が足りず、採用できない。

また、Com Port を多重化して利用する場合、最大 100MB/sec のバンド幅あるが、送信、受信のために 3 本ずつの利用となる。よって、最大 60MB/sec のバンド幅しかなく、同様にバンド幅が足りない。

よって、結論として DSP 間の通信には DSP のバスを使用することとした。

FIFO の利用

DSP のバスを利用して通信を行う場合、通信の実現には次のような方法が考えられた。

1. DSP 間に通信用のバッファをおかず、DSP 間が直接通信を行う。
2. DSP 間に 1 ポートの共有メモリをおき通信を行う。
3. DSP 間に 2 ポートの FIFO メモリをおき通信を行う。

ピクセル値計算を DSP の並列命令十分に用いて処理の高速化を目指す時、DSP で行われる通信には、でき得る限りウェイトの入らないような動作を保証する必要がある。

全 DSP が同じ動作をし、データの送信したい時刻と、それを受信したい時刻が全く一致している場合は、通信用のバッファなしで、DSP 間が直接通信を行っても処理速度に影響はない。しかし、実際は DSP 間で処理が同じである保証はないため、1. のバッファなしの通信では、ほぼ確実に通信待ちが生じる。また、ある箇所で処理遅延が起こると、その遅延が伝搬し、処理速度が著しく低下する。

2. の 1 ポートの共有メモリを通信バッファとした通信では、処理の遅延の伝搬は起こらない。しかし、並列命令の十分な利用のためには、ポートの競合を気にしないデータ転送の実現が必要である。

そこで、DSP 間の通信には間に FIFO メモリを配置し、通信を FIFO メモリへの書き込み、読み出しで実現する。

FIFO メモリを用いるにあたって、ウェイトなしの読み出しを、データの先読みを行うことで実現した。詳細は 4.3 節で述べる。

4 ピクセル値計算ユニットの設計

4.1 ピクセル値計算ユニット (PCU) の構成

ピクセル値計算ステージはピクセル値計算ユニットを 1 次元に接続させた構成となっている。*ReVolver*/C40 では 128 台のピクセル値計算ユニットで構成している。図 9に ピクセル値計算ユニットの構成を示す。

以下にピクセル値計算ユニットを構成している各要素について説明する。

- **DSP**:ピクセル値計算、交差判定を行う。
- **Address Generator**:サンプリングするボクセルのアドレス計算を行い、ボクセル値の読み出しを行い、隣接する Address Generator 間の通信を行う。また通常の DRAM Controller としての処理も行う。DSP の Local Bus や Voxel Latch の制御も行う。
- **FIFO**:次段の DSP への送信を行うためのバッファ。TI 社の SN74ABT3611 を用いる。DSP から FIFO へのアクセスは、通常のロード / ストア命令で実現でき、ウェイトなしでのアクセスが行える構成になっている。
- **3D Volume Memory**:ボリューム・データを格納している DRAM。*ReVolver*/C40 は、最大 512^3voxelの ボリューム空間を扱う。$1voxel$が 1B なので、ボリューム・データのデータ量は全体で 128MB となる。これを、各ピクセル値計算ユニットにローカルメモリとして持たせると、1 台のピクセル値計算ユニットに対して 1MB となる。ここで、2.3 節で述べたように、データの 3 重化を行うので、3MB のメモリが必要となる。さらにプログラムの格納なども考慮して、4MB の DRAM を用いる。
- **Voxel Latch**:Address Generator によってサンプリングされたボクセル値を DSP によって読み出されるまで保持しておく。DSP からの読み出しの要求があると、ウェイトなしでの読み出しを実現する。

以下では、ピクセル値計算ユニットの主な構成である、DSP、Voxel Latch、Address Generator についてその処理と構成について詳細に述べる。

4.1.1 Voxel Latch の構成

voxel Latch の制御は、Address Generator が行う。Voxel Latch の構成を、図 10に示す。

以下に、Voxel Latch の構成要素について述べる。

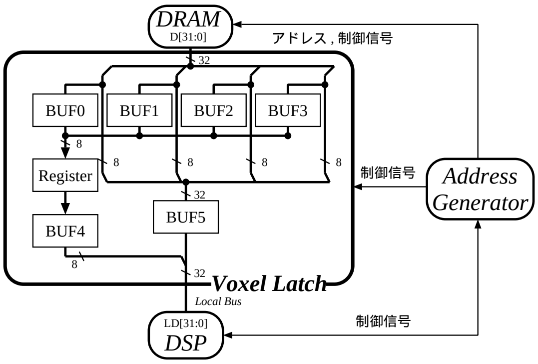

図 10: Voxel Latch の構成

- **Latch**:Address Generator によってプリロードされたボクセル値を保持する。クロックに同期してデータを取り込む 8bit のレジスタである。
- **BUF0～BUF3**:8bit のバッファである。ボクセル値は 8bit で表現されているが、格納する DRAM は 1M ワード×32bit であるので、ボクセル値を読み出すと、32bit 読み出されるが、必要な 8bit だけを Latch への入力とする必要がある。そこで、BUF0～BUF3 を制御することで必要な 8bit を選択し Latch への入力とする。この制御は、Address Generator が行う。
- **BUF4**:Latch の出力を制御する 8bit のバッファ。DSP がボクセル値の読み出しを行う時に、Latch の出力を DSP の Local Bus の下位 8bit への出力とする。これにより、DSP がボクセル値を読み出す時は、DRAM 内の格納場所に関係なく、常に下位 8bit(LD[7:0]) から読み出すことが出来る。
- **BUF5**:DSP の Local Bus の出力を制御する 32bit のバッファ。DSP から通常の DRAM へのアクセスの場合、DRAM のデータバスと DSP の Local Bus を接続する。

4.2 DSP の処理

本節では、DSP の処理の実現方法について述べる。

4.2.1 DSP 間通信

DSP 間には、ピクセル値計算に必要な種々のデータが流れる。DSP 間通信は DSP の Global Bus を用い、DSP 間に FIFO メモリを置くことで実現する。FIFO メモリは、TI 社の SN74ABT3611 を用いる。

DSP 間を流れるデータ

DSP 間を流れるデータのフォーマットを図 11に示す。

以下に、図 11のデータについて説明する。

2.4や 3で述べたように、ピクセル値の計算は主軸方向が正の時は式 (2) を計算し、負の時は式 (3) を計算する。よって、各 DSP は実行するプログラムを決定するのに主軸方向 (DIR) が必要となるので、最初に送信するデータ $(DATA0)$ に主軸方向を割り当てる。

主軸方向が正の場合は、前段から透明度の途中結果のデータ (透明度) と色の途中結果のデータ (R, G, B) を受信する必要があり、これらは $32bitfloat$ で表現されているので、$DATA1, DATA2, DATA3, DATA4$ にそれぞれを割り当てる。

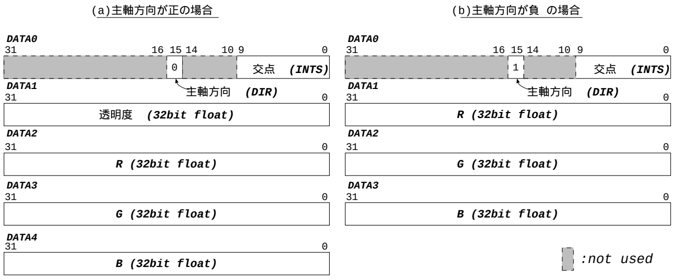

図 11: Global Bus を流れるデータ

主軸方向が負の場合は、前段から色の途中結果のデータ (R, G, B) を受信する必要があるので、$DATA1, DATA2, DATA3$ にそれぞれを割り当てる。

さらに、シェーディングステージでのシェーディング処理に、交点 $(INTS)$ のデータが必要となる。このデータは $10bit$ で表される。一方、$DATA0$ に割り当てた主軸方向は $1bit$ で表されるので、図 11のようにビットフィールドを割り当てデータのパックを行う。これにより、通信量の削減となる。

0 番の DSP は、視線生成ステージからのデータ受信を行い、処理の最初であるので透明度や色に関するデータは必要がない。そこで、視線生成ステージから 0 番の DSP に送るデータは、$DATA0$ のみとする。この時、交点のビットフィールドには、初期値として’10 0000 0000’ を入れる。

シェーディングステージに必要なデータは、主軸方向、交点、ピクセル値であるので、127 番の DSP はこれらのデータをシェーディングステージに送信する。

通信時間の隠蔽

通信用の FIFO メモリは Global Bus により DSP と接続される。

DSP の Global Bus は、2 つの制御系を持っているので、1 系統を前段の FIFO メモリからの受信のために、もう 1 系統を次段の FIFO メモリへの送信のために用いる。これにより、DSP 間の通信が通常のロード／ストア命令によって実現でき、DSP 間の処理の遅延の伝搬を防ぐことが出来る。

ここで、DSP が直接 FIFO メモリへのアクセスを行うように、制御信号を接続すると、FIFO への書き込みはウェイトなしのアクセスとなるが、読み出しはウェイトが 1 サイクル入るアクセスとなってしまう。

ピクセル値計算処理は粒度が細かく、通信が頻繁に起きる。したがって、ピクセル値計算ステージの処理速度は、この通信速度でほぼ決定される。

そこで、Global Bus の制御を行う **GBUS Controller** を用いる。GBUS Controller は DSP の Global Bus へのアクセスに同期して次のデータを先読みすることで、ウェイトなしの読み出しを実現する。データの読み出しの様子を図 12に示す。

ただし、図 12において、FIFO メモリ側の F要求 i は、FIFO メモリへ $DATAi$ の出力要求を行う時刻を示す。この要求は、GBUS Controller が行う。また、F要求 i○ は、$DATAi$ の読み出しに成功したことを示し、F要求 i× は、$DATAi$ の読み出しに失敗したことを示す。

同様に、DSP 側の D要求 i は、DSP が $DATAi$ の読みだし要求を行う時刻を示し、読み出しの成功は D要求○ 、

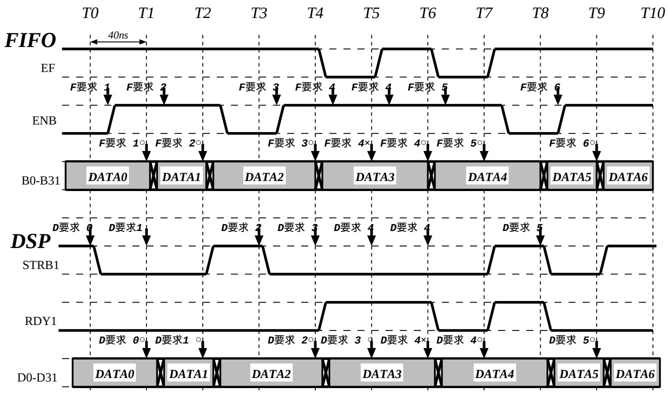

図 12: FIFO からデータを読み出す様子

失敗は D要求× 、でそれぞれ示す。

以上のような処理を GBUS Controller に行わせることで、FIFO にデータがある時はウェイトなしのアクセスが実現できる。なお、この処理を行うには、データの読み出しが行われるまで前のデータを出し続ける FIFO メモリが必要である [12]。

4.2.2 DSP のメモリマップ

ピクセル値計算の処理を行うための種々のデータを、図13に示す DSP のメモリマップにマッピングを行う。

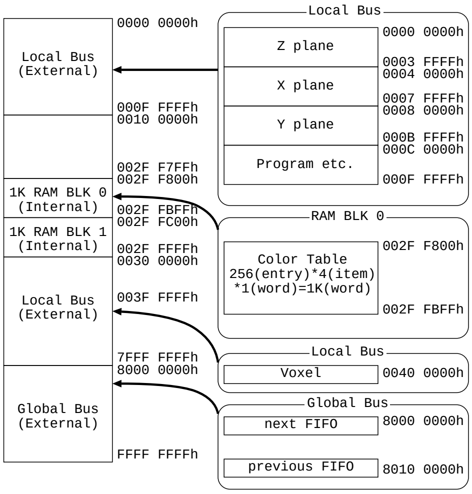

図 13: DSP のメモリマッピング

以下に、それぞれのデータについて述べる。

ボリューム・データ

ボリュームデータは DRAM に格納し、DRAM を 0000 0000h から 000F FFFFh 番地に割り当てる。ボリューム・データが 3MB に対し、DRAM は 4MB のものを使っている。ボリューム・データは、Z 平面を 0000 0000h から 0003 FFFFh 番値、X 平面を 0004 0000h から 0007 FFFFh 番地、Y 平面を 0008 0000h から 000B FFFFh 番値にそれぞれ割り当てる。

属性テーブル

ボクセル値から色と透明度に変換するのテーブルは、高速にアクセスできなければ処理速度に影響を及ぼすので、1 サイクルでアクセスの出来る DSP 内部の RAM ブロックを割り当てる。DSP 内部には 1K ワードの RAM ブロック 0, 1 が用意されている。属性テーブルの大きさは 1K ワードなので、属性テーブルに RAM ブロック 0 を割り当てる。よって、属性テーブルは 002F F800h から 002F FBFFh 番地を割り当てる。

また、属性テーブルは、α領域を 002F F800h から 002F F8FFh 番地、R領域を 002F F900h から 002F F9FFh 番地、G 領域を 002F FA00h から 002F FAFFh 番地、B領域を 002F FB00h から 002F FBFFh 番地にそれぞれ割り当てる。

プログラム

プログラムは、比較的小さいので DSP のキャシュに入る。よって高速にアクセスする必要がないため、DRAM に格納する。DRAM は、ボリューム・データを格納しても 1MB の空きがあるので十分な容量である。

作業領域

高速にアクセスできなければ行けないので、DSP 内部の RAM ブロック 1 の一部を割り当てる。

ボクセル値

Address Generator によってプリロードされたボクセル値は、Local Bus の制御系 1 によってアクセスすることで読み出すことが出来る。制御系 0 でアクセスできるアドレス空間を 4MB と設定し、ボクセル値の読み出しは 0004 0000h 番地へのロード命令で実現する。実際は、制御系 1 でアクセスのできる任意のアドレスへのロード命令でボクセル値の読み出しが行える。

FIFO メモリへのアクセス

前段の FIFO メモリからの読み出しは Globa Bus の制御系 1 でアクセスできる 8010 0000h 番地へのロード命令で実現し、次段の FIFO メモリへの書き込みは Global Bus の制御系 0 でアクセスできる 8000 0000h 番値へのストア命令で実現する。このとき Global Bus の制御系 0 でアクセスできるアドレス空間は 4MB と設定する。

実際は、前段の FIFO メモリからの読み出しは Global Bus の制御系 1 による任意のアドレスへのロード命令で実現でき、次段の FIFO メモリへの書き込みは制御系 0 による任意のアドレスへのストア命令で実現できる。

4.2.3 DSP プログラム

DSP で行う処理にかかる時間は、DSP のアセンブリ・プログラムを記述することで把握することが出来る。図 14 に DSP のプログラムを示す。

図 14 は、主軸方向 (DIR) が、正の場合は式 (2) を、負の場合は式 (3) をそれぞれ計算するプログラムである。ただし、1DSP で処理する平面数を 1 平面としている。

主軸方向が正の場合

隣接 DSP 間の通信について考える。送受信するデータが $5words$ であり、DSP 間のデータ通信は DSP プログラムのロード / ストア命令によって実現されるので、送受信に必要な命令数は 10 命令である。ロード / ストア命令は 1 サイクルで実行できるが、外部メモリへのアクセスの場合、連続した同一バスへのストアは 2 サイクルかかる。

次に、計算について考える。式 (2) は、減算 1 回、加算 3 回、乗算 5 回行うことで計算できる。DSP では、減算、加算、乗算とも 1 命令で行えるので、計算に必要な命令数は、9 命令である。

この他、主軸方向の判定のための、条件分岐のための比較・ブランチ命令が3命令必要であり、ボクセル値をロードするのに1命令必要である。

以上の処理を、行うには全部で23命令必要であるが、DSPには並列命令により、2命令を1サイクルで実行する機能があるので、これを利用すると、**17**サイクル必要であることがわかった。DSP内部のCPUでは、パイプライン処理によって高速化をはかっているが、様々な要因によってパイプライン処理が止まる場合がある。

主軸方向が負の場合

隣接DSP間の通信について考える。送受信するデータは$4words$であるので、主軸方向が正の場合と同様に考えると、通信に要する命令数は8命令となる。

計算について考えると、式(3)では、減算1回、加算3回、乗算6回、行わなければならない。よって、計算に要する命令数は10命令であるである。

また、主軸が正の場合と同様、主軸方向の判定のために3命令必要であり、ボクセル値のロードのために1命令必要である。

以上より、命令数の合計は22命令であるが、並列命令を利用すると、**15**サイクル必要であることがわかった。

主軸が正の場合と負の場合とでは、ほぼ同じ処理時間が必要となっているが、全く同じ処理時間ではない。

4.3 Address Generator

本節では、Address Generatorの動作と構成について詳細に述べる。

4.3.1 Address Generator 間の通信

隣接したAddress Genrator間の通信の実現方法について述べる。

Address Generator 間の通信データのフォーマット

レンダリング処理時の、Address Generator間を流れるデータのフォーマットを図14に示す。このデータは、視線生成ステージで生成され、パックされたものである。

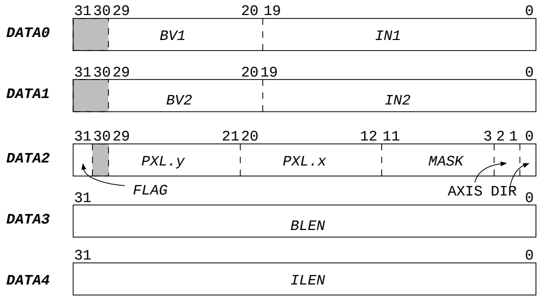

図14: Address Generator間を流れるデータ

送信されるデータについては、表1、表2に示している。

Address Generator 間の通信プロトコル

全Address Generatorが完全に、同じ動きをすることがわかっている場合は、データの送信する時刻と受信する時刻が決まっているので通信のための制御信号は必要ない。

しかし、主軸方向が正の場合と負の場合で処理時間が異なるため、ボクセル値の読み出しのタイミングが異なる。そこで、隣接したAddress Generator間で通信を行うための制御信号が必要となる。以下に、その制御信号と通信のプロトコルについて述べる。

0番のAddress Generatorは、視線生成ステージのAG_FIFOから視線データを受信することになる。AG_FIFOと0番のAddress Generator間、Address Generator間、127番のAddress Generatorとシェーディングステージ間、の通信が全て同じ通信プロトコルとするために、AG_FIFOと0番のAddress Generator間の通信を基準に、Address Generator間、127番のAddress Generatorとシェーディングステージ間の通信プロトコルを決定した。

Address Generator間の通信の様子を図15に示す。

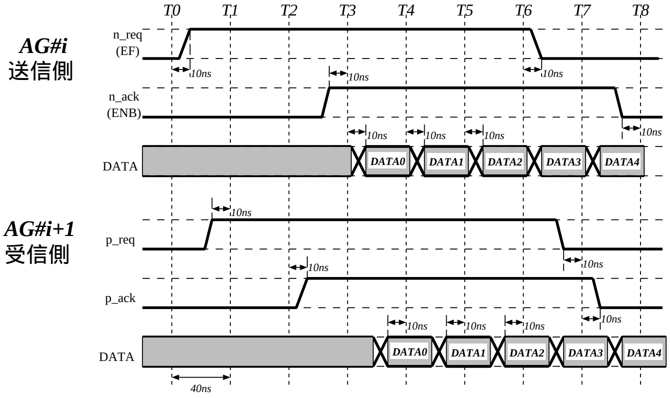

図15: Address Generator間の通信の様子

通信のための制御信号

Address Generator間通信を、次の制御信号によって実現する。

- 送信側の制御信号
  - **n_req**:HIGHにすることで送信の準備が出来たことを示す出力信号。実際には送信データの準備の整う$(T3)$の3サイクル前$(T0)$にこの信号をHIGHにする。この信号は、受信側のp_req信号に直結する。
  - **n_ack**:HIGHになるのを見て、受信側が受信の準備が出来たことを認識するための入力信号。送信側は、n_reqの信号をHIGHにしてから3サイクル後にはn_ackにかかわらず、$DATA0$の送信を開始する。送信側はn_ackがHIGHになるまで$DATA0$を送信し続け、n_ackがHIGHになると$(T3)$1サイクル後, $(T4)$に$DATA1$を送信する。その後は、1サイクル毎に$DATA2, DATA3, DATA4$を送信する。この信号は、受信側のp_ack信号に直結する。
- 受信側の制御信号
  - **p_req**:HIGHになるのを見て送信側の送信の準備が出来たことを認識するための入力信号。受信側で、既に受信の準備が出来ている時には、p_reqがHIGHになると$(T1)$1サイクル後$(T2)$にn_ack信号をHIGHにする。この信号は、送信側のn_req信号に直結する。
  - **p_ack**:HIGHにすることで、送信側に送信要求を出す出力信号である。この信号は、p_req信号がHIGHであり、かつ、受信の準備が出来てはじめてHIGHを出力する。受信の準備ができても、p_req信号が

LOW の時は、p_ack 信号は LOW を出力する。また、p_ack 信号を HIGH にする $(T2)$ と、その 5 サイクル後 $(T7)$ には LOW を出力する。T1 で p_ack を返すと同一基板への送信は間に合うが、異なる基板への送信が間に合わないため、$T2$ で p_ack を返す。

4.3.2 Address Generator(AG) の構成

Address Gnerator の構成を図 16に示す。

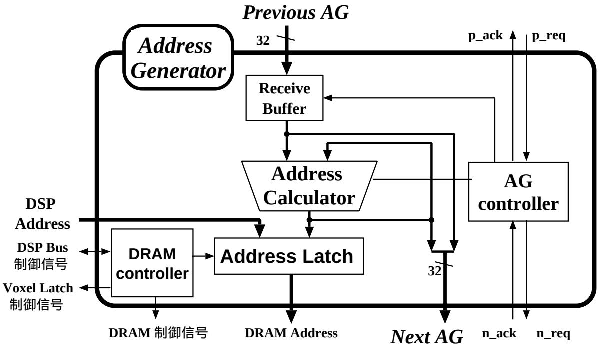

図 16: Address Generator の構成

以下では、Address Generator を構成する主な要素について述べる。

- **Address Calculator**:サンプリングするボクセルのアドレス計算を行う。
- **Address Latch**:Address Calculator で計算されたアドレスをラッチする。また、DRAM Controller の制御により、ボクセルの読み出しの場合ラッチしたアドレスを、DSP の DRAM アクセスの場合 DSP からのアドレスを、DRAM へ出力する。
- **Receive Buffer**:前段の Address Generator からの一視線分のデータをバッファリングするレジスタ。これにより、Address Generator 間での処理速度の遅延の伝搬を防ぐ。
- **AG Controller**:Address Generator の動作全体を制御する。具体的には、Address Generator 内のラッチのタイミング、マルチプレクサの制御、隣接の Address Generator との通信の制御を行う。
- **DRAM Controller**:DRAM へのアクセスを行う。また、DRAM にはリフレッシュ動作が必要なので、リフレッシュのタイミングの制御も行う。さらに、Address Latch の制御と、Voxel Latch の制御、DSP の Local Bus の制御も行う。

4.3.3 Address Calculator

サンプリングするボクセルのアドレスを計算する。また、マスク処理も行う。入力データは、初期座標 (IN1,IN2)、基本ベクトル (BV1,BV2)、サンプリング開始位置 (MASK)、主軸方向 (DIR)、主軸名 (AXIS) であり、初期設定データとして PE 番号 (PENUM)、ボクセル数 (VNUM)、平面数 (PLNUM) も用いる。初期設定データの表す意味は、以下の通りである。

- **PE 番号 (PENUM)**:Address Generator をパイプラインの下流から順に 1,2,$\cdots$,127 とつけた番号。
- **ボクセル数 (VNUM)**:処理するボリューム空間の 1 辺のボクセル数を表す。
- **平面数 (PLNUM)**:1PE が処理する平面数を表す。

Address Calculator の構成を図 17に示す。

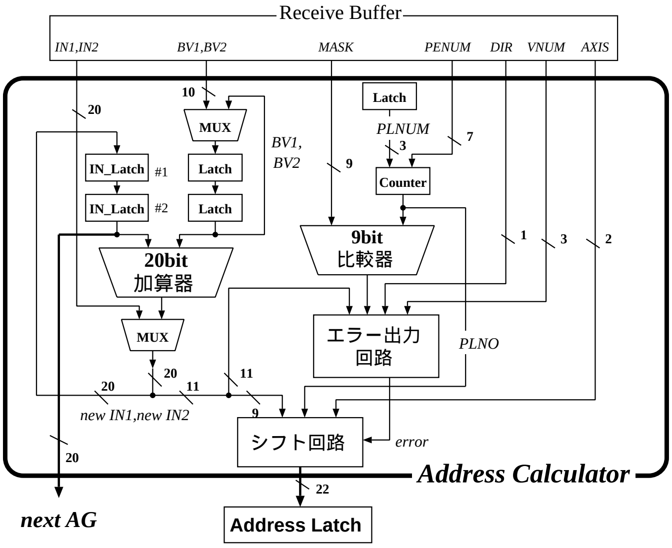

図 17: Address Calculator の構成

加算器

Address Calculator では IN1+BV1、IN2+BV2 の加算を行い、これらの結果からサンプリングするボクセルのアドレス計算を行う。FPGA の資源の都合上 20bit 加算器 1 つで実現し、パイプライン処理によって計算する。

その結果はアドレス計算で用いるが、次の平面の IN1、IN2 の値でもある。Address Generator は複数の平面の処理を行える構成となっている。複数の平面の処理は、加算器の結果を加算器の入力として、図 17の IN_Latch に取り込むよう帰還回路を設けることで実現している。

20bit 加算器の内部の構成と、データの流れを図 18に示す。

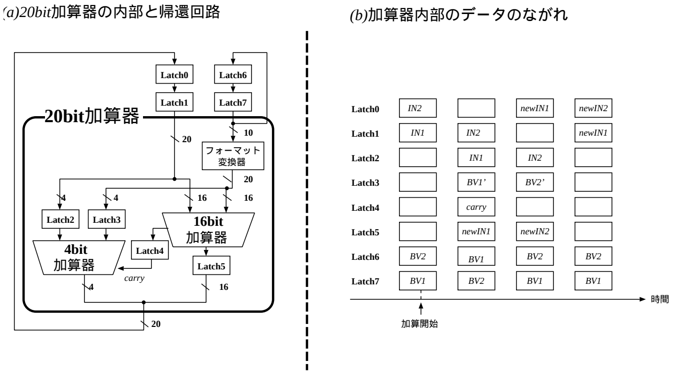

図 18: 20bit 加算器の詳細

20bit の加算は 1 サイクルでは行えないので、16bit 加算器と 4bit 加算器で構成し、間にラッチを置く。

IN1、IN2 は符合部 1bit、整数部 11bit、小数部 9bit、の 20bit 固定小数点であり、BV1、BV2 は符合部 1bit、小数部 9bit の 10bit 固定小数点であるので、加算を行う場合は、BV1、BV2 をそれぞれ 20bit 固定小数点の BV1'、BV2' に

変換してから、20bit 加算器への入力とする。

比較器

処理している平面番号とマスク開始位置 (MASK) を比較するための 9bit 比較器である。比較の結果はエラー出力回路への入力となる。

処理している平面番号 PENO は、初期設定データによって送られた PE 番号 (PENUM) を、1PE で処理する平面数 (PLNUM) の値に応じて PENO に変換して、カウンターにロードし、このカウンターの出力が処理している平面番号となる。AG Controller は 1 平面の処理が終了する毎に、カウンターへ出力を行い、PENO の値に 1 加える。

エラー出力回路

エラー出力回路は次の場合に、加算器の出力結果を無効化する。

1. マスク処理を行う:比較器の結果と、主軸方向からマスクをかけなければいけない場合。
2. サンプリング座標がボリューム外:加算器の出力結果、サンプリング座標がボリューム空間の外になる場合。ピクセル値計算ステージに送信される視線のデータは、ボリューム空間を (いつかは) 通過することが保証された視線だけであるが、この中には、サンプリング点の一部がボリューム空間の外となる視線も存在する。これは、加算器の出力結果とボクセル数 (VNUM) の値から判定できる。

上で述べたいずれかの場合、error 回路は error 信号をシフト回路に出力する。

シフト回路

加算器の出力結果を、主軸名 (AXIS)、処理している平面番号 (PENO)、から適切な bit 数だけシフトを行い、サンプリングするボクセルのアドレスを Address Latch へ出力する。また、error 信号が出力された場合、これらの値に関わらず 0x0070 0000 の値を出力する。

4.3.4 Address Latch

Address Latch は Address Calculator からの出力 22bit をラッチする。Address Latch の構成を図 19に示す。

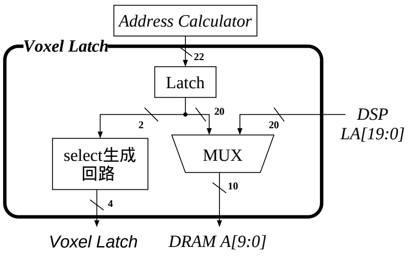

図 19: Address Latch の構成

ラッチした値の上位 20bit は、DRAM のアドレスとして用いる。ラッチした値の下位 2bit は、select 生成回路への入力データとなる。

select 生成回路

select 生成回路では、DRAM からの出力データの 4Byte の中から必要な 1Byte を Voxel Latch がラッチするように、Voxel Latch への制御信号を生成する。

マルチプレクサ

4bit マルチプレクサを 10 個並べたものである。DRAM へのアドレスは、カラム・アドレス、ロウ・アドレスの各 10bit に分割して送信しなければならない。また、DRAM へのアクセスは、Address Generator がボクセル値を読み出す場合と DSP が DRAM にアクセスする場合の 2 通りがある。そこで、このマルチプレクサで適切なアドレスを選択し DRAM へ送信する。このマルチプレクサの制御は DRAM Controller が行う。

22bit Latch のラッチのタイミングは AG Controller が行い、マルチプレクサは DRAM Controller が行う。

4.3.5 Receive Buffer

Receive Buffer は、Address Generator で視線 i の処理がまだ終っていない時に、前段の Address Generator から視線 $i+1$ のデータの送信要求があった場合、1 視線分のデータをバッファリングするためのレジスタである。視線 i の処理が終った後、視線 $i+1$ のデータを Receive Buffer から取り出すことで、処理の遅延の伝搬を防ぐことができる。

4.3.6 AG Controller

AG Controller は、Address Generator 内のラッチのタイミング、マルチプレクサを制御する。また、Address Generator 間の通信も行う。Address Generator 間の通信のタイミングは 4.3.1項で述べた通りである。

以下に、Address Generator が 2 平面 $p-1, p$ の処理をする場合を例にして、図 20を用いて述べる。

Address Generator 内を流れるデータは、初期座標 IN1,IN2 だけが処理を施され、データが書き換えられる。それ以外のデータは、視線生成ステージで作成されたデータがそのまま流れ、Address Generator は、その値を参照するだけである。

よって、次のようなことを実現して、処理の高速化をはかっている。よって図 20では Address Generator で行われる処理を、サンプリング開始位置 (IN1,IN2) のデータの流れに着目している。

ここで、図の IN_p, A_p, V_p はそれぞれ、平面 p を処理するために必要な、初期座標データ、これから求められるアドレス、アドレスを用いて読み出したボクセル値を表す。

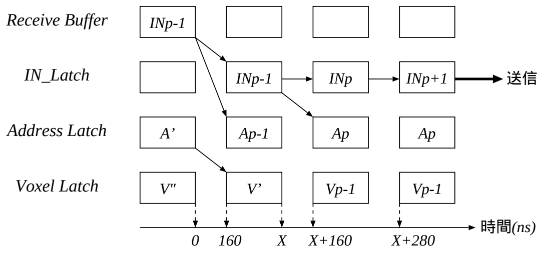

図 20: Address Generator の動作の様子

1. 図 20は、Voxel Latch、Address Latch がともにデータを保持しており、Address Latch が空でないため前段から送られてきた IN_{p-1} は Receive Buffer に保持される。
2. 時刻 0 で DSP によりボクセル値 V'' が読み出されると、これに同期して、Address Generator は次の処理をパイプライン動作で行う。

まず、Address Latchに保持されていたA'をもとにボクセル値V'を読み出し、Voxel Latchが保持する。同時に、Receive Bufferに保持していたIN_{p-1}をIN_Latchに移し、またIN_{p-1}からA_{p-1}を求め、Address Latchが保持する。これらの動作が完了すると、Receive Bufferが空になるので、次のデータの受信可能状態となる。これらの動作が完了するのに、$160ns$かかることが設計の結果わかった。

3. 次にV'が時刻Xに読み出されるとすると、これに同期して、次のような動作を開始する。

   先ほどと同様に、Address Latchに保持されていたA_{p-1}からボクセル値V_{p-1}を読み出しVoxel Latchが保持する。同時に、IN_Latchに保持されていたIN_pからA_pを求め、Address Latchが保持する。この動作が完了した時刻(X+160)に、次段のAddress Generatorに受信要求を前もって出す。

4. 3.の動作に続いて、IN_Latchに保持されているIN_pを用いて、新たなIN_{p+1}を求めIN_Latchに取り込む。その結果を、次段のAddress Generatorへ送信を開始する。

送信が行われることで、Address Generator内のデータを保持するLatchが空になった時点で、Address Generatorは次の動作に進むことが可能となる。

4.3.7 DRAM Controller

DRAM ControllerはDRAMの制御、DSPのLocal Busの制御、Voxel Latchがボクセル値をラッチするタイミングの制御を行う。これらを実現するために、DRAMの制御信号の生成、適切なタイミングでのアドレス送信、DSPのLocal Busの制御信号の受けとりおよび生成、Voxel Latchの制御を行う。

DRAMへのアクセス

DRAMへのアクセスは、DSPのDRAMへのアクセス、Address Generatorのボクセル値の読み出し、DRAMのリフレッシュがあげられる。ボクセル値の読み出しは、DRAMへのリード・アクセスを行い、Voxel Latchを制御することで実現できる。

DRAMのリフレッシュ

今回用いたDRAM(NEC社:MC-421000A32-60)は、1024リフレッシュ/16msのリフレッシュ・レートが必要なため、25MHzのクロックで動作させた場合、約390サイクル/リフレッシュの計算になる。リフレッシュ要求が発生した時に、DRAMにアクセスしている途中であった場合、アクセスが終了した後にリフレッシュを開始することにする。390サイクル/リフレッシュに余裕を持たせて、352サイクルに1度リフレッシュを実行することにする。この回路は、カウンタで実現する。また、リフレッシュ要求とメモリアクセス要求が同時に発生した場合は、リフレッシュ要求を優先させるものとする。

リフレッシュの要求とAddress Generatorのボクセル読み出しの要求が同時に発生した場合、ボクセルの読み出しが遅れるが、DSPの読み出し要求までには間に合い、DSPの処理には影響を与えない設計になっている。

4.4 性能の見積り

本節では、本研究で開発を行ったピクセル値計算ステージの性能の見積りを行う。性能の見積りに当たっては、ボリュームデータを$512^3 voxel$とし、スクリーンの大きさを$512^2 pixel$としている。

ピクセル値計算ステージの性能を見積もる場合、各ピクセル値計算ユニットにおける次の処理時間を見積もる必要がある。

1. DSPが視線iの平面P上のボクセルの値をVoxel Latchから読み出してから、平面$P+1$上のボクセルをVolxel Latchから読み出すまでの時間**DSP_t**。
2. Address Generatorが視線iの平面P上のボクセルの値をDRAMから読み出してから、次の平面$P+1$上のボクセルの値をDRAMから読み出すまでの時間**AG_t**。

DSP_t<AG_tの場合、Address Generatorの性能がPCSの性能を決定する。また、DSP_t>AG_tの場合、DSPの性能がピクセル値計算ステージの性能を決定する。DSP、Address Generatorともに全てのPCUにおいてほぼ同じ処理を行い、DSP間、Address Generator間のパイプラインとも処理の遅延の伝搬が生じない設計となっているので、DSP_t、AG_tを見積もることによって、ピクセル値計算ステージの性能を見積もることができる。

DSP_tの見積り

DSP_tは、4.2で述べたように、DSPのアセンブラ・プログラムから求めることが出来る。図5のプログラムから見積もった結果、1DSPが1平面を処理する場合に要するサイクル数は、主軸が正の時には**17**サイクルであり、主軸が負の時には**15**サイクルである。

DSPの1サイクルは40nsであるので、処理時間は主軸が正の時には**680ns**であり、主軸が負の時には**600ns**である。

次に、複数の平面の処理を1台のDSPで処理する場合に要する時間を考える。1台のDSPで2平面の処理をする場合、1平面の処理をする場合に要する時間の2倍にはならない。これは、主軸方向の判定とそれに伴う条件分岐を行う必要がなくなるためと、通信が約半分で済むためである。

2平面を処理するプログラムから見積もると、2平面を処理するのに要するサイクル数は、主軸が正の場合は**24**サイクルであり、主軸が負の場合は**22**サイクルとなった。プログラムは、図5に示す。

よって処理時間は、主軸が正の場合は**960ns**であり、主軸が負の場合は**880ns**である。

この結果、平面Pと平面$P+1$を同じDSPが処理する場合その処理時間は(960-680)ns=280nsであり、平面Pと平面$P+1$を異なるピクセル値計算ユニットが処理する場合**DSP_t=680ns**であると見積もることが出来る。

ただし、いずれの場合も、最長経路となる視線方向が正の場合で見積もっている。

AG_tの見積り

AG_tは、次の処理時間によって見積もることができる。

1. アドレス計算ための加算器の計算時間ADD_t。
2. 加算器の結果からアドレスを計算する時間SHIFT_t。
3. ボクセルを読み出すDRAMへのアクセス時間READ_t。

4. Address Generator 間の通信時間 COM_t。

設計した結果、ADD_t=120ns、SHIFT_t=80ns、READ_t=160ns、となった。COM_は、以下のように定める。

- 送信側は送信データの計算が終了した後に受信側に対し受信要求を出す。
- 受信側は受信データを全部受けとった後アドレスの計算を始める。

これによって COM_t = 440ns となる。

実際の通信は、送信側が送信データの計算が終了する前に、受信側に対し受信要求を出す。また、受信側は受信データを全部受けとる前に計算に必要なデータが揃い次第計算を始める。よって、実際は上で定義した COM_t より少ない時間で通信は実現されている。

平面 Pと平面 $P+1$ を同じピクセル値計算ユニットが処理する場合、AG_t = ADD_t+SHIFT_t = 200ns となる。これは、アドレスの計算と DRAM へのアクセスはパイプラインで処理するため、完全にオーバーラップさせせることができ、ADD_t+SHIFT_t>READ_であるアドレス計算の時間が Address Generator の処理時間を支配する。よって、**AG_t = 200ns** となる。

平面 Pと平面 P+1 を異なるピクセル値計算ユニットが処理する場合、AG_t = ADD_t + COM_t = 560ns となる。これは、平面 Pのアドレス計算と平面 P+1 のアドレス計算が同時に始まり、平面 Pのアドレス計算は ADD_t+SHIFT_t の時間を要するのに対し、平面 $P+1$ のアドレス計算は ADD_t+ADD_t+COM_t+SHIFT_t の時間を要するので、その差が AG_t となる。よって、**AG_t = 560ns** となる。

PCS の性能

以上の DSP_t と AG_t を比較すると、DSP の計算速度がピクセル値計算ユニットの処理速度を支配することがわかる。

先述した通り DSP が 1 枚の平面を処理する場合、その処理時間は 680ns であり、1 台の DSP が 2 枚の平面を処理する場合、その処理時間は 960ns である。

これより、1 台の DSP が n 平面の処理をする場合の処理時間は (n-1)×280 + 680(ns) と推測できる。

よって、$512^3 voxel$の処理を行う場合、128 台構成のピクセル値計算ステージでは、1 台の DSP が 4 平面を処理することになり、その処理時間は 1520ns という見積りになる。

ピクセル値計算ステージの性能は、$1520ns \times 512^2 pixel = 398ms/frame$ という性能になる。これは、約 2.5frame/sec の性能である。ただし、画面は $512^2 pixel$とし、以下の性能評価においても画面を $512^2 pixel$とする。

$128^3 voxel$の処理を行う場合、1 台の DSP は 1 平面を処理することになるので、ピクセル値計算ステージの性能は、$680ns \times 512^2 pixel = 178ms/frame$ の性能となる。

4.5 考察

実装の結果、$512^3 voxel$のレンダリングは $398ms/frame$ の速度で、$128^3 voxel$のレンダリングは $178ms/frame$ でそれぞれ行え、これらの処理時間は DSP の処理時間で決定されることがわかった。

ピクセル値の計算は、視線の方向に依らず単方向のパイプラインで行うため、DSP では主軸方向によってその計算方法を変える必要がある。今回行った実装では、視線毎にこの主軸方向の判定を行っており、この判定に要する時間によって性能の低下を招いている。

そこで、視線方向の判定の回数を減らすために、次の 2 つの方法が考えられる。

視線生成ステージと画面を単位としたパイプライン動作

本研究で行った設計では、視線生成ステージとピクセル値計算ステージは、視線を単位としてパイプライン動作を行っている。これを、画面を単位としたパイプライン動作させ、以下のような処理によって、視線方向の判定の回数を減らすことが可能となる。

視線生成ステージは、1 画面の視線データを生成し、全視線データの生成が終るまで、視線データの送信を行わず保持しておく。ただし、今回実装した視線生成ステージの PE に備え付けられた Local Memory の容量では不足するので、十分な容量のメモリが必要である。

全 PE の視線生成処理が終了すると、主軸方向が正の視線だけを連続して送信し、全 PE の正の視線の送信の終了の同期をとる。

全 PE の送信が終了すると、代表の PE が割り込み信号を生成する。通信用バッファとして今回採用した FIFO メモリは 36bit 幅のものであるが、利用しているのは 32bit だけである。そこで、図 21に示すように、生成した割り込み信号を FIFO メモリの利用していない 33bit 目に送信する。

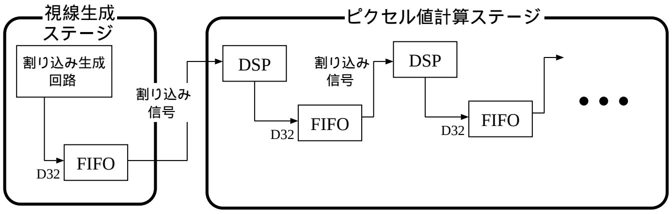

図 21: 割り込み信号の送信方法

FIFO メモリの出力ポートの 33bit 目を DSP の割り込み信号の入力とする。ピクセル値計算ステージの DSP はこの割り込み信号を受けとるまで、主軸方向が正の処理のプログラムを実行し、割り込み信号を受けとると主軸方向が負の処理のプログラムの実行を開始する。

割り込み処理は、たかだか数視線分の処理程度であり、この方法によって、1 画面の生成に主軸方向が 2 度しか変化しないので、性能の向上がはかれる。この方法による性能は以下のように見積もれる。

主軸の方向の変化による割り込み処理は、全処理量と比べて十分に小さいので無視できる。よって、今回実装したプログラムの中の、主軸方向の判定とそれに伴う分岐命令の 2 命令を実行する必要がなくなる。

$512^3 voxel$のレンダリングは、全視線の主軸方向が正の場合は、$1440ns \times 512^2 = 377ms/frame$ の性能であり、全視線の主軸方向が負の場合は $1320ns \times 512^2 = 246ms/frame$ の性能となる。この方法では、主軸方向が正の視線と負の視線の視線数の割合が、性能に大きく影響を与える。すなわち、負の方向の視線の割合が多いほど $246ms/frame$

の性能に近付き、逆に正の方向の視線の割合が多いほど $377ms/frame$ の性能に近付く。この性能は、今回実装したものに比べ、約 1.15 倍から 1.06 倍の性能の向上が予想される。

同様に 128^3voxelのレンダリングは、全視線の主軸方向が正の場合は $600ns \times 512^2 = 157ms/frame$ の性能となり、全視線の主軸方向が負の場合は $520\mathrm{ns}\times512^2 = 136ms/frame$ の性能となる。今回実装したものに比べ、約 1.31 倍から 1.14 倍の性能向上が予想される。

パイプラインの 2 重化

パイプラインの二重化によっても、主軸方向の判定回数を減らすことができる。

今回実装したピクセル値計算ステージは 1 次元に接続した 128 台の PE のパイプライン 1 本の構成である。これを、64 台の PE のパイプライン 2 本にし、2PE が 1 メモリを共有する構成とする (図 22参照)。1 本のパイプライン構成時と比べ、各 PE では 2 倍の平面数の処理をしなければならないが、全く独立に 2 視線が同時に処理でき、必要なメモリ量も半分になる。

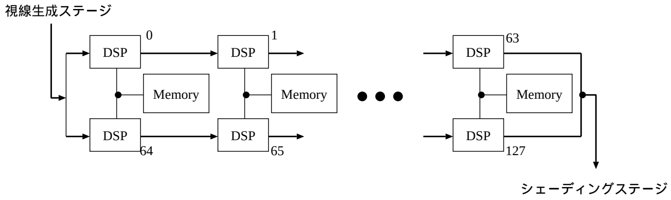

図 22: パイプラインの 2 重化

この構成によって、視線方向の判定の回数とともに DSP 間の通信量も半分となり、性能の大幅な向上がはかれると考えられる。そこで、この構成による性能を以下に見積もる。

512^3voxelをレンダリングする場合、1 台の DSP では 8 枚の平面を処理することになり、その処理時間は $(8-1)\times 280+ 680=2640ns$ となる。パイプラインが 2 本あるので、ピクセル値計算ステージの性能は、$2640ns\times512^2pixel/2 = 346ms/frame$ となる。これは、約 $2.9frame/sec$ の性能であり、パイプラインが 1 本時に比べ 1.15 倍の処理速度となる。

同様に 128^3voxelをレンダリングする場合、1 台の DSP では 2 枚の平面を処理することになり、その処理時間は $960ns$ となる。よって、ピクセル値計算ステージの性能は、$960ns\times512^2pixel/2 = 125ms/frame$ となる。これは、約 $8.0frame/sec$ の性能であり、1.38 倍の処理速度となる。

一方、この構成ではメモリを 2 台の DSP で共有しているため、ボクセル値読み出しに要求される速度が 2 倍になる。

5　おわりに

本論文では、ボリュームレンダリング専用並列計算機 it ReVolver/C40 について、ピクセル値計算ステージを中心に述べた。

$ReVolver$/C40 では、近年需要の高まってきている科学技術計算の結果の可視化を行うために、半透明ボリュームを遠近法によって高速に表示を行うことを目標とした。

この目標の実現のために、ピクセル値計算ステージでは、サンプリング方法を単純化し、データを 3 重化することで、メモリアクセスのバンク ·コンフリクト ·フリーを実現した。

ピクセル値計算ステージは、このメモリ構成のメモリバンクを 1 台の PE にローカルメモリとして割り当て、PE を 1 次元に接続した構成とした。そしてこれらの PE によるパイプライン処理とパイプラインの単方向化により、処理の高速化をはかった。

本研究では、処理の高速化を最優先し、さらに、コストの抑制、拡張性、様々な処理への柔軟な対応、を設計方針としてピクセル値計算ステージの設計を行った。

処理の高速化を達成するために、ボクセルの読み出し時間の隠蔽、ピクセル値計算と通信の高速化を実現した。

また、ピクセル値計算の処理に汎用の DSP を採用することで、柔軟性を高めた。今回設計したピクセル値計算ステージは 128 台構成であるが、512 台構成にも対応でき、また逆に小規模な構成にも対応している。

実装の結果、512^2pixelによる表示は、512^3voxelのレンダリングは $398ms/frame$ の速度で行え、128^3voxelのレンダリングは $178ms/frame$ で行える。この性能の見積りをした結果、次のようなことがわかった。

ピクセル値の計算は、視線の方向に依らず単方向のパイプラインで行うため、視線方向によって各 DSP の処理内容を変える必要がある。今回行った実装では、視線毎にこの視線方向の判定を行っており、この判定とそれにともなう分岐命令によって性能の低下を招いている。

そこで、1) 視線生成ステージとピクセル値計算ステージの画面を単位としたパイプライン処理や、2) パイプラインの 2 重化、の手法により性能の向上が見込める。

今回実装したピクセル計算ステージの処理能力は、今回採用した DSP の処理能力がネックとなっている。そこで今後、周波数の高い DSP を用いることで性能の向上がはかれる。

近年のプロセッサの処理能力の目覚しい向上とともに、ピクセル値計算ステージの処理能力も向上する可能性を秘めている。構造的には、メモリアクセスのサイクルでピクセル値計算が可能である。

今後の課題としては、先ほど述べた方法を実装することでさらなる性能の向上をはかり、また、DSP を用いて実装していることを利用して、レイトレーシングなど様々な画像生成手法の実装を行うことが挙げられる。

謝辞

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甚な謝意を表します。

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、木綿芳行氏、本多俊之氏、をはじめ、計算機工学講座計算機構成分野の諸氏に感謝いたします。

参考文献

[1] 中山明則, 他: ボリュームレンダリング専用並列計算機 ReVolver/C40 の試作, 情報処理学会第 50 回全国

大会講演論文集 (6) (1995).

[2] 對馬 雄次 and 他: ボリュームレンダリング専用並列計算機 *ReVolver*/C40, JSPP'95 論文集, pp. 11–18 (1995).

[3] 本田俊之: 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 (1996).

[4] 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 (1995).

[5] 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 (1996).

[6] 藤代一成, 茅暁陽: ダイレクト・アプローチ, *PIXEL*, No. 121, 図形処理情報センター, pp. 130–137.

[7] Pfister, H., Kaufman, A. and Chiueh, T.: Cube-3: A Real-Time Architecture for High-Resolution Volume Visualization, In *1994 Workshop on Volume Visualization*, pp. 75–83 (1994).

[8] 明石英也: ボリューム・レンダリング向き並列計算機のアーキテクチャ, 修士論文, 京都大学 (1992).

[9] Gordon: Image Space Shading of 3-Dimensional Objects, *CGIP*, pp. 361–376 (1985).

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

[11] 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).

[12] Texas Instruments: *High-Performance FIFO Memories Data Book* (1994).

付録

SCU の設計

SCU は、ホストマシンから送られる、*ReVolver*/C40 に対する処理コマンドやデータを受けとり、適切な処理を行って、*ReVolver*/C40 全体の制御を行う。

ホストマシンは SBus を装備したワークステーションをホストマシンとして使用することを想定しており、このバスを通してホストマシンとの間でデータ、コマンド等の授受を行う。

ホストマシンから *ReVolver*/C40 に対する処理は、以下のものがあげられる。

- 各ステージに対するリセット。
- 各ステージの DSP に対するデータアクセス。
- 視線生成ステージ、ピクセル値計算ステージに対する割り込み信号の生成。

SCU の構成と処理の概要

ホストマシンから *ReVolver*/C40 に対する処理は、全てホストマシンから SCU に対してのデータアクセスを行うことで実現する。

SCU の構成の概略を図 23に示す。SCU は FPGA(XILINX 社:XC4010DPQ160-5) で実現する。

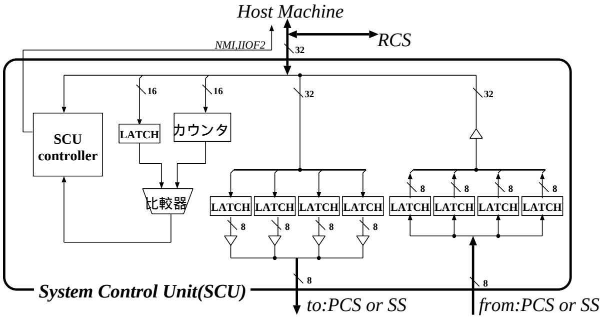

図 23: SCU の構成

- **SCU Controller** :SCU の動作全体を制御する。具体的には、ホストの Sbus カードから出ている Global Bus の制御、Sbus カードへの割り込み信号の生成、各ステージとの通信、SCU 内のラッチのタイミングの制御、SCU 内のバッファの制御を行う。
- カウンタ :各ステージへデータアクセスを行う場合に、アクセスを行った回数を数える 16bit カウンタ。
- 比較器 :ホストマシンから SCU に対して送られたデータアクセスを行う回数を示すデータの値と、カウンタの出力を比較する 16bit 比較器。

また、SCU の行う処理は、図 24に示す動作を行わせることで実現する。

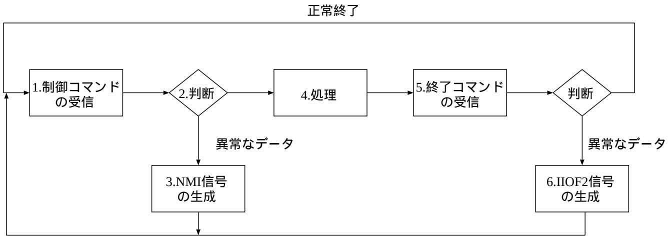

図 24: SCU の処理のながれ

1. SBus からの SCU 制御コマンドを受信する。
2. 送られてきたデータから、行う処理を判断する。
3. 送られてきたデータが不適切な場合、SBus カードへの割り込み信号 (NMI) を生成する。
4. 処理を行う。
5. 処理が終了すると、SBus カードから終了コマンドを受信する。
6. 終了コマンドが不適切な場合、ホストマシンへの割り込み信号 (IIOF2) を生成する。

SCU と各ステージとのデータ通信

本節では SCU と各ステージとのデータ通信について詳細を述べる。

視線生成ステージへのメモリアクセスRCS へのメモリアクセスは、RCS の全ての PE で共有されている View Buffer を、ホストマシンのメモリにマップし、ホストマシンから直接読み出し/書き込みを行うことで実現されている。

View Buffer への書き込み

n ワードの View Buffer への書き込みは、次に示すデータ列を送信することで実現する。

1. *DATA0* :SCU に対する SCU 制御コマンド送信であり、コマンドを受信した SCU は、View Buffer への書き込みと判断する。
2. *DATA1* :SCU に対するデータ送信であり、*DATA1* は書き込みを行う DATA 数を表す。よって、n ワードのデータを書き込む場合、*DATA1* の値は n となる。SCU はこのデータをラッチする。
3. *DATA2* ~ *DATAn+1* :View Buffer への書き込むデータ。
4. *DATAn+2* :SCU に対する終了コマンドの送信であり、処理が終了したことを示すデータである。

SCU は、ラッチした *DATA1* の値と、その後の View Buffer への書き込みの回数をカウントした値を比較し、同じ値になった時、View Buffer への書き込み処理を終了する。*DATAn+2* の値が、正しい値でない時は、SCU は Sbus カードに対しての割り込み信号を生成して処理を終了する。

また 1 度に書き込みできるデータ数は、View Buffer の容量と同じ最大 64Kword とする。

View Buffer からの読み出し

この処理は、先ほど述べた View Buffer への書き込みの処理とほぼ同様である。n ワードの View Buffer からの読み出しは、次に示すデータ列を送信することで実現する。

1. *DATA0* :SCU に対する SCU 制御コマンドの送信であり、このデータを受信した SCU は、View Buffer への書き込みと判断する。
2. *DATA1* :SCU に対するデータ送信であり、*DATA1* は読みだしを行うデータ数を表す。SCU はこのデータをラッチする。
3. *DATA2* :SCU に対する終了コマンドの送信であり、処理が終了したことを示すデータである。

DATA0 と *DATA1* を送信した後、読み出しを行う。読み出しが終了した後、*DATA2* を送信する。

1 度に View Buffer 読み出せるデータ数は、書き込みと同じく最大 64K ワードとする。

PCS、SS へのメモリアクセスホストマシンから、各ステージの DSP のメモリ空間へのアクセスは、次のような方法で実現する。各ステージに代表の DSP を設けて、SCU は、この代表の DSP に対してのみデータを送信し、残りの DSP に対しては、各ステージの持つ結合網によりデータ送信することで全 DSP への通信を実現する。

SCU と代表の DSP との通信は、Com Port により実現し、各ステージ内の結合も Com Port を用いて実現する。

SCU と各ステージの代表の DSP との結合と、各ステージ内の結合を図 25に示す。

図 25に示すように、PCS の 1 つのグループは 16 の DSP で構成されている。グループ内の DSP は 1 次元に接続され、単方向通信である。よって、SCU との通信は $DSP0$ が代表となって受信し、$DSP15$ が代表となって送信する。PCS では 128 の DSP が用いられているため、8 グループの構成となる。

SS 内では図 25で示すようにメッシュ状に接続されているが、1 次元の単方向通信は実現できるので、PCS と同様の通信を行う。よって、SS でも SCU との通信は $DSP0$ が代表となって受信し、$DSP15$ が代表となって送信する。SS では 16 の DSP が用いられているため、SS 全体で 1 グループとなる。

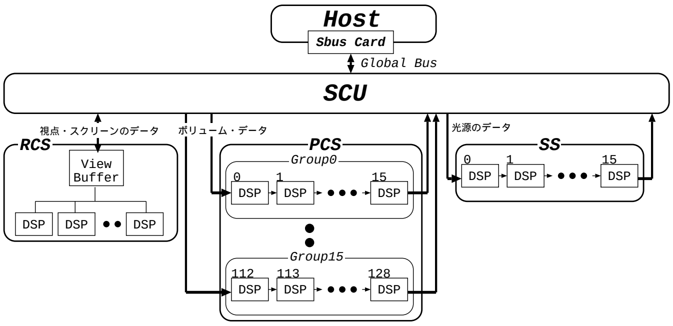

図 25: SCU と他のステージとの結合

PCS、SS へのデータ送信

n ワードのデータを DSP のメモリ空間へ書き込みを行う場合は、以下に示すデータ列を送信することで実現する。

1. *DATA0* :SCU に対する SCU データコマンドの送信であり、このデータを受信した SCU は、PCS または SS への書き込みと判断し、アクセスするグループも判断する。
2. *DATA1* :SCU に対するデータ送信であり、*DATA1* は書き込みを行う DATA 数を表す。SCU はこのデータをラッチする。n ワードのデータを書き込む場合、*DATA1* の値は $n+2$ となる。SCU はこの値をラッチする。
3. *DATA2* :各ステージの $DSP0$ に送信する最初のデータであり、DSP 番号と書き込みを行う先頭アドレスを表す。また、書き込みであることも示す。
4. *DATA3* :$DSP0$ に送信する 2 番めのデータであり、データ数を表す。*DATA3* の値は n となる。
5. *DATA4* ~ *DATAn+3* :実際に書き込むデータ。
6. *DATAn+4* :SCU に対する終了コマンドの送信であり、処理が終了したことを示すデータである。

$DSP0$ は SCU から最初のデータとして *DATA2* を受信し、続いて *DATA3* を受信する。

DATA2 の値から自分のメモリ空間へのアクセスであり、メモリ空間への書き込みであると判断すると、*DATA3* の値から書き込みを行うデータ数を認識する。そして、受信する *DATA4* ~ *DATAn+3* のデータを連続アドレスに書き込んでいく。

自分のメモリ空間へのアクセスではない場合、*DATA3* の値から SCU から送信されるデータ数を認識し、*DATA4* ~ *DATAn+3* を受信し、*DATA2*、*DATA3*、$\cdots$、*DATAn+3* の順で $DSP1$ に送信する。

$DSP1, \cdots, DSP15$ も同様の処理を行う。

また、1 度に送信できるデータ数は最大 (64M-2) ワードとする。

ピクセル値計算ステージ、シェーディングステージからのデータ受信

n ワードのデータを DSP のメモリ空間からの読み出しを行う場合は、以下に示すデータ列を送信することで実現する。

1. *DATA0* :SCU に対する SCU 制御コマンドの送信であり、このデータを受信した SCU は、ピクセル値計算ス

テージまたはSSからの読み出しと判断し、アクセスするグループも決定する。

2. *DATA1* :SCUに対するデータ送信であり、*DATA1*は読み出しを行うデータ数を表す。SCUはこのデータをラッチする。nワードのデータを読み出す場合、*DATA1*の値は$n+2$となる。SCUはこの値をラッチする。
3. *DATA2* :各ステージの$DSP0$に送信する最初のデータであり、読み出しを行うDSP番号と先頭アドレスを表す。また、読み出しであることも示す。
4. *DATA3* :$DSP0$に送信する2番めのデータであり、DATA数を表す。*DATA3*の値はnとなる。
5. *DATA4* :SCUに対する終了コマンドの送信であり、処理が終了したことを示すデータである。

ホストマシンは以上のデータを送信した後、受信状態になる。

$DSP0$は*DATA2*、*DATA3*を受信する。

*DATA2*の値から自分のアドレス空間へのアクセスであり、データの読み出しであることを判断すると、*DATA3*の値から読み出すデータ数を認識し、データの読み出しを連続アドレスで行う。*DATA2*、*DATA3*、読み出したデータの順で$DSP1$に送信する。

自分のアドレス空間へのアクセスでないと判断すると、*DATA2*、*DATA3*の順で$DSP1$に送信する。

$DSP1$～$DSP15$は*DATA2*、*DATA3*を受信すると、自分のアドレス空間への処理かどうかを判断する。

自分のアドレス空間へのアクセスである場合、$DSP0$と同様の処理を行って、データの読み出しを行う。

自分より小さい番号のDSPへのアクセスである場合、*DATA3*の値のデータ数だけ前段のDSPからデータを受信し、*DATA2*、*DATA3*、受信したデータの順で次段のDSPに送信する。

自分より大きい番号のDSPへのアクセスである場合、*DATA2*、*DATA3*を次段のDSPに送信する。$DSP15$の場合は、次段のDSPへの送信はSCUへの送信に相当する。

SCUは、トリプル・ハイトVMEボードに実装する。この基盤には、SCUを実現するFPGAの他に12.5MHzのクロック発信器も実装する。クロック発信器で発生したクロックは、PPLへ入力する。PPLによって12.5MHzのクロックは、25MHzと50MHzのクロックに変換される。25MHzのクロックはSCUのFPGAに配られ、50MHzのクロックはRCSの基盤に配られる。PPLには増幅回路も内蔵されており、増幅された12.5MHzのクロックはピクセル値計算ステージとSSの基盤に配られる。SCUとホストマシン、他のステージとはフラット・ケーブルで接続する。

DRAMのタイミング

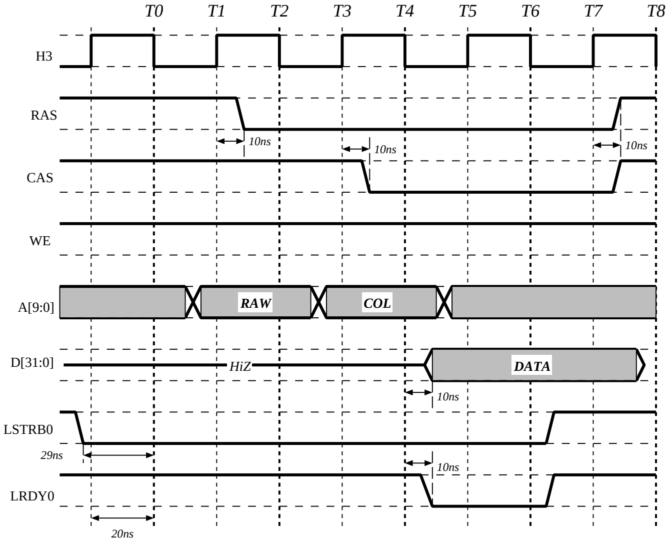

図 26: リードのタイミング

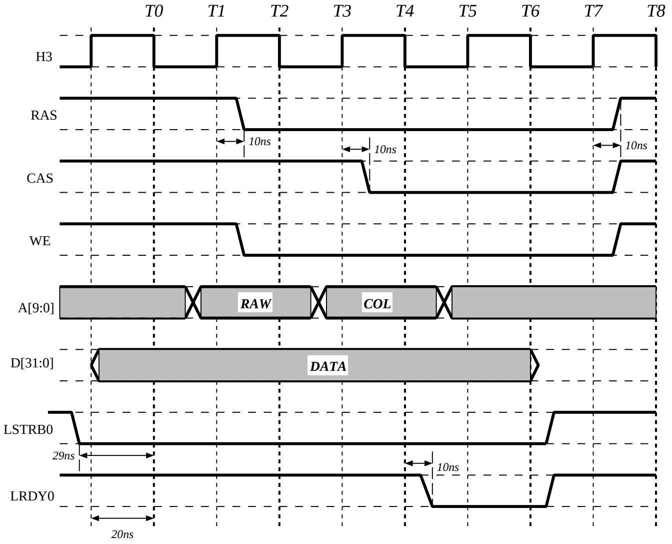

図 27: ライトのタイミング

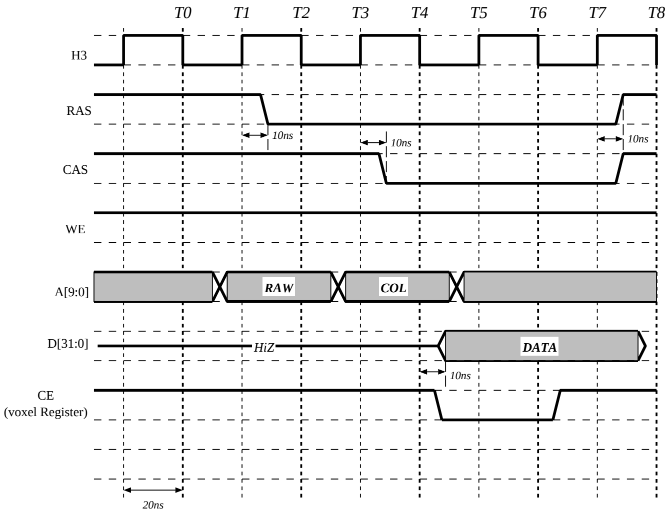

図 28: ボクセル読み出しのタイミング

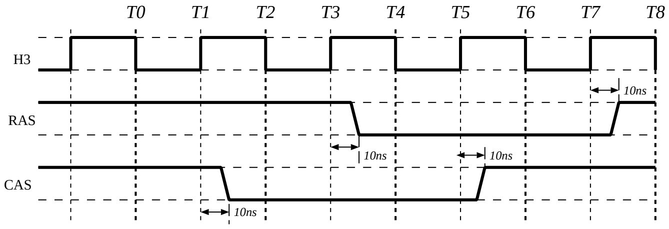

図 29: リフレッシュのタイミング

DSP のプログラム

Address Register(AR) の使用方法

AR0	→	属性テーブル (α) の先頭
AR1	→	属性テーブル (r) の先頭
AR2	→	属性テーブル (g) の先頭
AR3	→	属性テーブル (b) の先頭
AR4	→	作業領域
AR5	→	send(1)receive(0) 時のアドレス
AR6	→	1(0),0000 7FFFf(1)
AR7		ボクセル値の格納

() 内の数字は displacement

```
               LDF      *AR6,R7                 ; R7 ← 1
   LOOP:       LD       8000 0000h,RC           ; Repeat Counter のロード (MAX value)
               RPTB     ENDLOP                  ; リピート ・ブロックの開始
                                    ; 最内ループ
   STLOP:      LBU0     *AR7,IR0                ; IR0 ←ボクセル値の受信
               LDF      *AR5(0),R10             ; R10 ← next DATA0(DIR,INTS) の受信
               SUBF3    *AR6(1),R10,R9          ; DIRの判定
               BND      NEGATIVE                ; If(DIR==1) Brach after 3 order
               LDF      *AR5(0),R5              ; R5 ← DATA1 の受信
                                                ; R_{i+1}(DIR==1) or A_{i-1}(DIR==0)
               SUBF3    *AR0(IR0),R7,R6         ; R6 ← 1-α(v_i)
     ||        STF      R10,*AR5(1)             ; DATA0 の送信
               MPYF3    R5,*AR0(IR0),R4         ; R4 ← R_{i+1} × α(v_i)(DIR==1)
                                                ; R4 ← A_{i-1} × α(v_i)(= A_i)(DIR==0)
                                  ; 主軸が正の時の処理
               MPYF3    R5,R6,R6                ; R6 ← (1-α(v_i))× A_{i-1}
     ||        STF      R5,*AR5(1)              ;A_i(DATA1) の送信
               MPYF3    R6,*AR1(IR0),R5         ; R5 ← r(v_i) × (1 − α(v_i)) × A_{i-1}
               MPYF3    R6,*AR2(IR0),R3         ; R3 ← g(v_i) × (1 − α(v_i)) × A_{i-1}
               MPYF3    R6,*AR3(IR0),R1         ; R1 ← b(v_i) × (1 − α(v_i)) × A_{i-1}
     ||        ADDF3    R4,*AR5(0),R2           ; R2 ← R_{i-1}+r(v_i) × (1 − α(v_i)) × A_{i-1}(=R_i)
               ADDF3    R3,*AR5(0),R3           ; R3 ← G_{i-1}+g(v_i) × (1 − α(v_i)) × A_{i-1}(=G_i)
               ADDF3    R1,*AR5(0),R1           ; R1 ← B_{i-1}+b(v_i) × (1 − α(v_i)) × A_{i-1}(=B_i)
     ||        STF      R2,*AR5(1)              ; R_i(DATA2) の送信
               BRD      STLOP                   ; Branch STLOP After 3 order
               STF      R3,*AR5(1)              ; G_i(DATA3) の送信
               STF      R1,*AR5(1)              ; B_i(DATA4) の送信,1 delay after
                                  ; 主軸が負の時の処理
 NEGATIVE      MPYF3    *AR1(IR0),R6,R3         ; R3 ← r(v_i)×(1-α(v_i))
               MPYF3    *AR5(0),*AR0(IR0),R5    ; R5 ← G_{i+1} × α(v_i)
               MPYF3    *AR5(0),*AR0(IR0),R1    ;R1 ← B_{i+1} × α(v_i)
     ||        ADDF3    R4,R3,R3                ; R3 ← R_{i+1} × α(v_i)+r(v_i)×(1-α(v_i))(=R_i)
               MPYF3    *AR2(IR0),R6,R0         ; R0 ← g(v_i)×(1-α(v_i))
     ||        STF      R3,*AR5(1)              ; R_i(DATA1) の送信
               ADDF3    R5,R0,R5                ; R5 ← G_{i+1} × α(v_i)+g(v_i)×(1-α(v_i))(=G_i)
               MPYF3    *AR3(IR0),R6,R0         ; R0 ← b(v_i)×(1-α(v_i))
     ||        STF      R5,*AR5(1)              ; G_i(DATA2) の送信
               ADDF3    R0,R1,R0                ; R0 ← B_{i+1} × α(v_i)+b(v_i)×(1-α(v_i))(=B_i)
 ENDLOOP       STF      R0,*AR5(1)              ; B_i(DATA3) の送信
     BR        LOOP
```

図 30: 1 平面を処理する DSP プログラム

```
 1:               LDF              *AR6,R7                   ; R7 ← 1
 2:   LOOP:       LD               8000 0000h,RC             ; Repeat Counterのロード(MAX value)
 3:               RPTB             ENDLOP                    ; リピート ・ブロックの開始
                                                   ;最内ループ
 4:   STLOP:      LBU0             *AR7,IR0                  ; IR0 ←ボクセル値の受信
 5:               LDF              *AR5(0),R10               ; R10 ← next DATA0(DIR,INTS)の受信
 6:               SUBF3            *AR6(1),R10,R9            ; DIRの判定
 7:               BND              NEGATIVE                  ; If(DIR==1) Brach after 3 order
 8:               LDF              *AR5(0),R5                ; R5 ← DATA1の受信
                                                             ; R_{i+1}(DIR==1) or A_{i-1}(DIR==0)
 9:               SUBF3            *AR0(IR0),R7,R6           ; R6 ← 1-α(v_i)
          ||      STF              R10,*AR5(1)               ; DATA0の送信
10:               MPYF3            R5,*AR0(IR0),R4           ; R4 ← R_{i+1} × α(v_i)(DIR==1)
                                                             ; R4 ← A_{i-1} × α(v_i)(= A_i)(DIR==0)
                                                 ;主軸が正の時の処理
11:               MPYF3            R5,R6,R6                  ; R6 ← (1-α(v_i))×A_{i-1}
          ||      STF              R5,*AR5(1)                ;A_i(DATA1)の送信
12:               MPYF3            R6,*AR1(IR0),R5           ; R5 ← r(v_i) × (1 − α(v_i)) × A_{i-1}
13:               LBU0             *AR7,IR1                  ; IR1 ←ボクセル値の受信
14:               MPYF3            R6,*AR2(IR0),R0           ; R0 ← g(v_i) × (1 − α(v_i)) × A_{i-1}
          ||      ADDF3            R4,*AR5(0),R2             ; R2 ← R_{i-1}+r(v_i) × (1 − α(v_i)) × A_{i-1}(=R_i)
15:               MPYF3            R6,*AR3(IR0),R1           ; R1 ← b(v_i) × (1 − α(v_i)) × A_{i-1}
          ||      ADDF3            R0,*AR5(0),R3             ; R3 ← G_{i-1}+g(v_i) × (1 − α(v_i)) × A_{i-1}(=G_i)
16:               SUBF3            *AR0(IR1),R7,R6           ; R6 ← 1-α(v_{i+1})
          ||      STF R2,*AR4(1)   ; *AR4(1) ← R_i(退避)
17:               MPYF3            R4,R6,R6                  ; R6 ← (1-α(v_{i+1}))×A_i
18:               MPYF3            R4,*AR0(IR1),R0           ; R0 ← A_i × α(v_{i+1})(= A_{i+1})
          ||      ADDF3            R1,*AR5(0),R2             ; R2 ← B_{i-1}+b(v_i) × (1 − α(v_i)) × A_{i-1}(=B_i)
19:               MPYF3            R6,*AR1(IR1),R1           ; R1 ← r(v_i) × (1 − α(v_i)) × A_{i-1}
          ||      STF R0,*AR5(1)   ; A_{i+1}(DATA1)の送信
20:               MPYF3            R6,*AR2(IR1),R0           ; R0 ← g(v_{i+1}) × (1 − α(v_{i+1})) × A_i
          ||      STF R3,*AR4(0)   ; *AR4(0) ← G_i(退避)
21:               MPYF3            R6,*AR3(IR1),R1           ; R1 ← b(v_{i+1}) × (1 − α(v_{i+1})) × A_i
          ||      ADDF3            R1,*AR4(1),R3             ; R3 ← R_i+r(v_{i+1}) × (1 − α(v_{i+1})) × A_i(=R_{i+1})
22:               ADDF3            R0,*AR4(0),R0             ; R0 ← G_{i-1}+g(v_i) × (1 − α(v_i)) × A_{i-1}(=G_i)
          ||      STF              R3,*AR5(1)                ; R_{i+1}(DATA2)の送信
23:               BRD              STLOP                     ; Branch STLOP After 3 order
24:               ADDF3            R1,R2,R2                  ; R2 ← B_i+b(v_{i+1}) × (1 − α(v_{i+1})) × A_i(=B_{i+1})
25:               STF              R0,*AR5(1)                ; G_{i+1}(DATA3)の送信
26:               STF              R2,*AR5(1)                ; B_i(DATA4)の送信,1 delay after
                                                 ;主軸が負の時の処理
27:   NEGATIVE    MPYF3            *AR1(IR0),R6,R3           ; R3 ← r(v_i)×(1-α(v_i))
          ||      STF              R6,*AR4(0)                ; *AR4(0) ← 1-α(v_i)(退避)
28:               MPYF3            *AR5(0),*AR0(IR0),R5      ; R5 ← G_{i+1} × α(v_i)
29:               MPYF3            *AR5(0),*AR0(IR0),R1      ;R1 ← B_{i+1} × α(v_i)
          ||      ADDF3            R4,R3,R3                  ; R3 ← R_{i+1} × α(v_i)+r(v_i)×(1-α(v_i))(=R_i)
30:               LBU0             *AR7,IR1                  ; IR1 ←ボクセル値の受信
31:               MPYF3            *AR2(IR0),R6,R4           ; R4 ← g(v_i)×(1-α(v_i))
32:               MPYF3            *AR3(IR0),*AR4(0),R0      ; R0 ← b(v_i)×(1-α(v_i))
          ||      ADDF3            R5,R4,R2                  ; R2 ← G_{i+1} × α(v_i)+g(v_i)×(1-α(v_i))(=G_i)
33:               MPYF3            R3,*AR1(IR1),R0           ; R0 ← R_i × α(v_{i-1})
          ||      ADDF3            R0,R1,R3                  ; R0 ← B_{i+1} × α(v_i)+b(v_i)×(1-α(v_i))(=B_i)
34:               MPYF3            *AR1(IR1),R6,R4           ; R4 ← r(v_i)×(1-α(v_i))
35:               MPYF3            R2,*AR0(IR1),R5           ; R5 ← G_i × α(v_{i-1})
36:               MPYF3            *AR2(IR0),*AR4(0),R0      ; R0 ← g(v_{i-1})×(1-α(v_{i-1}))
          ||      ADDF3            R4,R0,R3                  ; R3 ← R_i × α(v_{i-1})+r(v_{i-1})×(1-α(v_{i-1}))(=R_{i-1})
37:               MPYF3            *AR3(IR1),R6,R0           ; R0 ← b(v_{i-1})×(1-α(v_{i-1}))
          ||      STF              R2,*AR5(1)                ; R_{i-1}(DATA1)の送信
38:               ADDF3            R5,R0,R5                  ; R5 ← G_i × α(v_{i-1})+g(v_{i-1})×(1-α(v_{i-1}))(=G_{i-1})
39:               STF              R5,*AR5(1)                ; G_{i-1}(DATA2)の送信
40:               ADDF3            R0,R1,R0                  ; R0 ← B_{i+1} × α(v_i)+b(v_i)×(1-α(v_i))(=B_i)
41:   ENDLOOP     STF              R0,*AR5(1)                ; B_i(DATA3)の送信
42:     BR        LOOP
```

図 31: 2 平面を処理する DSP プログラム