

コンピュータグラフィックス論

－アニメーション(3)－

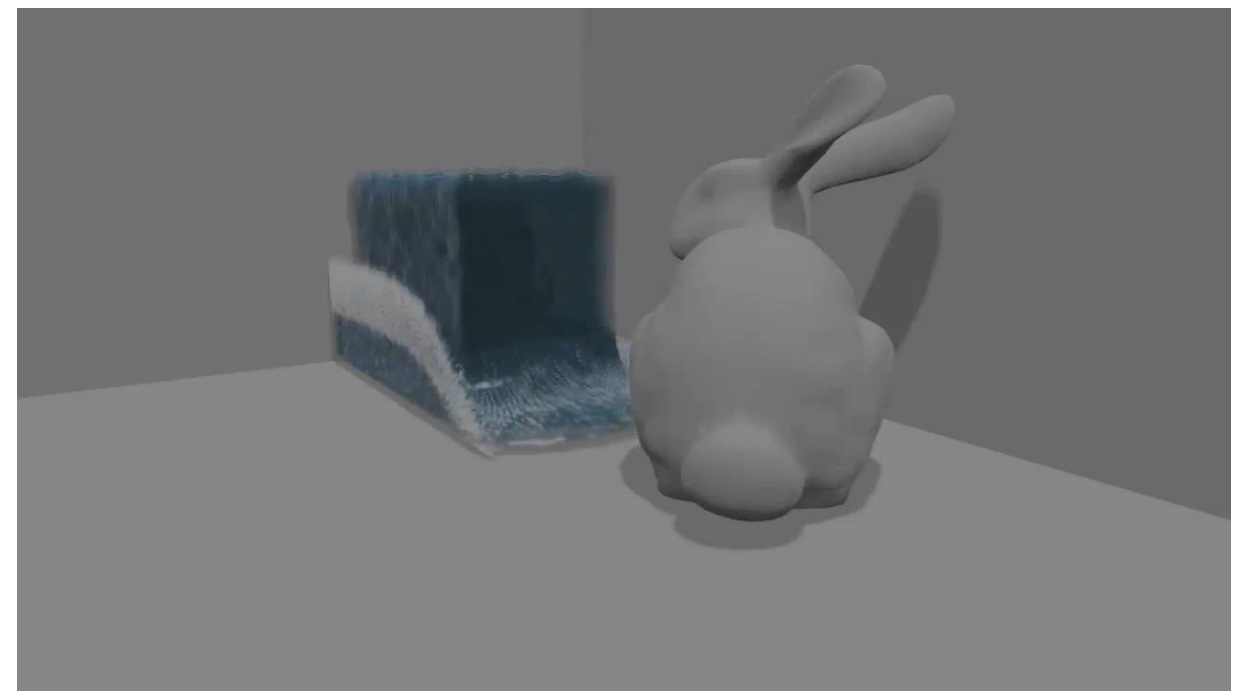
2018年6月7日

高山 健志

流体アニメーション



<https://www.youtube.com/watch?v=KoEbwZq2ErU>



<https://www.youtube.com/watch?v=6WZZARzpckw>

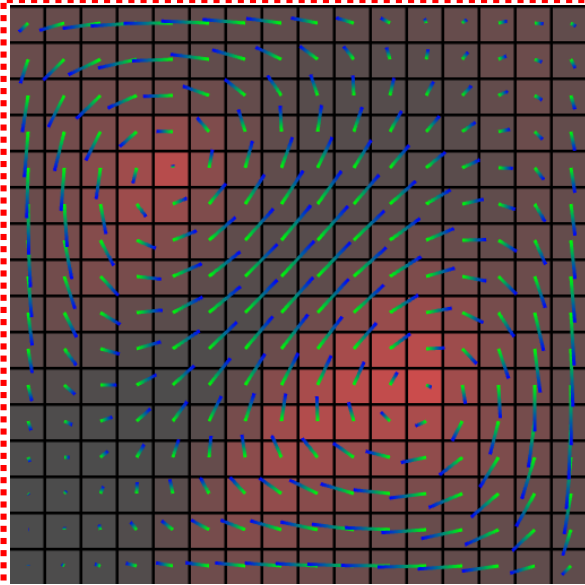


Our simulated oil paint

In this paper, we present a new algorithm to address this

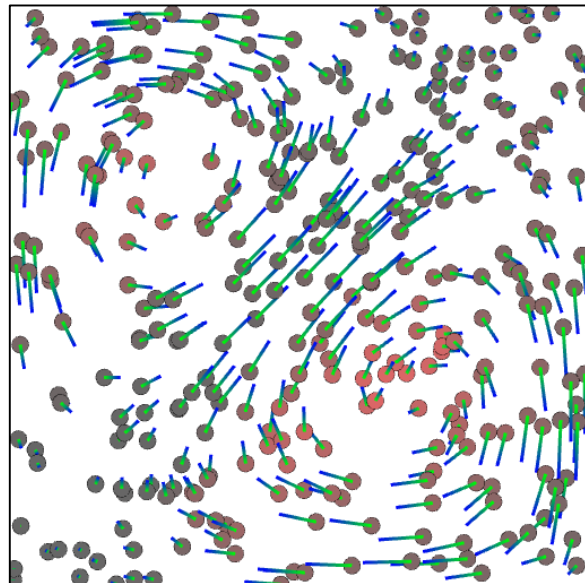
<https://www.youtube.com/watch?v=WFWi0qLV8hQ>

二つの異なるアプローチ



Eulerian

- 格子上的セルに速度とその他情報を保存
 - e.g. 煙の密度、温度
- 場の勾配等を計算しやすい → 流体計算の王道
- オフライン用途に適する



Lagrangian

- パーティクルに情報を持たせ、速度に従って動かす
- 場の勾配等の計算に工夫が必要 → ハック (?)
- リアルタイム用途に適する

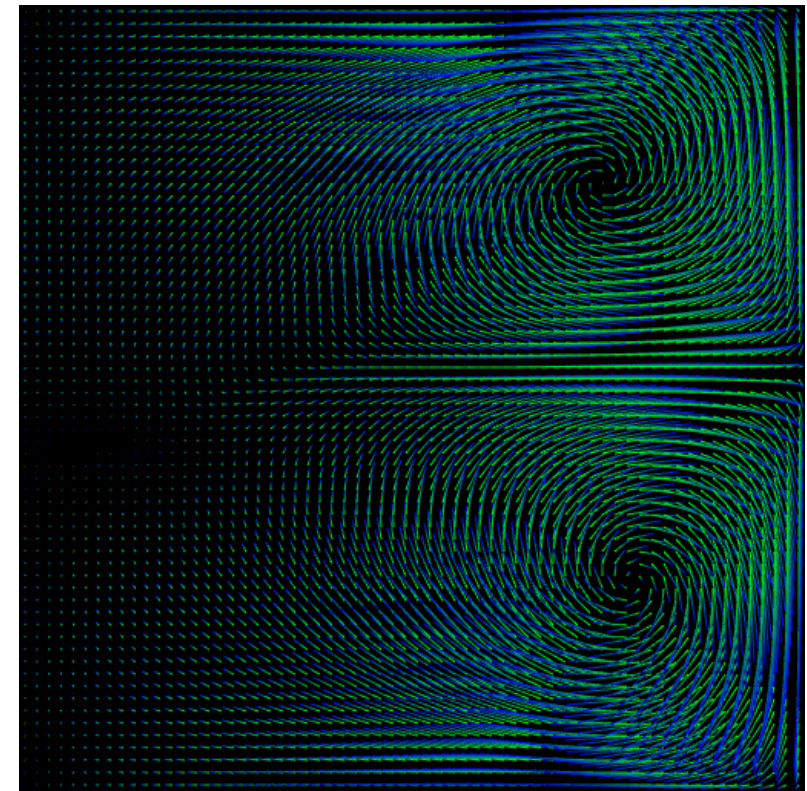
Stable Fluids [Stam, SIGGRAPH 99]

- 時間幅によらず無条件に安定 → ゲーム向き
- 超簡潔なサンプルコード
 - 500行未満、外部ライブラリ不使用
 - http://www.dgp.toronto.edu/people/stam/research/arch/zip/CDROM_GDC03.zip
- ゲーム開発者向けの易しい解説記事
 - Real-Time Fluid Dynamics for Games (GDC 2003)
- 目標：これを理解できるようにする

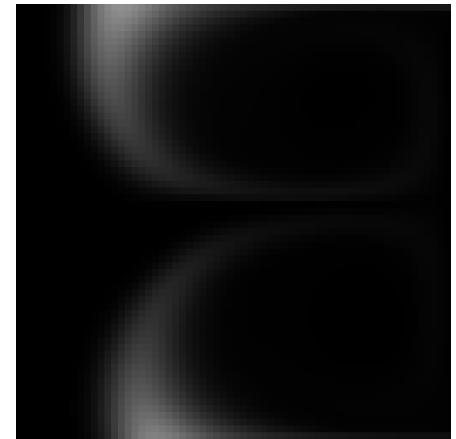
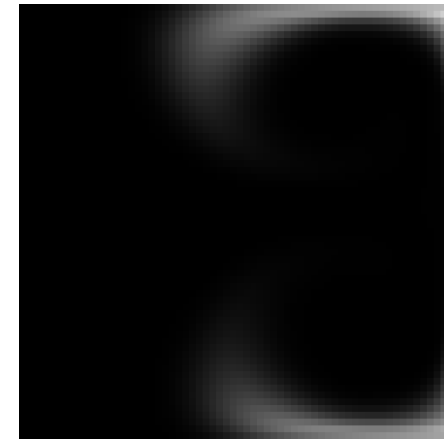
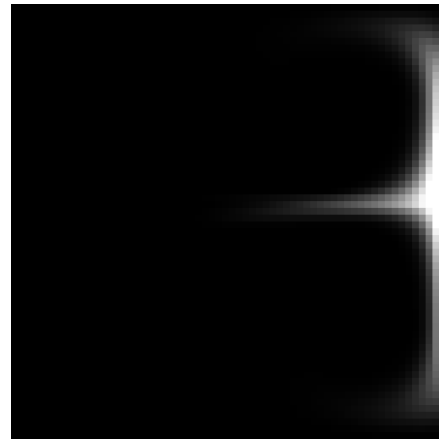
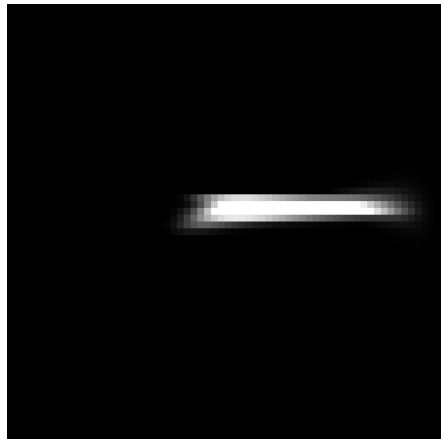
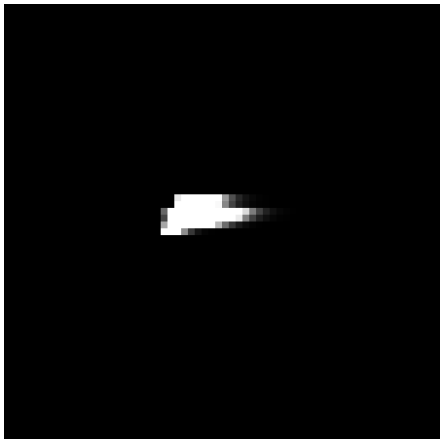


定常的な速度場に沿った物理量の移流

- 物理量：温度、煙の密度、etc
- 方法：
 - 陽的な方法 → 不安定
 - Semi-Lagrangian 法 → 安定



速度場



陽的な方法 [Foster 96]

- Given: 2D 領域上の (定常) 速度場 $\mathbf{u}: \mathbb{R}^2 \mapsto \mathbb{R}^2$
- ある物理量 q の時刻 t における分布を $q_n: \mathbb{R}^2 \mapsto \mathbb{R}$ とする
- 時刻 $t + h$ における分布 q_{n+1} を陽的に求める：

$$q_{n+1}(\mathbf{x}) = q_n(\mathbf{x}) - h \mathbf{u}(\mathbf{x}) \cdot \nabla q_n(\mathbf{x})$$

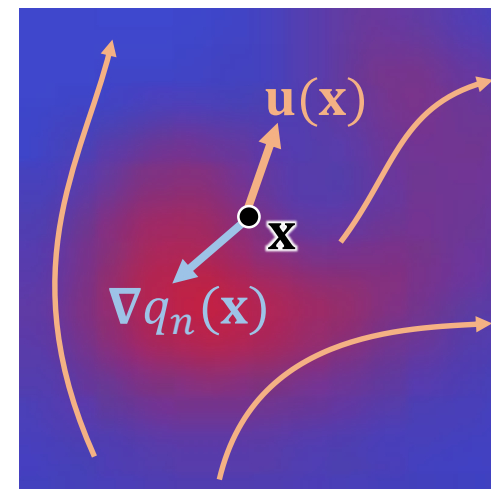
- $(\mathbf{u}(\mathbf{x}) \cdot \nabla)$ を微分演算子と見なした書き方：

$$q_{n+1}(\mathbf{x}) = q_n(\mathbf{x}) - h (\mathbf{u}(\mathbf{x}) \cdot \nabla) q_n(\mathbf{x})$$

- (後でまた出てくる)

- 問題点：数値的に不安定 ☹

- 変化量が時間幅 h に比例 $\rightarrow h$ を大きくしすぎると、物理量の総和が元よりも大きくなる



風上への微小な移動

物理量の勾配

$(\mathbf{u}(\mathbf{x}) \cdot \nabla)$ の直感的な意味：

速度場 \mathbf{u} の風が吹く中で、右に掛かる物理量が微小時間にどれだけ変化するか？

Semi-Lagrangian 法 [Stam 99]

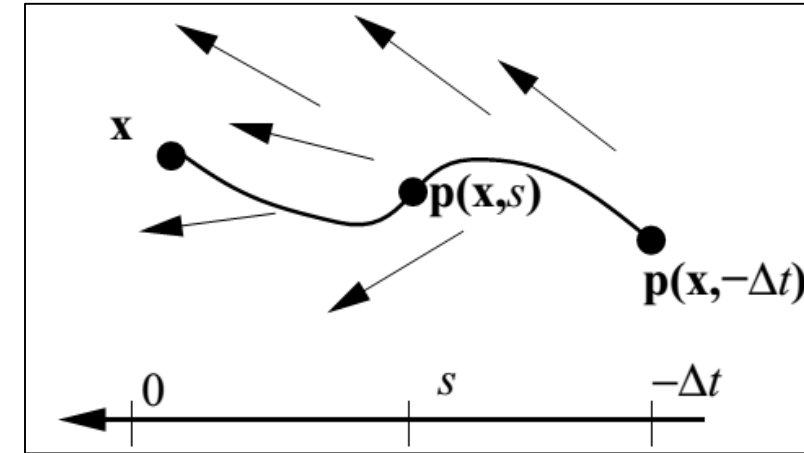
- 時刻 $t - h$ において位置 $\tilde{\mathbf{x}}$ にあったパーティクルが、時刻 t において位置 \mathbf{x} に流れ着いたとして、位置 $\tilde{\mathbf{x}}$ を現在の速度場 \mathbf{u} を逆に辿ることで推定する

$$\tilde{\mathbf{x}} = \text{trace}(\mathbf{u}, \mathbf{x}, -h)$$

- これを使って次の時刻の物理量を求める

$$q_{n+1}(\mathbf{x}) = q_n(\tilde{\mathbf{x}})$$

- パーティクル自体のデータは不要
- trace の方法：線形予測、Runge-Kutta, etc
- q_n をリサンプリングして q_{n+1} を求めるので、時間幅によらず安定！
 - 物理量の総和が元よりも大きくなることは原理的に起こり得ない



動的に変化する速度場

- 定常的な速度場によって移流するスカラー場：

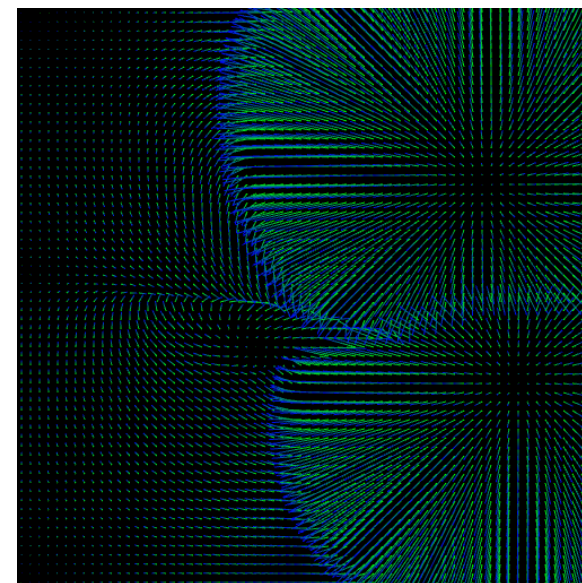
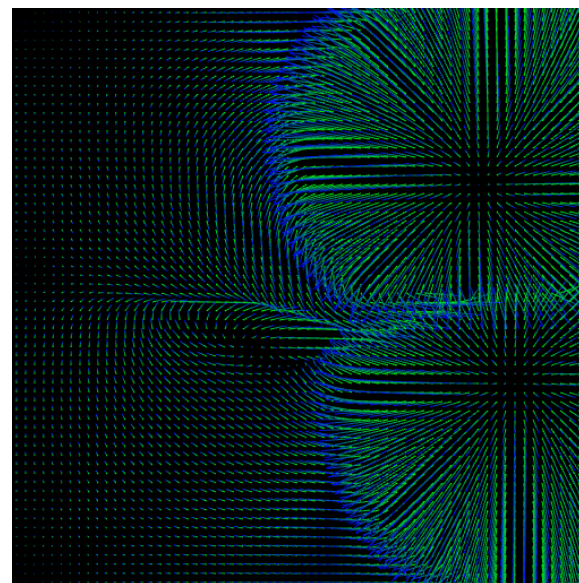
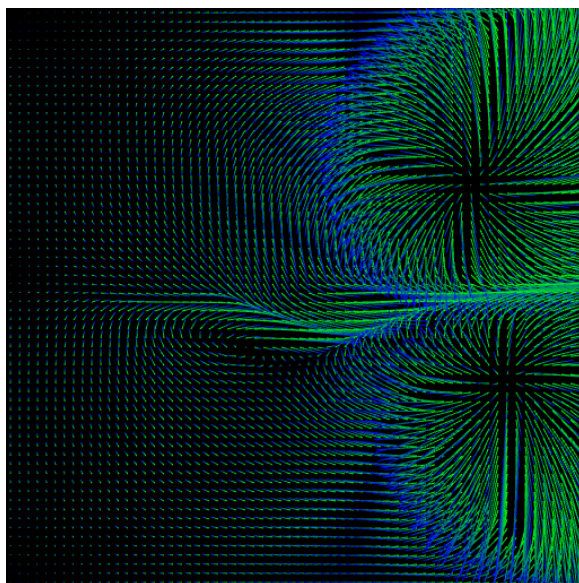
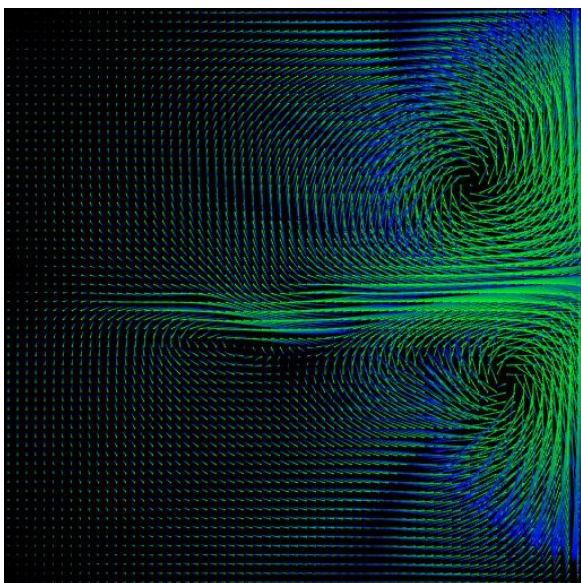
$$q_{n+1}(\mathbf{x}) = q_n(\mathbf{x}) - h (\mathbf{u}(\mathbf{x}) \cdot \nabla) q_n(\mathbf{x})$$

- それ自身によって移流し、動的に変化する速度場：

$$\mathbf{u}_{n+1}(\mathbf{x}) = \mathbf{u}_n(\mathbf{x}) - h (\mathbf{u}_n(\mathbf{x}) \cdot \nabla) \mathbf{u}_n(\mathbf{x})$$

- そのままでは全然流体っぽくならない！

もっと渦を巻くべき！



流体らしさのための必須条件：非圧縮性

$$\nabla \cdot \mathbf{u}(\mathbf{x}) = 0 \quad \forall \mathbf{x}$$

(以降簡単のため、位置 \mathbf{x} を適宜省略する)

微分演算子 $(\mathbf{u} \cdot \nabla)$ とは
意味が違うことに注意

- 至る所で発散がゼロ (divergence-free)
 - 各局所領域について、外部からの流入量と外部への流出量の合計がぴったり一致する
 - 視覚的には、渦を巻く現象として現れる
- 移流後のベクトル場 \mathbf{w} は、一般に非圧縮性条件を満たさない！
- ヘルムホルツの定理
 - 任意のベクトル場は、divergence-free なベクトル場とスカラー場の勾配の和に分解できる：

$$\mathbf{w} = \mathbf{u} + \nabla q$$

- 流体計算のアルゴリズム：条件 $\nabla \cdot \mathbf{u} = 0$ から q を求め、そこから \mathbf{u} を求める

ラプラシアン演算子

$$\Delta = \nabla \cdot \nabla$$

$$\nabla \cdot \mathbf{u} = 0 \quad \Leftrightarrow \quad \nabla \cdot (\mathbf{w} - \nabla q) = 0 \quad \Leftrightarrow \quad \Delta q = \nabla \cdot \mathbf{w} \quad \text{ポアソン方程式}$$

ポアソン (Poisson) 方程式

- 一般形：

$$\Delta q = f$$

- CG に限らず、様々な工学・自然科学分野で頻繁に登場

- 特別な場合：

$$\Delta q = \nabla \cdot \mathbf{w}$$

- ベクトル場 \mathbf{w} は guiding vector field と呼ばれることもある
- スカラー場 q は以下のエネルギーを最小化 \rightarrow projection と呼ばれる

$$E(q) = \int_{\Omega} \|\mathbf{w} - \nabla q\|^2$$

直感的な意味：

あらゆるスカラー場のうち、勾配がベクトル場 \mathbf{w} に最も近いものを求める

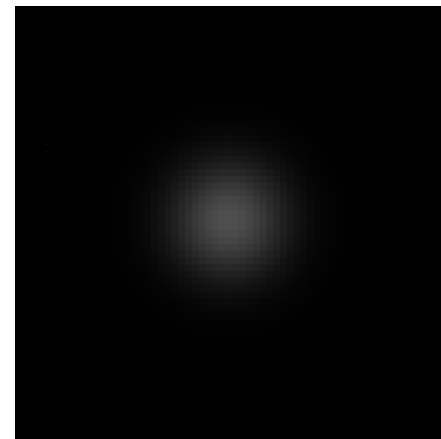
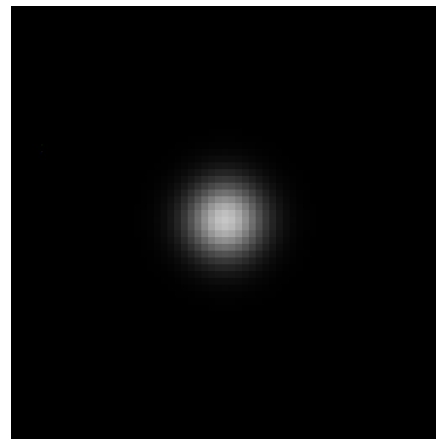
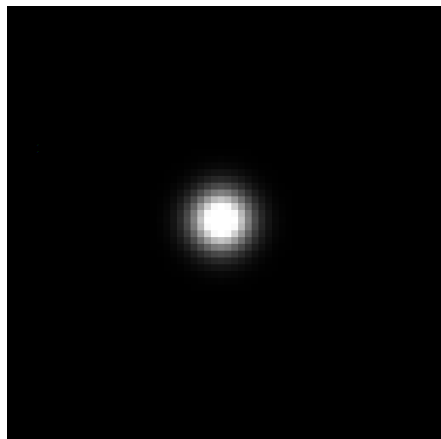
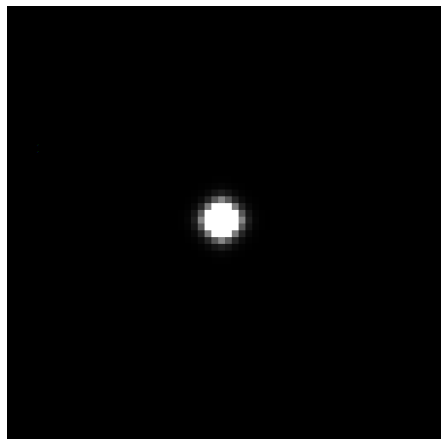
- 大規模疎行列で表される方程式

- よくある解法：

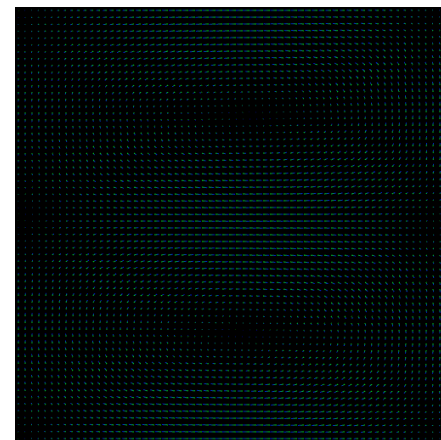
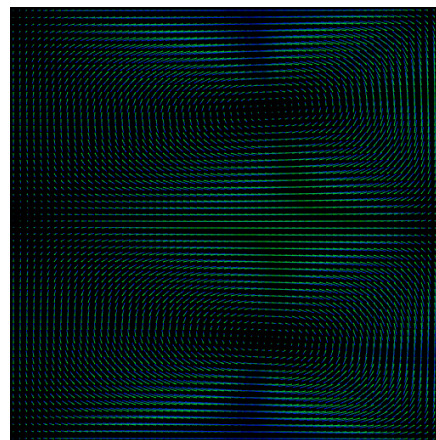
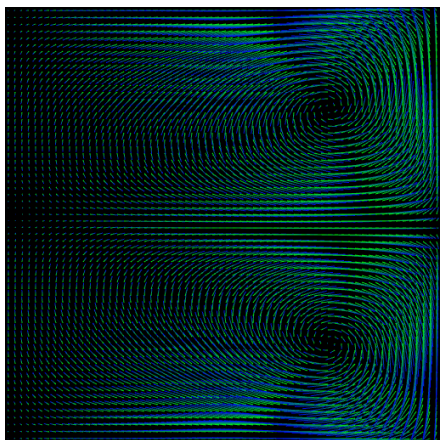
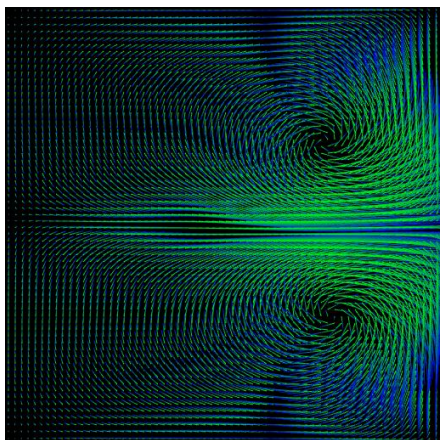
- Gauss-Seidel \rightarrow 実装が簡単、遅い (サンプルコードで採用)
- (Preconditioned) Conjugate Gradient \rightarrow 速い
- Multigrid \rightarrow かなり速い、実装が大変 (?)

拡散

- 分布がより滑らかになる効果



- 速度場に対して適用すると、粘性を表現できる



拡散方程式

$$\frac{\partial q}{\partial t} = \nu \Delta q$$

ν : 係数

- 直感的な意味：
 - ラプラシアン演算子 Δ は、(周囲の値の平均 - 中心の値) を表す
 - 時間の経過とともに、スカラー場の凸凹がならされていく

- 陽的な解法

$$q_{n+1}(\mathbf{x}) = q_n(\mathbf{x}) + h \nu \Delta q_n(\mathbf{x})$$

- 変化量が時間幅 h に比例 → 不安定

- 陰的な解法

$$q_n(\mathbf{x}) = q_{n+1}(\mathbf{x}) - h \nu \Delta q_{n+1}(\mathbf{x})$$

- 時間幅 h によらず安定
 - 大規模疎行列で表される方程式 (Poisson 方程式と同様)

非圧縮 Navier-Stokes 方程式

ρ : 係数

$$\frac{\partial \mathbf{u}}{\partial t} = \underbrace{-(\mathbf{u} \cdot \nabla) \mathbf{u}}_{\text{移流}} - \underbrace{\frac{1}{\rho} \nabla p}_{\text{圧力}} + \underbrace{\nu \Delta \mathbf{u}}_{\text{粘性}} + \underbrace{\mathbf{f}}_{\text{外力}} \quad \text{s.t.} \quad \nabla \cdot \mathbf{u} = 0$$

- x 成分 :

$$\frac{\partial u_x}{\partial t} = - \left(u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} \right) - \frac{1}{\rho} \frac{\partial p}{\partial x} + \nu \left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} \right) + f_x$$

- projection で求めたスカラー場 $q(\mathbf{x}) = p(\mathbf{x})/\rho$ は、圧力に相当
 - 圧力の高い所から低い所へ向かって加速度が発生

シミュレーションの流れ

- 速度場の更新 (vel_step)

- 外力の加算
- 拡散
- project
- 移流
- project

速度場 $\mathbf{u}(\mathbf{x})$ の方程式

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla) \mathbf{u} - \frac{1}{\rho} \nabla p + \nu \Delta \mathbf{u} + \mathbf{f}$$

- 煙の密度場の更新 (dens_step)

- 外部ソースの加算
- 拡散
- 移流

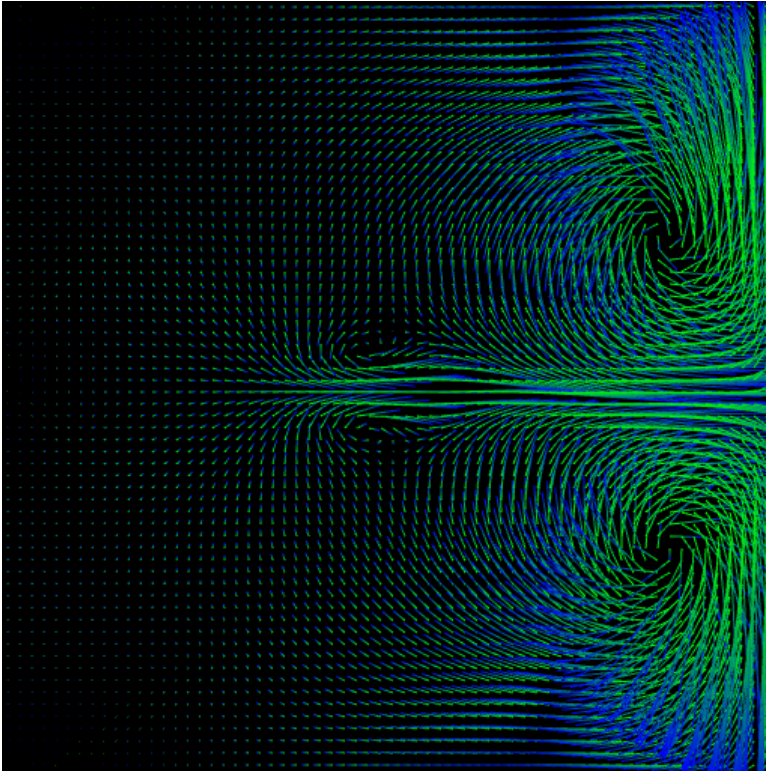
煙の密度場 $d(\mathbf{x})$ の方程式

$$\frac{\partial d}{\partial t} = -(\mathbf{u} \cdot \nabla) d + \nu \Delta d + s$$

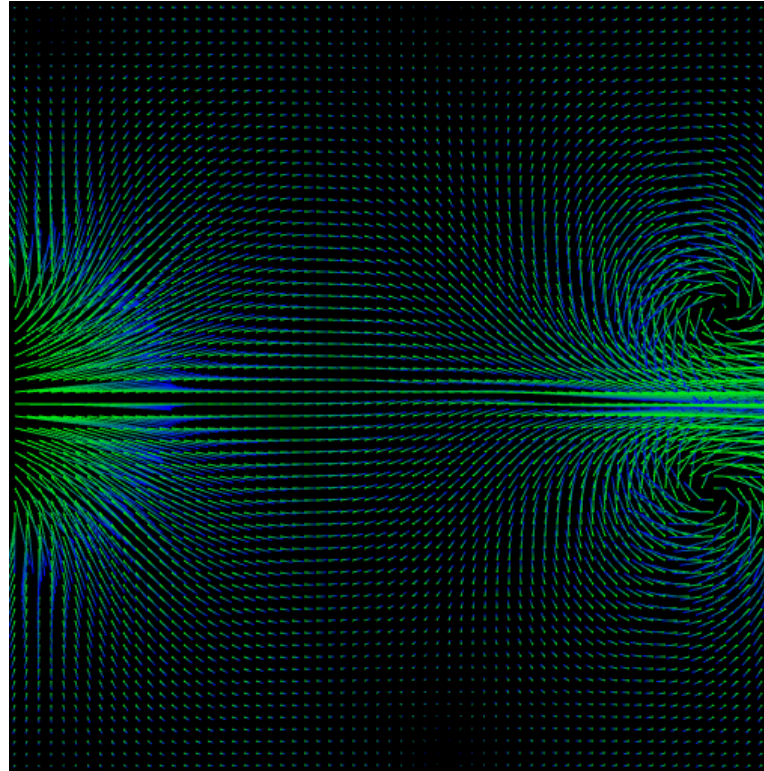
移流を行う前に project しておくこと！

境界条件の設定

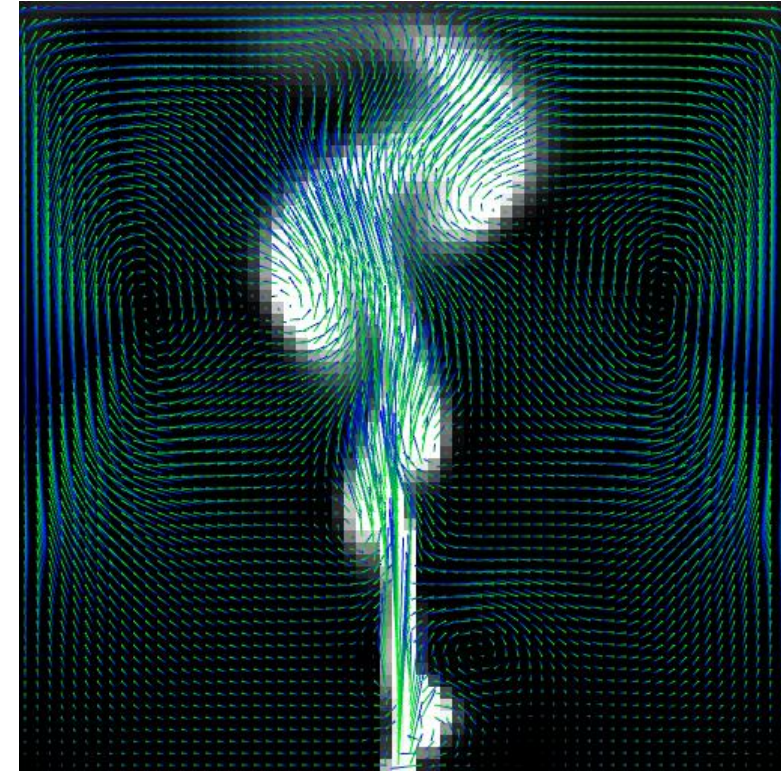
速度の壁方向の成分をゼロにする



左右と上下の壁を連続させる



一定の値を与え続ける




- より複雑なケースを扱うためには、高度な技術が必要
 - 丸みを帯びた形状、格子幅よりも薄いシート、etc

発展的な話題

レベルセット法による水面の表現

- 水面までの符号付き距離場 $\phi(\mathbf{x})$ を導入
 - $\phi(\mathbf{x}) < 0$ なら液体、 $\phi(\mathbf{x}) > 0$ なら空気
 - 初期状態を適当に与える
- 速度場に従って $\phi(\mathbf{x})$ を移流
- 圧力計算の際、水面 $\phi(\mathbf{x}) = 0$ において $p(\mathbf{x}) = 0$ という境界条件を設定

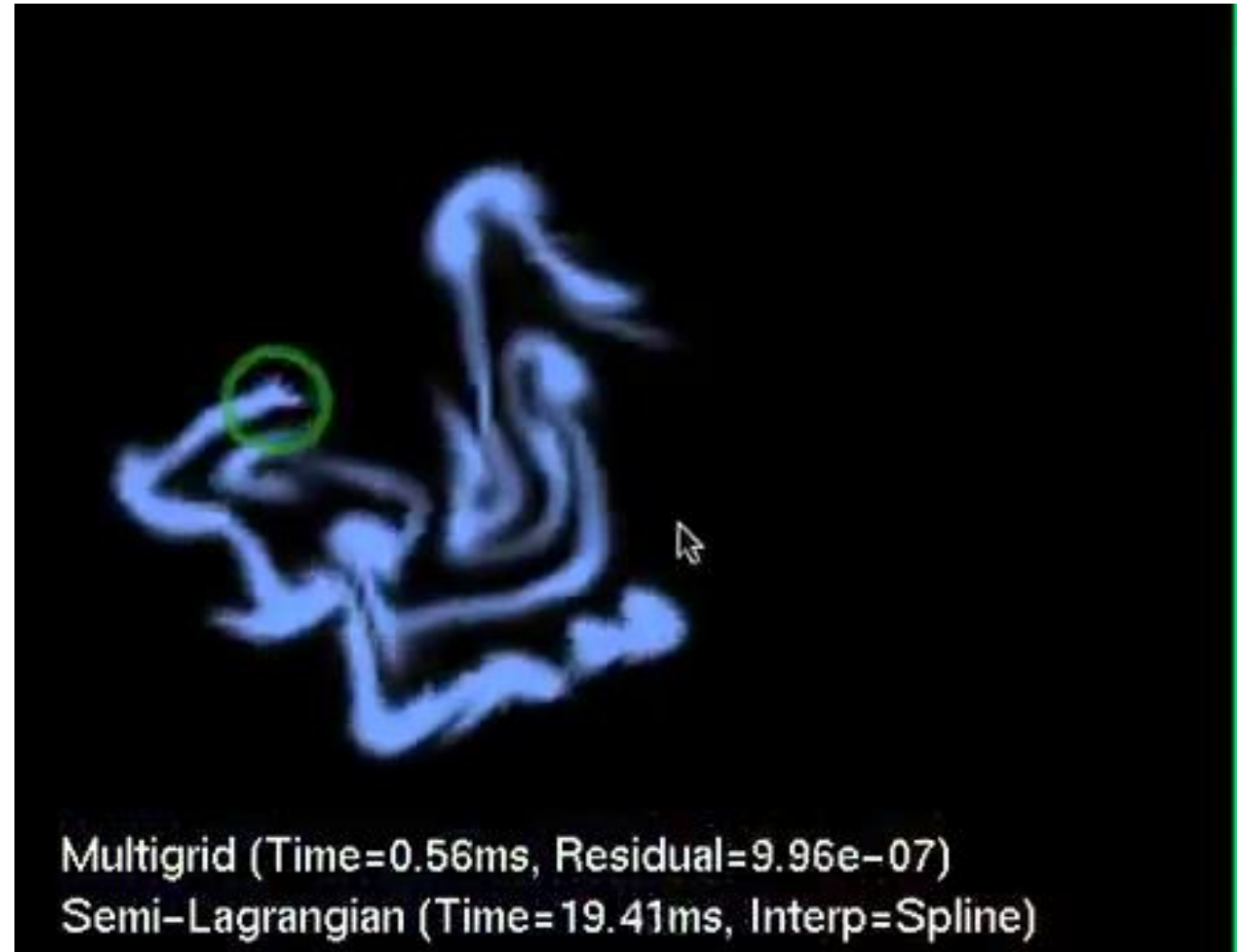


```
Push "p" to change the accuracy of boundary projection ( current: 2nd order.)
Push "d" to toggle distance field.
Push "f" to toggle liquid field.
Push "g" to toggle grid points.
Push "v" to toggle velocity.
Push "r" to reset.
Push "i" to toggle interpolation method. ( current: Catmull-Rom Spline.)
Push "a" to toggle redistance.
Push "c" to toggle volume correction. ( current: Enabled.)
Push "s" to toggle pressure solver. ( current: MICCG.)
```

https://www.youtube.com/watch?v=Ss89OpQ_u54
<http://code.google.com/p/levelset2d/>

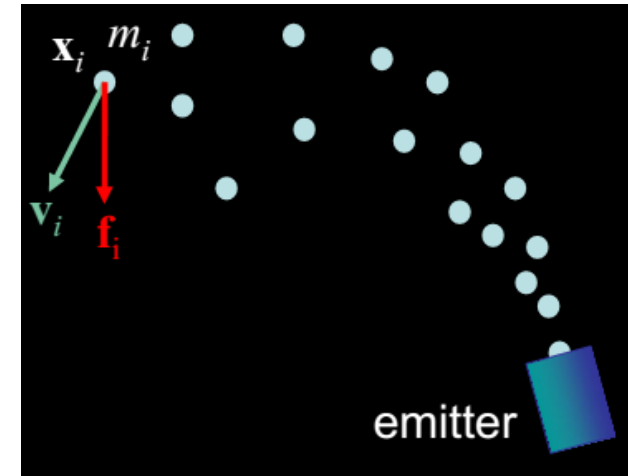
いろいろな移流アルゴリズム

- Semi-Lagrangian
- Upwind
- MacCormack
- WENO5
- QUICK



Smoothed Particle Hydrodynamics

- Lagrangian 法の代表格
- Navier-Stokes 方程式を、多数の粒子を使って近似



$$\frac{\partial \mathbf{u}}{\partial t} = \overbrace{-(\mathbf{u} \cdot \nabla) \mathbf{u}}^{\text{移流項}} - \underbrace{\frac{1}{\rho} \nabla p + \nu \Delta \mathbf{u}}_{\text{圧力項と粘性項}} + \mathbf{f}$$

移流項

→ 粒子自体を速度に従って動かすことで表現

圧力項と粘性項

→ 連続的な圧力場 p と速度場 \mathbf{u} を、多数の粒子の分布から推定

Smoothed Particle Hydrodynamics



<https://www.youtube.com/watch?v=M8WPINWAWPY>

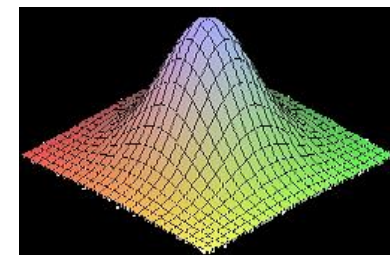
Particle-Based Fluid Simulation for Interactive Applications [Muller SCA03]

<http://matthias-mueller-fischer.ch/talks/gameFluids2007.pdf>

Smoothing kernel による連続的な場の近似

- Smoothing kernel の例 : $W(r) = \frac{315}{64\pi h^9} (h^2 - r^2)^3$

$$0 \leq r \leq h$$



- 密度場 $\rho(\mathbf{x})$ の近似 :

$$\rho(\mathbf{x}) = \sum_j m_j W(\|\mathbf{x} - \mathbf{x}_j\|)$$

- j 番目の粒子の密度 : $\rho_j = \rho(\mathbf{x}_j)$

- 速度場の近似 :

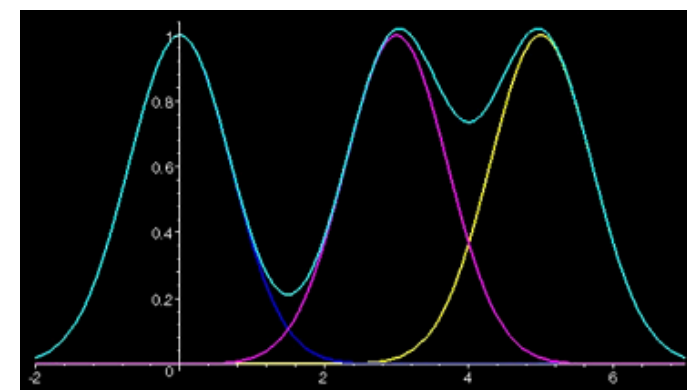
$$\mathbf{u}(\mathbf{x}) = \sum_j \frac{m_j}{\rho_j} \mathbf{u}_j W(\|\mathbf{x} - \mathbf{x}_j\|)$$

- 圧力は、密度に比例する !

$$p(\mathbf{x}) = k \rho(\mathbf{x})$$

- ポアソン方程式を解く必要が無い

- 関数の微分は、 W の形から解析的に求まる



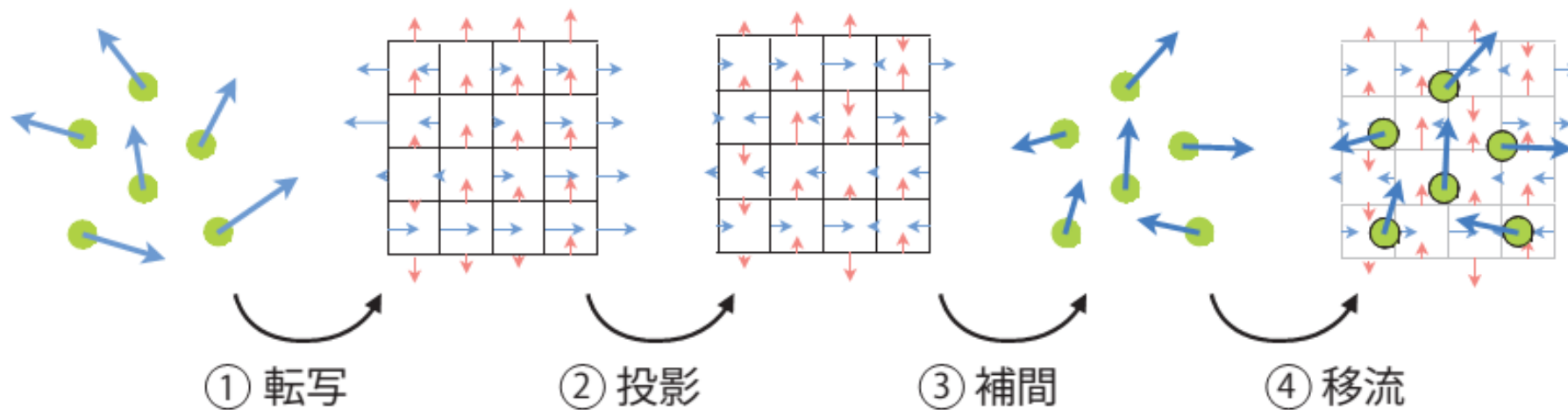
気体の状態方程式 :

$$pV = nRT$$

格子法と粒子法のハイブリッド

	格子法	粒子法
移流計算	数値拡散する ☹	数値拡散しない ☺
圧力計算	正確 ☺	不正確 ☹

- PIC (**P**article **I**n **C**ell) 法と FLIP (**F**luid **I**mplicit **P**article) 法



Height field による水面の近似

```
initialize u[i,j] as you like
```

```
set v[i,j] = 0
```

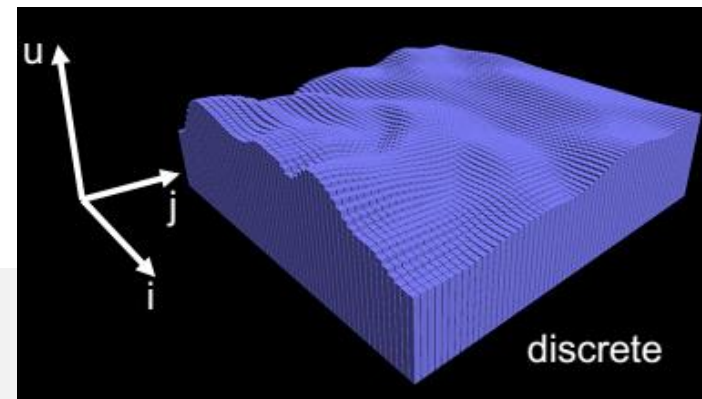
```
loop
```

```
    v[i,j] += (u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i,j+1])/4 - u[i,j]
```

```
    v[i,j] *= 0.99
```

```
    u[i,j] += v[i,j]
```

```
endloop
```



- WebGL実装

- <http://madebyevan.com/webgl-water/>
- <http://dblsai.github.io/WebGL-Fluid/>
- <http://jsdo.it/cx20/cAmU>

参考情報

- JavaScriptによる実装

- <http://www.ibiblio.org/e-notes/webgl/gpu/fluid.htm>
- <https://nerget.com/fluidSim/>
- <http://dev.miaumiau.cat/sph/>
- <http://www.miaumiau.cat/examples/SPH/v1/>
- <http://nullprogram.com/fun-liquid/webgl/>
- <http://p.brm.sk/fluid/>

- C++による実装

- <http://code.google.com/p/flip3d/>
- <http://code.google.com/p/levelset2d/>
- <http://code.google.com/p/smoke3d/>
- <http://code.google.com/p/2dsmoke/>
- http://www.cs.ubc.ca/~rbridson/download/simple_flip2d.tar.gz

- 書籍

- Fluid Simulation for Computer Graphics, by R. Bridson, 2008
- 安東遼一氏による Computer Graphics Gems JP 2012 の記事
 - Chapter 13: ベクタ形式で出力可能な美しいマープリング模様の生成法
 - Chapter 14: FLIP法による格子&粒子のハイブリッド流体シミュレーション
 - 付録コード：<http://book.borndigital.jp/support/CGGems2012/CGGems2012.zip>