CGI2014 manuscript No.
(will be inserted by the editor)

An Optimization Approach for Designing Fluid Flow Fields

Syuhei Sato · Yoshinori Dobashi · Kei Iwasaki · Hiroyuki Ochiai · Tsuyoshi Yamamoto · Tomoyuki Nishita

Abstract In entertainment applications, such as computer games and movies, animators are often requested to synthesize a realistic fluid flow with a particular behavior. This paper presents a method to help animators meet such requirements. Our method enables the user to design a realistic fluid flow. In order to generate the desired fluid flow in a realistic form, the flow is calculated by solving a minimization problem subject to user-specified constraints and the Navier-Stokes equations. The minimization problem is efficiently solved by representing the flow field by a linear combination of orthogonal basis functions. Consequently, the fluid flow is obtained by simple matrix operations.

Keywords fluid simulation · incompressible Navier-Stokes equation · Laplacian eigenfunctions · design

1 Introduction

Visual simulation of fluids has become one of the most important research topics in computer graphics. Many methods have been proposed for simulating smoke, water, fire, and so on [1]. Most of the recent methods are based on computational fluid dynamics to create realistic animations and these are used in many applications such as movies and computer games. However, one of the problems is the expensive computational cost. In those entertainment applications, rendering fluid with a particular motion is often requested. Animators usually attempt to create the desired motion by repeating fluid simulations with different parameter settings until a satisfactory result is obtained. However, this is an extremely tedious and time-consuming task incurring an expensive computational cost.

Syuhei Sato, Tsuyoshi Yamamoto
Hokkaido University

Yoshinori Dobashi
Hokkaido University, JST CREST, UEI Research

Kei Iwasaki
Wakayama University, UEI Research

Hiroyuki Ochiai
Kyushu University, JST CREST

Tomoyuki Nishita
UEI Research, Hiroshima Shudo University

Many methods have been proposed to address this problem. Controlling the fluid simulation is a promising approach [2,3,5]. However, a trial-and-error process is still required to tune the control parameters.

Several recent researches have focused on a different approach: using low-resolution fluid simulation as a guide to produce high-resolution results. In this approach, low-resolution fluid simulation is used for designing the fluid motion, which is subsequently converted into a high-resolution animation. Some methods control the simulation using low-resolution simulation results [4,7]. These methods allow the user to efficiently generate the desired high-resolution animation. However, creating a particular fluid motion is still a difficult and tedious task even when low-resolution fluid simulation is used.

To address this problem, we propose a method that enables the user to design a realistic fluid flow. Our method is based on grid-based fluid simulation and is suitable for synthesizing animations of incompressible flow such as smoke and fire. The key concept behind our method is to solve a minimization problem subject to the user's constraints and the Navier-Stokes equations. Our method decomposes the flow field into low and high frequency components. The minimization problem is solved for the low frequency components to control the overall motion, and the high frequency components are subsequently added to create detailed motion. The low frequency components of flow fields are represented

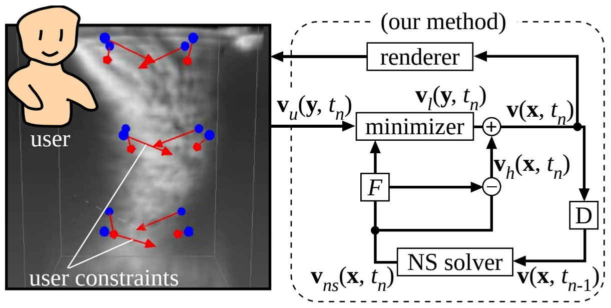
(our method)
renderer
user
$\mathbf{v}_u(\mathbf{y}, t_n)$
$\mathbf{v}_l(\mathbf{y}, t_n)$
minimizer
$\mathbf{v}(\mathbf{x}, t_n)$
$\mathbf{v}_h(\mathbf{x}, t_n)$
F
D
NS solver
$\mathbf{v}_{ns}(\mathbf{x}, t_n)$
$\mathbf{v}(\mathbf{x}, t_{n-1})$
user constraints

Fig. 1 Overview of our system. " D " is a delay buffer that takes a flow field as input and outputs it at the next time step. " F " is a low pass filter.

by a linear combination of incompressible orthogonal basis functions, which ensures that the velocity field satisfies the divergence-free condition. Moreover, our method uses 2D basis functions for 3D flow fields, and the fluid flow is generated at a low computational cost. Since the creation of the flow field is formulated as a least squares minimization problem using orthogonal basis functions, we can compute the flow field using simple matrix operations.

2 Overview of our system

Fig.1 shows an overview of our system. The flow field $\mathbf{v}$ is calculated at discrete time steps $t_n (n = 0, 1, \cdots)$, where the time interval is Δt. We decompose the flow field $\mathbf{v}$ into low and high frequency components.

$$\mathbf{v}(t_n) = \mathbf{v}_l(t_n) + \mathbf{v}_h(t_n), \tag{1}$$

where $\mathbf{v}_l$ and $\mathbf{v}_h$ are low and high frequency components of the flow field $\mathbf{v}$, respectively. In our computation, $\mathbf{v}$ and $\mathbf{v}_h$ are sampled at each grid point $\mathbf{x}$, where the grid size is N_h. $\mathbf{v}_l$ is sampled at each grid point $\mathbf{y}$, where the grid size is N_l that is smaller than N_h. Our method calculates desired fluid flow $\mathbf{v}_l$ in low frequencies, and then adds high-frequency components $\mathbf{v}_h$ to $\mathbf{v}_l$. Details are described in the following.

In order to generate the desired flow field, the user first places a set of control points at arbitrary positions as indicated by the blue spheres in Fig.1. The user then specifies the velocities at the control points (the red arrows). The user-specified velocity of the l-th control point at time step t_n is denoted as $\mathbf{s}_u(\mathbf{y}_l, t_n)(l = 0, 1, \cdots, N_u - 1)$, where $\mathbf{y}_l$ is the position of the l-th control point and N_u is the number of control points. Then, our system interpolates $\mathbf{s}_u(\mathbf{y}_l, t_n)$ in order to obtain a spatially continuous flow field $\mathbf{v}_u(\mathbf{y}, t_n)$ by using radial basis functions. On the other hand, a simulated flow field $\mathbf{v}_{ns}(t_n)$ is obtained by solving the Navier-Stokes equations using the flow field $\mathbf{v}(t_n - 1)$ computed at the previous time step (see Fig.1). $\mathbf{v}_{ns}(t_n)$ is calculated on the high-resolution grid whose grid size is N_h. Then, $\mathbf{v}_{ns}(\mathbf{x}, t_n)$ is decomposed into low and high frequency components. $\mathbf{v}_h(\mathbf{x}, t_n)$ is obtained as high frequency components of $\mathbf{v}_{ns}(\mathbf{x}, t_n)$. Our system computes an incompressible flow field $\mathbf{v}_l(\mathbf{y}, t_n)$ by solving a minimization problem (see Section 3). Our minimizer takes $\mathbf{v}_u(\mathbf{y}, t_n)$ and $[F * \mathbf{v}_{ns}](\mathbf{y}, t_n)$ as input, where F is a low pass filter, and $*$ is the convolution operator. After computing $\mathbf{v}(\mathbf{x}, t_n)$, scalar quantities, such as smoke density, are advected and visualized by using a volume rendering technique. The user then modifies the constraints to create the desired visual effect.

3 Definition of minimization problem

In our method, the low frequency components $\mathbf{v}_l(\mathbf{y}, t_n)$ of the flow field are represented by a linear combination of basis functions, $\mathbf{\Phi}_i$, as follows,

$$\mathbf{v}_l(\mathbf{y}, t_n) = \sum_{i=0}^{N-1} w_i(t_n)\mathbf{\Phi}_i(\mathbf{y}), \tag{2}$$

where $w_i(t_n)$ is the coefficient for the i-th basis function and N is the number of basis functions. We use Laplacian eigenfunctions [6] for $\mathbf{\Phi}_i(\mathbf{y})$. This enforces the divergence-free condition on the flow field. For reducing computation and storage costs, 3D flow fields are represented by using 2D basis functions (see Section 5). Our task is then to determine the coefficient $\mathbf{w}(t_n) = (w_0(t_n), w_1(t_n), \cdots, w_{N-1}(t_n))$ at each time step so that a realistic flow with the desired behavior is generated. The user can specify the constraint velocities at multiple positions in space and time in order to design the desired flow field. The Navier-Stokes equations are taken into account in computing the flow. The coefficient $\mathbf{w}(t_n)$ is obtained by minimizing the sum of two error functions, E_{usr} and E_{ns}. That is,

$$\arg\min_{\mathbf{w}(t_n)}(E_{usr}(t_n) + \alpha E_{ns}(t_n)), \tag{3}$$

where α is a user-specified constant used to adjust the influence of the two error functions described below.

$E_{usr}(t_n)$ measures the difference between the user-specified velocities and the resulting flow field. $E_{usr}(t_n)$ is defined by the following equation.

$$E_{usr}(t_n) = ||\mathbf{v}_u(\mathbf{y}, t_n) - \sum_{i=0}^{N-1} w_i(t_n)\mathbf{\Phi}_i(\mathbf{y})||^2, \tag{4}$$

where $||\cdot||$ is the L_2 norm of the function. $\mathbf{v}_u(\mathbf{y}, t_n)$ represents the constraint flow field and is defined by interpolating user-specified velocities $\mathbf{s}_u(\mathbf{y}_l, t_n)$ by using radial basis functions. That is,

$$\mathbf{v}_u(\mathbf{y}, t_n) = \sum_{l=0}^{N_u-1} b_l(t_n)\varphi(|\mathbf{y} - \mathbf{y}_l|), \tag{5}$$

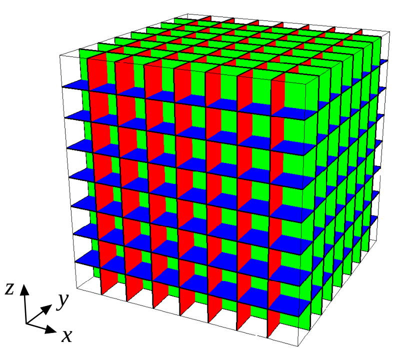

Fig. 2 Slices for representing a 3D flow field.

where φ is a radial basis function and $b_l(t_n)$ is obtained by solving the following linear system of equations.

$$\sum_{l=0}^{N_u-1} b_l(t_n)\varphi(|\mathbf{y}_m - \mathbf{y}_l|) = \mathbf{s}_u(\mathbf{y}_m, t_n), \tag{6}$$

where $m = 0, 1, \cdots, N_u - 1$. For φ, we use Gaussian basis functions.

$E_{ns}(t_n)$ measures the difference between the flow field and the incompressible inviscid Navier-Stokes equations. By discretizing the incompressible inviscid Navier-Stokes equations with the time step $\Delta t (= t_n - t_{n-1})$, we obtain the following equation.

$$\mathbf{u}(t_n) = \mathcal{G}(\mathbf{u}(t_{n-1})), \tag{7}$$

$$\mathcal{G}(\mathbf{u}(t)) = \mathbf{u}(t) + \Delta t\{-(\mathbf{u}(t)\cdot\nabla)\mathbf{u}(t) - \frac{1}{\rho}\nabla p + \mathbf{f}\}. \tag{8}$$

Let us now assume that we have obtained the flow field $\mathbf{v}(\mathbf{x}, t_{n-1})$ at time t_{n-1}. In order to satisfy the Navier-Stokes equations, the flow field at time t_n needs to be computed according to Eq.(7). That is, $\mathbf{v}_{ns}(\mathbf{x}, t_n) = \mathcal{G}(\mathbf{v}(\mathbf{x}, t_{n-1}))$. We use low frequency components of $\mathbf{v}_{ns}$ for defining E_{ns}. We define $E_{ns}(t_n)$ as the difference between $\mathbf{v}_l(\mathbf{y}, t_n)$ and low frequency components of $\mathbf{v}_{ns}$:

$$E_{ns}(t_n) = ||[F * \mathbf{v}_{ns}](\mathbf{y}, t_n) - \sum_{i=0}^{N-1} w_i(t_n)\mathbf{\Phi}_i(\mathbf{y})||^2. \tag{9}$$

Finally, $\mathbf{v}_h(\mathbf{x}, t_n)$ is obtained as follows: $\mathbf{v}_h(\mathbf{x}, t_n) = \mathbf{v}_{ns}(\mathbf{x}, t_n) - [F * \mathbf{v}_{ns}](\mathbf{x}, t_n)$.

4 Solving minimization problem

Given the definitions of the error functions in the previous section, we can now solve the minimization problem given in Eq.(3). By taking a derivative of Eq.(3) with respect to the coefficient $w_i(t_n)$, we obtain the matrix equation $\mathbf{A}\mathbf{w}(t_n) = \mathbf{c}(t_n)$, where $\mathbf{A}$ is an $N \times N$ matrix, $\mathbf{w}(t_n)$ and $\mathbf{c}(t_n)$ are N dimensional column vectors. The (i, j)-th element a_{ij} of $\mathbf{A}$ is given by $a_{ij} = (1+\alpha)(\mathbf{\Phi}_i(\mathbf{y}) \cdot \mathbf{\Phi}_j(\mathbf{y}))$, where $\cdot$ indicates the dot product between two functions. Since $\mathbf{\Phi}_i$ are orthogonal basis functions, a_{ij} computed by the above equation is zero if $i \neq j$. Therefore, $\mathbf{A}$ is a diagonal matrix and its diagonal elements can be precomputed. Next, i-th element c_i of $\mathbf{c}(t_n)$ is given by $c_i = \{\mathbf{v}_u(\mathbf{y}, t_n) + \alpha[F * \mathbf{v}_{ns}](\mathbf{y}, t_n)\} \cdot \mathbf{\Phi}_i(\mathbf{y})$. Our system computes the dot product between functions in the above equations on a grid with a specified resolution N_l. Then, the coefficient $w_i(t_n)$ is computed by $w_i(t_n) = c_i/a_{ii}$. The above approach allows us to compute the coefficient vector $\mathbf{w}(t_n)$ very efficiently since we do not have to compute $\mathbf{A}$ at run time. All we need to do at each time step is to project $\mathbf{v}_u + \alpha[F * \mathbf{v}_{ns}]$ onto $\mathbf{\Phi}_i$, and compute $w_i(t_n) = c_i/a_{ii}$.

5 Basis functions

As we have described previously, the Laplacian eigenfunctions are used as the basis functions. In our method, we use basis functions defined on a 2D rectangular grid for a 3D flow field. Although 3D basis functions exist [6], we found that those functions are not suitable for our purpose. In order to design a 3D flow field, we need a large number of basis functions. We tried 512 basis functions but we found that they were not sufficient. Furthermore, the storage cost for 3D basis functions sampled on a 3D grid is very expensive. We can compute the basis functions on the fly but we found that this significantly slows down the computation.

To address these problems, we propose to use the 2D basis functions for the 3D flow fields. For using 2D basis functions, we subdivide the simulation space using three sets of slices, indicated in red, green, and blue in Fig.2. The red, green, and blue slices are orthogonal to the x, y, and z axes, respectively. On the red slices, the yz components of the flow field are represented by 2D basis functions. Similarly, for the green and blue slices, the zx and xy components, respectively, are represented by 2D basis functions. Let us define the flow field $\mathbf{v}_{yz}^p$, $\mathbf{v}_{zx}^q$, and $\mathbf{v}_{xy}^r$ on the p-th red, q-th green, and r-th blue slices, respectively. The 3D flow $\mathbf{v}$ at grid point (p, q, r) is generated by combining these three flow fields:

$$\mathbf{v}_l(p, q, r) = \frac{1}{2}(\mathbf{v}_{yz}^p(q, r) + \mathbf{v}_{zx}^q(r, p) + \mathbf{v}_{xy}^r(p, q)). \tag{10}$$

Note that the 3D flow field computed in this way satisfies the divergence-free condition. This method significantly reduces the storage cost for the basis functions sampled at the grid points and the pre-computation time for calculating $\mathbf{A}$, because slices with the same size can share the same basis functions. If the shape of the simulation space is a cube, we can use the same basis function for all the slices.

6 Results

This section shows some examples created using our method. We used a desktop PC with a CPU (Intel

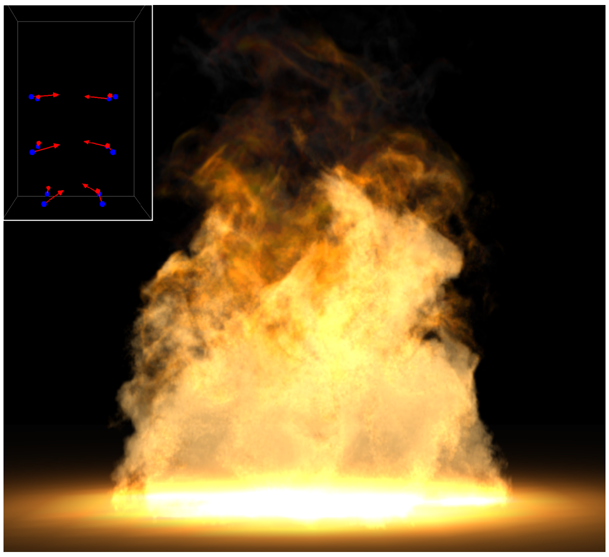
Fig. 3 Example of fire.

Core i7 2600K, 16GB memory) and GPU (NVIDIA GeForce GTX TITAN) to compute all the examples shown in this section. In all examples, the number of basis functions (N) is 256, the low resolution and high resolution grid sizes (N_l and N_h) are $32 \times 32 \times 48$ and $256 \times 256 \times 384$, respectively. The computation time for updating the flow field is 0.7sec/frame. The videos corresponding to the following examples can be found in the supplemental material.

Fig.3 shows an example of a fire animation with a circular swirling motion that is simulated by setting five disk-shaped heat sources at the bottom of the simulation space. The left-upper inset of Fig.3 shows the control points. Our method can be used to render fire simulations with buoyancy forces.

Next, Fig.4 shows an animation of a tornado. The right-bottom inset of Fig.4 shows the control points. By moving the control points, the user can design the motion of the tornado.

7 Conclusion

We have proposed a method for designing incompressible and inviscid flow fields. The user can specify a set of constraints in order to create the desired flow field. The method computes realistic flow fields by solving the minimization problem such that user-specified constraints are satisfied. The Navier-Stokes equations are also taken into account in solving the minimization problem. The low frequency components of flow fields are represented by incompressible orthogonal basis functions and this ensures the divergence-free conditions of the flow. We also proposed a method for representing a three-dimensional flow field using a set of two-dimensional basis functions. We demonstrated the capabilities of our method with a set of examples.

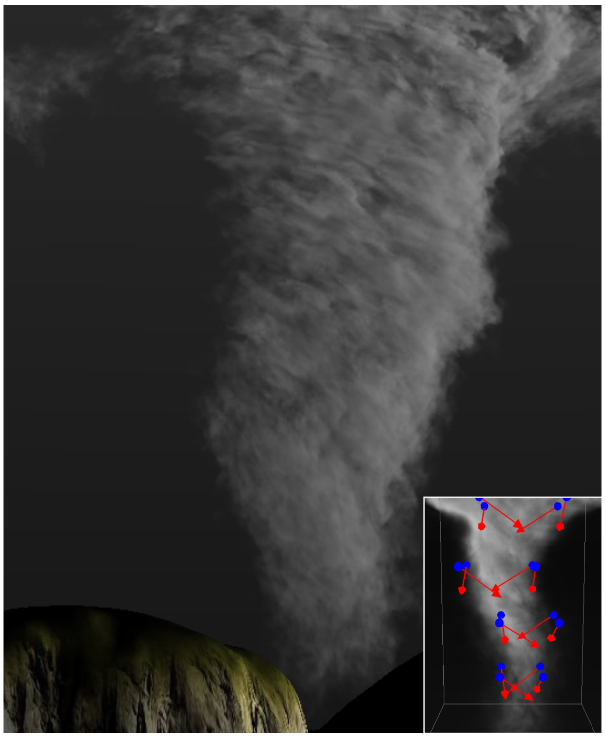
Fig. 4 Example of tornado.

One of the limitations is the fact that the computational cost of our method is proportional to the number of basis functions. The level of detail in editing the flow depends on the number of basis functions used. The user can design a detailed flow field at the expense of increased computational and storage costs.

References

1. R. Bridson, Fluid Simulation for Computer Graphics, Publisher, AK Peters, 2008.
2. R. Fattal, D. Lischinski, "Target-driven smoke animation," ACM Transactions on Graphics 23, 3, pp. 439-446, 2004.
3. Y. Kim, R. Machiraju, D. Thompson, "Path-based control of smoke simulaitons," In Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer Animation, pp. 33-42, 2006.
4. M. B. Nielsen, B. B. Christensen, " Improved variational guiding of smoke aniemtions," Computer Graphics Forum 29, 2, pp. 705-712, 2010.
5. A. Treuille, A. McNamara, Z. Popovic, J. Stam, " Keyframe control of smoke simulations," ACM Transactions on Graphics 22, 3, pp. 716-723, 2003.
6. T. D. Witt, C. Lessig, E. Fiume, " Fluid simulation using Laplacian eigenfunctions," ACM Transactions on Graphics 31, 1, Article 10, 2012.
7. Z. Yuan, F. Chen, Y. Zhao, " Pattern-guided smoke animation with lagrangian coherent structure," ACM Transactions on Graphics 30, 6, Article 136, 2011.