2007－CG－126 (12)
2007／2／20

GPUを用いた高品質ボリュームレンダリングに関する研究

高棹 大樹† 金井 崇†† 山口 泰†

GPUを用いたボリュームレンダリングには，主にテクスチャベース法，splatting，そしてレイキャスティング法などがある．本研究ではsplattingを対象とした画質と処理速度に関して検討を行った．従来splattingを用いる場合，オブジェクトを見る角度が変化するたびに，個々のボリュームデータ毎に変換行列を掛け，それらをソーティングしていた．一方，簡便な手法として，視線方向に最も平行な軸に垂直となる面をスライス面として，ボクセル描画の順序を定める方法があったが，この場合は角度によって画質が低下するという問題があった．そこで，本研究では正則なボクセル格子の関係を利用することとし，DDA(Digital Differential Analyzer)を拡張したスライス法を提案する．また，実際にGPUを用いたsplattingの一部として実装し，従来手法との比較を通して提案アルゴリズムの有効性を確認した．

High Quality Volume Rendering using GPU

Daiki Takasao,† Takashi Kanai†† and Yasushi Yamaguchi†

This paper aims to accelerate splatting algorithm, while keeping its quality. In order to achieve high quality images, splatting needs to sort all voxels each time viewing direction is changed. However, this sorting cost is not negligible. Thus, the previous studies approximate by the order along the major axis that is most parallel to the viewing direction. This approximation spoils its rendering quality, especially when the volume is observed in its diagonal directions.

This paper proposes a new ordering method based on DDA(Digital Differential Analyzer) concept. We implemented this method and made some experiments to see its validity.

1. はじめに

近年の計測装置の高度化に伴い，それらから得られる膨大な数値情報を視覚的にわかりやすく表現する可視化(visualization)への関心は高まる一方である．その中でも，画像内により多くの情報を持たせることができるボリュームレンダリングへの期待は大きく，医療分野や3Dゲームなど実際に利用される分野も多彩である．また近年のGPU(Graphics Processing Unit)の進歩はボリュームレンダリングのリアルタイム性を大きく向上させた．

GPUを用いたボリュームレンダリングには，主に3Dテクスチャをスライスして描画するテクスチャベース法[6]，各ピクセルについて視線方向に光線を飛ばし，ボクセルデータを再サンプリングするレイキャスティング[6]，そしてsplatting[3]などがある．本研究では高画質な画像を得られる可能性のあるsplattingについて検討を行った．

splattingは本来1ボクセルを1プリミティブとして描画する方法である．このとき視点から遠いものから順に，上描きを行うことで描画を実現している．そのため，オブジェクトを見る角度が変化するたびにCPU側で以下の作業をしなければならない[5]．

- 個々のボクセルに変換行列をかける．
- それらにソーティングを施し，GPUに渡す．

しかし，この手法はボリュームデータの構造を考慮しておらず，その点において非効率であるといわざるを得ない．一方，簡便な手法として，視線方向に最も平行な軸に垂直となる面をスライス面として，ボクセル描画の順序を定める方法があるが，この場合は角度によって画質が低下するという問題がある．

そこで，本研究では正則なボクセル格子の関係を利用することとし，DDA(Digital Differential Analyzer)を拡張したスライス法を提案する．また，実際にGPUを用いたsplattingの一部として実装し，従来手法との比較を通して提案アルゴリズムの有効性について議論する．

† 東京大学大学院学際情報学府
Graduate Shool of Interdisciplinary Information Studies, The University of Tokyo
†† 東京大学大学院総合文化研究科
Graduate Shool of Arts and Sciences
The University of Tokyo

2. 関連研究

1989年にWestover[3]によって提案されたsplattingはボリュームレンダリングの一手法である．splattingではボリュームデータを光の拡散，吸収を行うパーティクルと解釈し，各パーティクルを再構成核で囲み，それらを視線方向に積分することで最終的な画像を得る．

Zwickerら[1]によって提案されたEWA volume splattingはこの再構成核ごとにローパスフィルタをかけることにより，エイリアスの少ない高画質なボリュームレンダリングを可能にした．

Weiら[4]はGPUを用いてEWA volume splattingの高速化を図っている．しかし，視線方向に最も垂直となる面をスライス面として，ボクセル描画の順序を定めているため，後で説明するように画質に問題があった．

3. splatting

splattingとは，描画要素となる三次元空間上の点を再構成核(reconstruction kernel)で囲むことで有限の大きさを持たせ，再構成核の濃度と点の濃度との積を投影方向について積分し，スクリーンに投影するボリュームレンダリングの一手法である(**図1**)．再構成核で囲まれた三次元空間上の点のことをsplatといい，splatをスクリーンに投影したものをfootprintと呼ぶ．再構成核には一般的に楕円体ガウス核(elliptical Gaussian kernel)が用いられる．ボリュームデータ内の1ボクセルを1splatとして描画を行い，各splatのfootprintを積算することで最終的な像を得る．

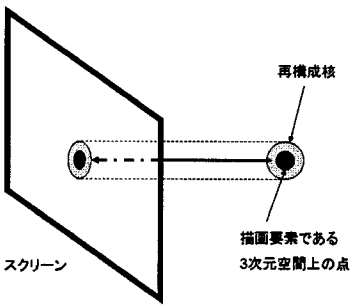


図1 splattingの基本概念

3.1 EWA volume splatting

splattingの問題点として，得られる画質が非常に低くなることが挙げられる．splattingではfootprintをスクリーンに投影し描画を行うため，透視投影を行う場合，視点からの距離に応じてスクリーン上に投影されるfootprintの大きさが変化する．このときスクリーン上のfootprintの大きさが1ピクセルより小さくなってしまうと，エイリアスが生じる．また，エイリアスを防ぐためfootprintを大きくすると，全体的にぼやけた画像になってしまう．

そこでZwickerら[1]はfootprintごとに適切なローパスフィルターをかけることにより，footprint全体の大きさを変えることなくアンチエイリアシングを行った．これにより，精細かつエイリアスのない高品質な描画が可能になった．

3.2 軸方向スライス

ボリュームレンダリングにおいて，スクリーン内の各ピクセルの色は，視点とそのピクセルを結ぶ直線上にぶつかる複数のfootprintの累積で計算される．したがってfootprintの視線方向の積算順序を考慮しないと，本来は隠れて見えない部分が見えてしまうなど不正確なものになってしまう．積算にあたっては視線方向に沿った距離でソーティングすべきだが，コストが高い．

そこで，Westover[3]はボリュームデータ内の各footprintをスクリーンへ投影する際，その順序を三次元ボリュームデータが持つローカル座標の三つの軸(x,y,z軸)に従って決めた．ボリュームデータを視線方向に最も平行な軸に垂直となる平面でスライスし，スライス単位でfootprintを計算した後に各スライスを視点から遠い順に重ね合わせることで最終的な画像を得る．

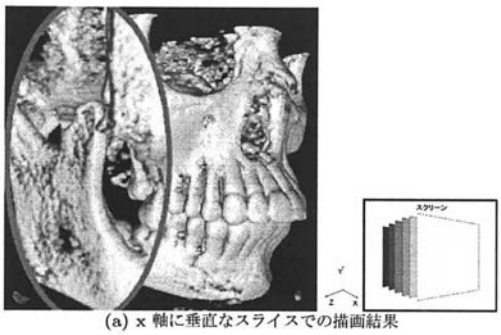
図3(a)ではボリュームデータが視線方向に最も平行な，z軸に垂直な平面でスライスされている．この手法を軸方向スライス(axis-aligned slices)と呼ぶ．なお，ZwickerらもWestover同様に軸方向スライスを採用している．

3.3 軸方向スライスの問題点

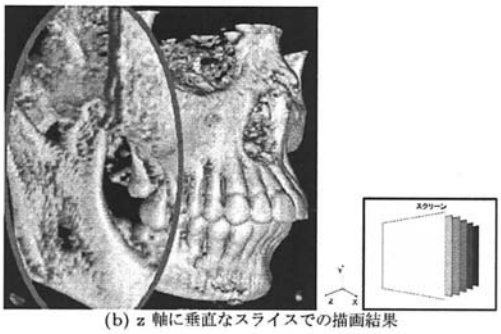
軸方向スライスは必要な計算量が少なく，そのため高速な描画が可能であるが，画質に関しては問題を持っている．**図2**は，軸方向スライスを用いたEWA volume splattingの描画結果を拡大したものである．二枚ともヒトの頭部頭蓋骨を斜め45度付近のほぼ同じ角度で描画しているが出力された画像の，特に円で囲まれた部分が多少異なっていることがわかる．これは，この角度付近でスライスを行う軸が切り替わったためである．**図2(a)**ではボリュームデータをx軸に垂直な平面にスライスしているのに対し，**図2(b)**ではz軸に垂直な平面でスライスしている．スライスの行われる軸は視線方向と最も平行なものが選ばれるので，斜め45度付近ではこのようにスライス面が突然切り替わってしまう．

そこで，Klausら[2]は，**図3(b)**のようにボリュームデータのスライスを，軸に対してではなく視線方向に垂直な平面で行う視線方向スライス(image-aligned slides)を考案した．軸方向スライスのようにある角度で突然スライス平面が切り替わることがないので，滑らかな描画が可能になる．

splattingで視線方向スライスを行う際，通常はボ



(a) x 軸に垂直なスライスでの描画結果



(b) z 軸に垂直なスライスでの描画結果

図 2 スライス軸が変更されることによる描画結果の違い

クセルごとに座標変換行列を掛け，その後カメラ座標の z 軸をキーにクイックソートなどを用いてソーティングを行う．この座標変換とソーティングはオブジェクトを見る角度が変わるたびに行う必要がある．ボクセル数が多くなるとこのソーティングに計算時間を要し，描画のサイクルタイムが低下する原因となる．

そこで本研究では，通常のソーティングを行わない代わりに，正則なボクセル格子の関係を利用することとし，DDA(Digital Differential Analyzer) を拡張した手法を提案する．

4. 提案手法

DDA(Digital Differential Analyzer) とは，幾何形状の微分量を利用して描画を行う手法である．DDA の代表的なものでは，ブレゼンハムの線分描画などが挙げられる．

4.1 DDA による線分の描画

図 4 のように点 P_0 から点 P_1 に向かう線分の描画法を考える．二点間の差分ベクトルの x 成分，y 成分を Δx，Δy とする．$\Delta y > \Delta x$ のとき，y 軸が描画の基準方向となり，点 P_0 から y 軸方向に 1 ピクセルずつ描画されていく．このとき x 方向の変位量は $\Delta x/\Delta y$ である．この変位量を $d \leftarrow d + \Delta x/\Delta y$ として累積し，$d > 1.0$ の場合に x 軸方向に 1 ピクセルシフトして $d \leftarrow d - 1.0$ とする．これを反復することで線分が描画される．

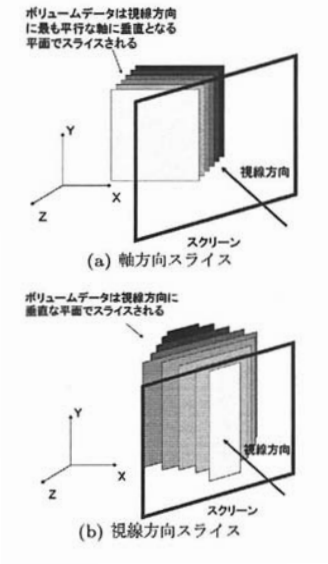


図 3 スライス法

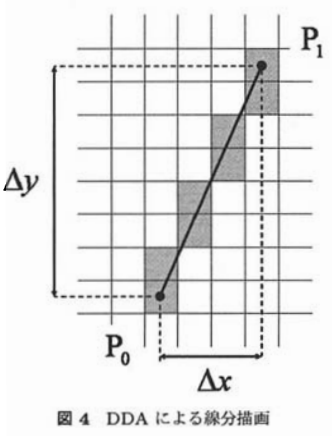
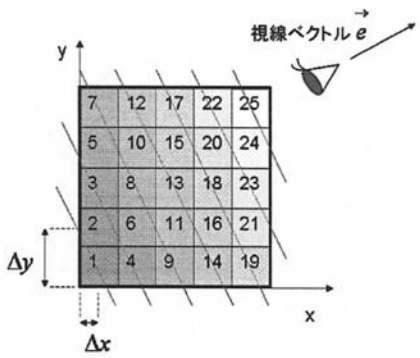


図 4 DDA による線分描画

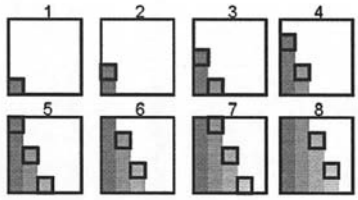
4.2 二次元空間での場合

はじめに，二次元空間内のボクセル面について，提案スライス法を説明する．なお，ここでは描画するボクセルの順番は視点から最も遠いものから描画する back-to-front 方式をとる．視線ベクトル $\vec{e}$ の x 成分，y 成分それぞれの逆数を Δx，Δy とする．ここからは $\Delta y > \Delta x$ の場合を例にとり説明する．

この手法では，ボクセルを列単位で扱う．この列の方向が描画の基準方向であり，Δx，Δy の大小で決められる．ここでは，$\Delta y > \Delta x$ なので y 軸が基準方向となる．また，このボクセル列のうち，各ステップにおいて描画対象となるもののリスト S を用意する．後で述べるように，このリストには視点から離れている



(a) 描画されるボクセルの順番（数字は描画される順番を表す）



(b) 各ステップでの描画（数字はステップ数を表す．色が付いている部分は各ボクセル列内の既に描画されたボクセルであり，太い線で囲まれたものがそのステップで描画される）

図 5 $\Delta x/\Delta y = 0.5$ の場合における二次元での提案アルゴリズム

順にボクセル列が並ぶ．

i. まず，視点から最も離れているボクセル列を S に入れ，$d \leftarrow 0$ で初期化する．

ii. S 内の各ボクセル列について，視点から最も離れている未描画ボクセル 1 つを描画する．S 内の全ボクセル列を処理し終えたところで 1 ステップとする．

iii. 1 ステップを終えたら，以下の 2 つの処理を行う．
- $d \leftarrow d + \Delta x/\Delta y$ とする．このとき $d > 1.0$ となったら，視点から最も離れている未描画のボクセル列を S の最後に加え，$d \leftarrow d - 1.0$ とする．
- S の先頭ボクセル列の全ボクセルが描画済みであれば，そのボクセル列を S から取り除く．

iv. S が空でなければ ii. に戻る．

図 5 に $\Delta x/\Delta y = 0.5$ のアルゴリズムのイメージを示す．

4.3 三次元空間でのスライス

次に，このアルゴリズムを三次元に拡張する．三次元の場合は前節で説明した二次元のボクセル面が積み重なったものとして処理される．各ステップにおいて描画対象となるボクセル面のリスト P，及び P 内の各ボクセル面 $index$ において描画対象となるボクセル列のリスト $S[index]$ を用意する．視線ベクトル $\vec{e}$ の x 成分，y 成分，z 成分それぞれの逆数を Δx，Δy，Δz とし，$\Delta z > \Delta y > \Delta x$ の場合の擬似コードを**図 6** に示す．

5. 実　　験

5.1 ソーティングの単純比較

表 1 に各ボリュームサイズでの全データのソートに要する時間の比較を示す．この値は視線を変えながら測定したソート時間の平均を取ったものである．実験に使用した CPU は Pentium4 3.2GHz である．提案手法は軸方向スライスと同様にデータサイズに対してほぼ線形のコストで処理されており，高々2〜3 倍程度の処理時間しか必要としない．

表 1 全データのソート時間の比較（msec）

		視線方向スライス	
サイズ	軸方向スライス	提案手法	クイックソート
64^3	0.352	0.982	73.1
100^3	0.946	2.43	306
128^3	2.82	5.67	615
256^3	17.6	34.8	5.12×10^3

5.2 splatting への組み込み

次に前項で述べた各アルゴリズムを，splatting に組み込み実験を行う．本実験では，GPU に GeForce 7800 GS，また，グラフィックス API は OpenGL，シェーディング言語は Cg を用いた．splatting の主要アルゴリズムはプログラマブルシェーダーを用いて実装している．**図 7** の hydrogenAtom（ボリュームサイズ，128^3）を splatting で描画をしたところ，軸方向スライスを用いた場合ではサイクルタイムが平均 2.5fps であった．したがって，1 フレームの描画には 400msec かかっており，表 1 のソート時間 2.82msec と比較すると GPU が律速しているのは明らかである．そのため，一般に GPU 内の処理量を軽減させる処置が取られる．すなわち，一定の閾値以下で描画に影響を及ぼさないとみなされるボクセル（ボクセル値がゼロなどの場合）は GPU に渡さずに負担を軽減させるのである．

クイックソートの場合は，予め閾値判定を行っておき，選ばれたボクセルのみをソーティング対象にすることで GPU に限らず CPU の負担も軽減できる．軸方向スライスの場合も，予め閾値判定に選ばれたボクセルを視線方向に応じて軸方向でソートしたデータを全て保存しておけば，描画の際，視線方向の変化に従って，データを切り替えることで必要な CPU の計算量を大幅に削減できる．しかし提案手法の場合は，図 6

```
ボクセル列リスト S[index] とボクセル面リスト P を空にする;
for (∀index) d_S[index] ← 0;
d_P ← 0;
最も離れているボクセル面を P に入れる;
do {
    for (P 内の全各ボクセル面 index){
        for (S[index] 内の全ボクセル列){
            最も遠いボクセル 1 つを描画する;          (1)
        }
        d_S[index] ← d_S[index] + Δy/Δz;
        if (d_S[index] ≥ 1.0){
            視点から最も離れている未描画のボクセル列を S[index] の最後に加える;
            d_S[index] ← d_S[index] − 1.0;
        }
        if (S[index] の先頭ボクセル列内の全ボクセルが描画済み)
            先頭ボクセル列を S[index] から外す;
    }
    d_P ← d_P + Δx/Δz;
    if (d_P ≥ 1.0){
        視点から最も離れている未描画のボクセル面を P の最後に加える;
        d_P ← d_P − 1.0;
    }
    if (P の先頭ボクセル面内の全ボクセルが描画済み)
        先頭ボクセル面を P から外す;
} while (P が空でない);
```

図 6 提案アルゴリズムの擬似コード

の (1) の段階で，閾値とボクセル値を比較し GPU に渡すか否かの判断を行う．つまり描画時に全ボクセルの閾値判定を行うことになる．

表 2 は充填率を変化させた際に hydrogenAtom（ボリュームサイズ，128^3）を描画するのに要する時間を示したものである．ここで充填率とは，閾値によって描画の対象となるボクセルの割合である．充填率が増して描画するボクセル数が増えると，提案手法，クイックソートともに描画速度が落ちる．しかし，提案手法では充填率が 8%以上のとき，充填率 ×fps が 1.6〜1.9fps 程度になっている．前に 128^3 の全データを軸方向スライスで描画すると 2.5fps であったことを考えると，概ね GPU で律速していると結論づけられる．

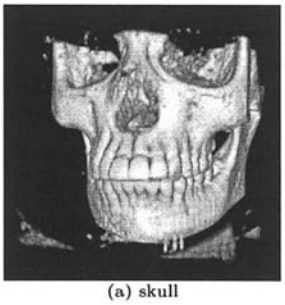
一方クイックソートの場合，充填率 1〜2%のときは提案手法よりも速いがその後急激に速度が低下している．これは，CPU でのソーティング処理がボトルネックになっているものと思われる．すなわち，ボクセル数が $128^3/50$ よりも多くなると，ソート時間の影響によって提案手法が有利になる．

ここまでの実験からは，提案手法が GPU による描画で律速されるように思われるが，実はもう少し複雑な要素が存在する．**表 3** は充填率を変化させた際の，skull（ボリュームサイズ，256^3）の描画時間を示したものである．クイックソートは充填率の増加に伴って急速に処理速度が落ちている．一方提案手法の場合には，充填率 25%の際にほぼ 1fps となっている．これはデータサイズが 128^3 の 8 倍であることを考えると，128^3 の全データを 2fps で処理していることに相当し，やはり GPU が律速していると考えて良い．

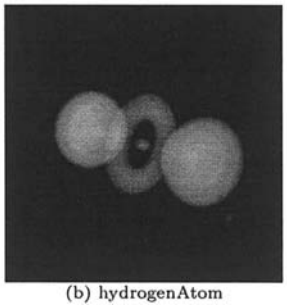
ところが，充填率が 1〜4%の範囲では 4fps 程度に留まっている．さらに注意深く描画速度を検証すると，視線方向によって速度が大きく変動することがわかった．特にボリュームの対角線方向（斜めの方向）から見る場合に急速に速度が低下する．このことから，CPU のキャッシュメモリに影響されているものと考えている．

表 2 hydrogenAtom(サイズ，128^3) を用いた各充填率でのサイクルタイムの比較 (fps)

充填率	提案手法	クイックソート
0.0103	48.3	59.9
0.0203	39.3	40.0
0.0400	29.0	21.3
0.0831	21.0	10.1
0.135	13.3	6.29
0.204	9.38	4.19
0.327	6.00	2.57



(a) skull



(b) hydrogenAtom

図 7 実験に用いたボリュームデータ（提案手法で描画した）

前にも書いたように提案手法では，図 6 の (1) の段階で閾値判定を行っている．256^3 のボリュームデータの場合，描画/非描画のフラグを各ボクセルにつき 1bit で表したとしても閾値データは 2MB になる．軸に沿った方向であれば閾値データを連続にアクセスできるが，斜めの方向になるとランダムではないものの閾値データの広い範囲を飛び飛びにアクセスすることになる．今回実験に用いた Pentium4 3.2GHz は L2 キャッシュメモリが 2MB あり，閾値データがキャッシュに乗り切らずにキャッシュミスが頻繁に発生し，速度低下に繋がっていると考えられる．実際充填率が 1〜4%で描画ボクセル数が変化してもサイクルタイムが変化しないのは CPU がボトルネックになっている証拠である．

表 3 skull(サイズ，256^3) を用いた各充填率でのサイクルタイムの比較 (fps)

充填率	提案手法	クイックソート
0.0144	4.14	7.28
0.0264	4.00	3.92
0.0423	3.99	2.41
0.0936	2.67	1.07
0.123	1.98	0.82
0.257	1.05	0.38

6. ま と め

本研究では，正則なボクセル格子の関係を利用し，DDA を拡張した splatting のためのスライス法を提案した．また，従来手法であるクイックソートとの比較実験を通して提案アルゴリズムの有効性について検証を行った．提案手法はデータをソーティングすることは非常に高速にできるが，CPU のキャッシュサイズに依存する問題があることがわかった．

参 考 文 献

1) Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, Markus Gross. EWA Volume Splatting. In the Proceedings of IEEE Visualization 2001,pp.29-36.
2) Klaus Mueller, Naeem Shareef, Jian Huang, Roger Crawfis. High-Quality Splatting on Rectilinear Grids with Efficient Culling of Occluded Voxels. IEEE Transactions on Visualization and Computer Graphics,1999,pp. 116-134.
3) L. Westover. Footprint evaluation for volume rendering. In Computer Graphics,Proceedings of SIGGRAPH 90,pp.367-376.
4) Wei Chen, Liu Ren, Matthias Zwicker, Hanspeter Pfister. Hardware-Accelerated Adaptive EWA Volume Splatting. In Proceedings of IEEE Visualization 2004.
5) Daqing Xue, Roger Crawfis. Efficient Splatting Using Modern Graphics Hardware. journal of graphics tools, 2003, 8(3):1-21
6) Klaus Engel, Markus Hadwiger, Joe M. Kniss, Christof Rezk-Salama, Daniel Weiskopf. Real-Time VOLUME GRAPHCS. AK Peters, Ltd. 2006