Vol.2011-HPC-130 No.23
2011/7/28

ボリュームレンダリングにおける効率的なテクスチャ参照

杉本祐樹[†1]　伊野文彦[†1]　萩原兼一[†1]

本稿では，CUDA（Compute Unified Device Architecture）環境を用いたボリュームレンダリングの高速化を目的として，視点の位置に応じてスレッドブロック（TB）の形状を選択する手法を提案する．提案手法は，GPU（Graphics Processing Unit）が同時に実行する一連のスレッド（ワープ）ごとに参照領域を局所化し，テクスチャキャッシュ（TC）ヒット率の向上を図る．そのために，レンダリングの参照ストライドがボリュームおよび投影面の位置関係に応じて決まることに着目する．具体的には，ボリュームを構成する 3 軸のうち，参照ストライドの小さい軸に対してワープの並びが平行となるように，TB の形状を選択する．実験の結果，提案手法により TC ヒット率を最大で 32.6% ほど向上でき，実行時間を半減できた．

Efficient Texture Access in Volume Rendering

Yuki Sugimoto,[†1] Fumihiko Ino[†1]
and Kenichi Hagihara[†1]

This paper presents a view-dependent method for selecting the shape of thread blocks (TBs) in order to accelerate volume rendering on the compute unified device architecture (CUDA). Our method improves the hit rate of texture cache (TC) by localizing the memory region accessed by warps, or series of threads simultaneously processed on the graphics processing unit (GPU). To achieve this, we focus on the stride of memory access, which depends on the geometric relation between the volume and the screen. Our method selects the shape of TBs such that series of warps are parallel to the volume axis that has small strides. Experimental results show that our method increases the TC hit rate by at most 32.6% and reduces execution time into half.

†1 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

1. はじめに

ボリュームレンダリング（VR）[1] は 3 次元データ（ボリューム）を投影面へ投影し，可視化する技術である．この技術は，CT （Computed Tomography）像[2] や流体シミュレーション[3] の分析を視覚的に支援する．ボリュームは，ボクセルと呼ばれる 3 次元単位格子の集合であり，各ボクセルは値を持つ．VR では，視点から投影面上の各画素に視線を結び，視線が貫くボクセルの値を等間隔で積算することにより，画素値を算出する．この際，ボクセル値は近傍領域でのみ再利用できる．したがって，実行時間は積算に要する計算時間よりもメモリ参照に要する参照時間に支配される．そこで，GPU（Graphics Processing Unit）[4] 上の並列処理により，参照時間を計算時間で隠蔽し高速化することが多い．

GPU は数万個のスレッドを並列処理できる．画素ごとの計算は互いに独立であるため，各スレッドに画素を 1 つずつ割り当てることが多い．また，ボリュームを 3 次元のテクスチャとして保持し，テクスチャメモリ（TM）に格納する．TM の参照機構は，メモリ参照時の遅延を短縮するためにテクスチャキャッシュ（TC）を持つ．TC 上にデータが存在すれば，その参照時間を数百分の 1 程度に短縮できる．したがって，局所性の高いデータ参照により TC ヒット率が高まり，性能を向上できる．

そこで本研究では，視点の位置に応じてスレッドブロック（TB）の形状を選択することにより，局所性の高い TM の参照を実現する．提案手法は，隣接ボクセル間の参照ストライドがボリュームを構成する軸ごとに異なることに着目する．TM は 32 スレッド（ワープ）ごとに同時に参照されるため，同一ワープ内の 32 スレッドが小さなストライドで TM を参照できるようにすればよい．そのような参照を実現するために，提案手法はボリューム空間および投影面の位置関係を基に TB の形状を選択する．提案手法は開発環境 CUDA（Compute Unified Device Architecture）[5] を前提とする．

以降では，まず 2 章で関連研究を紹介する．次に，3 章で GPU を用いた VR について述べ，4 章で視点位置に応じた TB 形状の選択手法を提案する．さらに，5 章で評価実験の結果を示し，提案手法による TC ヒット率の向上に対する効果および TC ヒット率と実行時間の結果について考察する．最後に，6 章で本稿をまとめる．

2. 関連研究

額田ら[6] は CPU を用いた VR においてキャッシュを効率的に利用している．彼らはボリューム空間をキューボイドと呼ばれる直方体に分割し，キューボイドごとに投影を進める．

Vol.2011-HPC-130 No.23
2011/7/28

キャッシュヒット率を最大化するために，視点の位置に応じて投影の順序を変更している．GPU を用いる VR ではワープ単位の同時処理においてキャッシュ最適化を図る必要がある．

GPU 上で TC ヒット率を向上した応用例として，成瀬ら[7] の流体アプリケーションの高速化がある．彼らは，GPU 上で安定して高いメモリバンド幅を実現するためには，スレッドの同期，参照パターンの局所化およびスレッド数の最適化が重要であると結論づけている．提案手法はスレッドブロックの形状を最適化する．

3. GPU を用いたボリュームレンダリング

本章では，VR アルゴリズムとしてよく知られているレイキャスティング（RC）法およびその並列化について説明する．また，GPU が持つ TM の構造をまとめる．

3.1 レイキャスティング法

図 1 に RC 法において視線がボクセルを貫く様子を示す．RC 法[1] は，視点 O から投影面 S 上の各画素に視線 R を結び，R が貫くボクセルの値を一定間隔で積算して画素値を決定する．以降，ボリュームを $N \times N \times N$ ボクセルの立方体とし，ボリュームを構成する各軸をそれぞれ x，y および z 軸とする．各ボクセルは不透明度を持つ．不透明度に応じた輝度値を積算することにより，ボリューム内部を可視化できる．投影面 S 上の座標を (u, v) とすると，その座標における画素値 $S(u, v)$ は式 (1) で計算できる．

$$S(u,v) = \sum_{i=1}^{n} \left(\alpha(v_i) c(v_i) \prod_{j=0}^{i-1} (1 - \alpha(v_j)) \right) \tag{1}$$

ここで，v_i は視線 R 上において投影面に近い方から数えて i 番目のボクセルを表し，投影面からもっとも遠いボクセルを v_n とする．$c(v_i)$ および $\alpha(v_i)$ はそれぞれボクセル v_i の輝度値および不透明度を表す．式 (1) に基づいて，投影面上のすべての画素 $\{S(u,v) \mid 1 \leq u \leq W, 1 \leq v \leq H\}$ を計算し，描画結果を得る．ここで，W および H は投影面の横および縦の大きさである．

3.2 レイキャスティング法の並列化

式 (1) は画素値ごとに独立であるため，複数のスレッドで並列計算できる．そこで，多くの研究では画素ごとの計算を各スレッドに割り当てている．CUDA[5] では，GPU は複数のマルチプロセッサ（MP）を持つ．MP は，TB と呼ばれる複数のスレッドの集合を並行実行する．MP は複数の TB を切り替えて並行実行することでメモリ参照を隠蔽する．並行実行する TB の数は，各 TB のスレッド数や使用するレジスタの数により決まる．現行のアーキテクチャは，最大で 8 つの TB を並行実行できる．TB はさらに 32 スレッドごとに分割される．この 32 スレッドをワープと呼び，同時実行の処理単位である．

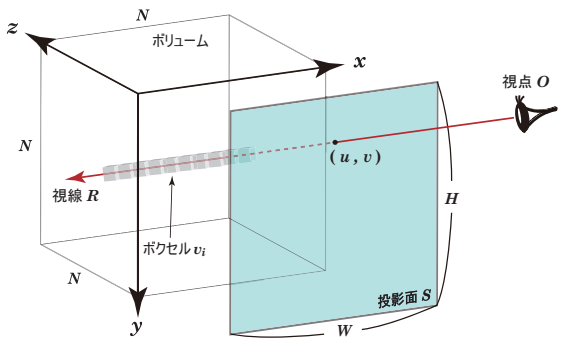


図 1 レイキャスティング法

本研究では，投影面を格子状に分割し，分割領域の処理を各 TB に割り当てる．TB は 2 次元で指定する．したがって，投影面および TB のサイズをそれぞれ $W \times H$ および $w \times h$ とすると，投影面を $\lceil W/w \rceil \times \lceil H/h \rceil$ に分割し，分割した領域の計算処理を 1 つずつ TB に割り当てる．

3.3 テクスチャメモリの構造

図 2 に，TM における物理アドレスおよび論理アドレスの対応関係を示す．図中の赤い矢印は物理アドレスの並びを表す．このように，3 次元のテクスチャは 2 次元スライスの集合とみなせ，スライスごとのメモリ参照に最適化されている[8]．

VR 時のストライドを調べるために，隣接する 2 つのボクセル v_i および v_{i+1} を順に参照することを考える．両ボクセルが異なるスライスに存在する場合（z 軸に沿って存在する場合），両者の物理アドレスの差は N^2 であり，ストライドが大きい．一方，両ボクセルが同一スライスに存在する場合，両者は x 軸もしくは y 軸に沿って存在する．物理アドレスの差は軸に依存していて，x 軸の場合と比較して，y 軸の場合は 2 倍の差となる．例えば，xy 平面上で 8 個のボクセルを同時参照する場合，参照範囲の物理アドレスの差は x 軸において 21 であり，y 軸では 42 である．

このように，ワープがいくつかのボクセルを同時に参照するとき，それらの並びと各軸の位置関係に依存してストライドが決まる．ストライドは x 軸，y 軸および z 軸の順に大きくなるため，この順番で軸に平行なボクセルを同時参照すべきである．

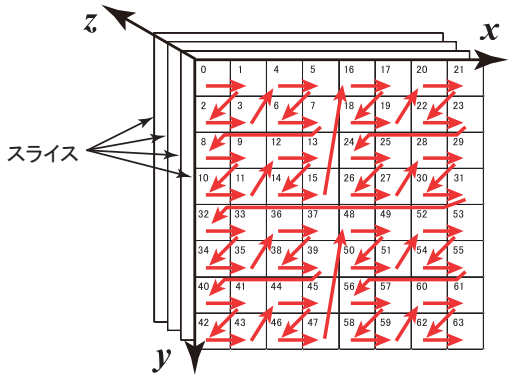


図 2 3 次元のテクスチャメモリにおける物理アドレスの並び

4. 提案手法

本章では，効率的なテクスチャ参照のために，まず TB を構成するスレッドの数について述べ，TB とワープの対応関係を示す．次に，視点位置の変更に伴うボリューム空間と投影面の位置関係から，視点の位置に応じた TB の形状の選択手法を説明する．

4.1 スレッドブロックとワープについて

VR の処理はメモリ参照の回数が多いため，並行実行する TB の数を最大化することにより，メモリ参照時間が隠蔽される．また，演算はワープ単位で同時実行されるため，スレッド数は 32 の倍数がよい．したがって，8 つの TB を割り当てることができ，かつ 32 の倍数のスレッド数のうち最多のスレッド数である 128 を選択する．TB の各スレッドにはスレッド ID が割り振られており，若い方から順に 32 ごとに区切られ，ワープとなる．TB を 2 次元で指定すると，ワープも 2 次元の形状をもつ．プログラマは TB の形状を指定することにより，ワープの形状が決まる．また，ワープの実行順序はスケジューラが決め，プログラマは制御できない．スレッド数が 128 のとき，TB およびワープの形状は**表 1** に示す 8 通りが考えられる．以降では，$w < h$ の場合は形状が縦長，$w > h$ の場合は横長と考える．

4.2 投影面に対するボリューム空間上の参照ストライド

視点位置の変更に伴い，ボリューム空間と投影面の対応関係が変動する．ここで，投影面を固定し，ボリュームを回転させることで視点位置を変更をする．簡単のために，視点位置の移動を 90 度単位で考える．このとき，投影面に対するボリュームの位置関係は投影面

表 1 スレッド数が 128 のときの TB およびワープの形状 $\langle w, h \rangle$

TB	$\langle 1, 128 \rangle$	$\langle 2, 64 \rangle$	$\langle 4, 32 \rangle$	$\langle 8, 16 \rangle$	$\langle 16, 8 \rangle$	$\langle 32, 4 \rangle$	$\langle 64, 2 \rangle$	$\langle 128, 1 \rangle$
ワープ	$\langle 1, 32 \rangle$	$\langle 2, 16 \rangle$	$\langle 4, 8 \rangle$	$\langle 8, 4 \rangle$	$\langle 16, 2 \rangle$	$\langle 32, 1 \rangle$	$\langle 32, 1 \rangle$	$\langle 32, 1 \rangle$

に並行な平面上の 2 つの方向および垂直な方向の 3 つの方向に対して x, y, z の軸が対応する．したがって，**図 3** のように合計 6 つの視点位置に場合分けできる．

ここで，隣接するボクセル間のアドレスの差を考える．ボリュームのあるボクセルにおける座標を (x, y, z) とする．このとき，3 つ軸方向に隣接するボクセル間のアドレスの差は最悪時にそれぞれ α，2α，N^2（一定）となる．ただし，α は式 (2) で表される．

$$\alpha = \frac{2^{2\lceil \log_2 N \rceil - 1} + 1}{3} \quad (N \geq 2) \tag{2}$$

この式より，N が大きくなるにつれて N^2 は 6α に収束する．したがって，3 つ軸方向に隣接するボクセル間の最悪時の物理アドレス差は，N が大きいとき近似的に 1:2:3 で表せる．このように，3 つの軸方向で参照ストライドに差があるため，3 軸に対する参照ストライドの大小関係から同時参照する領域を考えることに有効性がある．

4.3 スレッドブロック形状の選択

図 3 の各視点位置において，ボリューム空間の 3 軸に対する参照ストライドの大小関係から，局所性が高いワープの同時参照を考える．画素値を算出するために参照するボリュームの領域は，視点から等間隔の距離にあるボクセル値である．したがって，ワープ単位で同時参照するボリューム領域は，実際には球面状である．しかし，ここではこの領域を近似的に平面とみなす．このとき，投影面とワープの同時参照する平面領域は並行となる．ゆえに，ボリューム空間において投影面と並行な平面に着目し，その平面上の 2 つの軸の参照ストライドの大小関係から局所性の高い同時参照ができるようなワープの形状を考える．具体的には，参照ストライドの小さい方向に対応する，ワープの形状における幅が大きくなるように TB の形状を指定する．

この TB 形状の選択手法を図 3 の 6 つの視点位置について適用する．ボリューム空間において投影面と並行な平面は**図 3(a)** および**図 3(e)** では xy 平面，**図 3(b)** および**図 3(d)** では xz 平面，**図 3(c)** および**図 3(f)** では yz 平面となる．次に，平面上の 2 つの軸の参照ストライドの大小関係から xy 平面および xz 平面では x 軸の方向が，yz 平面では y 軸の方向が参照ストライドが小さいため，それぞれの軸の方向に対応するワープの形状における幅が大きくなるように TB の形状を設定する．したがって，図 3(a)，図 3(b) および図 3(c)

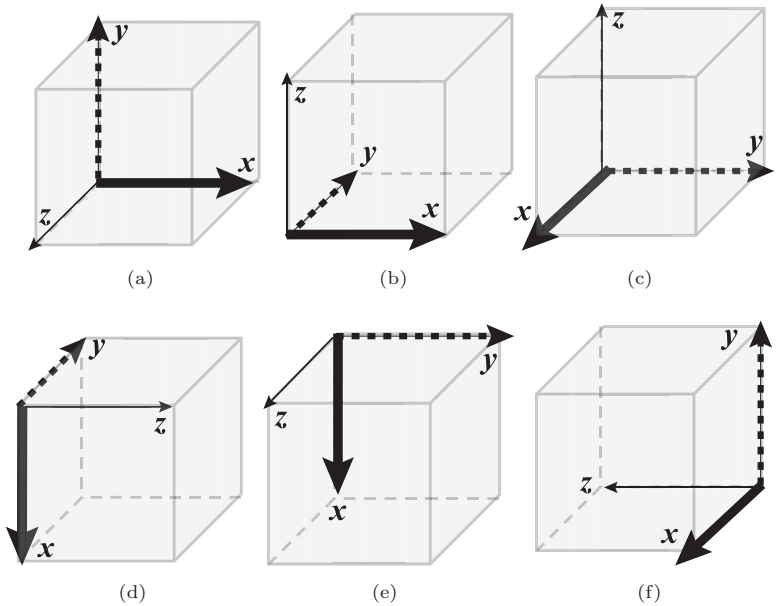


図 3 3 次元ボリュームにおける参照ストライドの大小関係

の視点位置ではワープの形状が横長になるように，図 3(d)，図 3(e) および図 3(f) の視点位置ではワープの形状が縦長になるように TB の形状を設定する．

5. 評価実験

提案手法の効果を評価するために，図 3 の 6 つの視点位置における TC ヒット率および FR を計測し，それらの結果を考察する．**表 2** に実験環境を示す．実装は，CUDA SDK を基にしており，テクスチャにはボリュームのみ保持する．実験に用いるボリュームおよび投影面のサイズは $N = W = H = 1024$ である．したがって，図 3 の 6 つの視点位置における投影処理においてすべて対称性がある．実験では，対称性に基づいて考察するため，画素値の積算を途中で打ち切らない．

図 4 は，表 1 で示した 8 通りの TB の形状について，各視点位置ごとの TC ヒット率および FR の関係を表す．横軸の（a）～（f）はそれぞれ図 3(a)～図 3(f) の視点位置に対応

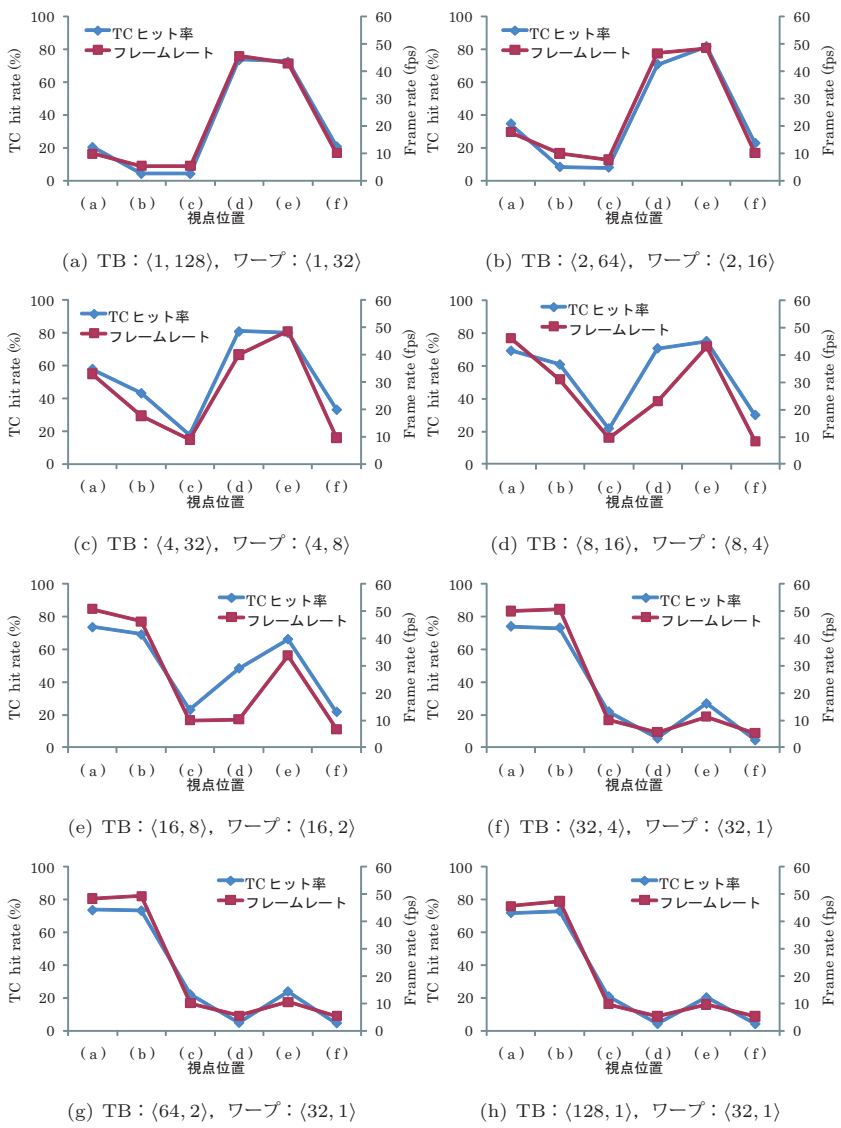
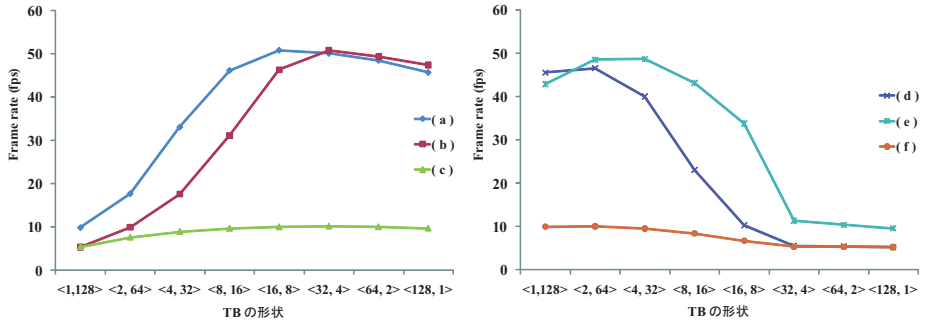


図 4 各 TB の形状における TC ヒット率とフレームレートの変化

表 2　実験環境

OS	Windows 7 Professional 64-bit
CPU	Intel Core i7 930 2.80 GHz
主記憶	12 GB
GPU	NVIDIA GeForce GTX 480
ビデオメモリ	1536 MB
CUDA バージョン	CUDA 3.2
プロファイラ	CUDA Visual Profiler
ビデオドライバ	260.61

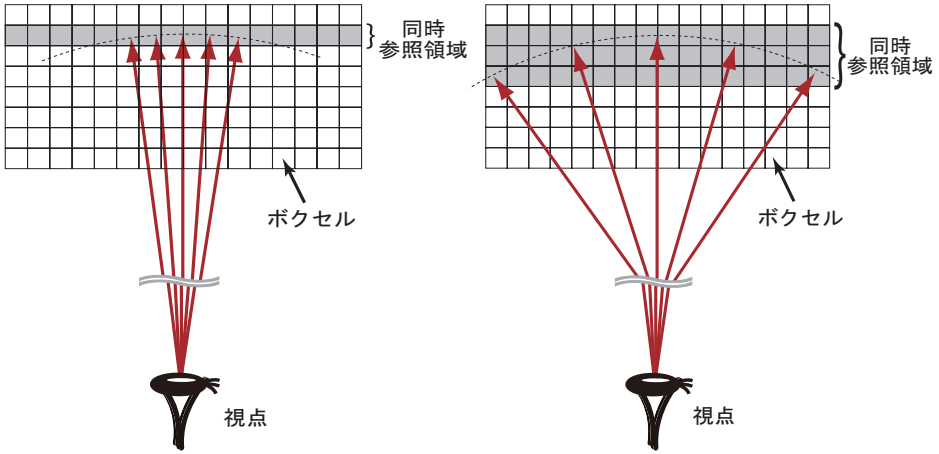


(a) 視点位置（a），（b）および（c）　　(b) 視点位置（d），（e）および（f）

図 5　ワープを同様の形状に設定する場合の視点位置におけるフレームレートの変化

する．図 4 のグラフにおいて，TC ヒット率および FR のグラフの形状は，いずれの TB の形状の場合も対応しており，TC ヒット率と FR には相関がある．したがって，TC ヒット率を向上することで実行時間を短縮できている．

次に，各視点位置における FR の変化を TB の形状別に考える．**図 5** は各視点位置において TB の形状の変化に伴う FR の変化をグラフにしたものである．**図 5(a)** のグラフは視点位置（a），(b）および（c）における各 TB の形状での FR の変化である．これら 3 つの視点位置は，4.3 節においてワープの形状が横長になるように TB の形状を設定すると提案した．一方，**図 5(b)** のグラフは視点位置（d），(e）および（f）における各 TB の形状での FR の変化であり，これら 3 つの視点位置では，ワープの形状が縦長になるように TB の形状を設定すると提案した．表 1 から，ワープの形状が横長であるか縦長であるかは $\langle 8, 4\rangle$ と $\langle 4, 8\rangle$ の間を境界として決まる．また，そのとき TB の形状はそれぞれ $\langle 8, 16\rangle$ および $\langle 4, 32\rangle$



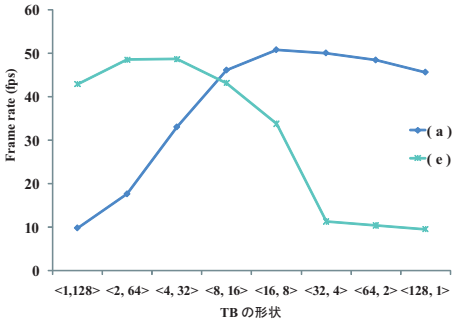
(a) TB の形状の縦横比が小さい場合　　(b) TB の形状の縦横比が大きい場合

図 6　TB の形状の縦横比による同時参照領域の違い

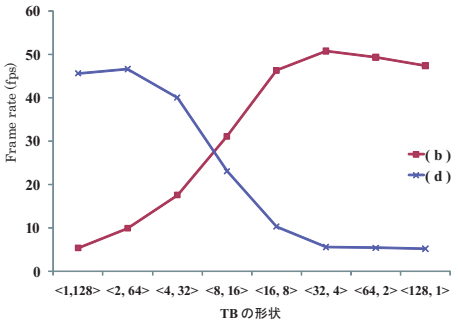
である．図 5(a) のグラフでは，いずれの視点位置においてもワープの形状が横長のときの方が縦長のときよりも FR が高い．これに対し，図 5(a) のグラフでは，いずれの視点位置においてもワープの形状が縦長のときの方が横長のときよりも FR が高い．

一方で，図 5(a) の視点位置（a）および（b）と，図 5(b) の視点位置（d）および（e）のグラフから，TB の形状の縦横比を大きくし過ぎると，逆に FR が低下することがわかる．これは，ワープ単位で同時参照するボリューム領域が球面状であるため，同時参照するメモリ領域が大きくなることが原因である．**図 6** に TB の形状の縦横比の大小によるワープが同時参照する領域の違いを示す．図中の矢印は，ある時刻においてワープが同時に参照するときの参照するボクセルを示す．画素値の積算は視点から等距離にあるボクセル値を参照する．したがって，視点から奥行き方向に参照するボリューム上の幅が，TB の形状の縦横比が小さい場合は**図 6(a)** のように小さいが，TB の形状の縦横比が大きい場合は**図 6(b)** のように大きい．

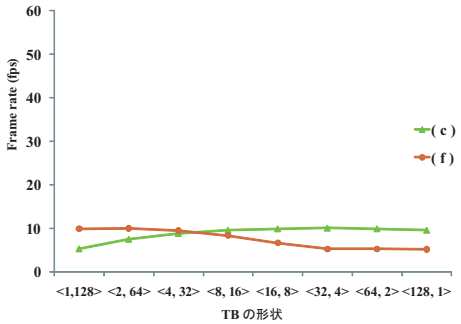
また，視点位置（c）および（f）では，図 5(a) および図 5(b) のグラフにおいて FR の変化がなだらかである．この 2 つの視点位置の共通点は，図 3 ボリューム空間と投影面の位置関係において参照ストライドの最も小さい x 軸の方向が投影面に対して垂直なことである．提案手法では，ボリューム空間において投影面と並行な平面に着目する．したがって，



(a) 視点位置（a）および（e）



(b) 視点位置（b）および（d）



(c) 視点位置（c）および（f）

図 7 対称的な視点位置におけるフレームレートの変化

参照ストライドの最も小さい x 軸方向が無視され，同時参照メモリ領域の局所性が全体的に低くなっている．

図 7 は，図 3 に 6 つの視点位置の中で，投影面に並行な平面上の 2 つの方向が対称な，2 つの視点位置の組み合わせ 3 通りについてそれぞれ FR 変化を比較したグラフである．**図 7(a)**，**図 7(b)**，**図 7(c)** のいずれもグラフの交点は TB の形状が $\langle 4, 32\rangle$ と $\langle 8, 16\rangle$ の間，すなわちワープ形状が $\langle 4, 8\rangle$ と $\langle 8, 4\rangle$ の間にある．したがって，ワープの縦長と横長の境界を基準として，視点位置に応じた TB の形状を提案したことが正しいといえる．

6. ま と め

本稿では，VR の高速化を目的として，視点の位置に応じて TB の形状を選択する手法を提案した．ボリューム空間と投影面の位置関係に着目することにより，視点の位置に応じて参照ストライドの小さいボリューム空間上の軸方向に対して，ワープの並びが並行となるように TB の形状を選択する．これにより，TM の参照における局所性を向上できる．

評価実験では，$1024 \times 1024 \times 1024$ ボクセルのボリュームに対し，処理内容に対称性のある 6 つの視点位置において TC ヒット率および FR を計測した．結果として，TB の形状を $\langle 8, 16\rangle$ または $\langle 16, 8\rangle$ に固定した場合と比較して TC ヒット率を最大で 32.6%ほど向上でき，実行時間を半減できた．

今後の課題として，最悪時の実行時間を短縮し滑らかな描画を実現することが挙げられる．

謝辞 本研究の一部は，科学研究費補助金基盤研究（B）（2330007）の支援を受けた．

参 考 文 献

1) Drebin, R. A., Carpenter, L. and Hanrahan, P.: Volume Rendering, *Computer Graphics (Proc. SIGGRAPH'88)*, Vol.22, No.3, pp.65–74 (1988).
2) Ney, D.R., Fishman, E.K., Magid, D. and Drebin, R.A.: Volumetric Rendering of Computed Tomography Data: Principles and Techniques, *IEEE Computer Graphics and Applications*, Vol.10, No.2, pp.24–32 (1990).
3) Uselton, S.P.: Volume Rendering for Computational Fluid Dynamics: Initial Results, Technical Report RNR-91-026, Nasa Ames Research Center (1991).
4) Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E. and Phillips, J.C.: GPU Computing, *Proceedings of the IEEE*, Vol.96, No.5, pp.879–899 (2008).
5) NVIDIA Corporation: CUDA Programming Guide Version 3.2 (2010).
6) 額田匡則，小西将人，五島正裕，中島康彦，富田眞治：参照の空間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.SIG11(ACS 3), pp.137–146 (2003).
7) 成瀬 彰，住元真司，久門耕一：GPGPU 上での流体アプリケーションの高速化手法〜1GPU で姫野ベンチマーク 60GFLOPS 超〜，情報処理学会研究報告，2008-HPC-117, pp.49–54 (2008).
8) Pharr, M. and Fernando, R.(eds.): *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Addison-Wesley, Reading, MA (2005).