6Y-9

自由表面を伴う流体シミュレーションの GPU による高速化

松田 靖広 †　　土橋 宜典 ‡　　西田 友是 †

† 東京大学 大学院情報理工学系研究科　‡ 北海道大学 大学院情報科学研究科

1　はじめに

コンピュータグラフィクスにおいて流体のリアルな動きを生成するために数値流体解析に基づく手法が広く用いられている。自由表面を伴う流体シミュレーションにおいては、自由表面の動きを正確に追跡することが重要となるが、精度の高い表面追跡手法としてパーティクルレベルセット法 [1, 2] が知られている。しかしパーティクルレベルセット法はその高い精度の代償として、非常に計算コストが高いという欠点がある。そこで本稿ではパーティクルレベルセット法を GPU(Graphics Processing Unit) を利用することにより高速に実行する手法を提案する。流体シミュレーションなど様々な手法を GPU で計算を行うことで高速化する研究は以前にもあったが、パーティクルレベルセット法のような格子と粒子のハイブリッドな手法を GPU で行った例はなかった。なぜならハイブリッドな手法では格子と粒子の間でデータのやりとりが必要となるが、そのような処理を GPU で効率的に実現するのは難しかったためである。提案法ではこの問題を render to vertex array、z-cull といわれる GPU の機能を用いることで解決する。提案手法により精度の高い流体シミュレーションを高速に行うことができることを示す。

2　関連研究

Stam は流体の動きの安定的な解法としてセミラグランジアン法を用いた手法 [6] を提案した。自由表面を伴う流体シミュレーションでは自由表面を追跡する手法が必要となるが、Enright らは精度の高い表面追跡手法としてパーティクルレベルセット法 [1, 2] を提案した。近年ではシミュレーションを GPU を利用して高速に行う研究が多数なされており、Lefonhn らはレベルセット法を GPU を用いて高速に実行する手法 [5] を提案した。また Harris らは GPU を用いることで雲や流体の動きを高速に計算する手法 [3, 4] を提案した。

3　パーティクルレベルセット法

本節ではパーティクルレベルセット法の概要を述べる。詳細については [1, 2] を参考にされたい。

レベルセット関数 ϕ の値は自由表面からの符号付距離として定義される。自由表面はこのレベルセット関数の値が 0 の部分として表現する。この値を移流方程式 $\phi_t + \mathbf{u}\cdot\nabla\phi = 0$ のもとで更新する。ここで $\mathbf{u}$ は速度、t は時間、ϕ_t はレベルセット関数 ϕ の時間に関する偏微分を表す。また $|\nabla\phi| = 1$ という性質を常に満たすようにするために再初期化といわれる処理をおこなう。パーティクルレベルセット法では表面付近に配置したパーティクルを独立に更新し、パーティクルの情報でレベルセット関数値を補正する。パーティクルの位置は $\frac{d\mathbf{x}_p}{dt} = \mathbf{u}(\mathbf{x}_p)$ に基づいてルンゲクッタ法で計算して更新する ($\mathbf{x}_p$ は各パーティクルの位置)。またパーティクルは定期的に表面付近に追加する。パーティクルによるレベルセット関数値の補正処理は以下のように行う。まず任意の位置 $\mathbf{x}$ に対して、パーティクルに付随するレベルセット関数の値を $\phi_p(\mathbf{x}) = s_p(r_p - |\mathbf{x} - \mathbf{x}_p|)$ で定義する。r_p はパーティクルの半径、s_p はパーティクルの符号でありパーティクル生成時の位置におけるレベルセット関数の値の符号として定義される。レベルセット関数の値の符号が負の領域に飛び出した正の符号を持ったパーティクルの集合を E^+、正の領域に飛び出した負の符号を持ったパーティクルの集合を E^- とする。これら E^+, E^- による補正値を保持する ϕ^+, ϕ^- を ϕ の値で初期化して、パーティクルの近傍の格子点において以下の式に基づいて計算する。

$$\phi^+ = \max_{\forall p\in E^+}(\phi_p, \phi^+)\ ,\ \phi^- = \min_{\forall p\in E^-}(\phi_p, \phi^-) \tag{1}$$

得られた ϕ^+, ϕ^- を用いて ϕ の値を以下のように更新する。

$$\phi = \begin{cases} \phi^+ & |\phi^+| \le |\phi^-| \text{ の場合} \\ \phi^- & |\phi^+| > |\phi^-| \text{ の場合} \end{cases} \tag{2}$$

4　流体の速度場の計算

流体の速度場については下式で定義される Navier-Stokes 方程式に基づいて計算する。

$$\nabla\cdot\mathbf{u} = 0, \tag{3}$$

$$\frac{\partial\mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \nu\nabla^2\mathbf{u} + \mathbf{f}, \tag{4}$$

t は時間、p は圧力、ρ は密度、ν は動粘性係数、$\mathbf{f}$ は外力である。Navier-Stokes 方程式の解法としては Stam の手法 [6]、その GPU 化に関しては Harris の手法 [3] を用いた。

5　GPU による実装

提案法の GPU による具体的な実装について述べる。

まず図 1 で示すように、CPU における配列と GPU におけるテクスチャの各ピクセルを一対一に対応させる。このように対応させることで CPU における各要素についての計算を、GPU ではフラグメントプログラムでの各ピクセルの計算処理として実行することができる。

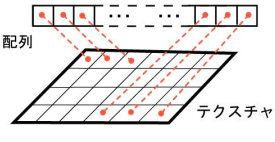

図 1　配列とテクスチャの対応

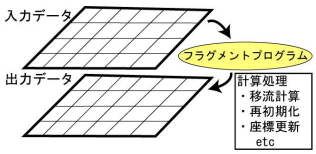

図 2　GPU を用いた計算の流れ

シミュレーションの各時刻での値を保持するためにレベルセット値、速度場、パーティクルの座標と半径を保存する浮動小数点テクスチャを用意する。現在の GPU ではテクスチャは読み書きを同時に行うことができないが、レベルセット値とパーティクル値は読み書きを同時に行う必要がある。そこでこれらのテクスチャはそれぞれ 2 枚用意しダブルバッファとして読み込み書き込みを交互に切り替えて使用する。

レベルセット関数の移流計算、再初期化、パーティクル座標、半径の更新処理は各要素を独立に更新できるため CPU の各配列の要素を更新するのと同様に、図 2 のような流れでフラグメントプログラムで実現できる。レベルセット関数値の

GPU での処理については [5] が詳しい。パーティクルによるレベルセット関数値の補正を表す式 (1) および式 (2) のうち、式 (1) に相当する処理は簡単には実現できない処理であるが、本稿では GPU の render to vertex array と z-cull という機能を組み合わせてこの処理を可能とした。render to vertex array とは vertex array と同等の機能を提供する GPU の Pixel Buffer Object に値を書き込む機能のことで Open GL では ARB_pixel_buffer_object 拡張を用いることで実現できる。z-cull とは各ピクセル値に depth 値といわれる値の大小でバッファへの書き込みを制御できる機能である。これらの機能を用いて次のように補正処理を実現する。以下では ϕ^+ を作成するものとする。まずパーティクル座標のデータを Pixel Buffer Object に送る。図 3 で示すように、ϕ の値を depth 値に変換する単調増加関数 $f : \mathbf{R} \to [0, 1]$ (本稿では線形関数を用いた) を定義し、ϕ の値とその値を f で変換した depth 値を持つ ϕ^+ のバッファを用意する。Pixel Buffer Object のデータを vertex array として E^+ のパーティクルの近傍グリッドに描画し、パーティクルに付随するレベルセット関数値を更新値とし、f で変換した depth 値で書き込む。depth テストにより最終的に式 (1) で示される値のみ残ることとなる。処理の概要を図 4 で示した。

この処理は GPU においてテクスチャの特定の部分を更新する手法として以下のように一般化できる。

- depth テストを有効にし depth 値の大きいもののみ書き込むように設定する
- 更新値として残したい値を大きくする関数 $f : \mathbf{R} \to [0, 1]$ を定義する
- 更新候補の位置を格納したテクスチャ $\mathbf{T}$ を用意する
- Pixel Buffer Object に $\mathbf{T}$ のデータを転送する
- 更新先バッファの depth 値を f で計算し、設定する
- Pixel Buffer Object に格納されたデータを vertex array として描画する
- フラグメントプログラムにおいて更新値と f で変換した depth 値を更新先バッファに書き込む

以上で f が最も高くなるように部分的に要素を更新することができる。

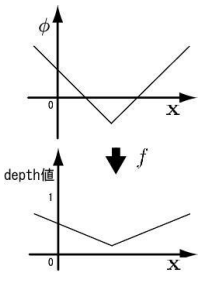

図 3　ϕ の depth 値への変換

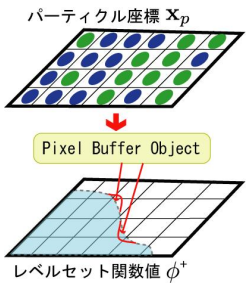

図 4　ϕ^+ を作成する概要図

6　結果

本稿で提案した手法を実装した結果を示す。図 5 は 2 次元、図 6 は 3 次元の流体シミュレーションの結果である。表 1 は 2 次元流体シミュレーションでの CPU と GPU の実装 (提案法) の計算速度の比較である。ただし本稿ではパーティクルの追加処理と 3 次元でのパーティクルでの補正処理、および流体の速度場計算は CPU 実装とした。

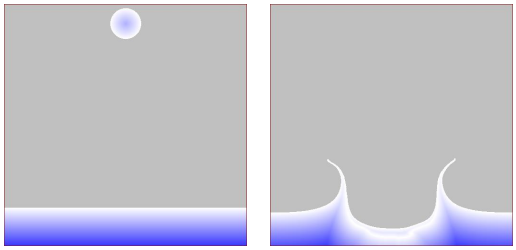

図 5　水滴が水面に落下して衝突する様子 (グリッド数:512^2)

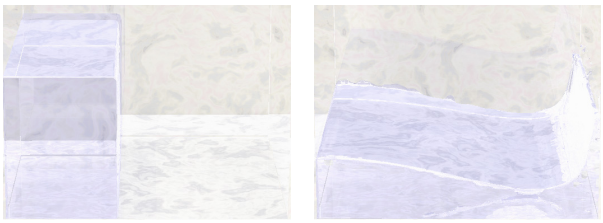

図 6　水槽の水柱 (左図) が崩壊して壁にあたる様子 (レンダリングにはフリーソフト POV-Ray を使用)

表 1　GPU(提案法),CPU それぞれの実装による 2 次元流体シミュレーションのフレームレート (fps) の比較 (N:グリッド数、実行環境:Intel Xeon 3.8G, 2.0G RAM, NVIDIA GeForce7950 GX2)

	$N = 128^2$	$N = 256^2$	$N = 512^2$
CPU	23.4	5.83	1.00
GPU	98.2	53.0	12.3

7　まとめと今後の課題

本稿では GPU を用いてパーティクルレベルセット法を実装することで流体シミュレーションを行う手法を提案し、高速に表面追跡を行えることを確認した。

今後の課題としては 3 次元でのパーティクルの補正処理の GPU 実装および、頂点を GPU で生成することができる GPU の新しい機能であるジオメトリシェーダを用いて、パーティクルの追加処理を GPU で行うことが考えられる。

参考文献

[1] Douglas Enright, Frank Losasso, and Ronald Fedkiw. A fast and accurate semi-lagrangian particle level set method. *Computers and Structures*, 83:479–490, 2005.

[2] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. Animation and rendering of complex water surfaces. In *ACM SIGGRAPH*, pages 736–744, 2002.

[3] Mark J. Harris. Fast fluid dynamics simulation on the gpu. In *GPU Gems*, pages 637–665. Addison Wesley Pub., 2004.

[4] Mark J. Harris, William V. Baxter, Thorsten Scheuermann, and Anselmo Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 92–101, 2003.

[5] Aaron E. Lefohn, Joe M. Kniss, Charles D. Hansen, and Ross T. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proceedings of the 14th IEEE Visualization*, page 11, 2003.

[6] Jos Stam. Stable fluids. In *ACM SIGGRAPH*, pages 121–128, 1999.