情報処理学会論文誌　Vol. 49　No. 12　4080–4087 (Dec. 2008)

GPUを用いたメタボールの実時間描画

三　嶋　　　仁[†1]　田　中　敏　光[†1]　佐　川　雄　二[†1]

本論文では，グラフィックスハードウェアを使って，メタボールを直接リアルタイム表示する方法を提案する．提案手法は，レイキャストと同様に，視線上を等間隔に区切ったサンプル点でポテンシャルを計算する．ただし，ポテンシャルが存在しない場所で無駄な計算を行うことを避けるために，各粒子の最大影響範囲を表面とする球を包含する立方体を作成し，その視点側の半分を計算候補領域に選ぶ．そして，計算候補領域が不連続の場合には，Z バッファによる前後判定を使って次の計算候補領域にサンプル点を移動することで，効率的なサンプリングを実現する．また，選ばれたサンプル点のポテンシャルを，それぞれの粒子によるポテンシャルを画像に加算合成する方法で計算する．閾値を超えたサンプル点を物体の中にあると判定し，直前のサンプル点との間を補間して，物体の表面を推定する．さらに，視線方向に並んだ 4 点を同時に計算することで．処理コストを減らす．このような，GPU の特性を活用した実装により，メタボールの実時間表示を実現した．実験では，4,096 個の粒子を使って表現した物体を，256×256 画素のサイズで，30 fps 前後で表示することができた．

Real-time Rendering of Metaballs Using GPU

Hitoshi Mishima,[†1] Toshimitsu Tanaka[†1]
and Yuji Sagawa[†1]

This paper presents a real time rendering method for metaballs. The method computes potential at equal interval sampling points like ray-casting. However, in order to remove unnecessary sampling points, a bounding cube is created for each particle then its front-half is selected as the sampling area. At each pixel, the initial sampling point is selected on the front surface of the nearest are. The sampling point shifts at same interval step by step. However, if the sampling areas are not continuous on the ray, the sampling point is made to jump to the next area by using modified Z-buffer algorithm. For each particle, potential at the sampling points is added to the accumulation buffer, which is the same size of the image. Then potential at each pixel is evaluated. If the value is over the threshold, surface of the object is estimated by interpolating between the sampling point and the proximate point. In addition, the cost is reduced by computing 4 sampling points along the ray together. By considering feature of GPU, real-time rendering is realized. The experimental result confirmed that 256×256 images including 4096 metaballs are created in 30 fps.

1. はじめに

本論文では，陰関数表現の一種であるメタボール[5]を，グラフィックスハードウェアを使って，直接リアルタイム表示する方法を提案する．

陰関数表面をリアルタイム表示する手法は，間接法と直接法の 2 つに大別できる．間接法の代表例はマーチングキューブ[6]を使った手法で，処理速度が速いため，近年多く使われている[1]．

この手法では，まず，空間を均一な立方格子に分割し，格子の各頂点でポテンシャルを計算する．すなわち，ポテンシャルをボリュームデータに変換する．次に，格子の最小単位である立方体のそれぞれで，8 つの頂点が物体の内部か外部かを判定し，その内部に境界面を割り当て，それをレンダリングすることで陰関数表面を表示する．

間接法の表面精度は，空間分割の細かさに比例するので，画素の大きさほど細かく空間分割しないと，正しい形状が表示できない．しかし，処理時間や必要とするメモリも空間分割の細かさに比例して増大するので，リアルタイム表示ではそれほど細かく分割できない．データの圧縮[10),11)]やハードウェアの利用[12)–14)]による処理効率の改善により，医療データのように限られた領域にデータが収まっている場合には実用的な表示が可能だが，広い空間にポテンシャルが分布する場合には，解像度が不足する．

直接法の代表例はレイキャスティング[2]を使った手法（Volume ray casting）[3),15)]である．この手法では，視点から各ピクセルの中央を通る視線（ray）を飛ばし，視線上を等間隔に分けたサンプル点のそれぞれで各粒子が与えるポテンシャル値の総和を求める．ボリュームレンダリングでは格子点の値から計算点の値を推定するが，陰関数表面の表示では，計算回数と誤差の増加を防ぐため，サンプル点の値をポテンシャル関数から直接求める．

視点側から，この値が閾値を超えるサンプル点を求め，さらに，直前のサンプル点との間を細かく探索することで，閾値と一致する位置を求める．この点を物体表面とすることで，計算精度の範囲内で表面を正しく求めることができるが，処理に時間がかかる．

†1 名城大学大学院理工学研究科
Graduate School of Science and Engineering, Meijo University

© 2008 Information Processing Society of Japan

このため，GPU を使って処理を高速化する方法が提案されている．Müller らはイメージプレーンを使ってポテンシャルを計算している[9]．しかし，この手法では，粒子が遠く離れていてもその間を等間隔にサンプリングするため，効率が悪い．金森らは GPU による Depth-peeling で視点に近い粒子を求めている[4]．この方法は必要なところだけで計算を行うため効率が良いが，影響を与える粒子をバッファに保存するため，メモリの消費量が多い．また記録できる粒子数に上限があるため，粒子が集中すると表面を正しく求められない場合がある．ほかにも，SLIM 曲面に限定して表面を求める手法[8] などが提案されている．

本論文では，3D ゲームなどを想定して，水流のように粒子が広い範囲を運動する状況を CG でリアルタイム表示することを目的としている．現状のハードウェア環境では処理速度と画像品質のトレードオフは避けられない問題なので，リアルタイム処理できる範囲で品質の良い手法を選択しなければならない．マーチングキューブ法では，格子の間隔を適正に保たないと表示が劣化するため，粒子が運動する範囲が広い場合には適していない．これに対して，レイキャスト法の処理時間は粒子数に比例するので，移動空間に対して粒子数が少ない場合にはこちらのほうが有利になる．

そこで本研究では，レイキャストを GPU で高速処理することで，実時間性を向上させる．具体的には，Z バッファを使って無駄なサンプル点を省き，加算合成を利用してポテンシャルを効率良く計算することで，従来手法よりも実時間表示できる粒子数を増す．

前提条件として視点に最も近い表面のみを計算する．他の手法と同様に，ポテンシャル関数に有効範囲を持たせ，一定以上の距離が離れれば影響量を 0 とする．

2.　提案手法の概要

本手法の処理手順を**図 1** に示す．まず，画素ごとに視点から最も近いポテンシャルが存在する位置を調べ，初期計算点とする．次に加算合成を使って，計算点のポテンシャルを求める．このポテンシャルを閾値と比較することで，物体の内外を判定する．外部であれば，次の計算点を探索し，位置を更新する．内部であれば，その位置を保持する．

実時間で処理を完了するため，ポテンシャル計算，内外判定，計算点の更新のループは，指定した回数だけ繰り返した後に強制終了する．最後に，物体表面を挟む計算点のポテンシャルの値から物体表面の位置を推定し，描画する．

3.　ポテンシャル計算点の算出

ポテンシャルは，レイキャスティング法と同様に，視線を等間隔に区切った位置で計算する．**図 2** は，右方向が視点からの距離，上下方向が視線の角度として描いている．各画素について，図に示すオレンジの線で示した面（視点を中心とする球面）上の点がポテンシャルを計算する候補となる．ただし，この中にはポテンシャルが存在しない位置も含まれているので，以下に示す手法で，計算を始める位置と次の計算位置を決めることで，不要な計算を減らす．

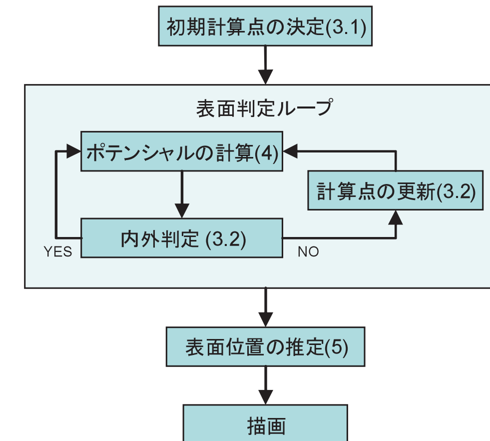


図 1　1 フレームの処理の流れ
Fig. 1　Flow chart of one-frame.

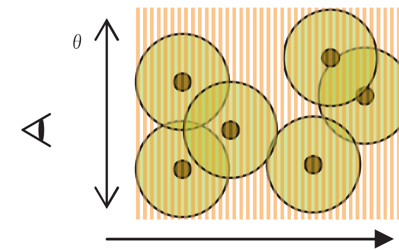


図 2　一定間隔で分割した例
Fig. 2　Evaluation points in constant interval sampling.

3.1　初期位置の決定

ポテンシャル関数に有効範囲を与えているので，粒子の位置を中心とし，最大影響範囲

© 2008 Information Processing Society of Japan

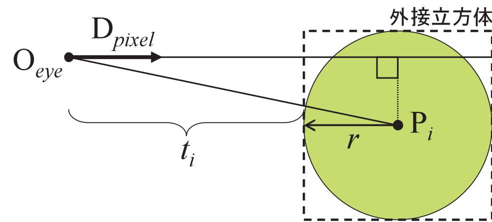


図 3　ビルボードまでの距離の計算
Fig. 3　Distance to a billboard.

を半径とする球の内部しか，ポテンシャル計算の必要はない．そこで，Z バッファ法を使って，視点に最も近い球面を求める．

ただし，ピクセルシェーダで球面を描画するには，球面と視線のとの交点を求めなければならないので，描画コストが大きくなる．そこで，この球に外接し，かつ，視点側の面が視線と直交する立方体を作成し，その視点側の正方形をビルボードとして描画する．このビルボードは，初期化時に作成したポリゴンを頂点シェーダの頂点テクスチャフェッチにより変形して作成する．こうすることで，CPU から GPU へ送る情報は，ビルボードを構成する 4 頂点だけとなる．

ビルボードまでの距離は，ピクセルごとに

$$t_i = \mathbf{D}_{pixel} \cdot (\mathbf{P}_i - \mathbf{O}_{eye}) - r \qquad (1)$$

で計算する．**図 3** に示すように，$\mathbf{O}_{eye}$ は視点の位置，$\mathbf{D}_{pixel}$ はその画素の中心方向を示す単位ベクトル，$\mathbf{P}_i$ は i 番目の粒子の中心位置で，いずれも世界座標系での値である．r は粒子の最大影響距離で，式 (1) で r を引くのは，粒子を囲む立方体の一番手前の面までの距離を求めるためである．

複数の粒子が重なっている場合は，最も手前の粒子までの距離を求めなければならない．この処理は Z バッファで行う．ビルボード内の画素ごとに式 (1) で距離を計算し，同じ値を Z 値（深度値）と画素値とする．これは，GPU では Z バッファの値の読み出しに制限があるためである．Z バッファは，入力が記録している Z 値より小さい場合に限り値を更新し，描画が許可される．距離値を保存するバッファを画像に割り当てることで，そこに最も近い面までの距離が記録される．

こうして求めた距離だけ，視点から視線方向に進んだ点が，ポテンシャルを計算する点の初期位置となる．**図 4** では初期位置を水色で示している．このように，画素ごとに異なる距離から処理を開始する．

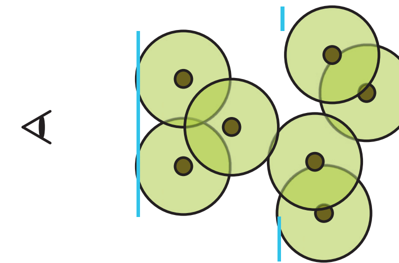


図 4　求められた初期位置
Fig. 4　Initial position of the points.

図 5　計算候補領域
Fig. 5　Evaluation space for a particle.

3.2　計算位置の更新

それぞれの粒子でポテンシャル計算の必要な範囲は，最大影響距離を半径とする球の内部であるが，初期値の計算と同様の理由で，外接立方体で近似する．さらに，裏側は求めないので，**図 5** に赤色で示した，粒子の中心より視点側の部分だけでポテンシャルを計算する．この範囲を計算候補領域と呼ぶことにする．

図 6 に示すように，ある粒子の計算候補領域の後端と次の粒子の計算候補領域の前端が離れていた場合には，無駄な計算を省くために，間を跳ばして次の粒子へ移動する．この処理にも，Z バッファを用いる．処理の概要を以下に示す．

まず，距離バッファを 2 枚用意し，交互に更新するダブルバッファ構造とする．また，距離バッファに計算点の更新停止フラグを持たせ，すでに表面が決まっている場合は現在の距離を維持する．

現在の位置 t_{now} は初期位置の計算で求められているので，これにサンプリング間隔 w を加えた，

© 2008 Information Processing Society of Japan

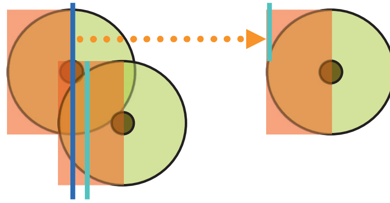


図 6　計算点の移動
Fig. 6　Jump of the evaluation point.

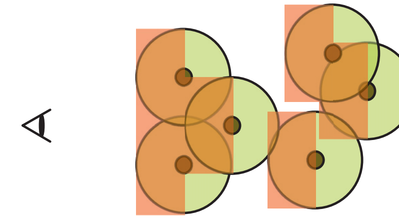


図 7　最終的な計算候補領域
Fig. 7　Final evaluation space for all particles.

$$t'_{next} = t_{now} + w \quad (2)$$

が次の計算位置の候補となる．この値を距離バッファ1 に保存する．

距離バッファ2 と Z バッファを初期化する．この距離バッファ2 に次の計算位置までの距離を保存する．初期値決めと同様に粒子を 1 つずつ取り出し，その候補領域の視点側の正方形を距離バッファ2 に投影する．投影された範囲内の画素のそれぞれで，距離バッファ1 の値を参照し，次の式で計算点の候補より奥にあるかを判定する．

$$t_b = \mathbf{D}_{pixel} \cdot (\mathbf{P}_i - \mathbf{O}_{eye}) < t'_{next} \quad (3)$$

この不等号が成り立つ場合には，候補領域の後端が現在の候補点より視点側にあるので，その粒子は無視できる．式 (3) を満たさない粒子については，

$$t_f = \mathbf{D}_{pixel} \cdot (\mathbf{P}_i - \mathbf{O}_{eye}) - r < t'_{next} \quad (4)$$

を評価する．この不等号が成り立てば，候補領域の内部にあるので，t'_{next} が次の計算位置となる．成り立たない場合には，t_f が次の計算位置の候補になる．

こうして求めた距離を，初期位置決めと同様に，正方形ビルボードの深度値および画素値として与え，Z バッファを使って，距離バッファ2 に描画する．ただし，式 (3) を満たした場合は，この処理を棄却する．以上の処理をすべての粒子について行うと，画素ごとに，次の計算点までの距離が定まる．したがって，視点から各画素の視線方向にこの距離だけ進んだ位置が，次の計算点となる．この処理を繰り返すと，**図 7** の赤色の範囲だけがポテンシャル計算を行う領域となるので，不要な計算を減らすことができる．

4. ポテンシャルの計算

4.1 加算合成を用いたポテンシャル計算

計算点の位置が求められたら，前章と同様に粒子を四角形のビルボードとして描画する．このとき，描画先のピクセル座標で計算点までの距離を格納した距離バッファを参照し，計算点の三次元位置を復元する．そして，この画素における粒子と計算点のポテンシャル値を計算し，値をポテンシャル計算用の 16 bit 浮動小数点バッファに加算合成で書き込む．この処理をすべての粒子に対して行うことで，スクリーン上の全計算点でその点のポテンシャルが計算できる．

これに対し，画素単位で近傍の粒子からのポテンシャルを調べる手法も考えられる．その場合は，近傍探索用の空間分割の作成ならびに，画素単位での近傍探査処理が必要となる．しかし GPU で近傍探査処理に使うツリー探査のような非連続メモリアクセスを行うと，パフォーマンスが低下する．このため，提案手法を採用した．

4.2 連続した計算点の同時計算

1 つの画素は RGBA の 4 つの要素を持てるので，視線方向に並んだ 4 個の計算点でポテンシャルを同時に計算する．計算点の総数は変わらないため，メモリアクセスの負荷は同じであるが，頂点シェーダ部分および，テクスチャ参照といったポテンシャル計算に必要なセットアップ処理を減らす効果があるため，処理時間の短縮が見込める．

同時に 4 点を計算する場合は，1 つ目の計算点の位置は 2 章に示した方法で求め，残りの 3 つは，この計算点の距離にサンプリング間隔を足すことで求める．このため，計算位置更新処理の回数も削減される．この場合，必要のない場所でポテンシャル計算を行うこともあるが，計算領域のサンプル点数を 4 の倍数に選んでおけば，それほど多くの無駄は生じない．

4.3 低解像度距離バッファによる粒子のカリング

計算点へのポテンシャル値が 0 となる粒子を描画対象から外す処理を導入する．従来手法にも，オクルージョンクエリと呼ばれる描画面積を取得する命令を使って描画対象から外す処理の例があるが[4]，描画面積の取得には GPU からのリードバックが必要となるため処理速度が低下する．

© 2008 Information Processing Society of Japan

本手法は 2 次元のスクリーン上に計算点までの距離を保持しているので，これを端から 2×2 画素の小領域に分けて，その中の最大値と最小値を求める．この値を，解像度を縦横 1/2 にした画像 2 枚に保存する．次に，最大値を格納した画像を縦横 1/2 に圧縮して，4 画素の最大値を求める処理と，最小値を格納した画像を縦横 1/2 に圧縮して，4 画素の最小値を求める処理を，指定した低解像度になるまで繰り返す．再帰的に縮小するのは，GPU で効率良く低解像度画像を作るためである．

この最大値と最小値の間に粒子の影響範囲が入っていなければ，その粒子をポテンシャル計算から排除できる．解像度を落としているため，正確な包含判定とはならないが，視点のすぐ近くにある粒子のように，多くの画素に投影される粒子に使うと，各画素で距離を計算して判定するよりも少ないコストで粒子を排除できる．

5. 表面位置の推定

ポテンシャルが始めて非負となる計算点を物体表面と判定するのだが，計算点は一定間隔でサンプリングしているので，物体の表面上にあるわけではない．その 1 つ前の計算点では，ポテンシャルは負となっているので，真の表面はこの間に存在する．

そこで，反復法の一種である Bezier clipping [7] を使用して表面の位置を近似計算する．具体的には，前の計算点までの距離を t_{neg}，後ろの計算点までの距離を t_{pos}，距離 t での陰関数の値を $I(t)$ とするとき，表面までの距離を次の一次関数で推定する．

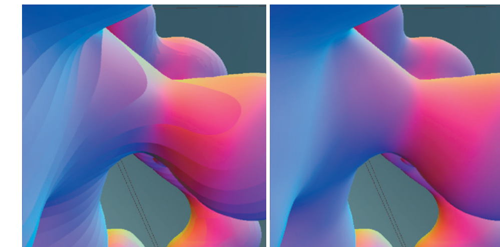
$$t = \frac{t_{neg}I(t_{pos}) - t_{pos}I(t_{neg})}{I(t_{pos}) - I(t_{neg})} \tag{5}$$

次に，距離 t での陰関数の値を計算し，値が正なら後方の，負なら前方の t の値を置き換える．

$$\begin{cases} t_{pos} = t & P(t) > 0 \\ t_{neg} = t & P(t) < 0 \end{cases} \tag{6}$$

この処理を繰り返すことで表面位置をより正しい位置に修正する．

メタボールに適用した例を**図 8** に示す．左側はサンプリングした計算点の位置をそのまま使う場合の画像で，右側は 1 回だけサンプル点の関数値を線形補間して表面位置を求めた場合の画像である．色は表面の法線ベクトルの方向を示しており，色が大きく変わるところでは法線方向が不連続になっている．修正しない画像では不連続の箇所が目立つが，修正した画像では法線が滑らかに変化している．



(a) サンプル位置を表示　　(b)補間した位置を表示

図 8　表面位置の補間

Fig. 8　Estimation of surface position by interpolation.

6. 評価実験

CPU が Core 2 DUO E6600，GPU が Geforce8800GTX の環境で性能を評価した．実装には DirectX9.0c および ShaderModel3.0 を使用した．ポテンシャル $I(t)$ は

$$P(r) = \begin{cases} (1 - r/h)^2 & (0 \le r \le h) \\ 0 & (h < r) \end{cases} \tag{7}$$

$$I(t) = \sqrt{\sum_j P\left(|\mathbf{O}_{eye} + t\mathbf{D}_{pixel} - \mathbf{P}_j|\right) - T} \tag{8}$$

で計算する．粒子の最大影響半径 h は 0.61 に，表面を決める閾値 T は 0.765625 にしている．また，画素あたりの計算点の最大数は，特に指定がない限り，32 点である．

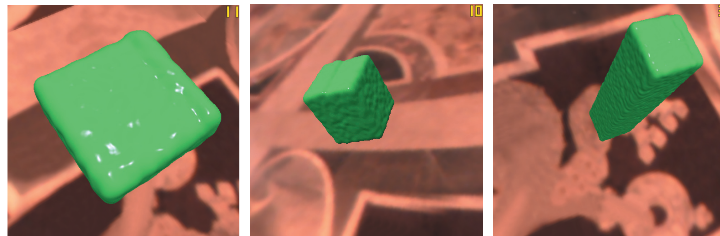
粒子数と解像度を変えた場合のフレームレート（fps）を**表 1** に示す．計測には fraps を利用した．実験に使用した画像を**図 9** に示す．メタボールは 24×24 個の正方形配置を 1 層として，粒子数になるまで積み重ねている．実験に用いたハードウェアでは，画像解像度は 256×256 のときに，粒子数 4,096 個で 35 fps，16,384 個で 21 fps となり，このあたりがリアルタイム表示の限界であった．

図 10 と**図 11** は水の表示例で，図 10 は 4,096 個の粒子を，図 11 は 16,384 個の粒子を使って水の流れを表現している．詳細は省略するが，粒子の運動は近傍の粒子から受ける力を考慮して決めている．それぞれの画像のフレームレート（fps）を，画像の右下に黄色の文字で示している．粒子 4,096 個の場合では，512×512 画素の画像でも 19 fps で描画できている．粒子 16,384 個の場合は，128×128 画素でも 18 fsp とかなり遅く，512×512 画素

© 2008 Information Processing Society of Japan

表 1　粒子数と解像度に応じたフレームレート
Table 1　Frame rate by number of particles and resolution.

平均フレームレート [fps]		粒子数		
		4096	16384	65536
画像サイズ	128x128	60	37	12
	256x256	35	21	7
	512x512	11	10	3
画像（図番）		9(a)	9(b)	9(c)



(a)メタボール 4098 個　(b) メタボール 16834 個　(c) メタボール 65536 個

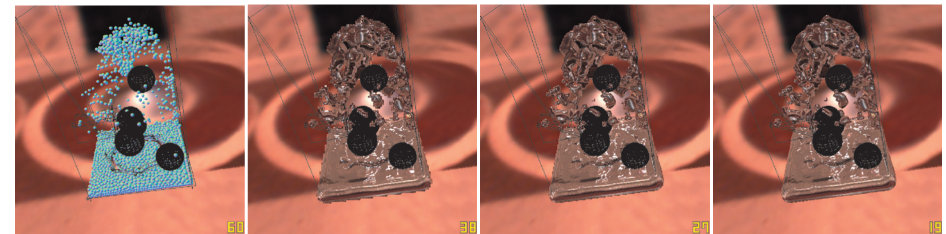
図 9　フレームレート計測実験に使った画像
Fig. 9　Test images for evaluating frame rate.

では 10 fps まで落ちた．どちらの場合も，球による表示の 3 倍程度の時間がかかっており，メタボール表示のコストが高いことが分かる．

メタボールを描画する画素数に比例して処理コストが増えるので，視点近傍のメタボールが多いと処理速度が低下する．このため，一概に粒子数や画素サイズだけで議論はできないが，表 1 や図 10 (c) の結果から，メタボールが画像上に大きく広がっている場合でも，256×256 画素，4,096 粒子の条件ならば，十分に実時間表示できる．

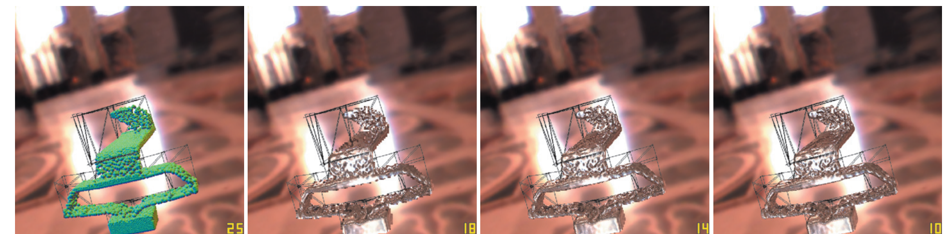
1 画素の計算点の数に上限を設けているため，条件が悪いと穴が開いてしまう．**図 12** はその例で，右上は 1 画素あたり 16 点，左下は 32 点，右下は 52 点計算した場合の結果である．画像更新速度は，それぞれ，36，25，18 fps だった．画像は 512×512 画素で 4,096 個の粒子を使って表示している．

16 点の場合は黄色の円で囲んだ部分に大きな穴が開いている．32 点の場合にも，小さな穴が残っている．穴の発生は，ポテンシャル関数や表面を決める閾値にも依存するが，粒子が離散的に分布している場合に起こりやすい．ただし，提案手法の回数制限は純粋に処理時間によるものなので，悪条件の場合には，描画速度を犠牲にして参照点を増すことで，穴の発生を防ぐことができる．この点は，メモリサイズで回数が制限されている文献 4) の手法に比べて有利である．また，提案手法は等間隔に計算点を移動するため，ポテンシャル関数が多数重なる場合にも単独の場合にもサンプル間隔は変わらない．このため，各ポテンシャル関数の影響限界で分割する文献 4) の手法と比べて，より少ない更新回数で奥へ進むことができる．



(a)球による表示 512² 画素　(b)メタボール表示 128² 画素　(c)256² 画素　(d)512² 画素
（図の右下の数字は各画像のフレームレートを示している．単位は fps.）

図 10　4,096 個の粒子で水流を表現した例
Fig. 10　Water flow depicted with 4,096 particles.

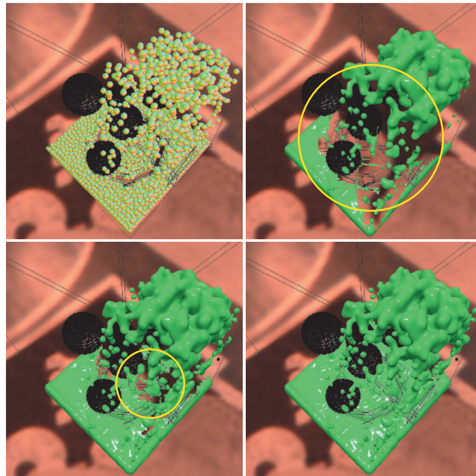


(a)球による表示 512² 画素　(b)メタボール表示 128² 画素　(c)256² 画素　(d)512² 画素

図 11　16,384 個の粒子で水流を表現した例
Fig. 11　Water flow depicted with 16,384 particles.

ただし，提案手法では，ある視線方向について，隣り合う計算点の間だけに影響するポテンシャルにより表面が発生する場合には，それを見落とす可能性がある．これが問題になるのは粒子が密集する場合なので，動的にサンプリング間隔を狭くする．ポテンシャルを与える可能性のある粒子の数は 3.2 節に示した計算点の移動処理で検出できるので，それに応じ

© 2008 Information Processing Society of Japan



左上は粒子表示，右上，左下，右下はそれぞれ，
16，32，54 計算点／画素で画像表示

図 12　計算点数による精度の変化
Fig. 12　change of precision by calculation points.

て移動距離を小さくする．この改良により見落としが軽減されると考えているが，計算点の数が増えるため，品質と速度の両方を考慮して移動距離を変える条件を決める必要がある．

経験的に計算回数を 64 回前後にすることで穴の発生は見られなくなる．また，計算点のサンプリング間隔は，使用するポテンシャル関数に依存するが最大影響範囲の 1/8 より小さくすると良好な結果が得られた．この場合，サンプリング間隔を粒子が単独で表示される最小影響範囲の半径の 1/2 以下にすることで，さらに良好な結果になった．

7. おわりに

提案手法は，視線上を等間隔に区切ったサンプル点でポテンシャルを計算する．ただし，無駄な計算を省くために，粒子の影響範囲を半径とする球面を外接する立方体を作り，視点に近い半分を対象とした．計算候補領域が不連続の場所では，Z バッファによる前後判定を使って，対象領域内にある計算点に移動することで，効率の良いサンプリングを実現した．

また，選ばれた計算点のポテンシャルを，それぞれの粒子によるポテンシャルを画像に加算合成する方法で計算した．閾値を越えた計算点が物体の中にあると判定し，直前のサンプル点との間を線形補間して，物体の表面を推定した．さらに，視線方向に並んだ 4 点を同時に計算することで．処理コストを減らした．

このように，GPU の特性を利用した実装により，メタボールの実時間表示を実現した．実験では，4,096 個の粒子を使って表現した物体を，256×256 画素のサイズで，30 fps 前後で表示することができた．

参　考　文　献

1) Muller, M., et al.: Particle-Based Fluid Simulation for Interactive Applications, *Eurographics/SIGGRAPH Symposium on Computer Animation* (2003).
2) Levoy, M.: Display of Surfaces from Volume Data, *IEEE Computer Graphics and Applications*, Vol.8, No.5 (1988).
3) Kruger, J. and Westermann, R.: Acceleration techniques for gpu-based volume rendering, *Proc. IEEE Visualization 2003*, pp. 287–292 (2003).
4) 金森由博，西田友是：GPU を用いたメタボールの高速レンダリング，情報処理学会研究報告 CG-127(3), pp.13–18 (2007).
5) Blinn, J.: A Generalization of Algebraic Surface Drawing, *ACM Trans. Graphics*, Vol.1, No.3, pp.235–256 (1982).
6) Lorensen, W. and Cline, H.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm, *Computer Graphics*, Vol.21, No.4, pp.163–169 (1987).
7) Nishita, T. and Nakamae, E.: A method for displaying metaballs by using bezier clipping, *Computer Graphics Forum* (*Proc. Eurographics '94*), Vol.13, No.3, pp.271–280 (1994).
8) 金井　崇ほか：GPU による点群ベース陰関数曲面の直接的レンダリング，グラフィクスと CAD/Visual Computing 合同シンポジウム (2006).
9) Müller, C., et al.: Image-Space GPU Metaballs for Time-Dependent Particle Data Sets, *Proc. VMV '07*, pp.31–40 (2007).
10) Brodlie, K. and Wood, J.: Recent advances in volume visualization, *Computer Graphics Forum*, Vol.20, No.2, pp.125–148 (2001).
11) Nguyen, G. and Saupe, D.: Rapid high quality compression of volume data for visualization, *Proc. Eurographics 2001*, pp.C49–C56 (2001).
12) Binotto, F., et al.: Real-Time Volume Rendering of Time-Varying Data Using a Fragment-Shader Compression Approach, *Proc. IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003).
13) Sherbondy, A., et al.: Fast volume segmentation with simultaneous visualization using programmable graphics hardware, *Proc. IEEE Visualization*, pp.171–176 (2003).
14) Engel, K., et al.: High-quality pre-integrated volume rendering using hardware-

© 2008 Information Processing Society of Japan

accelerated pixel shading, *Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp.9–16 (2001).

15) Müller, C., et al.: Optimized Volume Raycasting for Graphics-Hardware-based Cluster Systems, *Proc. Eurographics Symposium on Parallel Graphics and Visualization* (2006).

(平成 20 年 3 月 26 日受付)
(平成 20 年 9 月 10 日採録)

三嶋　　仁（正会員）

2006 年名城大学理工学部情報科学科卒業，2008 年名城大学大学院理工学研究科情報科学専攻前期課程修了．株式会社カプコンに勤務．

田中　敏光（正会員）

1984 年名古屋大学大学院情報工学専攻博士（前期）修了．NTT に勤務．1994 年名古屋大学大型計算機センター助教授．2000 年 4 月より，名城大学理工学部情報工学科教授．CG の研究・教育に従事．工学博士．

佐川　雄二（正会員）

1992 年名古屋大学大学院情報工学専攻後期課程単位取得退学．同年同大学助手，同講師を経て，2000 年名城大学理工学部講師，2008 年同教授．現在に至る．自然言語処理の研究・教育に従事．工学博士．

© 2008 Information Processing Society of Japan