View-Dependent Control of Elastic Rod Simulation for 3D Character Animation

Yuki Koyama*
The University of Tokyo

Takeo Igarashi†
The University of Tokyo

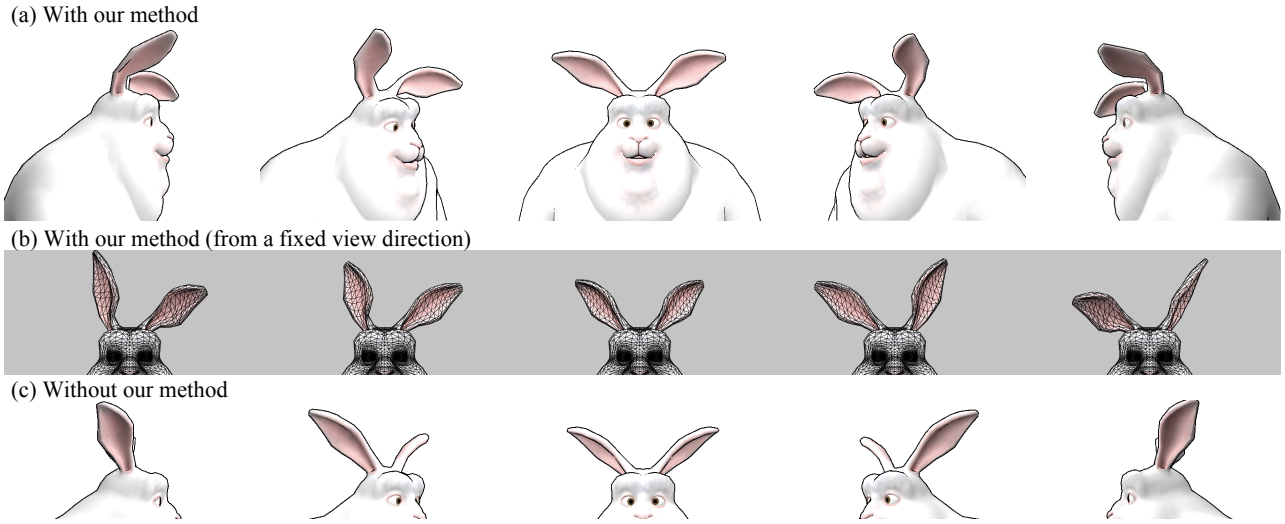

Figure 1: *(a) An animation of the camera moving around Bunny. His swinging ears are view-dependently simulated by our method so that they always have the best configurations according to each view direction. (b) The same scene from a virtual fixed view direction. (c) An animation of the same scene without our method.*

Abstract

This paper presents view-dependent control of elastic rod simulation for 3D character animation. Elastic rod simulation is often used in character animation to generate motion of passively deforming body parts such as hair, ear, and whiskers. Our goal is to allow artistic control of the simulation in a view-dependent way, for example to move a hair strand so that it does not hide the eye regardless of the view direction. To achieve this goal, the artist defines several example rest poses of the rod in preparation, each of which is associated with a particular view direction. In run time, the system computes the current rest pose by blending the example rest poses associated with the view directions near the current view direction, and then pulls the pose to the current rest pose. Technical contribution is in the formulation of example-based rod simulation using view direction as an input, and an algorithm to suppress undesirable increase of momentum caused by dynamically changing rest poses.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: example-based materials, view-dependent control, elastic simulation

*e-mail:koyama@is.s.u-tokyo.ac.jp
†e-mail:takeo@acm.org

1 Introduction

3D animations that look like 2D traditional cell animations (e.g. the movie *009 RE:CYBORG* [2012]) have become popular thanks to the advancement in sophisticated cartoon shading methods. In order to imitate 2D-like animations using 3D, or to realize the know-how of 2D animations in 3D, it is important to stylize contents according to view directions; for example, Ramemacher [1999] presented a method to change object geometry depending on the view direction. Such view-dependent stylizations or adjustments are quite common in creating 2D animations.

In this paper, we present a method to control elastic rod simulation in a view-dependent way. Elastic rod simulation is often used in character animation to generate motion of passively deforming body parts such as hair, ear, and whiskers. The artist defines the motion of the active body parts such as arms, legs, and head, and the motion of these passive body parts are generated by physical simulation. Fig. 2(a) shows examples of elastic rods used in character animation. Physical simulation frees the artist from manually specifying the motion, but at the same time makes artistic control difficult. Our goal is allow artistic control in such a simulation process.

Our method works as follows. The artist first defines several example rest poses of the rod, each of which is associated with a particular view direction. At run time, the system computes the current rest pose by blending the example rest poses associated with the view directions near the current view direction. The system then runs the simulation using the computed current rest pose. The overall framework is based on example-based materials [Martin et al. 2011], but we present multiple extensions to make it work appropriately in our problem setting. Specifically, we present a formulation of example-based material for rod simulation and a method to blend example rest poses using the view direction as an input. We also present an

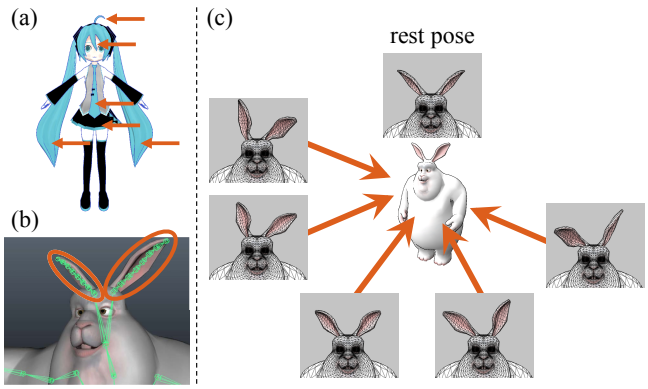

Figure 2: *(a) Examples of elastic rods in a 3D character. (b) Examples of joint chains. (c) Examples of additional user input for our method. The input consists of an arbitrary number of (here 5) example poses of joint chains, each of which is associated with a view direction and an extent (magnitude of its influence, here not showed).*

algorithm to suppress undesirable increase of momentum caused by dynamically changing rest poses.

1.1 Related Work

View-Dependent Control. View-dependent control is considerably important, especially for creating cartoon-like highly stylized animations. For this purpose, Rademacher [1999] proposed View-Dependent Geometry (VDG), which allows a 3D surface mesh to adapt its shape according to view directions. To achieve this, in addition to the base surface mesh, users make additional deformed surface meshes for each key view direction. At run time, they dynamically blend the input meshes by weights calculated from view directions. There are some inspired view-dependent animation systems such as [Chaudhuri et al. 2004; Chaudhuri et al. 2007]. To our knowledge, our method is the first one that attempts view-dependent control for physical simulation.

Example-Based Material. The concept of example-based materials was first proposed by Martin et al. [2011] based on the elastic simulation by Finite Element Method (FEM). The basic idea is, given some desired deformed poses as additional input, they define an *example manifold* as a non-linear interpolation space of input poses. This example manifold is considered a pose space of the desired deformations, and used for calculating the rest pose of the elastic materials. Koyama et al. [2012] presented an implementation of example-based materials using shape matching dynamics [Müller et al. 2005] to achieve real-time simulation.

Extending this concept, Schumacher et al. [2012] introduced the concept of *explicit weight control* as an attractive application of example-based materials. The original Martin et al.'s work produces passive materials, which means the blending weights are calculated by only the conditions of materials, while the weights in explicit weight control can be calculated from any other conditions; for example, velocities, joint angles, and so on. Our idea is to use this explicit weight control concept with the weights calculated from view directions.

There is a method for generating locomotions of elastic objects [Coros et al. 2012], which exploits the concept of example-based materials in the algorithm. Their goal is making elastic objects as if they were alive. In our case, the targets, such as hair, are usually not alive. Therefore, we present an algorithm to suppress the effect that makes the object look alive (section 4).

2 User Input

Our method takes a skinned mesh as input, in which a surface mesh deforms according to the configuration (pose) of a skeleton. The skeleton consists of active joints and passive joints in our representation. Active joints represent main body parts such as arms and legs, and passive joints represent secondary body parts such as hairs and ears. The artist explicitly specifies the motion of active joints (mainly by key framing), and the system computes the motion of passive joints via physical simulation. We group a set of connected passive joints together and call it as a *joint chain*. We assume that the base of a joint chain is connected to an active joint, and the end of the joint chain is not connected any joint. We also do not handle a joint chain with branches or loops. Fig. 2(b) shows examples of joint chains. In the rest of this paper we will discuss how to compute the motion of a joint chain.

Our method also takes an arbitrary number of view-dependent example poses for each joint chain as input (see Fig. 2(c)). Each example pose is associated with a view direction and an extent (magnitude of its influence). Note that each example represents a shape (vertex positions) of a mesh in the original VDG, while each example represents a pose of a joint chain in our method.

Given a pose of a joint chain consisting of a set of N joints, the pose $\mathbf{P}$ can be represented as:

$$\mathbf{P} = (\mathbf{X}, \mathbf{Q}), \quad (1)$$

$$\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N), \quad (2)$$

$$\mathbf{Q} = (\mathbf{q}_1, \ldots, \mathbf{q}_N), \quad (3)$$

where $\mathbf{x}_i \in \mathbb{R}^3$ and $\mathbf{q}_i \in \mathbb{R}^4$ are the position of the i-th particle in Eulerian coordinates and its orientation in quaternion representation, respectively. Here, the 1st joint is attached to an active joint (base joint). Note that the positions and the orientations are represented in an absolute way, not relative to the parent joint. We omit the scale information for our definition of poses because it is seldom used for posing; however, our method could be extended to deal with scale information if needed.

The view direction is represented as a unit 3-dimensional vector $\mathbf{d}$: the direction of the camera as seen from the base joint in its local coordinate frame. Our definition of the view direction does not include other camera parameters such as the tilt, pan, roll, angle of view, and the distance between the camera and the model.

Convention. we denote $\mathbf{P}^j = (\mathbf{X}^j, \mathbf{Q}^j)$, $\mathbf{d}^j$, and M as the j-th example pose, view direction, and the number of example poses, respectively. When we use $j = 0$, it means the base input.

3 View-Dependent Rod Simulation

In this section, we explain how to simulate example-based materials in real-time rod simulation controlled in a view-dependent way. The basic idea is to exploit the concept of explicit weight control [Schumacher et al. 2012] and calculate the blending weights similarly to VDG [Rademacher 1999]. We make non-trivial extensions to these methods to make it work appropriate in our problem setting. First, we explain how to blend the example poses in rod simulation. Second, we discuss how to compute the blending weights from the given view direction.

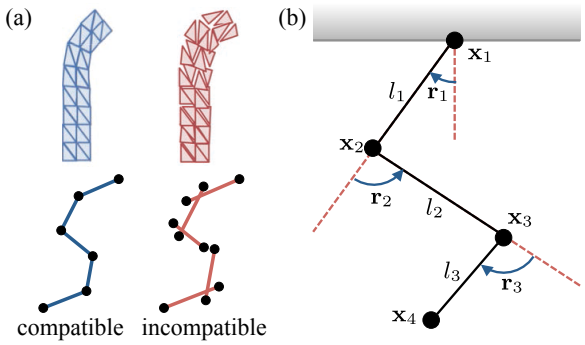

Figure 3: *(a) 2D and 1D examples of compatible (left) and incompatible (right) poses. Incompatible poses cannot be used in our case. (b) To blend input poses, our method use the rotations* $\mathbf{r}_1, \ldots, \mathbf{r}_{N-1}$. *Here,* $N = 4$.

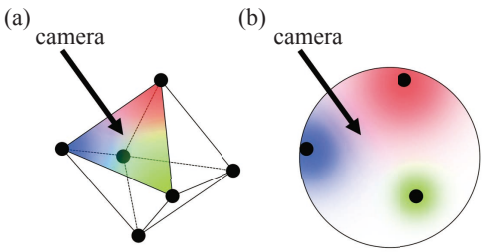

Figure 4: *(a) Rademacher's algorithm for calculating blending weights from view directions. They use barycentric coordinates of a triangle as weights. (b) Our algorithm. We use Gaussian functions on a sphere surface. Note that if the total of the Gaussian weights is less than 1 (the white area on the sphere), we give the remaining weight to the base pose.*

3.1 Example-Based Rod Simulation

We use Oriented Particles (OP) [Müller and Chentanez 2011] as the rod simulation framework. It can simulate elastic solid, shell, and rod in the same framework in real time. As OP is based on Position-Based Dynamics (PBD) [Müller et al. 2007], our method is also implemented upon PBD. For further information of PBD, please refer to [Bender et al. 2013]. For ensuring inextensibility of rods, which is often expected in 3D characters' elastic parts, we use distance constraints presented in [Müller et al. 2007]. The definition of local regions of OP is arbitrary in our method; if you make the region larger, the rod becomes stiffer.

Interpolation of example poses is not a trivial problem. Martin et al. [2011] use non-linear interpolation in their method, because simple linear interpolation of their deformation descriptor breaks the *compatibility* of the deformation, which is undesired in their method. This non-linearity makes the computation cost expensive. Schumacher et al. [2012] and Koyama et al. [2012] proposed methods that allow *incompatible* interpolated results so that they can use simple linear interpolation, which makes their methods faster. Fig. 3(a) shows examples of compatible and incompatible poses, and please refer to [Schumacher et al. 2012; Koyama et al. 2012] for details. In our case, we cannot handle incompatible interpolated results because of the *ghost force* problem, which is mentioned in [Müller and Chentanez 2011]. At the same time, our interpolation should be linear because of the computational cost, and the interpolated poses should preserve the lengths for ensuring inextensibility.

We therefore use relative rotations as the interpolation space. Using rotations allows us to interpolate linearly, and we can always reconstruct a compatible pose from relative rotations by using forward kinematics. Here we assume that the lengths between neighboring joints $l_1, \ldots, l_{N-1}$ are constant.

Specifically, given blending weights $\mathbf{w} = (w_0, \ldots, w_M)$, the result of linear blending $\mathbf{P}(\mathbf{w})$ can be written as

$$\mathbf{P}(\mathbf{w}) = (\mathbf{X}(\mathbf{w}), \mathbf{Q}(\mathbf{w})), \tag{4}$$

$$\mathbf{X}(\mathbf{w}) = \mathrm{LB}_{\mathbf{X}}(\mathbf{w}; \mathbf{X}^1, \ldots, \mathbf{X}^M), \tag{5}$$

$$\mathbf{Q}(\mathbf{w}) = \mathrm{LB}_{\mathbf{Q}}(\mathbf{w}; \mathbf{Q}^1, \ldots, \mathbf{Q}^M), \tag{6}$$

where $\mathrm{LB}_{\mathbf{X}}$ means the linear blending of $\mathbf{X}$s, and $\mathrm{LB}_{\mathbf{Q}}$ means the linear blending of $\mathbf{Q}$s.

For the algorithm of $\mathrm{LB}_{\mathbf{X}}$, we use relative rotations as interpolation space (see Fig. 3(b)). Here we denote $\mathbf{x}_{i,i'} = \mathbf{x}_i - \mathbf{x}_{i'}$ for simplicity. First, for each pose $j = 0, \ldots, M$ and each joint $i = 1, \ldots, N-1$, the angle and axis of the rotation that makes the direction of $\mathbf{x}^j_{i,i-1}$ into the direction of $\mathbf{x}^j_{i+1,i}$ can be calculated as

$$\theta^j_i = \cos^{-1} \frac{\langle \mathbf{x}^j_{i,i-1}, \mathbf{x}^j_{i+1,i} \rangle}{\|\mathbf{x}^j_{i,i-1}\| \|\mathbf{x}^j_{i+1,i}\|}, \quad \mathbf{e}^j_i = \frac{\mathbf{x}^j_{i,i-1} \times \mathbf{x}^j_{i+1,i}}{\|\mathbf{x}^j_{i,i-1} \times \mathbf{x}^j_{i+1,i}\|}. \tag{7}$$

Then, from the angle and axis, the rotation can be obtained as a quaternion

$$\mathbf{r}^j_i = \cos \frac{\theta^j_i}{2} + \mathbf{e}^j_i \sin \frac{\theta^j_i}{2}. \tag{8}$$

$\mathbf{x}^j_{1,0}$ is constant, for example $\mathbf{x}^j_{1,0} = (1, 0, 0)$, for any pose j. In addition, we set $\mathbf{r}^j_i$ directly to $(1; 0, 0, 0)$ if $|\theta^j_i| < \epsilon$ for stability reasons. Now, our pose descriptor for interpolation can be described as

$$\mathbf{R}^j = (\mathbf{r}^j_1, \ldots, \mathbf{r}^j_{N-1}). \tag{9}$$

In order to blend these descriptors with $\mathbf{w}$ and obtain $\mathbf{R}(\mathbf{w})$, we apply QLERP (the linear interpolations of quaternions) [Kavan and Žára 2005] for each joint. After blending, $\mathbf{X}(\mathbf{w})$ can be obtained using forward kinematics from $\mathbf{R}(\mathbf{w})$ and $l_1, \ldots, l_{N-1}$.

In contrast to $\mathrm{LB}_{\mathbf{X}}$, the algorithm of $\mathrm{LB}_{\mathbf{Q}}$ can be simple; $\mathbf{Q}(\mathbf{w})$ can be obtained by simply using QLERP for each joint.

3.2 View-Dependent Control

As with VDG [Rademacher 1999], we determine the blending weights $\mathbf{w}$ by view directions. However, we use a different scheme for weight computation.

In Rademacher's algorithm, they consider a triangle mesh that covers the model, where each vertex corresponds to the input key view direction. In runtime, the system finds a triangle enclosing the given current view direction, and then uses its barycentric coordinates as weights to blend example shapes (Fig. 4(a)). However, this method is inconvenient for artists because it is necessary to cover the model entirely with the triangle mesh (at least 4 vertices) and it is not easy to arbitrarily change the extent (influence) of each example.

We therefore use scattered data blending for the computation of blending weights. Given the current view direction $\mathbf{d}$, the system first computes the Gaussian weight for each example, $\phi_j(x) = \exp(-(x/\alpha_j)^2)$ where x is the angle between the two directions $\mathbf{d}$ and $\mathbf{d}^j$, and $\alpha_j > 0$ is the user defined extent of the example (for example, we use $\alpha_j = 0.5$ and 1.3 to make our results). We then blend the example poses with these Gaussian weights. If the total of the Gaussian weights exceeds 1, then the system modulates the

weights so that total equals one. If the total is less than 1, we give the remaining weight to the base pose $\mathbf{P}^0$. Algorithm 1 describes these computations. This algorithm returns the pose similar to $\mathbf{P}^j$ when $\mathbf{d}$ is close to $\mathbf{d}^j$, and returns a pose similar to $\mathbf{P}^0$ when $\mathbf{d}$ is far away from all examples.

Algorithm 1 Calculating weights from view directions.

```
for j = 1 to M do
    angle ← ArcCos(DotProduct(d^j, d));
    w_j ← φ_j(angle);
end for
sum ← w_1 + ··· + w_M;
if sum ≤ 1.0 then
    w_0 ← 1.0 − sum;
else
    for j = 1 to M do
        w_j ← w_j / sum;
    end for
    w_0 ← 0.0;
end if
```

4 Suppression of Ghost Momentum

The direct application of the algorithm described in the previous section causes *ghost momentum*, that is, the joint chain gains unnecessary momentum as the user changes the view direction. This makes the joint chain look alive, which is undesirable for passive body parts such as hairs and whiskers. A naive approach to suppress such ghost momentum is to use damping. However, it does not work well because damping also suppresses the momentum caused by external forces and internal forces that pull the current pose towards the example manifold, making the entire animation unnaturally slow and dull.

In order to selectively suppress the ghost momentum caused by the change of the rest pose, we modify the velocity update step in the PBD formulation. The idea is to use different preferred positions for position updates and velocity updates. In generalized PBD for OP [Müller and Chentanez 2011], each particle is updated by

$$\mathbf{v} \leftarrow (\mathbf{x}_p - \mathbf{x})/h, \tag{10}$$
$$\mathbf{x} \leftarrow \mathbf{x}_p, \tag{11}$$
$$\boldsymbol{\omega} \leftarrow \mathrm{axis}(\mathbf{q}_p\mathbf{q}^{-1}) \cdot \mathrm{angle}(\mathbf{q}_p\mathbf{q}^{-1})/h, \tag{12}$$
$$\mathbf{q} \leftarrow \mathbf{q}_p, \tag{13}$$

where $\mathbf{x}_p, \mathbf{q}_p, h$ are the estimated position, orientation, and the time step, respectively. Please refer to [Müller et al. 2007; Müller and Chentanez 2011] for the details. In our algorithm, we replace Equation 10 and 12 with

$$\mathbf{v} \leftarrow (\mathbf{x}'_p - \mathbf{x})/h, \tag{14}$$
$$\boldsymbol{\omega} \leftarrow \mathrm{axis}(\mathbf{q}'_p\mathbf{q}^{-1}) \cdot \mathrm{angle}(\mathbf{q}'_p\mathbf{q}^{-1})/h, \tag{15}$$

where $\mathbf{x}'_p, \mathbf{q}'_p$ are the newly introduced position and orientation, which can suppress the undesired increase of momentum. Note that the modifications are on the update of velocity $\mathbf{v}$ and angular velocity $\boldsymbol{\omega}$. The position $\mathbf{x}$ and orientation $\mathbf{q}$ are updated the same way as generalized PBD for OP.

Fig. 5 describes the concept of our algorithm. The key idea is the decomposition of the internal elastic force into two forces: the force that pulls the current pose $\mathbf{P}$ towards the nearest pose $\mathbf{P}^{\mathrm{proj}}$ in the example manifold, and the other force that pulls $\mathbf{P}^{\mathrm{proj}}$ to the modified rest pose $\mathbf{P}(\mathbf{w})$. The latter force is the main cause of the ghost momentum, so we only use the former force for velocity update.

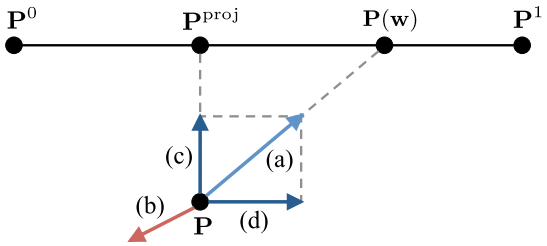

Figure 5: *The concept of our algorithm to suppress ghost momentum. There is always an internal force (a) that pulls the current deformed pose $\mathbf{P}$ towards the goal pose $\mathbf{P}(\mathbf{w})$. There is an external force (b), too. We decompose the internal force to the force that makes the pose move to the example manifold (c) and the force that is caused roughly by changing the weights for blending the poses (d). Finally, we use only (b) and (c) for the velocity updates.*

Note that we still use $\mathbf{P}(\mathbf{w})$ for positional update so that the pose approaches to the blended rest pose. That is, $\mathbf{x}_p$ and $\mathbf{q}_p$ for position updates are computed using $\mathbf{P}(\mathbf{w})$ as the target position also considering the external forces and inertia, while $\mathbf{x}'_p$ and $\mathbf{q}'_p$ for velocity updates are computed using $\mathbf{P}^{\mathrm{proj}}$ as the target position.

$\mathbf{P}^{\mathrm{proj}}$ is a given as a linear projection of the current pose $\mathbf{P}$ towards the example manifold defined by the example poses $\mathbf{P}^j$. Specifically, $\mathbf{P}^{\mathrm{proj}}$ is defined as $\mathbf{P}(\mathbf{w}^{\mathrm{proj}})$ (see Equations 4-6) and we compute $\mathbf{w}^{\mathrm{proj}}$ as the weights that minimize a quadratic energy

$$\mathbf{w}^{\mathrm{proj}} = \arg\min_{\mathbf{w}} \left\| \sum_{k=0}^{M} w_k \mathbf{R}^k - \mathbf{R} \right\|^2, \tag{16}$$

where $\sum_{k=0}^{M} w_k = 1$, and it can be obtained by

$$(w_1^{\mathrm{proj}}, \ldots, w_M^{\mathrm{proj}})^T = \left(\mathbf{L}^T\mathbf{L}\right)^{-1}\mathbf{L}^T\left(\mathbf{R} - \mathbf{R}^0\right) \tag{17}$$
$$w_0^{\mathrm{proj}} = 1 - \sum_{k=1}^{M} w_k^{\mathrm{proj}} \tag{18}$$

where $\mathbf{R}, \mathbf{R}^j \in \mathbb{R}^{4(N-1)}$ are the concatenations of the elements of quaternions from Equation 9, and $\mathbf{L} = (\mathbf{R}^1 - \mathbf{R}^0, \ldots, \mathbf{R}^M - \mathbf{R}^0) \in \mathbb{R}^{4(N-1)\times M}$. This projection process is similar to [Koyama et al. 2012], but these is a subtle difference in the post processing. They modify the result of projection so that it stays within the convex combination of the examples. That is, they clamp all the weights between 0 and 1, while maintaining the total to be 1. On the other hand, we modify the result of projection $\mathbf{w}^{\mathrm{proj}}$ so that it remains sufficiently close to the weights for the current rest pose $\mathbf{w}$. We do this to avoid visual artifacts caused by the inconsistency between positional update (by $\mathbf{w}$) and velocity update (by $\mathbf{w}^{\mathrm{proj}}$). We omit the details of this implementation, but in short, we set a sliding window around $\mathbf{w}$, and force the $\mathbf{w}^{\mathrm{proj}}$ to stay within the sliding window. We also adjust the values so that the total equals to 1.

Note that all the quaternions should satisfy $\langle \mathbf{q}, \mathbf{q}_{\mathrm{pivot}} \rangle > 0$ with a pivot quaternion $\mathbf{q}_{\mathrm{pivot}}$ before the projection. We use $\mathbf{q}_{\mathrm{pivot}} = \mathbf{r}_1^0$. If not, we replace $\mathbf{q}$ with $-\mathbf{q}$. This is necessary for ensuring the interpolation paths of the example manifold are the shortest ones, analogous to QLERP [Kavan and Žára 2005].

Figure 6: *The top part of the character's hair is always facing the camera in the best configuration, even when the character moves or rotates.*

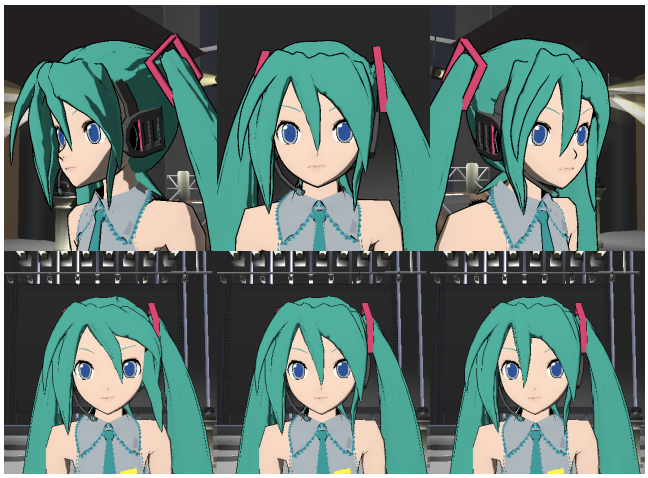

Figure 7: *The front hair attempts to avoid the eyes. The top row shows the camera view. The bottom row shows a fixed view.*

5 Results and Discussions

Fig. 1, 6, and 7 show the results of our method[1]. The conditions and performance are shown in Tab. 1. Our current implementation works on Unity [2013], and the results are exported as interactive games. All animations are generated at over 60 FPS, but it is difficult to accurately measure the performance of each computation.

In practice, ghost momentum can be ignored when the elastic part is moving constantly, for example, being swayed by wind. In that case, it is better not to use our algorithm of suppression because it doubles the computational cost.

Limitations. There are several limitations in our method and implementation. Our current method cannot handle a structure that has loops or branches. The algorithm of suppression cannot completely suppress the increase of the momentum (see the supplemental video, 01:35). View-dependent behavior is inherently incompatible with physical law, and thus our formulation lacks physical validity.

Discussions. As we mentioned above, the suppression algorithm cannot completely solve the problem of ghost momentum. It would be possible to explore the alternatives to this algorithm; for example, taking view directions into account more directly for updating velocity. Another possibility is to avoid the use of the example-based elasticity in the first place in order to achieve the view-dependence; for example, using direct constraints on position and velocity based on view directions in addition to the standard elastic rod simulation.

[1]The 3D models are provided by Blender Foundation (Fig. 1), Yamamoto (Fig. 6), and Kio (Fig. 7), and we modified them for simplicity. Big Buck Bunny is copyrighted by Blender Foundation. © Blender Foundation. Hatsune Miku is copyrighted by Crypton Future Media, INC. © Crypton Future Media, INC.

Table 1: *The conditions and performance of the animations. The numbers put in parentheses indicate the total number of joint chains including those simulated without our method (by pure OP). FPS indicates the frames per second of the entire system including simulation, rendering, and so on. Note that all animations are generated in over 60 FPS using a consumer-level laptop.*

model	#joint chains	#particles	#examples	FPS
Bunny	2 (2)	8, 8	5, 5	60+
Front hair	3 (6)	7, 5, 7	3, 5, 1	60+
Top of head	1 (4)	3	5	60+

Conclusion. In this paper, we proposed a method for controlling elastic rod simulations by view directions for real-time 3D character animations. View-dependent control could help the artists to achieve traditional 2D-like stylizations, or to adjust results of simulation more effectively according to view directions. Our method is fast enough to be used in real-time environments such as games. The algorithm for suppressing undesired increase of momentum is empirically derived and lacks physical validity, but it is very practical and generates visually plausible results in our experiments.

Acknowledgements

We would like to thank Yasunori Harada and Nobuyuki Umetani for many useful discussions, and anonymous reviewers for their helpful comments. We also thank Yamamoto and Kio for their model data and the permission for use of them, Crypton Future Media, Inc. for the permission for use of the character, Hatsune Miku, and Blender Foundation for the Bunny data under a Creative Commons Attribution 3.0 License. This project was funded in part by grants from the Japanese Information-Technology Promotion Agency (IPA).

References

009 RE:CYBORG PRODUCTION COMMITTEE. 2012. 009 re:cyborg. http://009.re-cyb.org/.

BENDER, J., MULLER, M., OTADUY, M. A., AND TESCHNER, M. 2013. Position-based methods for the simulation of solid objects in computer graphics. In *EG 2013 - STARs*, Eurographics Association, 1–22.

CHAUDHURI, P., KALRA, P., AND BANERJEE, S. 2004. A system for view-dependent animation. *Computer Graphics Forum 23*, 3, 411–420.

CHAUDHURI, P., KALRA, P., AND BANERJEE, S. 2007. Reusing view-dependent animation. *The Visual Computer 23*, 9-11, 707–719.

COROS, S., MARTIN, S., THOMASZEWSKI, B., SCHUMACHER, C., SUMNER, R., AND GROSS, M. 2012. Deformable objects alive! *ACM Trans. Graph. 31*, 4 (July), 69:1–69:9.

KAVAN, L., AND ŽÁRA, J. 2005. Spherical blend skinning: a real-time deformation of articulated models. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '05, 9–16.

KOYAMA, Y., TAKAYAMA, K., UMETANI, N., AND IGARASHI, T. 2012. Real-time example-based elastic deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '12, 19–24.

MARTIN, S., THOMASZEWSKI, B., GRINSPUN, E., AND GROSS, M. 2011. Example-based elastic materials. *ACM Trans. Graph. 30*, 4 (July), 72:1–72:8.

MÜLLER, M., AND CHENTANEZ, N. 2011. Solid simulation with oriented particles. *ACM Trans. Graph. 30*, 4 (July), 92:1–92:10.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (July), 471–478.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2007. Position based dynamics. *J. Vis. Comun. Image Represent. 18*, 2 (Apr.), 109–118.

RADEMACHER, P. 1999. View-dependent geometry. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '99, 439–446.

SCHUMACHER, C., THOMASZEWSKI, B., COROS, S., MARTIN, S., SUMNER, R., AND GROSS, M. 2012. Efficient simulation of example-based materials. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '12, 1–8.

UNITY TECHNOLOGIES. 2013. Unity. `http://unity3d.com/`.