Abstract

We present a novel fluid control method that is capable of driving particle-based fluid simulation to match a rapidly changing target while keeping natural fluid-like motion. To achieve the desired behavior, we first generate control particles by sampling the target shape and then apply a non-linear constraint to each control particle, with its neighboring fluid particles keeping a constant fluid density within its influence region. This density constraint is highly in line with the incompressible nature of the fluid, which can drive the fluid to match the target shape in a natural way. In addition, to match a fast moving or deforming target, we add an adaptive spring for each fluid particle in the control region, connecting with its nearest control particle. The spring constraint takes effect only when the fluid particle is far from its corresponding control particle to avoid introducing artificial viscosity. Therefore, the fluid particles are well controlled even if the target shape changes rapidly. Furthermore, we integrate a velocity constraint to adjust the stiffness of the controlled fluid. All these three constraints are solved under position-based framework which enables our simulation fast, robust and well-suited for interactive applications. We demonstrate the efficiency and effectiveness of our method in various scenarios in real time.

Keywords: Fluid control, Position-based, Fluid simulation

1 Introduction

In the past decade, fluid simulation has been a hot research topic in the field of computer graphics and many efficient algorithms have been proposed to reveal the rich visual details of the realistic fluid motion. Besides realism of fluid, the controllability for fluid is equally important in computer graphics. In the procedure of animation production, to produce certain special fluid effects, such as an animated character made of water or river flowing along a specified path, animators have to tune numerous parameters repeatedly, which is laborious and time-consuming without the help of efficient fluid control techniques.

Due to the non-linear feature of fluid simulation, it is a challenge to control fluid while maintaining a naturalistic-looking fluid motion. In particular, matching fluid to a rapidly changing target, such as a dancing character, is interesting for visual effects in animation production but becomes more complex, in which the fast velocity and acceleration may lead the fluid out of control. Furthermore, with the emergency of interactive applications, such as games, controlling such fluid in real time becomes interesting and even challenging.

In this paper, we propose a control method for driving fluid simulation to follow a rapidly changing target while preserving plausible fluid-like motion. In our pipeline, animators only need to provide the target shape, which is represented by a general mesh model. Then, our method automatically generates control particles by sampling the input model. Additionally, our method allows the animator to specify the joints of the generated control particles and produce a smooth skeleton-driven animation for the control particles. These control particles specify the density distribution in the simulation space. Then, we provide a position-based control technique to drive fluid particles to match the rapidly changing target shape defined by the control particles, while keeping plausible fluid motion. By tuning a few control parameters that are exploited in our method, animators can simulate various special effects ranging from stiff-like fluid to low-viscosity fluid. In addition, our method can direct a fluid simulation to flow along a path specified by the user. Furthermore, our control method can be simulated in real-time.
time and allows users to interact with the simulation directly. Four criteria were proposed by Shi and Yu [Shi and Yu 2005] for evaluating a fluid control method: 1) Control capability, 2) Ease to use, 3) Fluid-like motion, and 4) Stability. Our control method can well meet all these four criteria. It is capable of tracking rapidly changing targets while keeping fluid-like motion. By using our sampling and deforming algorithm, animators only need to provide the target model with corresponding skeleton motion data as input and our method formulates control particles automatically. Additionally, our method inherits the efficiency and stability of position-based fluid framework [Müller et al. 2007] and allows users to interact with the simulation directly.

Our main contributions in this paper include following:

- A density-based control method is proposed to control fluid simulation to match a user-specified shape efficiently while preserving rich visual fluid details.
- By adaptively adding springs between fluid particles and control particles, our method allows the target shape to move or deform rapidly.
- Integrating with position-based methods makes our algorithm simple, robust and efficient. So controlling fluid in interactive applications is not out of reach.

2 Related work

Foster and Metaxas [Foster and Metaxas 1997] was the first to control fluid simulation by using the concept of an embedded controller. Later, Foster and Fedkiw [Foster and Fedkiw 2001] extended their method by using 3D parametric space curves with a local velocity control. Treuille et al. [Treuille et al. 2003] proposed a keyframe control technique to drive a smoke simulation to match a user-specified keyframe by exerting an optimal wind force. However, it was computationally expensive due to the need of solving a non-linear optimization problem. To improve the efficiency, McNamara et al. [McNamara et al. 2004] used the adjoint method to solve the gradient-based non-linear optimization problem. They also extended their method to deal with both smoke and water control.

Fattal and Lischinski [Fattal and Lischinski 2004] achieved the fluid control by adding a driving force term and a gathering term to the standard momentum equation. Their method avoided expensive optimization and improved efficiency significantly. Pighin et al. [Pighin et al. 2004] investigated an Advected Radial Basis Function (ARBF) model as a new fluid representation and allowed users to control the fluid flow as a deformable object interactively.

Control particles were used to control a grid-based fluid simulation and produced a directable photorealistic liquid in [Rasmussen et al. 2004]. Similar to our motivation of matching rapidly changing models, Shi and Yu [Shi and Yu 2005] introduced a simple grid-based technique controlling free surface fluid to follow a rapidly changing target objects. In their work, the shape and velocity of the target object were considered and used as the feedback force to pull fluid back to the target. In addition, they formulated a potential function according to the shape and skeleton of the target.

Recently, Ravendran et al. presented an approach [Ravendran et al. 2012] that could track the desired shape while producing highly detailed secondary effects. Their method used the velocities of control mesh as boundary conditions during the projection step in the Euler fluid simulation. Pan et al. [Pan et al. 2013] proposed the notion of interactive editing fluid motion. They provided users the ability to edit fluid motion locally, such as sketching local fluid features using strokes, dragging a local fluid region and controlling a local shape with a small mesh patch.

There were also many works [Nielsen et al. 2009] [Nielsen and Christensen 2010] [Huang et al. 2011] aiming to direct a high-resolution fluid simulation based on a coarse preview simulation. Because of the chaotic nature of fluid motion and the error of numerical simulation, small variations in simulation resolution often produced a divergent result compared to the preview version. These fluid directing techniques were used to correct the high-resolution simulation to be consistent with the low resolution simulation.

The above works were all under the grid-based fluid simulation framework which is computationally expensive and limited with fixed simulation area. Instead, in this paper, we focus on the particle-based fluid simulation method, which is simple, flexible and popular for interactive applications.

The methods of controlling particle-based fluid simulation are much less than that of grid-based methods. A recent pioneering work [Thirey et al. 2009] proposed a detail-preserving fluid control method. They used an attraction force to pull fluid particles towards control particles, and a velocity force to control the flow of fluid. To avoid artificial viscosity introduced by the control forces, the fine-scale detail of the fluid motion was separated using a low-pass filter on the current velocity field, and the velocity forces were only applied to the coarse-scale components of the flow. Their method aimed to guide the fluid flow without disturbing the small scale fluid motion.

Different from their work, our emphasis is controlling the fluid to match a desired shape with fast movement or large deformation, where animators only need to provide the desired fluid state and our method will automatically drive the fluid to match the target shape in a natural-looking way. Instead of using a linear attraction force, we use a density constraint to drive fluid to match the target shape. Our density constraint is highly in line with the incompressible nature of the fluid. This feature makes the fluid simulation controllable while exhibiting plausible visual effects and keep stable even if the user defines a hard constraint. In order to avoid the fluid out of control when the target shape moves fast, we also incorporate spring-like constraints which could pull the fluid particles back to the target shape region. Furthermore, we allow animators to control fluid in real time.

The density and spring constraints in our method are all solved using the position-based dynamic methods (PBD) [Müller et al. 2007]. PBD is more and more popular in computer games and other interactive applications for its simplicity and unconditional stability. It has been a sophisticated framework that is capable of performing a variety of physical simulation in real-time [Macklin et al. 2014]. Particularly, by enforcing fluid incompressibility using density constraints. [Macklin and Müller 2013] proposed a position-based fluids (PBF) method under PBD framework. PBF inherited the unconditional stability of PBD framework and made it practical to simulate large scale fluid at an interactive frame rate.

Although our control algorithm can be integrated into any particle-based fluid simulation methods such as [Müller et al. 2003], [Becker and Teschner 2007] and [Solenthaler and Pajarola 2009], our primary focus is on PBF mainly because of two reasons. First, PBF is more suitable for interactive applications compared to other algorithms. Second, our control solver can share the same PBD framework with PBF.

Our control method includes a sampling algorithm and a deforming algorithm to generate animated control particles. In previous works, voxelization is a common method for sampling mesh models. There are many algorithms [Dong et al. 2004] [Schwarz and Seidel 2010]
that were able to voxelize the surface and the internal of the model efficiently. In this paper, we do a subtle modification to the existing voxelization algorithms and produce a satisfied particle-based representation of the original mesh model. The existing works to skin a mesh model are also numerous [Lewis et al. 2000] [Kry et al. 2002] [Dionne and de Lasa 2013]. We borrow from their techniques to deform the particle-representing model with a skeleton motion data and produce a smooth skeleton-driven deformation.

3 Algorithm

The first part of our method is generating control particles. It contains two major stages: 1) sampling the model into the particle representation; 2) deforming the sampled particles using a skeleton motion data. Note that this part is performed before the simulation loop and the generated control particles are used as input for the control solver.

The core of our method is a position-based control solver that contains three different constraints: density constraint, spring constraint, and velocity constraint. The density constraint is used to drive the fluid to match the target shape while preserving details of the fluid motion. The spring constraint is used to pull fluid back to the control domain when the target is moving rapidly. The velocity constraint is capable of controlling the velocity of the flow and making fluid follow the target tightly.

3.1 Generate Control Particles

This part involves two algorithms: sampling and skeleton-driven deformation. The sampling algorithm is used to sample a target model provided by the animator to generate corresponding control particles. The skeleton-driven deformation algorithm works on the generated set of control particles, which allows the animator to mark the joints and then drive the particle set to deform based on the skeleton motion data.

To sample a mesh model, we first follow an existing voxelization method [Dionne and de Lasa 2013]. However, the result is a suboptimal representation of the original model especially with a low sampling resolution, as shown in Figure 2(top). The key reason for the poor result is that the position of the generated control particle is fixed at the center of its corresponding voxel cell. We find that a subtle modification to the basic voxelization algorithm can sidestep this problem. We first perform a naive mesh subdividing process, i.e. dividing a mesh triangle into two if its edge-length is greater than the voxel cell size, as shown in Figure 3. After this division process, the vertex located in the voxel cell is used as the location of its corresponding control particle instead of the center of the voxel cell. If multiple vertices fall into the same voxel cell, we simply average these positions. Finally, we voxelize the internal solid of the model using a seed filling algorithm. The refined particle representation of the mesh model is shown in Figure 2(bottom).

Our sampling method produces a smoother representation than traditional voxelizing techniques. Although our method can not guarantee the even distribution of the generated control particles on the surface of the model, the subsequent control solver has no strict requirement on the distribution of control particles.

To deform the generated control particles with skeleton motion data, we first need the animator to mark the position of each joint. Then, the weights of every voxel influenced by each bone are computed as follows [Dionne and de Lasa 2013]: first, computing the shortest distance $d_j^i$ between voxel $j$ and its corresponding bone $i$ using the Djikstra’s algorithm; then the weight influence $w_j^i$ of bone $i$ voxel cell $j$ is given as:

$$w_j^i = \left(\frac{1}{(1-\alpha)(d_j^i)^2} + \alpha\frac{d_j^i}{d_j^i} \right)^2$$

where $\alpha$ is a parameter allowing animators to control bind smoothness which is 0.3 in our experiment. Finally, we perform a normalizing step for the weights:

$$w_j^i = \frac{w_j^i}{\sum_k w_j^k}.$$  

The smooth character deformation after applying a skeleton motion is illustrated in Figure 4.

3.2 Density Constraint

The generated control particles actually specify the target distribution of fluid in the simulation space. To drive fluid to match a target shape, the most straightforward way is using a constant attraction force to pull fluid particles towards control particles. However, this naive method will cause a problem of fluid particles which tend to be clustering and clumping at the center of control particles and cause the control domain having a higher density than that of the free state fluid. To address this problem, Thürey et al. scaled down the attraction force according to the density of control particles [Thürey et al. 2009]. However, hard constraints are difficult to define using force-based methods.

Different from the attraction-force-based method, we use a density constraint which is in line with the incompressible nature of fluid.
The control particle treats itself and its neighboring fluid particles as a system and manages to reach a constant density by moving its neighboring fluid particles, as illustrated in Figure 5.

Our idea is inspired by PBF [Macklin and Müller 2013] which solves a density constraint for each fluid particle associated with its neighbors to enforce incompressibility. Differing from their work, we introduce the control particles to the simulation and apply a density constraint to each control particle with its neighboring fluid particles.

In this paper, the \( i \)th control particle is denoted by its position \( P_i \), and the \( j \)th fluid particle is represented by its position \( p_j \). The density constraint on the \( i \)th control particle is formulated as:

\[
C_i(P_i, p_1, \ldots, p_n) = \frac{\bar{\rho}_i}{\bar{\rho}_0} - 1
\]

\[
\bar{\rho}_i = \sum_j m_j W(P_i - p_j, H)
\]

where \( \bar{\rho}_0 \) is the rest density of the control particles, and \( \bar{\rho}_i \) corresponds to the fluid density in the support radius of the \( i \)th control particle. \( \bar{\rho}_i \) is defined using the standard SPH density estimator. \( m_j \) denotes the mass of fluid particle \( j \). We treat it as a unit value, so it is omitted in the subsequent equation. \( H \) represents the support radius of control particles. The controllability is stronger with larger support radius of the control particles.

The state equation (3) formulates a non-linear density constraint for each control particle. So a displacement represented by \( \Delta p_i \) for each fluid particle is needed to make equation (3) satisfied as below:

\[
C_i(P_i, p_1 + \Delta p_1, \ldots, p_n + \Delta p_n) = 0
\]

\[
\Delta p_j = \hat{\lambda}_i \nabla p_j C_i(P_i, p_1, \ldots, p_n)
\]

where \( \hat{\lambda}_i \) is a scalar value for constraint \( i \) and each constraint has the same \( \hat{\lambda} \) for all the fluid particles under this constraint. The \( \hat{\lambda}_i \) is given by:

\[
\hat{\lambda}_i = -\frac{C_i(P_i, p_1, \ldots, p_n)}{\sum_j |\nabla p_j C_i(P_i, p_1, \ldots, p_n)|^2 + \varepsilon}
\]

where \( \varepsilon \) is a soften coefficient which avoids dividing by a very small value and making simulation unstable. And the derivation of constraint \( C_i \) for the \( j \)th fluid particle is given as [Monaghan 1992]:

\[
\nabla p_j C_i = -\frac{1}{\bar{\rho}_0} \nabla p_j W(P_i - p_j, H)
\]

Finally, we sum up all the displacements exerted by control particles on a fluid particle to obtain the density constraints:

\[
\Delta p_i^{density} = -\alpha \frac{1}{\bar{\rho}_0} \sum_j \hat{\lambda}_j \nabla p_j W(P_j - p_i, H)
\]

Here, we incorporate the \( \alpha \) ranging from 0 to 1 for each control particle to represent the intensity of density constraints.

### 3.3 Spring Constraint

It is a challenging task to control the fluid simulation matching a rapidly changing object. Solving this problem by simply setting a strong constraint will lead to obvious artificial viscosity. As a result, the controlled fluid will look like a rigid body and lose rich fluid details.

Compared to the grid-based fluid simulation framework, this problem is more difficult for the particle-based control method. The reason is that particles have a finite length of the support radius.

Figure 4: Illustration of the output of the sampling and deforming algorithms in Section 3.1: the original mesh model (top left), the sampling result (top right), the weighting result using different color (bottom left), and the smooth skeleton driven deformation result (bottom right).

Figure 5: This figure illustrates our density constraint for a control particle (green). In its support radius (grey), when the fluid density is smaller than its rest density, the fluid particles (yellow) will be attracted towards the control particle (left); otherwise, the fluid particles will be rejected from the center (right).
This figure illustrates how the spring constraints are used to cope with the fast moving target. **Left:** When the fluid particles (yellow) are inside the control domain (light blue), they find a reference control particle respectively. **Right:** Once the fluid particles are far away from the control domain, the spring constraints (red arrow) will help to pull them back. We also find that other constraints that based on SPH formulation are unable to work, because the influence area of control particles can not reach them.

Once the target moves out of the radius in some certain frames, the constraints of control particles will have no influence on fluid particles, which makes fluid particles out of control (Figure 6). Here, we propose a novel method which is suitable for particle-based fluid simulation framework, to sidestep this long-standing problem. We add an adaptive spring constraint between a fluid particle and its nearest control particle which we refer to as the reference. When the fluid particle is far away from its reference, the spring constraint will pull the fluid particle back to its reference. Otherwise, the spring constraints have little impact on the fluid particles and thus will not introduce side effects such as artificial viscosity.

In order to formulate a spring satisfying above features, we use the following algorithm to add and update the spring of each fluid particle.

**Algorithm 1 Update the references of the fluid particles**

1: for all fluid particles $f_i$ do
2: find its nearest control particle $c_j$
3: if $c_j$ exists and Distance($f_i, c_j$)$\leq d/2$ then
4: update reference $r_i \leftarrow j$
5: reset life time $m_i \leftarrow T_{life}$
6: end if
7: if $m_i > 0$ then
8: damp spring $m_i \leftarrow m_i - t_{damp}$
9: apply spring constraint
10: end if
11: end for

Here, $d$ represents the maximum sampling interval for control particles and is equal to the support radius in our experiment. When the fluid particle is in the control area, it will update its reference continuously and the corresponding spring length will be always less than half of $d$. The intensity of spring constraint is defined as:

$$f(x) = \frac{1}{1 + e^{b - ax}}$$ (10)

This formulation is widely used as the logistic regression function in machine learning. We use this function for its advantage of having an upper and lower bound (Figure 7). Additionally, by tuning the two parameters, $a$ and $b$, we can easily make the spring constraints have little or no influence on the fluid particles in the control area. Once the fluid particles get far away from the control region, they will not be able to find a nearest control particle and thus will not update their reference. In this situation, the spring constraints will take effects significantly and pull the fluid particles back to the control area immediately. For the upper bound of $f$, the intensity of our spring constraints will not grow unlimited, thus avoid causing unstability to the simulation.

In addition, we define a life time for each spring constraint and there is a little time elapse per frame. Based on this property, we can achieve the visual effects like droplets, because the spring constraints will not keep alive forever and part of fluid particles will get rid of the control. In our experiment, we set the life time as a random value within a fixed range.

Together, this simple but dedicated spring constraints allow us to control fluid simulation to follow a rapidly changing target while keeping rich visual details. Figure 8 illustrates the intensity of our spring constraint by a 2D example.
\[ \Delta p^{\text{spring}}_i = \beta f(|r_i - p_i|) \frac{(r_i - p_i)}{|r_i - p_i|} \]  

where \( \beta \) is a parameter used to adjust the intensity of the spring constraint. \( r_i \) represents the reference of the fluid particle \( i \). Certainly, if a fluid particle has no reference, the spring constraint is not applied.

### 3.4 Velocity Constraint

The velocity constraint is used to modify the flow of fluid. In some scenarios, animators may hope to control fluid to flow along a specified path or simulate a stiff-like fluid. In such cases, the above two constraints are not sufficient. A velocity constraint used to modify the entire flow is necessary. It can drive fluid particles to follow the target shape’s movement tightly.

For each fluid particle, we interpolate its velocity constraint exerted by its nearby control particles.

\[ \Delta v_i = \frac{\sum_j V_j W(p_i - P_j, H)}{\sum_j W(p_i - P_j, H)} \]  

where \( V_j \) refers to the velocity of the \( j \)th control particle, and \( v_i \) is the velocity of the fluid particle. Then, the displacement of velocity constraints is given as:

\[ \Delta p^{\text{velocity}}_i = \gamma \Delta t (\Delta v_i - v_i) \]  

where \( \gamma \) is the intensity of the velocity constraints.

As mentioned earlier, the most relevant work [Thüray et al. 2009] to our method mainly uses this velocity constraint to control the overall flow. To avoid suppressing small scale details of the fluid, they apply this constraint only on the low-frequency part of the fluid velocity.

Here in our method, the velocity constraint is just auxiliarily used. Generally, it will not be defined too hard unless hoping to produce a stiff-like fluid model. Thus, we simply impose the velocity constraint on the overall velocity field and omit the velocity-decomposing step.

### 4 Implementation

We integrate our control solver into the PBF fluid simulation framework. The simulation loop is outlined by Algorithm 2. Besides integrating our control solver from line 8 to line 21, the other parts are the same to the original PBF.

Same to most of the particle-based fluid simulation, the primary computational time is spent on finding the neighboring particles. In our method, each control particle needs to find its neighboring fluid particles to calculate its density, and each fluid particle also needs to find its neighboring control particles to accumulate the control displacements. We use the method of [Green 2008] for neighbor finding. By choosing the length of control particles’ support radius to be integer multiple of that of fluid particles, we omit the steps of recomputing the grid hash of fluid particles and sorting them. In addition, only when the control particles have movements, we need to recompute the control particles neighbors for fluid particles. As a consequence, our control solver will only increase a little extra computation time to the original fluid simulation.

### 5 Results

We implement our control algorithm and PBF using C++ and CUDA 4.2. All our examples except in Figure 1 are rendered in real-time using the Isosurface Raycasting method [Kruger and Westermann 2003]. The fluid density field is generated by splatting fluid particles onto a regular grid with the Poly6 kernel function [Müller et al. 2003]. We use an environment cubemap to produce refraction and reflection effects. All of our simulations run on a NVIDIA GTX 780 GPU.

Figure 9 shows that the fluid is under control to form a static bunny model automatically. User can exert external force on the model and move it interactively. The controlled fluid will deform drastically to produce plausible visual effects during the process of tracking and matching of the model. The large scale of both the control particle number and fluid particle number demonstrates the scalability of our control method.

In Figure 1, we demonstrate our method by controlling a dancing character made of fluid. Even if the movement of control particles are very large between frames, our algorithm is still very stable and manages to keep fluid under control. Additionally, by tuning the velocity constraint parameter, we can animate from low-viscosity fluid to stiff-like fluid. To achieve the high quality rendering results in Figure 1, we use Mitsuba [Jakob 2010] for offline rendering.

In Figure 10, our control method is used for controlling fluid to follow a path. In this example, we first make a heart-like curved path model using modeling software. After sampling the model into control particles, we specify a velocity for each control particle on the path. In our result, the velocity constraint is very effective to drive the fluid to flow along the specified path.

Figure 11 is an example of controlling fluid to match different mod-
Figure 9: The fluid is controlled to form the Stanford’s bunny (top row). The bottom row is rendered as particles, and the controlled fluid model is hit by a solid sphere (bottom right).

Figure 10: Path control example: the user specifies a flow path and the fluid is controlled to flow along this path. When all the control constraints are removed, the fluid falls down (bottom right).

Table 1 summarizes timings of our control algorithm in different scenarios. Our control solver usually takes 7-25% of the total computational time. The number of control particles and the support radius are the two main factors affecting the performance of our control solver. In all of our cases, we set the control particle’s support radius two times of the fluid particle’s support radius.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Control Particles</th>
<th>Fluid Particles</th>
<th>Control Time[msec]</th>
<th>Total Time[msec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny</td>
<td>63k</td>
<td>480k</td>
<td>7.30</td>
<td>28.57</td>
</tr>
<tr>
<td>Dancer</td>
<td>17k</td>
<td>100k</td>
<td>2.25</td>
<td>9.10</td>
</tr>
<tr>
<td>Heart</td>
<td>4k</td>
<td>120k</td>
<td>0.69</td>
<td>8.63</td>
</tr>
<tr>
<td>Multi-models</td>
<td>15k</td>
<td>90k</td>
<td>1.59</td>
<td>8.65</td>
</tr>
</tbody>
</table>

6 Conclusion and Future Works

In this paper, we describe a simple and effective fluid control method to drive fluid simulation to follow and match a rapidly changing target in an artistic yet naturalistic way. Our method does not require laborious model processing and pre-simulation, and only need the user to provide the mesh model and the skeleton motion data. An animated model made of fluid can be produced automatically. The combination with PBD framework makes the method fast and stable. It can be integrated into the existing fluid simulation framework easily and will only add a small ratio of computation time to the original simulation. The technique is efficient and easy to implement, which has strong potential to be applied into industry include movies, animations, games and other interactive applications.

There are several possible improvements for the future. Firstly, the support radius of control particles is currently unified. Using the techniques similar to adaptive SPH [Adams et al. 2007] [Hong et al. 2008] [Orthmann and Kolb 2012], setting different length of support radius according to the density or motion of the control particles will decrease the number of control particles to some extent. Secondly, the current spring constraint in our method has a limit that the control particles should keep a consistent index during the simulation. Combining the shape-matching method [Müller et al. 2005] to modify our spring constraint model may be a promising direction.

Acknowledgements

The authors would like to thank anonymous reviewers for their valuable comments. This work was supported by the National Natural Science Foundation of China (Grant No. 61373085 and 61173105).

References


