社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－133（18）
2008／1／17

並列ボリュームレンダリング・アクセラレータVisAの開発とその予備実装

川原 崇宏† 三輪 忍† 嶋田 創† 森 眞一郎†† 富田 眞治†

† 京都大学 〒606-8501 京都市左京区吉田本町
†† 福井大学 〒910-8507 福井市文京3-9-1
E-mail: †{kawahara,miwa,shimada,tomita}@lab3.kuis.kyoto-u.ac.jp, ††moris@fuis.fuis.fukui-u.ac.jp

あらまし 我々は並列処理可能なボリュームレンダリング・アクセラレータ『VisA』を開発している．VisAはディスプレイ用汎用インタフェースであるDVI-Dケーブルを通信路として用いた単方向リンクにより，ノード間の画像合成処理とノード内での画像生成処理を統合した細粒度パイプラインを実現している．本稿ではVisAの設計手法と予備実装による評価について述べる．

キーワード 可視化，ボリュームレンダリング，並列処理，FPGA，ハードウェアアクセラレータ

Development of Parallel Volume Rendering Accelerator VisA and its Preliminary Implementation

Takahiro KAWAHARA†, Shinobu MIWA†, Hajime SHIMADA†, Shin-ichiro MORI††, and Shinji TOMITA†

† Kyoto University Kyoto-shi, 606-8501, Japan.
†† University of Fukui Fukui-shi, 910-8507, Japan
E-mail: †{kawahara,miwa,shimada,tomita}@lab3.kuis.kyoto-u.ac.jp, ††moris@fuis.fuis.fukui-u.ac.jp

Abstract We are developing the parallel volume rendering accelerator VisA. VisA communicates with one-way link over DVI-D which is usualy used as a display interface, so it achieves the fine grain pipeline system which integrates both image composition between nodes and image rendering inside a node. This paper describes the design concept of VisA and its preliminary implementation.

Key words Visualization, Volume Rendering, Parallel Processing, FPGA, Hardware Accelerator

1. はじめに

1.1 背景

3次元データを直接可視化する技術が注目を集めている．可視化したい3次元データには，計算機による数値シミュレーションにより得られる結果や，医学におけるCTスキャン・MRIといった人体解析によって得られるデータなどがある．だが，3次元データを直接可視化するための解析処理は非常に莫大であり，特に大規模かつ高精細な実時間インタラクティブシミュレーションのような用途で扱うには困難である．

こうした背景をもとに可視化機構の並列処理による高速化に関する研究は各所で活発に行われており，中でも汎用GPUを直接接続して並列処理するSLIやCrossFireは強力な画像処理技術を提供する．しかし現段階では並列度に限りがあり，それをクリアする技術としてVGクラスタ[1]やORAD DVG[2]といったシステムがある．これは分散配置した汎用GPUで画像生成を分割処理し，専用並列アクセラレータで合成処理を行う技術である．また，複数の演算器を内包し，画像の生成から合成まで同じユニットで行う並列グラフィックスアクセラレータとしてPixelFlow[3]がある．これらの技術はVisA同様高速なデイジーチェーンを用いて並列処理を行う．しかしながら，いずれも並列処理はフレーム単位で行われるため，並列度が上がるとそれに応じて遅延が増大するという欠点がある．

それに対して我々は，高い並列度を提供すると同時に，ピクセル単位での並列処理システムを目指している．これまでの成果として並列処理可能な可視化ハードウェアReVolver/C40[4]を開発・評価し，その後継機であるVisA[5]を提案した．

本稿では，ReVolver/C40アーキテクチャを継承しつつ，実

時間インタラクティブシミュレーション中に得られる大規模な3次元データでも実時間可視化可能な専用並列ハードウェアVisAの開発とその予備実装について述べる．

1.2 ボリュームレンダリング

上述した可視化技術の代表的な手法にボリュームレンダリングというものがある．ボリュームレンダリングとは，3次元空間(ボリューム空間)に広がる数値データ(ボリュームデータ)に色と透明度を対応付け，2次元スクリーンへ投影することで複雑な内部構造や動的特性を可視化する手法である．具体的な処理手順として，まず可視化対象のボリューム空間を単位立方体格子で区切り，個々の格子上に存在する情報に色と透明度を対応付ける．色と透明度が対応付けられた格子をボクセルと呼ぶ．次に可視化する視点の位置を決め，視点からボリューム空間を眺めたとき投影スクリーン上の各ピクセルを通過する視線を算出する．投影スクリーン上の各ピクセル値は，算出された視線がボリューム空間を通過する際に衝突するボクセルを累積計算していくことによって決定される．

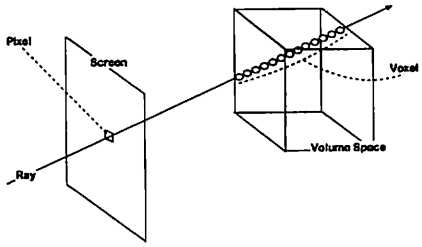

図1 ボリュームレンダリング

ある視線が n 個のボクセルを通過する際，通過するボクセルの色を c，不透明度を α としたとき，視線を累積計算するための式は次のようになる．

$$B_n = \sum_{i=0}^{n}(c_i \times \alpha_i \prod_{j=0}^{i-1}(1-\alpha_j)) = B_{n-1} + \alpha_n \times c_n \times A_{n-1} \quad (1)$$

$$A_n = (1-\alpha_n) \times A_{n-1} \quad (2)$$

ここで A_n は累積透明度を表しており，視線がボリューム空間を進む際に累積される透明度である．この式での累積計算はFront-to-Backアルゴリズムと呼ばれ，視線方向と同じ順番にボクセルを累積していくときに用いられる．これとは逆に，視線が通る最奥のボクセルから視線方向と逆順にボクセルを累積するBack-to-Frontアルゴリズムは以下の式により実現できる．

$$B_{k-1} = c_k \cdot \alpha_k + (1-\alpha_k) \sum_{i=k+1}^{n}(c_i \times \alpha_i \prod_{j=0}^{i-1}(1-\alpha_j)) = c_k \cdot \alpha_k + (1-\alpha_k)B_k \quad (3)$$

2. VisA

本章では我々が提案している並列ボリュームレンダリング・アクセラレータVisA(Visualization Accelerator)の紹介する．まずVisAの構成を述べた後，VisAの持つ特徴について述べる．

2.1 構　　成

VisAはボリュームレンダリングに必要不可欠な α ブレンディング処理を行うPCU(Pixel Calculate Unit)を複数搭載し，内部で直列に接続している(図2)．接続されたPCUの先頭には前ノードからデータを受け取るための外部入力端子が接続されており，後尾には次ノードへデータを転送するための外部出力端子が接続されている．通信路には，主にPCディスプレイインタフェースとして利用されているDVI-D Dual Linkを採用している．単方向ながら高速なデータ通信を可能とするDVI-Dを利用することで，ノードを接続すると各ノードPCUの直列接続数を接続したノード数だけ増設することが可能となる．

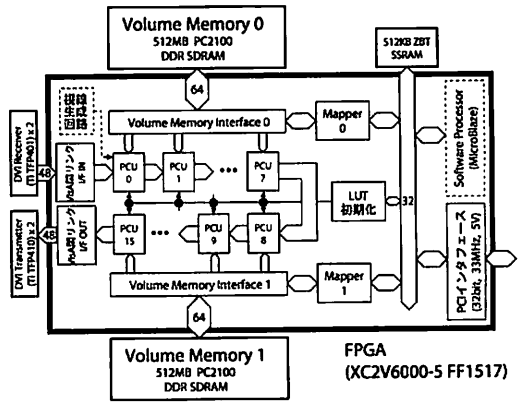

図2 VisA Proブロック図

2.2 DVI-Dによるデータ転送

DVI-D(Digital Visual Interface - Digital)は一般的にはPCからディスプレイへ映像を出力するために使用するインタフェースである．その用途から，このインタフェースはデータ線に映像用のRGB各8bitずつ合計24bitのバス幅を持っている．一方，DVI-D入出力制御に用いる汎用LSIは165MHzの動作周波数で動作し，最大転送速度は $165M(cycle) \times 24(bit/cycle) \times 2(dual) = 7.92Gbps$ となる．また，Single Link構成時には転送速度は3.96Gbpsとなる．

DVI-Dは1サイクル単位で24bitずつデータ転送可能だが，ディスプレイ用インタフェースとして設計されているため，連続転送データ数は1スキャンライン分までしか想定されていない．しかし我々は，スーパーHD(7680×4320)での1スキャンライン7680pxを大幅に超える1Gpxでの連続データ転送にも耐えることを実証している[7]．そのため，ピクセル単位で任意の連続データ転送が可能である．

また，DVI-Dでのデータ通信は初期データ通信前に数サイクルのプリアンブルを送信するだけで可能となる。そのため低遅延でのデータ転送が可能という特徴も持っている．

2.3 並 列 処 理

VisAは視線に対してボクセルを累積計算するPCUを直列に接続する構成をとっている．各々のPCUを同時に処理することで，スクリーン上の全画素に対応する各視線をパイプライン処理することが可能となる．そのため，VisAではボリュー

ムレンダリングを視線並列で処理する．

続いてボリュームデータの分散配置の仕方について述べる．VisA の単方向リンクを利用したボリュームデータの分散配置法にはいくつかの議論があるが [5], [6]，ここではボリュームデータ 3 重化によるデータ配置について述べる．

可視化対象のボリュームデータを 3 つにコピーし，それぞれのボリュームデータを x 軸，y 軸，z 軸に垂直な平面で分割する．分割する間隔はノード内の PCU 数と同一とする．3 重化を施す利点として，どのような視線でも視線が全く通らないノードの発生率が極めて低くなり，各ノードの処理がある程度均等になる点がある．また，各視線主軸に対して整列されたサブボリュームがメモリに格納されることになるので PCU へのボリュームデータのプリフェッチ処理も単純になるため，実装が容易となる．欠点にはメモリ容量が 3 倍必要なことと，シミュレーション系からのデータ分配が複雑になることが挙げられる．

2.4 サンプリング方法の単純化

視線のサンプリング方法には，視線方向に対して等間隔にサンプリングする手法と，視線ベクトル成分絶対値のうち最大値をもつ座標軸（主軸）に対して等間隔にサンプリングする手法がある．VisA では主軸等間隔サンプリング法を採用することで，視線が通過する PCU の順番とボリューム空間内の主軸方向のボクセルの順番を対応付ければ，スムーズにピクセル値を累積計算することが可能になる．

2.5 単方向リンクでのボリュームレンダリング

1.2 節ではボリュームレンダリングアルゴリズムには二つの手法があることを述べた．一見，累積透明度を必要としない Back-to-Front アルゴリズムのほうが通信コストが低い分有利に見える．しかし，VisA は単方向リンクを用いて並列処理を実現するため，累積計算を行う際に取得するボクセルの順番はある程度決定されている．そのため，Back-to-Front アルゴリズムを採用した場合，視線が進む方向と通過するサブボリュームの順番が一致しない（負方向視線）とき，前ノードまでの累積透明度を次ノードに渡す必要がある．視線方向によって通信するデータが変わると処理が複雑になるため，VisA では Front-to-Back アルゴリズムを採用する．

VisA で負方向視線を処理するときは次のように行う．まずノード内で PCU にプリフェッチするボクセルの順番を正方向視線のときと逆にして累積計算を行う．求めた結果を一つのボクセルとみなし，累積透明度をボクセルの透明度とする．次に前ノードまでに求めた結果とこのボクセルを α ブレンディングすることで，負方向視線でのピクセル値を求めることができる [5]．

これはノード内部では Front-to-back アルゴリズムで視線累積計算を行い，ノード間では Back-to-front アルゴリズムで視線累積計算を行うことになる．ノード間でのみアルゴリズムを切り替えることで，ノード内部での負方向視線のための回路増設を最小限に抑えることができる．

2.6 メモリからのボクセル読み出し

VisA は，保持するボリュームデータを格納するメモリとして DDR SDRAM を採用している．この DDR SDRAM は 1 語 64bit であり，ランダムアクセスと 2・4・8 バースト転送に対応している．1 ボクセルのデータサイズは 8bit であるため，1 語読み込むと 8 ボクセル取得することができる．VisA に搭載する PCU 数は 16 個を予定しているため，全 PCU を同時実行するには 1 サイクルあたり 16 ボクセルを取得する必要がある．そのためには毎サイクルデータを取得しなければならないので，DDR SDRAM に対するアクセスをパイプライン化してデータ線を休ませないようにする必要がある。バースト数はリードリクエストを隠ぺいするために 8 バーストを採用する．

次にメモリからサブボリュームを読み出すことを考える．ボリュームレンダリングを行う際の視線方向には様々な向きが考えられる．一回のリードリクエストでどの視線に対してもロスの少ないサブボリュームを読み出すことを考えると，立方体のサブボリュームを読み出すことが理想的である．8 バーストの場合，一回のリードリクエストで 64 ボクセルを取得するため，$4 \times 4 \times 4 (= 64)$ の塊ごとにメモリへボリュームデータを格納することで，最も効率の良いボクセル読み出しが可能となる．

2.7 視線累積計算の順番

前に述べた手法でメモリからサブボリュームを取得すると，視線主軸方向でスライスされて $4 \times 4 \times 1$ で固まったボクセルが各 PCU にプリフェッチされる．このプリフェッチされたボクセルを効率よく消費するために，累積計算する視線もスクリーン上の縦・横 4×4 単位で処理する [8]．

3. 評価ボードによる実装と動作検証

本章では，我々が東京エレクトロンデバイス (株) と共同企画で開発を行った DDR SDRAM 搭載 PCI カード型 DVI 評価ボード TD-BD-PCI2DVI(VisA Pro カード：図 3) での実装と動作検証について述べる．

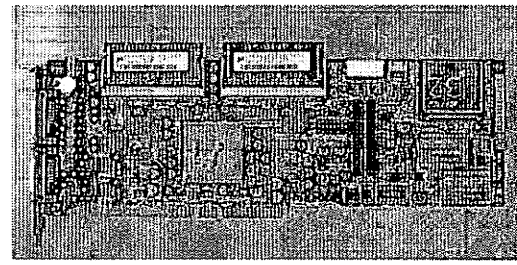

図 3 VisA Pro カード

3.1 VisA Pro カードでの実装

今回の動作検証では最低限のボリュームレンダリング機能を実装した．現在は図 4 の点線で囲んだ回路が機能している．投影方法には並行投影を採用し，動作周波数は DDR SDRAM Controller が 166MHz，その他は 100MHz である．

3.1.1 PCU

PCU は入力された視線データに対して対応するボクセルを累積計算する回路である．内包する要素には，演算器のほかに SDRAM からプリフェッチしたボリュームデータを格納するバッファと，ボクセル値を元に色データを引き出すためのカ

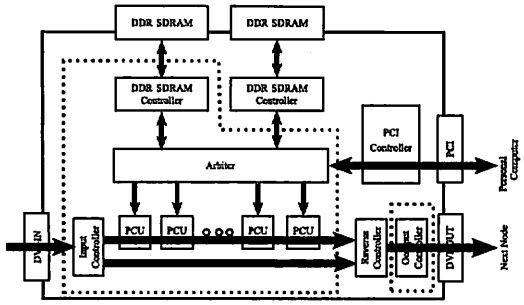

図 4 実装状況

ラーテーブル (32bit × 256 階調) がある.

PCU は式 (1)(2) に示す Front-to-Back アルゴリズムにより累積計算を行うが, ピクセル値及び累積透明度を高速に演算する必要があるため, 計算式を分割してパイプライン処理を行うことを考える. 二つの計算式を分割する際, 双方に共通部分を見出すことができれば演算効率が上がることは明白である. そのため, 各計算式を式 (4)(5) のように変形し, 図 5 に示すパイプラインによって演算を行うことにした.

$$B_n = B_{n-1} + \alpha_n \times c_n \times A_{n-1}$$
$$= B_{n-1} + c_n \times \underline{\alpha_n \times A_{n-1}} \quad (4)$$

$$A_n = (1-\alpha_n) \times A_{n-1}$$
$$= A_{n-1} - \underline{\alpha_n \times A_{n-1}} \quad (5)$$

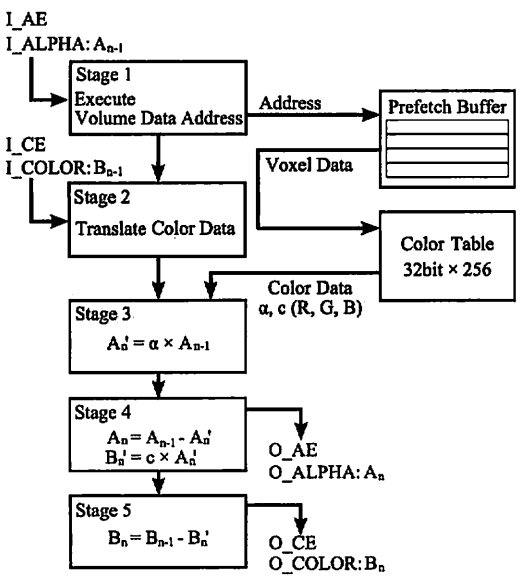

図 5 PCU Pipeline Flow

PCU のパイプライン処理は次のようになる. Stage1 で累積計算すべきボクセルのアドレスを計算し, プリフェッチバッファからボクセルを読み出す. Stage2 でボクセル値からボクセルの色及び透明度を読み出す. Stage3 でピクセル値及び累積透明度の共通部分を計算する. Stage4 で累積透明度を計算して次 PCU に求まった累積透明度を送ると同時に, ピクセル値の前計算を行う. Stage5 でピクセル値を求め, 次 PCU に送信する. 図 5 中 I_AE および I_CE は累積透明度と視線データの有効を示すものであり, これに 0 が入力されればそのサイクルに入力されたデータに対するパイプライン処理は行われない.

今回の実装では, FPGA への配置配線時の信号遅延の問題から, 演算精度はピクセル値, 累積透明度ともに 8bit とした.

3.1.2 DDR SDRAM Controller

DDR SDRAM を扱うためのコントローラは 1 チャネルのみ利用可能となっている. また, 読み出しのパイプライン化は行っておらず, 8 バーストによるデータアクセスのみ実装している. 本コントローラで読み出しを行う場合, リードリクエストを行ってから全データを取得するまで 13 サイクルを要する. つまり一度のリードリクエストで 64 ボクセルを 13 サイクルかけて取得することになるので, $64/13 \simeq 4.9 voxel/cycle$ のスループットが得られる.

全 PCU に必要なボクセルをプリフェッチしなければ視線の累積計算は行えないが, 本コントローラでは 4.9voxel/cycle しか得られない. 16 個のボクセルを得るためには $16/4.9 \simeq 3.2$ サイクル必要である. つまり, 1 視線の累積計算には 4 サイクルを要することになる.

3.1.3 VisA Pro 間の通信

DVI-D による通信では, 24bit 幅の通信路で, 累積透明度および色 (R,G,B) の 32bit を通信する必要がある. PCU で行う演算に着目すると, 累積透明度を使用した演算が視線・累積透明度計算の共通部分にある. そこで, 先に累積透明度を 8bit×3px 分まとめて送信し, 対応するピクセル値を 1px(24bit) ずつ続けて送信する方式をとる. また, 1 視線の処理に 4 サイクル必要なので 3px 送信 (4 サイクル) 毎に 8 サイクル待つことにする. フレーム同期信号には DVI-D に備わっている水平同期信号を利用する.

3.1.4 Input Controller

本コントローラは 3px まとめて送られてきた累積透明度を分割し, 各々に対応する視線データを PCU に対して順に送信する機能を持つ. また, 前ノードから送られてきた視線にボクセルを累積計算するためには各 PCU に累積すべきボクセルがプリフェッチされている必要がある. しかし, 視線が送られてきた時点ですでにメモリから必要なボクセルが読みだされているとは限らない. そうした視線を一時待機させておくための FIFO バッファを持たせている. フレーム同期信号を受信したときにはその情報を PCU および Arbiter に知らせる機能も持つ.

3.1.5 Arbiter

Arbiter はボリュームレンダリング開始前, 全 PCU に対してカラーテーブルをセットする. その後は各 PCU のプリフェッチバッファを監視し, 消費されて不要となったバッファを見つけ次第, 次に必要となるであろうボクセルをメモリから読み出して PCU に渡す. 累積計算する視線の順番はあらかじめ決まっているので, 次に必要なボクセルはこれまでに送られてきた視線の順番から割り出すことが可能である. フレーム同期の

通達があったときは送られてきた視線数をリセットし，初めから PCU に対してボクセルをプリフェッチしなおす．

3.2 動作検証回路の実装

実装した VisA Pro の動作を検証するにあたり，別の TD-BD-PCI2DVI 上で図 6 に示す回路の実装を行った．この回路は二つの機能を有する．一つは，DVI-OUT に接続された VisA Pro に対して空のスクリーンデータを送信し，DVI-IN に接続された VisA Pro より出力された，ボリュームレンダリングが施されたスクリーンデータを画像データとして DDR SDRAM に書き込む機能である．もう一つは，DVI-IN から入力されたディスプレイ出力用の画素データを DDR SDRAM 内の画像データに置き換えて，DVI-OUT に接続されているディスプレイに出力する機能である．TD-BD-PCI2DVI には，DVI-IN および DVI-OUT はそれぞれ一つずつしか搭載されていないので，機能を切り替えるときは接続も切り替える必要がある．

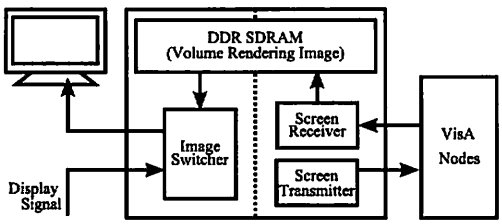

図 6 ボリュームレンダリング動作検証回路

3.3 検証用ボリュームデータ

今回の動作検証では PCI コアの実装までは至らなかったので，半透明ボクセルを使ったボリュームレンダリングが確認でき，かつ VisA Pro 内部で生成できる，単純なボリュームデータを用意した．ボリュームデータの大きさは $384 \times 384 \times 16$ でボクセルの透明度は全て $21/255 (\simeq 0.083)$，スクリーンの縦・横サイズはボリュームデータの高さ・幅と同じサイズである．

3.4 動作検証

動作検証は次の二段階で行う．

（1） VisA Pro と動作検証回路を接続し，動作検証回路から空のスクリーンデータ (の全ピクセルに対応する視線データ) を送信する．VisA Pro から順次，ボリュームレンダリングされた結果が出力されるので，それを動作検証回路の DDR SDRAM 内に書き込む．

（2） 動作検証回路の DVI-IN に PC のディスプレイ用 DVI ケーブルを接続し，DVI-OUT とディスプレイを接続する．PC から出力される映像を DDR SDRAM 内の画像データに置き換えて出力し，ディスプレイに表示して結果を確認する．

VisA Pro による出力結果とソフトウェアによる出力結果を並べた画面を図 7 の左に示す．図中ディスプレイ上の左上が VisA Pro によるもので，右下がソフトウェアによるものである。また図 7 右には評価用ボリュームデータの右側面図を示しておく．主観では，VisA Pro で出力した映像のほうが若干暗く見えるが，色の並びや半透明ボクセルによるグラデーションが同じように出力されていることが確認できた．

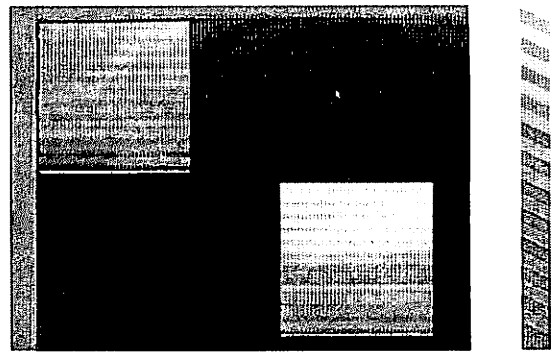

図 7 出力結果の比較と評価用ボリュームデータの側面

続いて，高さおよび幅をそれぞれ 3 倍の大きさにした $1152 \times 1152 \times 16$ サイズのボリュームデータで同様の実験を行った．スクリーンサイズも 1152×1152 とした．実験の結果，正常に動作していることが確認できた．

3.5 考　　察

本節では今回の動作検証による考察を述べる．

3.5.1 PCU 演算精度によるピクセル値の誤差

まず PCU 演算精度によるピクセル値の誤差について考察する．図 7 にも示したように，今回の実装によるボリュームレンダリングはソフトウェアによる結果に比べて若干暗く出力された．VisA は本来 16bit の精度で累積計算を行うように設計されているが，実装時の都合により 8bit の演算精度しか得られなかったため，累積計算後のピクセル値に誤差が生じたことに起因していると考えられる．PCU では固定小数点による演算を採用しているので，演算精度の低さは主に乗算時の丸め誤差を増大させ，PCU の段数を重ねる度に誤差は累積されていく．そこで，視線累積計算の共通演算部でもある図 5 中 Stage3 の乗算に着目し，乗する透明度 (α) と演算精度による誤差について調査した．評価環境として，PCU を 16 個搭載した VisA を 128 台接続したシステム構成を想定し，全ボクセルが同一透明度をもつボリュームデータに対して double 型 (64bit) で累積計算を行った時の結果を基準とした．比較する PCU の演算精度は 8bit 及び 16bit とし，PCU 間の通信バンド幅は演算精度と同一であり，VisA 間通信時はデータを 8bit に丸めることにする．

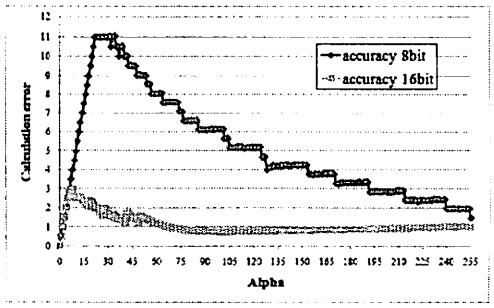

図 8 演算精度による誤差

比較結果のグラフを図 8 に示す．グラフ横軸は累積するボクセルの透明度 (256 階調) であり，縦軸はピクセル値 (256 階調) の誤差である．図 8 からわかるように，8bit 精度では透明度が 26/255〜34/255 のときにピクセル値は最大 11 程度の誤差が生じることが確認できる．今回の動作検証で用いたボリュームデータのボクセルの透明度はすべて 21/255 であり，このときの誤差は 10.5 と最大値に近いこともわかる．だがこれほどの誤差が生じても，レンダリングの結果を実際に視認すると若干暗くなる程度の違いしか表れないことがわかる．

しかしながら，より高精細なボリュームレンダリングを行うには 16bit の演算精度が必要である．今回の実装では配線遅延のクリティカルパスは乗算器の出力結果を他演算に渡す部分にあることが確認できている．この問題を解決するためには，クリティカルパスの部分にレジスタを挿入することで配線距離を縮め，それに伴い演算パイプラインを伸ばす方法が考えられる．

3.5.2 動作速度

次にボリュームレンダリング処理速度について考察する．

VisA Pro は累積計算にパイプラインを利用しているので視線 1 ピクセルあたり 1 サイクルで処理を行う．そのため，処理速度は視線データの転送速度に依存することになる．視線転送速度は 3.1.3 項でも述べたように 4cycle/px である．今回の実装では DDR SDRAM Controller 以外の回路は全て 100MHz で動作している．そこで，1024 × 1024 のスクリーンサイズで，ボリュームデータの深さが PCU 数と同じ場合にボリュームレンダリングを行ったときの処理速度を考える．全スクリーンの視線を送信（処理）するために必要なサイクル数は $4 \times 1014 \times 1024 = 4,194,304$ サイクルであり，1 秒間あたりの処理回数は 23.84 回となる．このとき動作速度のオーバヘッドはメモリからのプリフェッチにあるので，DDR SDRAM を 2 系統使い，読み出しをパイプライン化すれば 4 倍の 1px/cycle での処理速度が見込める．

続いて DDR SDRAM 読み出しのパイプライン化について考える．DDR SDRAM の読み出し動作は次からなる．

（1） アクティブ命令/Row アドレス指定
（2） RAStoCAS 遅延後リード命令/Column アドレス指定
（3） CAS 遅延後データアクセスの開始
（4） アクセスしたバンクの活性化（プリチャージ）

DDR SDRAM では一度アクセスしたバンクに対して再アクセスするためには，バンクを一度プリチャージしなければならない．プリチャージが開始できるまでの遅延及びプリチャージ期間はデバイスにより多少ことなるが，概ね ROW アドレス指定から全データ読み出しまでの時間と一致する．そのため，同一バンクに対して連続データアクセスをする際はデータをすべて読み出すまで次のリードリクエストが行えない．

だが，DDR SDRAM は内部で 4 バンク構成をとっており，各バンクを独立して制御することができる（マルチ・バンク・オペレーション [9]）．そのため，あるデータを読み出した後，次に読みだすべきデータが別バンクにある場合はバンクプリチャージを待つことなくリードアクセスできることになる．

VisA に搭載する DDR SDRAM はボリュームデータを格納することに用いられ，データ読み出しは視線主軸方向に対して連続で行われる．そこで，VisA に格納される 3 重化されたスライス状のサブボリュームをそれぞれ主軸方向に 4 分割し，別バンクに振り分ける．そうすることで，1 バンクに格納されるサブボリュームデータは主軸方向に 4 ボクセルの深さをもつことになる．ある視線に対するボリュームデータを各 PCU にプリフェッチする際，DDR SDRAM からは $4 \times 4 \times 4$ 単位で読み出す方針なので，視線主軸方向に連続でデータ読み出しを行うと同一バンクに対する連続アクセスがなくなる．

このようにボリュームデータの配置を工夫することで，DDR SDRAM に対するアクセスは最適化される．VisA に搭載している DDR SDRAM は RAStoCAS 遅延+CAS 遅延が 5.5 サイクルであり，8 バーストでの読み出しに必要な時間が 4 サイクルであることを考えると，上述した最適化を施すことで更に 2 倍近い速度向上が見込める．

4. おわりに

本稿では，並列ボリュームレンダリング専用アクセラレータ VisA のシステム構想と，その予備実装による評価について述べた．実装による動作検証により，ボリュームレンダリングが正常に行われることを確認することができ，大規模ボリュームデータに対する実時間可視化の可能性についても確認することができた．今回の実装ではボリュームデータ正面からの投影しかできなかったが，今後は様々な角度での投影や透視投影を可能とする実装を行い，それらに対しての投影時間に対する検証を行いたい．

謝　辞

日頃より御討論いただく京都大学大学院情報学研究科富田研究室の諸氏に感謝します．本研究の一部は，日本学術振興会 科学研究費補助金 基盤研究 S (課題番号 16100001) による．

文　献

[1] 村木, 他, “VG クラスタ:スケーラブルビジュアルコンピューティングシステム”, Vosual Computing グラフィクスと CAD 合同シンポジウム 2001, pp.85-90,2001.
[2] Orad homepage, “http://www.orad.co.il/”,2006.
[3] John Eyles et al., “PixelFlow: The Realization”, 1997.
[4] 劉馬, 他, “ボリューム・レンダリング専用並列計算機 ReVolver のアーキテクチャ”, 情報処理学会論文誌, 第 36 巻, 第 7 号, pp.1709-1718, 1995.
[5] 生雲 公啓, “時変ボリュームデータの実時間可視化のための専用グラフィックスカード VisA の開発”, 京都大学大学院情報学研究科 修士論文, 2003.
[6] 岡村 大, “超高速単方向リンクを用いた並列ボリュームレンダリングシステム”, 京都大学大学院情報学研究科 修士論文, 2006.
[7] 野田 裕介, “DVI-D インタフェースによる高速低遅延データ転送”, 京都大学工学部特別研究報告, 2006.
[8] 額田, 他, “参照局所性を最大化するボリューム・レンダリング・アルゴリズム”, 情報処理学会論文誌：コンピューティングシステム, Vol. 44, No. SIG 11(ACS 3), pp.137-146(2003.08).
[9] Elpida Memory, Inc., “DDR SDRAM の使い方 ユーザーズマニュアル”