Vol.2011-MPS-82 No.22
2011/3/8

データ通信時間を隠蔽したGPGPUによる粒子ベースボリュームレンダリングの高速化

松井　絵里佳[†1]　高田雅美[†1]　城　和貴[†1]

本稿では粒子ベースボリュームレンダリング手法に対するGPUをプラットフォームとした最適化を行う．一般にCPUとGPU間のデータ転送には大きなデータ遅延時間を伴う．そこでCUDAランタイムAPIのページロックメモリを使って，CPUとGPU間のデータ転送を高速に行うことが可能な領域を確保しつつ，ストリームを利用したカーネルの実行とデータ転送のオーバーラップにより実行時間の大幅な短縮を図る．台風のボクセルデータ (1188 × 979 × 64，140MB) を用いて実験した結果，データ通信時間とカーネルの実行速度は高速化され，GPUの理論性能の約3割を引き出した．

Optimization of a Particle based Volume Rendering Method for GPGPU with Hiding Data Transfer Latency

ERIKA MATSUI,[†1] MASAMI TAKATA[†1]
and KAZUKI JOE[†1]

In this paper, we present the optimization of a Particle based Volume Rendering method for GPU platforms. In general, data transfer between CPU and GPU accompanies long latency. Using page lock memory of the cuda runtime API, data area is selected so that the data transfer between CPU and GPU becomes faster to reduce the execution time. In the meantime, Using streams, the overlap of data transfer and the execution of kernels is achieved. As the result of experiment with the voxel data of a typhoon (1,188 × 979 × 64, 140MB), data transfer time and kernel execution time are improved and bring out about 30performance of the GPU.

†1 奈良女子大学大学院人間文科研究科
Graduate School of Humanities and Sciences, Nara Womens's University

1. はじめに

3次元可視化技術は幅広い分野で必要不可欠となっている．科学技術分野では，シミュレーションや計測の結果をわかりやすく提示するために，医療分野ではCTやMRIなどでとったデータを画像化するのに3次元可視化技術が利用されている．近年，コンピュータの性能が高まっていくにつれて，複雑高度化した問題の解決に必要とされるシミュレーション技術は，高精度・高分解能化されていく傾向があり，結果としてそのシミュレーション結果であるボリュームデータは大規模・複雑化する．

ボリュームデータの可視化方法の1つにボリュームレンダリング[1)2)3)]という手法がある．ボリュームレンダリングは，3次元画像を作成する時に，物体の表面だけでなく，内部情報も可視化することができる．しかし，データ量と計算量が多いため，大規模なデータの可視化には，非常に膨大な計算時間が必要となる．ボリュームレンダリングを利用してデータを解析し，新たな知見の発見に役立てるために快適な研究を行うには，リアルタイムに3次元可視化を行う必要がある．そこで本研究では，大規模ボリュームデータの可視化に適した手法である粒子ベースボリュームレンダリング[4)]に注目し，並列計算を得意とするGPGPUを用いて高速化を行う．

粒子ベースボリュームレンダリングには2つの特徴がある．1つ目は，粒子を不透明発光粒子とすることにより，粒子をスクリーンに投影する際，Zバッファ法による陰点消去処理が可能であるため，サンプリング点の奥行きソートを必要としないことである．そのため大規模3次元データの可視化に適している．2つ目はモデルが粒子なので細かな表示ができることである．はっきりと実態を持たないもの，例えば霧や煙，雲などの可視化に適している．

高速化はGPGPU[5)]を用いて行う．GPGPUとは，本来グラフィックを出力することが目的であるGPUを汎用計算に応用する技術のことである．高速化を行うGPUというものは，画像処理に必要な計算を行う半導体チップで，高い計算能力を持っており，その上価格は数万円程度であり，簡単に設置することも可能なため，誰にでも簡単に導入することができる．本研究では粒子ベースボリュームレンダリングのGPUへの効率のよい適応法を述べる．また，開発環境はNVIDIAが提供するGPU向けのC言語の統合開発環境であるCUDA[6)]で行う．

第2章では，関連研究として，粒子ベースボリュームレンダリング，GPGPU，CUDAについて紹介する．第3章で，具体的な高速化手法について述べ，第4章において，実験と考察を行う．最後に第5章にて本稿のまとめを行う．

Vol.2011-MPS-82 No.22
2011/3/8

2. 関連研究

2.1 節にて粒子ベースボリュームレンダリングについて述べる．次に 2.2 節で高速化で利用する GPGPU について述べる．2.3 節にて使用する言語である CUDA について述べる．

2.1 粒子ベースボリュームレンダリング

データの可視化方法の 1 つにボリュームレンダリングという手法がある．ボリュームレンダリングは，ボリューム情報から直接レンダリングする方法である．3 次元画像を再構成する時に，物体の表面だけでなく，内部情報も反映してレンダリングを行うことができる．ボリュームレンダリングでは物体の表面の表示を行った場合，メモリに表面の情報だけでなく，ボリューム情報も残る．ボリューム情報を常に保持しているため，いつでも様々な可視化方法で処理ができるという利点がある．しかし，情報の維持と計算量の多さのため，この処理には膨大なメモリとそれを高速に処理できる演算プロセッサが必要となる．

この問題を解決するために，粒子ベースボリュームレンダリングが提案されている．粒子ベースボリュームレンダリングは，京都大学の小山田研究室によって開発されたボリュームレンダリングの手法である．従来のボリュームレンダリングでは，サンプリング点を視線にそって視点方向か，その逆方向に順次設定し，視線計算の積分結果を再利用するため，サンプリング点について視点を基準とした順序付けが必要となる．その順序付けの計算のため，計算時間が膨大となる．これに対して粒子ベースボリュームレンダリングでは，ボリュームデータから求められる粒子密度の値を基に粒子を生成する．つまり，粒子ベースボリュームレンダリングでは，ボリュームデータ内部に不透明度属性を持つ発光粒子を生成し，ボリュームデータをこれら不透明発光粒子集合という離散モデルとして表現している．これによって，奥行きソートの必要がなくなり，任意の順序で計算することができるため，大規模ボリュームデータの可視化に特化した可視化手法であると言える．可視化の際，粒子の投影順序は任意で良く，他粒子との奥行き比較のみで陰点消去処理を行うことにより粒子自体の透明度も考慮することなくレンダリングが可能である．また，投影の際は粒子からスクリーンへ視線を向けるため，空間にデータがない場合には計算をしなくてもよい．そのため，疎なデータの可視化に適している．図 1 に従来のボリュームレンダリングと粒子ベースボリュームレンダリングの計算手法の違いを示す．

粒子ベースボリュームレンダリングのアルゴリズムは，以下の通りである．

(1) ボクセルデータの読み込み
(2) 伝達関数により不透明度を計算

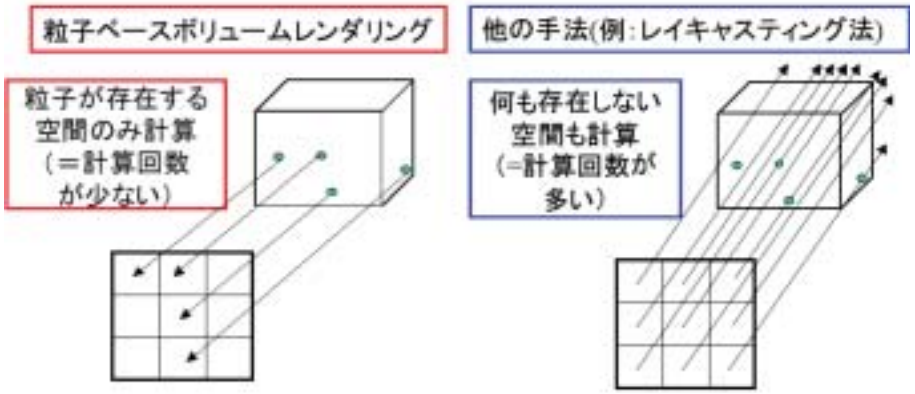

図 1 ボリュームレンダリングの計算手法

(3) 不透明度を利用して粒子密度を計算
(4) 粒子密度より生成粒子個数の決定
(5) 生成粒子の位置，色を求める
(6) スクリーンに投影
(7) サブピクセル処理

まず手順 (1) ではボクセルデータを読み込む．このボクセルデータは，バイナリの形式で保存されている．そのため，利用可能なスカラ値に変換する．手順 (2) では読み込んだデータを伝達関数を通し，ボクセルごとに不透明度を求める．手順 (3) では，不透明度を利用して，ボクセル毎の粒子密度を求める．手順 (4) では，粒子密度から，ボリューム内に発生させる粒子の個数を決定する．手順 (5) では，乱数により粒子の生成位置を決定し，ボクセルデータを参照して，粒子の色を決定する．手順 (6) では，生成した粒子に対し透視投影を行い，スクリーンの色を決める．手順 (7) では，ピクセルの色を平均化し，アルファブレンディングを行い，より細やかな描画を行う．

2.2 GPGPU

GPU の描画性能は，半導体技術の進歩とともに急速に向上している．近年，映画やゲー

ムなど幅広い分野での利用により，急速に GPU の性能が向上している．特に G 浮動小数点演算が 1TFlops を超え，倍精度計算の性能も飛躍的に伸びている．

GPU の構造は，複数のマルチプロセッサを持ち，各マルチプロセッサにはストリーミング・プロセッサ (SP) という演算ユニットが 8 個ある．ストリーミング・プロセッサはクロックごとに単精度の積和演算が実行可能であり，同一のマルチプロセッサの中で並行して計算を行うことが出来る．GPU は構成が単純であるため，CPU と比べて性能比で安価であり，さまざまな分野への応用が期待されている．その応用方法の 1 つとして GPGPU がある．

GPGPU とは，General-Purpose Computation on GPU のことで，GPU を用いて汎用計算を行うことである．GPGPU を用いるためのプログラマブルシェーダなどがアセンブラ言語から高級言語に進化することによって，3 次元グラフィックの表示以外の演算にも使われるようになってきている．GPGPU には，3 つの長所がある．1 つ目は高い演算性能をもつこと，2 つ目は価格が安価であり数万円程度で購入できること，3 つ目はパソコンへの搭載の容易さであり GPU を自宅のパソコンに装着するだけで誰でも簡単に利用できることである．

GPGPU の計算では，データの受け渡し時間がボトルネックとなる．そのため，GPGPU による高速化が図れる処理として，演算に依存関係がなく，演算数がデータ数よりも大きいものが挙げられる．以上のことから，GPGPU による高速化は，データ量が巨大で，各データ間の依存関係が低く，並列度が大きいものが適している．

2.3 CUDA

CUDA とは，NVIDIA 社が提供している GPU を利用するための C/C ++言語の統合開発環境でありコンパイラやライブラリなどから構成されている．

GPU で計算するために利用する言語の代表的なものとしては，Cg(C for Graphics)[7]，GLSL(OpenGL ShadingLanguage)[8]，HLSL(High-Level Shader Language)[9] が挙げられる．本研究では，CUDA を使用する．汎用的なプログラムの開発がしやすいからである．CUDA でプログラミングを行うためには，NVIDIA 社の Geforce8 以降の GPU が必要である．

CUDA 対応 GPU では，スレッドという概念を導入している．スレッドとは，ある処理を GPU 上で実行する際に，ハードウェアが自動的に行う処理分割単位である．ハードウェアは，各スレッドを時分割でそれぞれの統合シェーダへ割り当てている．あるスレッドが読み込み命令を行ってから GPU が外部メモリからデータを受け取るまでのレイテンシは，GPU のクロック・サイクルに比べて，非常に長い時間が必要となる．スレッドがデータを待っている間，統合シェーダはアイドル状態になってしまう．そこで，ハードウェアが自動的にスレッドの切り替えを行い，別スレッドを統合シェーダに割り当て，統合シェーダが休みなく稼働するようになっている．このように，GPU と外部メモリの間の高速データ転送能力，スレッド処理と高速なオンチップ・メモリの導入，複数個の統合シェーダによる多数のスレッドの並列処理により，CUDA 対応 GPU は，非常に高い演算性能を発揮できるようになっている．[10]

GPU の種類	TeslaC1060	Geforce8400GS
ピーク演算性能 [GFlops]	933	67
SP 数	240	16
SP 動作クロック [MHz]	1300	900
メモリバンド幅 [GB/sec]	102	6.4
メモリ容量 [MB]	4096	512
メモリクロック [MHz]	1600	800

表 1 TeslaC1060 と Geforce8400GS の性能

3. 高速化手法

本章では，GPGPU を用いた粒子ベースボリュームレンダリングの高速化手法を提案する．3.1 節では開発対象の GPU について述べる．3.2 節では CPU と GPU 間の通信について説明し，3.3 節では粒子ベースボリュームレンダリングの並列計算方法について述べる．

3.1 開発対象の GPU

表 2 は，本研究で開発の対象とする GPU である TeslaC1060 と GeForce8400GS の性能表である．

TeslaC1060 は NVIDIA 初の GPGPU 専用チップである．従来の GPU に比べて浮動小数点演算性能が高く，主な用途はシミュレーション，大規模な計算，高品質の画像生成などで，高性能計算市場での使用を意図した製品である．しかしながら，ビデオ出力端子が搭載されていないため，OpenGL による描画をすることができない．そこで OpenGL での描画には GeForce8400GS を利用する．

GeForce8400GS は GeForce 8 シリーズのローエンド向けモデルである．対応メモリは DDR2 である．Hybrid SLI に対応している表 1 に TeslaC1060 と Geforce8400GS の性能を示す．

3.2 CPU と GPU 間のデータ転送

CPU と GPU 間のデータ転送には多くの時間を要する．そこで，転送時間を短縮するた

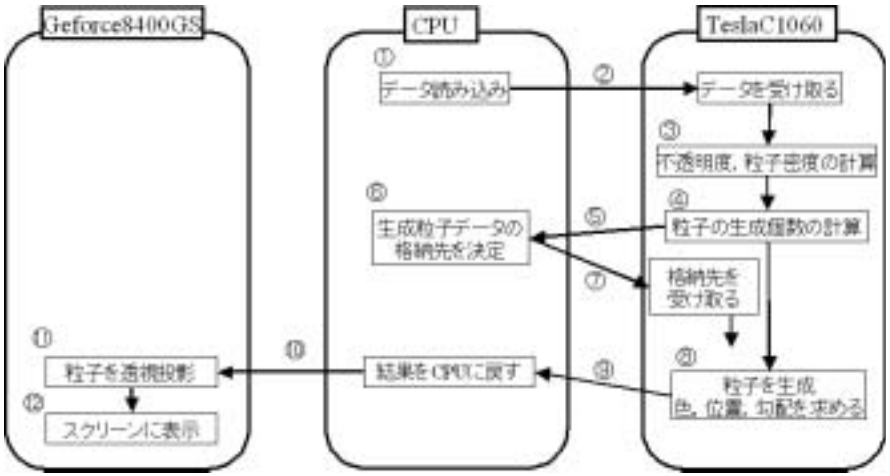

図 2 プログラムの流れ

めに，CUDA ランタイム API のページロックメモリを使うことで，CPU と GPU 間のデータ転送を高速に行うことが可能な領域を確保する．

CUDA 対応 GPU は 1 つの CUDA カーネル関数と，1 つの CPU と GPU 間の通信を同時に実行することができる．そのための条件はカーネル関数とデータ転送の間に依存関係がないことである．そこでストリームを用いる．ストリームはストリーム番号で管理をする．同じストリーム番号を与えられた処理は依存関係があるとみなされる．なお，ストリームは，ページロックメモリで CPU の領域を確保されたデータに対してのみ利用できる．

3.3 GPU での並列処理

2.1 節で紹介した粒子ベースボリュームレンダリングのアルゴリズムに対して，GPGPU による高速化を行う．図 2 は処理手順を示す．

$$index = (gridDim.x * blockDim.x * blockIdx.y) + (blockDim.x * blockIdx.x + threadIdx.x \quad (1)$$

なお，4 章の実験で用いるボクセルデータは 1188 × 979 × 64 であるため，ブロックを 2 次元，スレッドを 1 次元として対応させる．

(1) ボクセルデータ読み込み
(2) TeslaC1060 にボクセルデータを送信

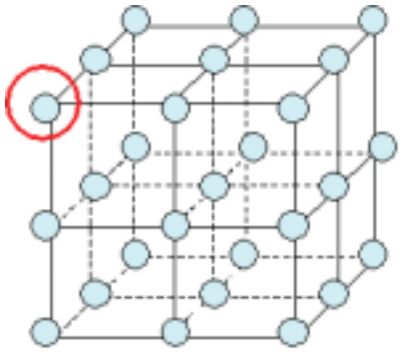

図 3 ボクセルデータ毎に並列化

(3) 不透明度，粒子密度の計算
(4) 粒子の生成個数の計算
(5) 生成粒子個数を CPU に戻す
(6) CPU で生成粒子データの格納先を決定
(7) TeslaC1060 に生成粒子データの格納先を送信
(8) 粒子の色，位置，勾配を求める
(9) 結果を CPU に戻す
(10) Geforce8400GS に粒子データを送信
(11) 粒子を透視投影
(12) 描画結果をスクリーンに表示

手順 (1) ではボクセルデータ読み込み，手順 (2) で読み込んだボクセルデータを CPU から TeslaC1060 へ送信する．手順 (3) では不透明度の計算と粒子密度計算を行う．この高速化では図 3 のようにボクセルデータ毎に並列化を行う．手順 (4) と手順 (8) の粒子生成個数の決定と，生成粒子の位置，色勾配の計算は，図 4 のようにボリュームごとに並列化を行う．手順 (5) は手順 (4) で求めた生成粒子個数を CPU に戻す．手順 (6) では CPU で生成粒子データの格納先を決定する．この計算はブロック間のデータのやりとりを行う必要があるため，CPU に戻して計算する必要がある．手順 (7) では TeslaC1060 へ生成粒子データの格納先を送信する．手順 (9) では TeslaC1060 では描画処理ができないため，粒子データを CPU に戻し，手順 (10) は粒子データを描画に用いる GPU の Geforce8400GS に送信する．手順 (11) と手順 (12) は OpenGL の関数を用い，透視投影計算，スクリーンに描画結

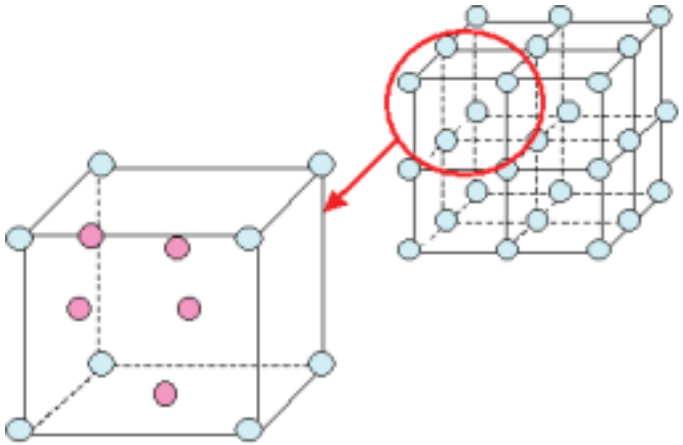

図 4 ボリュームごとに並列化

果を表示する．

手順 (2)，(5)，(7)，(9)，(10) では，GPU と CPU 間で通信が必要となる．そこで，高速化のために，データ，生成粒子の書き込み位置，粒子の色，位置，勾配の配列にページロックメモリを使用する．また，データ転送時にはストリームを使用する．手順 (2) と手順 (3) をストリーム 1，手順 (7)，(8)，(9) をストリーム 2 とする．CUDA でのメモリアクセスには，グリッド・ブロックという概念を利用する．その際，ブロック ID とスレッド ID を取得し，その ID を利用してメモリにアクセスし，並列計算を行う．ID として，gridDim はブロックのグリッドの次元，blockDim はスレッドのブロックの次元，blockId はグリッド内のブロック番号，threadId はブロック内のスレッドの番号を表す．式 (1) により，計算を行っているボクセルデータ若しくはボリュームの位置を求めることができる．

4. 実験と考察

実験環境は，CPU は Intel(R) Core(TM) i5 CPU 760@2.80GHz，メモリ 4GB GPU は TeslaC1060 と Geforce8400GS を利用し，OS は Fedra10，使用言語は CUDA2.3 とする．ボクセルデータは，1188 × 979 × 64 (74,435,328) の 140MB の台風の密度のボクセルデータを使用する．TeslaC1060 での不透明度，粒子密度の計算，生成粒子数の決定，生成した粒子の位置，色，勾配の計算の高速化効果について検証実験を行う．そのために，実行時間

	ストリーム未使用	ストリーム使用
カーネル（粒子密度）	0.066	0.025
カーネル（粒子の位置，色，勾配）	0.013	0.013
データ転送 (ボクセルデータ)	24.646	0.044
データ転送 (粒子データの格納先)	48.450	0.005
データ転送 (粒子の色，位置，勾配)	96.741	0.008

表 2 ストリーム使用時の計算速度の比較 [msec]

を CPU のみと，TeslaC1060 を適用した時のものをそれぞれ計測する．粒子ボリュームレンダリングの設定は，TF（S）= S/19840，α =TF （TF:伝達関数 S:スカラ値）とする．

4.1 節ではストリームの使用時と未使用時の計算時間の比較を行い，4.2 節では GPU と GPU の計算速度の比較を行い，実行性能やピーク比を示す．

4.1 ストリーム使用時の計算速度の比較

本節では，ストリームの効果を確認する．

3.3 節より，本提案には，2 つのストリームが存在する．ストリーム 1 には，カーネルとして粒子密度の計算が，データ転送としてボクセルデータの通信が含まれている．ストリーム 2 には，カーネルとして粒子の位置，色，勾配に関する計算が，データ転送として粒子の格納先の指定と，粒子の色，位置，勾配に関するデータが含まれている．

表 2 は，ストリーム使用時と未使用時の各カーネルとデータ転送の実行時間である．

カーネルの実行時間に関して，ストリーム 1 に含まれる粒子密度の計算が速くなっている．これは，次のボクセルデータの通信を待つ間に，先のボクセルデータに含まれる粒子の情報を用いて計算をすることができるからである．つまり，何も処理を行わない待機時間が減少していることを意味する．

ストリーム 2 に含まれるカーネルの実行時間に関して，ストリームの利用の有無に関わらず，あまり変化していない．これは，必要とするデータが少量であるために，待機時間がほとんどないためであると考えられる．

データ転送に関して，ストリームを用いた場合，ボクセルデータでは 560 倍，粒子データの格納先では 9690 倍，粒子の位置，色，勾配では 12729 倍速くなっている．これは，通信速度そのものが高速になったのではなく，通信がカーネル処理と同時に行われているため，単純に通信のためだけに必要となる時間が大幅に減少したものと考えられる．

以上より，ストリームを用いることによって，通信時間は隠蔽され，全体として大幅な時間短縮になることがわかる．

	CPU	GPU
粒子密度	5396.108	0.025
生成粒子数の決定	4292.623	0.015
粒子の位置，色，勾配	4744.648	0.013

表 3 CPU と GPU の計算速度の比較 [msec]

	演算量	実行性能 [GFlops]	ピーク比
粒子密度	3126283776	125.05	0.13
生成粒子数の決定	2486617812	165.77	0.18
粒子の位置，色，勾配	3903282340	300.25	0.32

表 4 実行性能

4.2 CPU と GPU の計算速度の比較

本節では，2.8GHz の CPU と TeslaC1060 を用いて，3.3 節で説明したストリーム 1 および 2 に含まれるカーネルである粒子密度の計算，生成粒子数の決定，粒子の位置，色，

勾配の決定に関する計算の時間を比較する．表 3 は，各カーネルの実行時間を表す．この表より，CPU と比較して，TeslaC1060 では，粒子密度計算が 215844 倍，生成粒子数の決定が 286174 倍，粒子の位置，色，勾配に関する計算が 364972 倍，高速になっていることがわかる．これは，CPU よりも非常に速い TeslaC1060 を用いて計算時間が短縮されたことに加え，TeslaC1060 において各カーネル内の計算が並列処理されているためであると考えられる．また，TeslaC1060 で計算するために必要となる通信に関しても，4.1 節で示されているように，通信時間がカーネルの処理に隠蔽されているため，実行時間の遅延が生じていないことも要因であると考えられる．

表 4 は，TeslaC1060 における各カーネルの実行性能とピーク比である．表 3 より，CPU と比べ TeslaC1060 では非常に高速に計算できているが，TeslaC1060 のピーク比としては一番効率的に計算できている粒子の位置，色，勾配に関する計算でも 3 割ほどしかない．これは，GPGPU を有効に用いるためのテクスチャメモリやシェアードメモリを利用していないためであると考えられる．ゆえに，今後の課題として，これらのメモリの活用を考慮した高速化手法の改良が必要であると思われる．

以上より，TeslaC1060 のピーク性能を十分に引き出してはいないが，ストリームや並列処理を用いて，CPU よりも非常に高速に計算できる本提案方法は，有効であると考えられる．

5. ま と め

本稿では，大規模ボリュームデータの可視化に適した手法である粒子ベースボリュームレンダリングに注目し，並列計算を得意とする GPGPU を用いて高速化を行った．

高速化手法としては不透明度の計算と粒子密度計算の高速化はボクセルデータ毎に並列化を行い，粒子生成個数の決定と，生成粒子の位置，色勾配の計算は，ボリュームごとに並列化を行った．CPU と GPU 間のデータ転送には多くの時間を要するため，CUDA ランタイム API のページロックメモリを使って，データ，生成粒子の書き込み位置，粒子の色，位置，勾配の配列の確保を行い，ストリーム機能でデータ転送とカーネル実行を同時に行った．カーネル（粒子密度），データ転送 (ボクセルデータ) を 1 つのストリームとしてカーネル（粒子の位置，色，勾配)，データ転送 (粒子データの格納先)，データ転送 (粒子の色，位置，勾配) をもう 1 つのストリームとしてデータ転送とカーネルの非同期に実行した．データ転送時間はボクセルデータでは 560 倍，粒子データの格納先のデータ転送では 9690 倍，粒子の色，位置，勾配データ転送では 12729 倍高速になった．これより，ストリームを利用することは十分効果的であると言える．

CPU と GPU の計算時間の比較では，粒子密度の計算では 215844 倍，生成粒子数の決定では 286174 倍，粒子の位置，色，勾配では 364972 倍もの高速化ができた．ただしピーク比はそれぞれ 0.13，0.18，0.32 と，1 番高いピーク比でも 3 割ほどしか出ていない．これはテクスチャメモリやシェアードメモリなどの高速なメモリを使用していないためだと考えられ，今後改良することによって，より高速な計算が可能になるものと期待される．

参 考 文 献

1) CG 教科書:　http://chiyo.sfc.keio.ac.jp/cgsoft/Release/Textbook/index.html
2) 千葉則茂・土井章男:　3 次元 CG の基礎と応用　サイエンス社
3) 伊藤貴之: CG とコンピューティング入門 サイエンス社
4) 坂本尚久，小山田耕二：粒子ベースボリュームレンダリング，可視化情報学会論文集 Vol.27 No.2，pp.7-14(2007/02)
5) GPGPU: http://gpgpu.org/
6) CUDA: http://www.nvidia.co.jp/object/cuda_home_new_jp.html
7) Cg: http://developer.nvidia.com/page/cg_main.html
8) GLSL:http://www.opengl.org/documentation/glsl/
9) HLSL: http://msdn.microsoft.com/ja-jp/library/cc973422.aspx
10) 青木尊之・額田彰:　はじめての CUDA プログラミング　工学社