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

Yuki Koyama      Takeo Igarashi
The University of Tokyo
Motivation

• 2D-like stylizations in 3DCG
  – View-dependent, inconsistent shapes

Example of inconsistency:
Existing method

- **View-dependent geometry (VDG)**
  - [Rademacher, 1999]
  - Changing the geometry according to the view direction
Existing method

- View-dependent geometry (VDG)
  - [Rademacher, 1999]
  - Changing the geometry according to the view direction
Existing method

• View-dependent geometry (VDG)
  – [Rademacher, 1999]
  – Changing the geometry according to the view direction

Only for static geometry 😞
Our goal

- Extending VDG for physical simulation
  - Passively deformable rod structures

Target: hairs, ties, long ears, ...
Big Buck Bunny

DEMO
Demo

- Side-by-side comparison

Fixed view  Camera view
OTHER RESULTS
Other results

- Front hair avoiding the eyes

Video
Other results

• Front hair avoiding the eyes

Without our method

With our method
Other results

• Hair always facing the camera
Other results

- Hair always facing the camera

- This “cowlick” effect is popular especially in recent Japanese 2D animations
User inputs

• A skinned mesh
  – Whose deformable rods are represented by joint chains
User inputs

• A skinned mesh
  – Whose deformable rods are represented by joint chains

• Pairs of…
  – Key example pose
  – Key view direction
Rod simulation framework

- **Oriented Particles**
  - [Müller and Chentanez, 2011]
  - Based on position-based dynamics

- **Simple distance constraint**
  - For ensuring inextensibility
Overview of the runtime operations

1. Calculate weights
2. Blend poses
3. Simulate

\[ P^0 \]
(base pose)

\[ P^1 \]
(example pose)

Current deformed pose
Overview of the runtime operations

1. Calculate weights
2. Blend poses
3. Simulate

\[ \mathbf{P}^0 \quad \text{(base pose)} \]

\[ \mathbf{P}^1 \quad \text{(example pose)} \]

\[ \mathbf{W} \]

Current deformed pose

(view direction)
Overview of the runtime operations

1. Calculate weights
2. Blend poses
3. Simulate

$P^0$ (base pose)

$P^1$ (example pose)

$P(w)$

Current deformed pose
Overview of the runtime operations

1. Calculate weights
2. Blend poses
3. Simulate

\[ P(w) = \text{goal pose} \]

\[ P^0 \]
(base pose)

\[ P^1 \]
(example pose)

Current deformed pose

Internal elastic force
Technical details

• Weight calculation

• Suppression of ghost momentum
Technical details

- Weight calculation
- Suppression of ghost momentum
Weight calculation

• The algorithm of VDG [Rademacher, 1999]
  – Wrapping the model with a triangle mesh
    • Each vertex corresponds to a key view direction

Video
Weight calculation

- The algorithm of VDG [Rademacher, 1999]
  - Wrapping the model with a triangle mesh
  - Each vertex corresponds to a key view direction
  - Linear interpolation on a triangle
Weight calculation

• The algorithm of VDG [Rademacher, 1999]
  – Wrapping the model with a triangle mesh
    • Each vertex corresponds to a key view direction
  – Linear interpolation on a triangle
  – Difficulties
    • Necessary to give at least 4 inputs
    • No base (default) pose
Weight calculation

- Our algorithm (scattered interpolation)
  - Consider **Gaussian weights** on a sphere

\[
\omega_i = \phi_i \left( \left\| \theta - \theta_i \right\| \right) = \exp \left( - \left( \frac{\left\| \theta - \theta_i \right\|}{\alpha_i} \right)^2 \right) \quad (i = 1, 2, \ldots)
\]

\[
\omega_0 = \max \left( 0, 1 - \sum_{i=1}^{\infty} \omega_i \right)
\]

\[
P(w) = \frac{\sum_{i=0}^{\infty} \omega_i P_i}{\sum_{i=0}^{\infty} \omega_i}
\]
Weight calculation

• Our algorithm (scattered interpolation)
  – Consider **Gaussian weights** on a sphere

  \[ w_i = \phi_i \left( \| \theta - \theta_i \| \right) = \exp \left( - \left( \| \theta - \theta_i \| / \alpha_i \right)^2 \right) \quad (i = 1, 2, \ldots) \]

  \[ w_0 = \max \left( 0, 1 - \sum_{i=1} \omega_i \right) \]

  – Arbitrary number of inputs
  – Base (default) pose
  – Influence control by \( \alpha_i \)
Technical details

• Weight calculation

• Suppression of ghost momentum
Problem: ghost momentum

- Ghost momentum
  - The rod increases undesired momentum as the view direction changes
  - It looks “alive”
Problem: ghost momentum

• A possible naïve approach
  – Suppressing ALL momentum
    • Simple damping technique
  – Undesirable

• Our solution
  – Damping ONLY the ghost momentum
    • “Suppression algorithm”
Suppression algorithm

- Separate velocity and position update

\[ \text{Current deformed pose} = \text{P(w)} \]

\[ P^0 \] (base pose)

\[ P^1 \] (example pose)

Internal force

External force
Suppression algorithm

- Separate velocity and position update

\[ P^0 \] (base pose) \[ P(w) \] \[ P^1 \] (example pose)

(a): force that causes the ghost momentum
(b): force that doesn’t cause the ghost momentum
Suppression algorithm

• Separate velocity and position update

(a): force that causes the ghost momentum
(b): force that doesn’t cause the ghost momentum
Suppression algorithm

- Separate velocity and position update
Suppression algorithm

• Comparison

Without suppression

With suppression
Suppression algorithm

• Comparison
  – Failure case (still ghost momentum remaining)
Suppression algorithm

• Limitations
  – Cannot completely remove the momentum
    • Ghost momentum still remains
  – No theoretical ground
    • But practically useful?
  – Doubled computational costs
    • Simulation runs twice (for position and velocity)
CONCLUSION
Conclusion

• Concept
  – View-dependent control of simulated rods

• Techniques
  – Calculating weights from view directions
  – Suppressing ghost momentum

• Limitations
  – Suppression algorithm is not complete
    • Empirically (not theoretically) derived algorithm
  – Not physically accurate
Thank you for listening

- Characters used in our experiments
  - Hatsune Miku © Crypton Future Media, Inc.
  - Big Buck Bunny © Blender Foundation
- 3D models by Yamamoto, Kio, and Blender Foundation