Vol.2012-CG-149 No.8
Vol.2012-CVIM-184 No.8
2012/12/3

ラプラシアン固有関数を用いた速度場の補間による流体アニメーションの生成

佐藤 周平[1,a)]　土橋 宜典[1,2,b)]　山本 強[1,c)]

概要：近年，CG 技術の発達により写実的な流体映像が映画やゲームなどに多く用いられている．これら流体映像は，流体シミュレーションを用いて作成される場合が多いが，既存のシミュレーションにおけるパラメータの調整のみでは，実現が難しいアニメーションが要求される場合もある．そこで，本研究では，シミュレーション空間を任意に伸縮させる手法を提案する．提案法では，任意に伸縮させたシミュレーション空間における速度場をラプラシアン固有関数により近似することで，流体の法則を保った流体アニメーションの伸縮を可能とする．

キーワード：流体シミュレーション，ラプラシアン固有関数，シミュレーション空間の伸縮

A Method of Interpolating Velocity Field using Laplacian Eigenfunctions for Fluid Simulation

Syuhei Sato[1,a)]　Yoshinori Dobashi[1,2,b)]　Tsuyoshi Yamamoto[1,c)]

Abstract: Recently, realistic fluid animations are often used in movies or TV games. These fluid animations are usually generated by using fluid simulation. However, it is difficult to generate desired animations by only adjusting parameters used in the fluid simulation. To address this problem, this paper proposes a method for extending and retracting a simulation space of fluid. Our method uses laplacian eigen functions to generate velocity fields that satisfy the fluid equations in the deformed simulation space.

Keywords: fluid simulation, laplacian eigenfunctions, simulation space extending and retracting

1. はじめに

近年，コンピュータグラフィックス (CG) として表現された煙や水などの写実的な流体アニメーションが，映画やゲームなどに多く用いられている．流体現象は，屋内外問わずシーンを構成する重要な要素の 1 つであり，様々な場面で利用されている．このような流体現象は，物理シミュレーションに基づいた方法により写実的な表現が可能であり，煙や水，炎などをシミュレーションするための手法が数多く提案されている [1][4][6][7]．しかし，流体解析を利用したシミュレーションは計算コストが非常に高い点が問題となる．また，所望の映像を作成するためには，一般に，数多くのパラメータを試行錯誤的に決定するといった煩雑な作業を必要とする．そのため，所望の映像を得るまでには，高コストのシミュレーションを繰り返し実行しなければならず，膨大な時間がかかってしまう．

上記の問題を解決するために，流体シミュレーションを制御することでユーザの所望するアニメーションを生成するための研究が行われている [2][3][8]．これらの手法には，外力などを用いてシミュレーションを直接制御することで，所望の形状の流体を得るものやシミュレーションパラ

[1] 北海道大学
Hokkaido University, Sapporo, Hokkaido 060–0814, Japan
[2] 独立行政法人科学技術振興機構 CREST
JST CREST
[a)] sato@ime.ist.hokudai.ac.jp
[b)] doba@ime.ist.hokudai.ac.jp
[c)] yamamoto@ist.hokudai.ac.jp

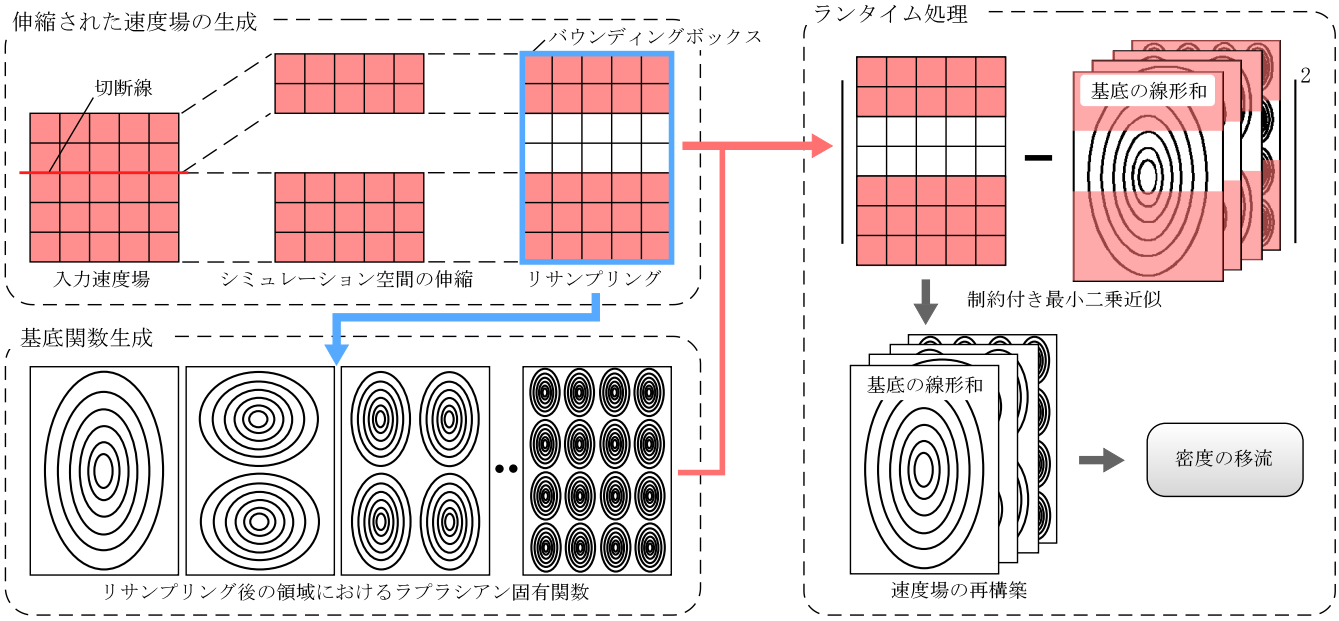


図 1 提案手法の概要

メータを自動で調整して所望の流体アニメーションを生成するものなどが存在する．このような手法により，ユーザの意図を反映した映像を表現できるようになってきた．しかし，過去の手法は，いずれもあらかじめ設定したシミュレーション空間内での流れを表現するもので，シミュレーション空間の大きさや形状を変更した場合には，始めから計算をやり直さなければならない．

そこで，本研究では，シミュレーション空間を任意に伸縮した場合でもシミュレーションをやり直すことなく，流れ場を生成出来る手法を提案する．これにより，ユーザはシミュレーション空間のサイズを調整しながら所望のアニメーションを生成することができる．提案法では，まず，シミュレーション空間を任意に伸縮させ，伸縮させた後のシミュレーション空間において定義された速度場をラプラシアン固有関数を用いて近似する．本稿では，実験として 2 次元の煙のシミュレーションを対象とする．また，合成された速度場に対し，実際に流体解析を行った場合との誤差を算出し可視化する．提案法により，流体解析により生成した速度場を再利用して様々な流体アニメーションを生成できる．

2. 関連研究

Stam は Navier-Stokes 方程式の安定的な解法を提案し，CG において実用的な流体シミュレーション手法を開発した [7]．Stam の手法以降，様々な流体現象を対象とした数多くの解析手法が提案されている．それらの手法の詳細は [1] にまとめられている．しかし，流体シミュレーションは計算コストが非常に高く，結果を生成するまでに多大な時間がかかってしまう．また，所望の流体の動きを作成するためにはシミュレーションパラメータを試行錯誤的に調整しなければならない．

流体シミュレーションを制御することで所望の映像を生成する研究もいくつか提案されている [2][3][8]．Fattal らの手法では，ユーザが入力したターゲット形状から，ポテンシャル場を計算し，そのポテンシャル場に比例した外力を発生させることで，流体の動きを制御する．Dobashi らは，目標形状と現在形状との差分をフィードバックし，シミュレーショパラメータを自動的に調整することで，所望の流体アニメーションを得る手法を提案した．しかし，これらは，事前に設定されたシミュレーション空間において流体を制御する．そのため，シミュレーション空間が伸縮された場合には，すべての計算をやり直さなければならない．

本研究と同様の目的をもった研究も存在する．ボリュームデータの変形手法を利用して，炎のアニメーションを変形する手法が提案されている [5]．しかし，この手法では，流体シミュレーションを用いていないため，写実性に欠ける．また，ボリュームデータを引き延ばしているため，不自然な動きとなる場合がある．

3. 流体シミュレーション

本稿では，流体を非粘性，非圧縮と仮定し，シミュレーションを行う．流体の動きは，次の Navier-Stokes 方程式（以下，NS 方程式）を解くことで計算される．

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

$$\nabla\cdot\mathbf{u} = 0 \qquad (2)$$

$\mathbf{u}$ は流体の速度場，ρ は流体の密度，p は圧力，$\mathbf{f}$ は重力や風などの外力である．式 (1) は速度場の時間発展を表す方

程式であり，右辺第 1 項から，移流項，圧力項，外力項と呼ばれる．また，非圧縮性流体を扱う場合のみ式 (2) が成り立ち，この式は連続の式と呼ばれる．また，Fedkiw らが提案した手法 [4] を用いて乱流を付加する．

以下では上記の方程式を用いて煙をシミュレーションする方法を説明する．NS 方程式により得られた速度場にしたがって煙の密度を以下の式によって移流させる．

$$\frac{\partial D}{\partial t} = -(\mathbf{u}\cdot\nabla)D + D_s \qquad (3)$$

D は煙の密度，D_s は煙の発生源から追加される密度量を表す．提案手法は，速度場にのみ適用され，密度など他の物理量は合成された速度場にしたがって移流させることで動きを解析する．そのため，式 (1)，(2) による速度場の解析は，入力となる速度場を生成する際の 2 次元流体シミュレーションでのみ行う．

4. 提案手法の概要

提案法の概要を図 1 に示す．本手法は，以下の処理から構成される．

シミュレーション空間の伸縮：まず，入力とするシミュレーション空間を任意に伸縮させ，伸縮後の空間を覆うようなバウンディングボックスを作成する．そして，そのバウンディングボックスを入力のシミュレーション空間と同様の格子幅でリサンプリングし，入力の速度場からバウンディングボックスへ速度をマッピングする（図 1 伸縮された速度場の生成部分を参照）．以上の処理により得られたバウンディングボックスを以降，伸縮後の速度場と呼ぶこととする．

ラプラシアン固有関数を用いた近似：次に，伸縮後の速度場をラプラシアン固有関数の線形和として近似する．ラプラシアン固有関数は，伸縮後のシミュレーション空間において定義される（図 1 の基底関数生成）．そして，伸縮後の速度場において，入力速度場がマッピングされている格子点の速度を基底関数の線形和で近似する．各基底関数にかかる重みを最小二乗法により算出し，その重みを用いて伸縮後の速度場全体を再構築することで，ラプラシアン固有関数で表現された伸縮後の速度場を得る（図 1 ランタイム処理）．

本稿では，以上の方法により各ステップにおいて合成された速度場について，実際に NS 方程式を計算した場合の速度場と提案法で作成した速度場の誤差を評価する関数を定義し，それを可視化する．これにより，どの程度伸縮させると，NS 方程式から逸脱してしまうのかを視覚的に確認する．以下，提案法の各処理について詳しく説明する．

5. シミュレーション空間の伸縮

入力のシミュレーション空間を任意に伸縮させる．本稿では，図 2 に示すように，シミュレーション空間の伸縮を

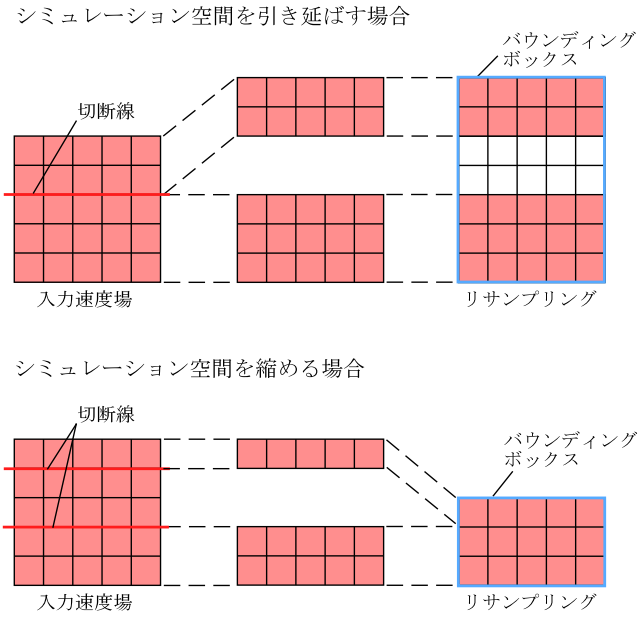


図 2 シミュレーション空間の伸縮

実現する．まず，伸長する場合，ユーザにより指定された切断線において入力のシミュレーション空間を 2 つの領域に分割する．そして，各領域を移動させることでシミュレーション空間の引き延ばす．続いて，縮小する場合，切断線を 2 ヵ所指定する．そして 2 つの切断線の内部の領域を消去し，残った領域同士を切断線で結合する．シミュレーション空間を伸縮させた後，伸縮後のシミュレーション空間を覆うバウンディングボックスを定義し，それを入力と同様の格子幅でリサンプリングすることで伸縮後のシミュレーション空間とする．

6. ラプラシアン固有関数を用いた近似

前節の処理により生成されたシミュレーション空間における速度場を基底関数の線形和で近似する．本稿では，2 次元のシミュレーションを対象としているため，2 次元の矩形領域に定義されるラプラシアン固有関数を用いる．格子数 $(N_{l,x}, N_{l,y})$ の矩形領域におけるラプラシアン固有関数 $\mathbf{\Phi}_k = (f_{k_1,k_2}(x,y), g_{k_1,k_2}(x,y))$ は，次式により定義される．

$$f_{k_1,k_2}(x,y) = \frac{k_2 \sin(k_1 x)\cos(k_2 y)}{k_1^2 + k_2^2} \qquad (4)$$

$$g_{k_1,k_2}(x,y) = \frac{-k_1 \cos(k_1 x)\sin(k_2 y)}{k_1^2 + k_2^2} \qquad (5)$$

k_1, k_2 はベクトル波数である．また，x は，$x = \Delta x, 2\Delta x, 3\Delta x, ..., N_{l,x}\Delta x$（$\Delta x = \pi/N_{l,x}$），$y$ は，$y = \Delta y, 2\Delta y, 3\Delta y, ..., N_{l,y}\Delta y$（$\Delta y = \pi/N_{l,y}$）として定義される．ラプラシアン固有関数に関しては，Witt らの手法 [9] を参考としているため，詳細についてはそちらを参照していただきたい．

上記 2 次元のラプラシアン固有関数の線形和として，伸

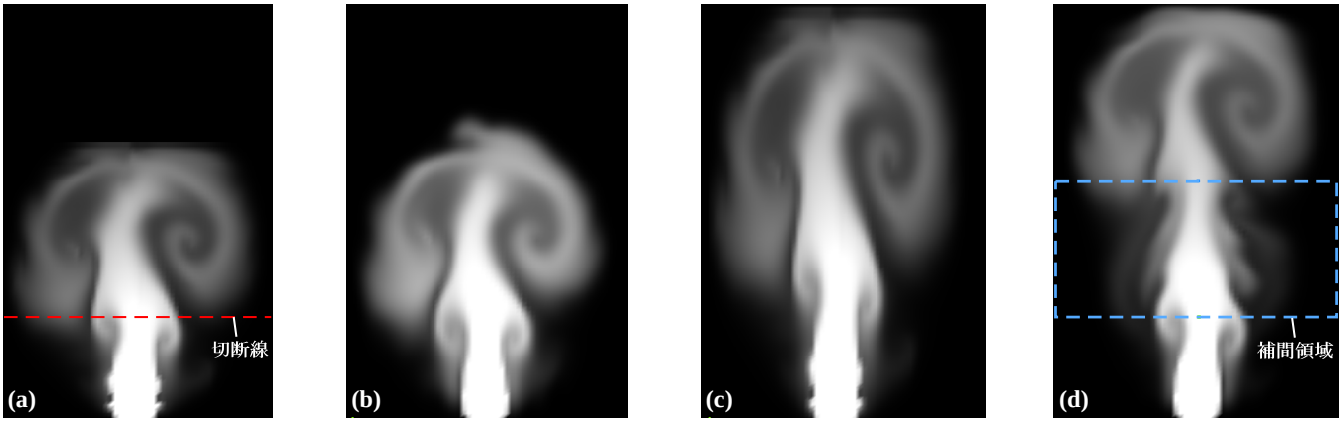


図 3 他の方法との比較

縮後の速度場を最小二乗法により近似する．ラプラシアン固有関数の格子数 $(N_{l,x}, N_{l,y})$ には，伸縮後の速度場と同様の格子数を設定する．本稿では，以下の式を満たす重み w_i を算出することで，速度場を近似する．

$$\min_{w_i}\{\sum_{id=0}^{N_{in}} ||\mathbf{V}(id) - \sum_{i=0}^{N_{LE}} w_i \mathbf{\Phi}_i(id)||^2 + \sum_{i=0}^{N_{LE}} w_i^2\} \quad (6)$$

$\mathbf{V}$ は入力の速度場，N_{LE} は使用する基底関数の数である．また，N_{in} は，図 2 右のリサンプリング後の速度場における赤色領域の格子全体を示しており，入力となる速度成分が存在する部分でのみ，最小二乗法を計算することを意味する．上記の式 (6) により算出された重みを用いて，速度場 $\mathbf{v}$ が以下の式により合成される．

$$\mathbf{v} = \sum_{i=0}^{N_{LE}} w_i \mathbf{\Phi}_i \quad (7)$$

ここで，本稿で使用するラプラシアン固有関数は $\nabla \cdot \mathbf{\Phi}_k = 0$ を満たすので，その線形和として表現される $\mathbf{v}$ についても同様に発散が 0 となり，非圧縮性の法則を満たす．上記の方法により，速度場を引き伸ばした際の速度が存在しない領域に関しても，尤もらしい速度を補間することができる．

7. シミュレーションとの誤差の評価

前節までの方法により生成した速度場は，NS 方程式のうち連続の式（式 (2)）は満たしているが，式 (1) については満たさない場合がある．そのため，本稿では，NS 方程式を基に，合成した速度場の誤差評価関数を構築する．まず，NS 方程式から外力項を除き，両辺の回転をとったものを考える．圧力場はスカラー場であるため，その回転は 0 となる．従って，次式が得られる．

$$\nabla \times (\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u}) = \nabla \times (-\frac{1}{\rho}\nabla p) = 0 \quad (8)$$

ここで，$\partial \mathbf{u}/\partial t$ を離散化すると次式を得る．

$$\nabla \times \mathbf{u}_n = \nabla \times (\mathbf{u}_{n-1} - \Delta t(\mathbf{u}_{n-1} \cdot \nabla)\mathbf{u}_{n-1}) \quad (9)$$

この式 (9) を用いて，以下の評価関数を定義することで，各格子点の速度がどの程度 NS 方程式から逸脱しているのかを評価する．

$$\begin{aligned} E &= |\nabla \times \mathbf{v}_n^{synth} - \nabla \times \mathbf{v}_n^{advect}| \quad (10) \\ &= |\nabla \times \mathbf{v}_n - \nabla \times (\mathbf{v}_{n-1} - \Delta t(\mathbf{v}_{n-1} \cdot \nabla)\mathbf{v}_{n-1}))| \end{aligned}$$

ここで，$\mathbf{v}_n, \mathbf{v}_{n-1}$ は，$n, n-1$ ステップ目に提案法により合成された速度場である．右辺第一項（$\mathbf{v}_n^{synth}$）が n ステップ目の速度を提案法により合成したもの，第二項（$\mathbf{v}_n^{advect}$）が提案法により合成された $n-1$ ステップ目の速度場に対し，NS 方程式における移流項を計算したものとなる．この E を全格子点について算出し，可視化することで，伸縮によるシミュレーションからの逸脱度合いをユーザに提示する．

8. 実験結果

提案手法を適用して生成した結果を図 3，4 に示す．実験環境は，CPU が Intel Core i7 2600K（メモリ 8GB），GPU が NVIDIA GeForce GTX 680 となっている．また，どの結果においても，使用したラプラシアン固有関数の数は 400 である．

図 3 は，提案法および他の方法により煙のアニメーションを伸長した場合の比較結果である．図 3(a) は，64x64 の格子でシミュレーションした例，(b) は 64x96 の格子数でシミュレーションを行った結果である．また，図 3(c) は (a) により算出された密度場を 64x96 の格子にマッピングした例であり，(d) は提案法により (a) の速度場を 64x96 に切断線の位置から伸長した例である．(b) では，格子数は増加しているが，シミュレーションパラメータは変更していないため，(a) を伸長した結果を得たい場合，パラメータを調整しなければならない．(c) は単純に密度場を伸長したものとなっているため，全体が引き延ばされたような結果となってしまっている．しかし，提案法による結果である (d) では，(a) の特徴を保ったまま伸長できているのが分かる．ここで，各結果の毎ステップの計算時間は，(a)

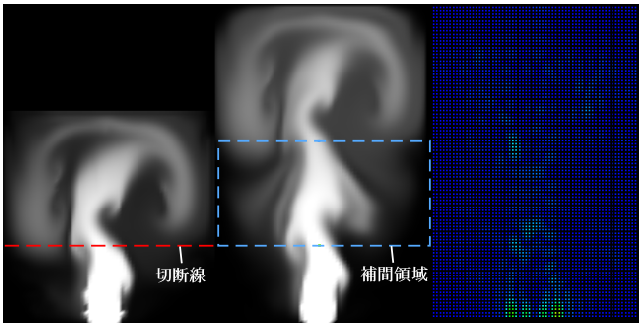
が 0.01 秒，(b) が 0.015 秒，(d) では前計算に 4.80 秒，近似に 0.01 秒，速度場の再構築に 0.02 秒ほどかかる．

図 4 は，提案法によりシミュレーションを伸縮させた結果および，第 7 節の方法により算出した誤差を可視化した結果である．図 4(a) では，(a) 左の 64x64 のシミュレーションを 64x96 の格子に伸長した結果，(b) では左の 64x96 のシミュレーションを 64x64 に縮小した結果を示す．どちらの結果においても，元のシミュレーションの特徴を保ったまま，シミュレーションの伸縮が実現できているのが分かる．また，毎ステップの計算時間は，(a) が図 3 の結果と同様であり，(b) のシミュレーションでは 0.015 秒，提案法では前計算に 1.80 秒，近似に 0.01 秒，速度場の再構築に 0.02 秒かかる．次に，図 4(a) および (b) の右の図は，合成結果に対し第 7 節の方法により算出した誤差を可視化したものである．この結果では，誤差の値をシミュレーションの速度場における速度の最大値で正規化して表示しており，0 に近いほど青色となり，1 に近いほど赤色となる．(a) では，補間領域の部分に誤差が出ている分布となっている．これは，補間領域外では流体解析を行った速度場を近似しているため，誤差は小さくなるが，補間領域内は領域外の近似から尤もらしい速度場が補間されているのみであり，NS 方程式は考慮されていないためであると考えられる．(b) では，結合位置の周辺に大きく誤差が出ているのが確認できる．これは 2 つの切断線の位置における速度場同士が大きく異なっているため，誤差が大きくなったのだと考えられる．また，どちらの結果においても中央下端の部分に誤差が確認できるが，この部分は煙の発生源であり大きな外力を与えているため，誤差が大きくなっている．

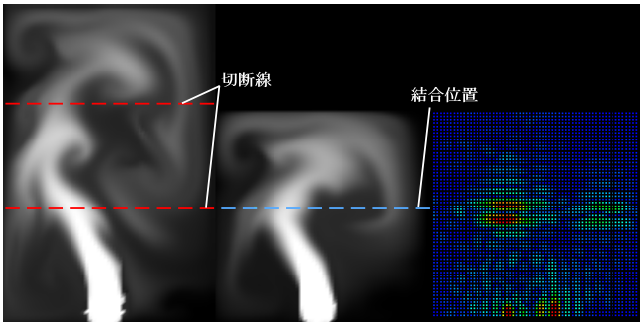
9. まとめと今後の課題

本稿では，流体のシミュレーション空間を任意に伸縮させ，その速度場をラプラシアン固有関数の線形和として表現することで，流体の法則を保ったアニメーションの伸縮を可能とした．また，誤差評価関数を構築し，提案法により合成された速度場が，どの程度 NS 方程式から逸脱しているか，可視化した．そして，実験において，2 次元のシミュレーションについて，提案法の有効性が確認できた．

今後の課題としては，3 次元のシミュレーションへの拡張があげられる．3 次元の場合，近似に必要な基底数が多くなり，計算コストやメモリ使用量が膨大になってしまう可能性がある．また，図 4(b) の結果のように，合成された速度場が NS 方程式から大きく逸脱してしまう場合への対処も今後の課題としてあげられる．これについては，速度場の近似において，今回構築した誤差評価関数を最小とするような項を追加することで，ある程度 NS 方程式に沿った流れの合成も可能なのではないかと考えている．さらに，水など，液体状の流体に対し適用可能か実験を行うことがあげられる．液体の場合，体積保存を考慮する必要があり，



(a)



(b)

図 4 適用例

本手法をそのまま適用できるか実験を行う必要がある．

謝辞 この研究は独立行政法人科学技術振興機構，CREST によりサポートされています．

参考文献

[1] R. Bridson : *Fluid Simulation for Computer Graphics*, AK Peters (2008).
[2] Y. Dobashi, K. Kusumoto, K. Nishita, and T. Yamamoto : Feedback control of cumuliform cloud formation based on computational fluid dynamics, *ACM Transactions on Graphics 27*, 3, Article 94 (2008).
[3] R. Fattal, and D. Lischinski : Target-driven smoke animation, *ACM Transactions on Graphics 23*, 3, 439-446 (2004).
[4] R. Fedkiw, J. Stam, and H.W. Jensen : Visual simulation of smoke, In *Proc. SIGGRAPH 2001*, 15-22 (2001).
[5] A.R. Fuller, H. Krishnan, K. Mahrous, B. Hamann, and K.I. Joy : Real-time procedural volumetric fire, In *Proceeding of the 2007 symposium on interactive 3D graphics and games*, 175-180 (2007).
[6] D.Q. Nguyen, R. Fedkiw, H.W. Jensen : Physically Based Modeling and Animation of Fire, In *Proceeding of ACM SIGGRAPH 2002*, 721-728, (2002)
[7] J. Stam : Stable fluids, In *Proceedings of ACM SIGGRAPH 1999, Annual Conference Series*, 121-128 (1999).
[8] A. Treuille, A. McNamara, Z. Popovic, and J. Stam : Keyframe control of smoke simulations, *ACM Transactions on Graphics 22*, 3, 716-723 (2003).
[9] T.D. Witt, C. Lessig, and E. Fiume : Fluid simulation using Laplacian eigenfunctions, *ACM Transactions on Graphics 31*, 1, Article 10 (2012).