INERTIALIZATION: HIGH-PERFORMANCE ANIMATION TRANSITIONS IN GEARS OF WAR

DAVID BOLLO
PRINCIPAL SOFTWARE ENGINEER
THE COALITION (MICROSOFT STUDIOS)
Animation Transitions
**Animation Transitions**

- Transition from one animation state to another
- Typically a cross-fade blend between poses
Animation Transitions

- Optimizations are often focused on the blend step
- Fast SLERP, optimizing cache and memory usage, etc

Source

Blend

Target
Animation Transitions

- Biggest cost is evaluating both Source and Target
- Source/Target cost is much greater than blend cost
MULTIPLE CHARACTERS

- If we’re lucky...

- Only a few active transitions at once
MULTIPLE CHARACTERS

- But in the worst case...
- Everybody transitions at the same time
- Double the animation cost
Can We Do Better?

- Intuition: Real humans don’t “blend”
- (but they do have inertia)
CAN WE DO BETTER?

- IDEA: Eliminate blended transitions!
- Fix the discontinuities as a post-process
Can We Do Better?

- IDEA: Eliminate blended transitions!
- Fix the discontinuities as a post-process
TRANSITIONS AS A POST-PROCESS
TRANSITIONS AS A POST-PROCESS
TRANSITIONS AS A POST-PROCESS
GOALS

- **Respect** the original animation
  - No changes when not transitioning
- **Believable** and aesthetically pleasing
  - Smooth and momentum-preserving
- **Stay on-model**
  - No bad / unnatural poses
  - No overshoot
IDEA #0: FILTER DISCONTINUOUS POSE

- Apply a filter to the output pose stream
- Difficult to tune
- Introduces lag
- Deviates from input even when not transitioning
Idea #1: Blend From Pose

- Capture pose difference between Source and Target
- Ease out the difference over time
Idea#1: Blend From Pose
Idea #2: Match Velocity

- Capture pose difference between Source and Target
- Ease out the difference over time
- Remember Source velocity (via finite differences)
- Match initial velocity
- Quintic polynomial [Flash and Hogan 1985]

Idea #2: Match Velocity
Idea #3: Limit Overshoot

- Capture pose difference between Source and Target
- Ease out the difference over time
- Remember Source velocity (via finite differences)
- Match initial velocity
- Limit overshoot by controlling initial acceleration
- Choose $a_0$ to give us zero jerk at $t_1$
Idea #3: Limit Overshoot
Inertialization
INERTIALIZATION
INERTIALIZATION – INITIAL VALUES

\[ x_0 = 10000 - 10000 \]
\[ v_1 = 0 \]
\[ a_1 = 0 \]
\[ x_0 = 10000 - 10000 - 1 \]
\[ v_0 = 0 - \frac{1}{dt} - 2 \]
\[ x_1 = 0 \]
\[ v_1 = 0 \]
\[ a_1 = 0 \]
\[ \begin{align*} x_0 &= 10000 - 10000 \frac{d}{dt}, \quad \text{or} \quad 0 \\ v_0 &= -1 - 2 \\ x_1 &= 0 \\ v_1 &= 0 \\ a_1 &= 0 \end{align*} \]
INERTIALIZATION – ACCELERATION

\[
x_0 = 10000 - 10000\cdot constant
\]

\[
v_0 = \frac{1}{dt} - 1 - 2 or 0
\]

\[
a_0 = \frac{-8v_0 t_1 - 20x_0}{t_1^2}
\]

\[
x_1 = 0
\]

\[
v_1 = 0
\]

\[
a_1 = 0
\]
Inertialization – Acceleration

\[ x_0 = \begin{cases} -1 & \text{or} & 0 \\ \end{cases} \]

\[ v_0 = \begin{cases} -1 & \text{or} & 0 \\ \end{cases} \frac{dt}{dt} \]

\[ a_0 = \frac{-8v_0 t_1 - 20x_0}{t_1^2} \text{ or } 0 \]

\[ x_1 = 0 \]

\[ v_1 = 0 \]

\[ a_1 = 0 \]
**Inertialization — \( x(t) \)**

\[
\begin{align*}
    x_0 &= 10000 - 10000 \\
    v_0 &= 0 \\
    a_0 &= -8v_0 t_1 - 20x_0 \\
    a_1 &= 0 \\
    x_1 &= 0 \\
    v_1 &= 0 \\
    a_1 &= 0
\end{align*}
\]

\[
A = -\frac{a_0 t_1^2 + 6v_0 t_1 + 12x_0}{2t_1^5}
\]

\[
B = \frac{3a_0 t_1^2 + 16v_0 t_1 + 30x_0}{2t_1^4}
\]

\[
C = -\frac{3a_0 t_1^2 + 12v_0 t_1 + 20x_0}{2t_1^3}
\]

\[
x_t = At^5 + Bt^4 + Ct^3 + \frac{a_0}{2} t^2 + v_0 t + x_0
\]
OVERSHOOT REVISITED
OVERSHOOT REVISITED

\[ x_0 = 1.2 \]
\[ v_0 = -12.6 \]

\[ t_1 = 0.5 \]
Overshoot Revisited

\[ x_0 = 1.2 \]
\[ v_0 = -20.0 \]
\[ t_1 = 0.5 \]
IDEA #4: CLAMP TRANSITION TIME

\[ x_0 = 1.2 \]
\[ v_0 = -20.0 \]

\[ t_1 = 0.5 \]
Idea #4: Clamp Transition Time

\[ x_0 = 1.2 \]
\[ v_0 = -20.0 \]

\[ t_1 = 0.3 \]
Idea #4: Clamp Transition Time

\[ x_0 = 1.2 \]
\[ v_0 = -20.0 \]

\[ t_1 = 0.3 \]

\[ t_1 = \min(t_1, -5 \frac{x_0}{v_0}) \]
Inertialization on One Slide

\[ x_0 = \begin{cases} -1 & \text{or} \\ -2 & \text{or} \end{cases} \]

\[ v_0 = \frac{-1 - \frac{-2}{dt}}{0} \]

\[ a_0 = \frac{-8v_0 t_1 - 20x_0}{t_1^2} \text{ or } 0 \]

\[ t_1 = \min(t_1, -5 \frac{x_0}{v_0}) \]

\[ x_t = At^5 + Bt^4 + Ct^3 + \frac{a_0}{2} t^2 + v_0 t + x_0 \]

\[ A = -\frac{a_0 t_1^2 + 6v_0 t_1 + 12x_0}{2t_1^5} \]

\[ B = \frac{3a_0 t_1^2 + 16v_0 t_1 + 30x_0}{2t_1^4} \]

\[ C = -\frac{3a_0 t_1^2 + 12v_0 t_1 + 20x_0}{2t_1^3} \]
VECTORS AND QUATERNIONS
Inertializing Vectors

- Obvious choice:
  - Inertialize x, y, z independently
  - Visual artifacts if \( \vec{v}_{x0}, \vec{v}_{y0}, \vec{v}_{z0} \) are too dissimilar (because of transition time clamping)

- Instead:
  - Decompose vector into direction and magnitude
  - Inertialize the magnitude
INERTIALIZING VECTORS

\[ \vec{x}_0 = 10000 - 10000^{-1} \]

\[ \vec{x}_{-1} = 10000 - 10000^{-2} \]
Inertializing Vectors

$$\vec{x}_0 = \vec{x}_{-1} - $$

$$x_0 = |\vec{x}_0|$$
INERTIALIZING VECTORS

\[ \vec{x}_0 = \vec{x}_{-1} - \vec{x}_{-2} \]

\[ x_0 = |\vec{x}_0| \]

\[ \vec{x}_{-1} = \vec{x}_{-1} \cdot \frac{\vec{x}_0}{x_0} \]
INERTIALIZING VECTORS

\[ \vec{x}_0 = \vec{x}_{-1} \]

\[ x_0 = |\vec{x}_0| \]

\[ v_0 = \frac{x_0 - x_{-1}}{\Delta t} \]

\[ \vec{x}_{-1} = \vec{x}_{-2} \]

\[ x_{-1} = \vec{x}_{-1} \cdot \frac{\vec{x}_0}{x_0} \]
INERTIALIZING VECTORS

\[
\vec{x}_0 = \vec{x}_{-1} - 10000
\]

\[
x_0 = |\vec{x}_0|
\]

\[
v_0 = \frac{x_0 - x_{-1}}{\Delta t}
\]

\[
\vec{x}_t = x_t \frac{\vec{x}_0}{x_0} + \vec{x}_t
\]
INERTIALIZING QUATERNIONS

- Similar construction to vectors:
  - Decompose quaternion into axis and angle
  - Inertialize the angle
Inertializing Quaternions

\[ q_0 = q^{-1} \quad \ast \quad q^{-1} \]

\[ q_{-1} = q^{-2} \quad \ast \quad q^{-1} \]
**Inertializing Quaternions**

\[ q_0 = \begin{pmatrix} -1 & * \end{pmatrix} \]

\[ \hat{x}_0 = \text{Axis}(q_0) \quad x_0 = \text{Angle}(q_0) \]

\[ q_{-1} = \begin{pmatrix} -2 & * \end{pmatrix} \]
INERTIALIZING QUATERNIONS

\[ q_0 = -1 \quad \ast \quad -1 \]

\[ \vec{x}_0 = \text{Axis}(q_0) \quad x_0 = \text{Angle}(q_0) \]

\[ q_{-1} = -2 \quad \ast \quad -1 \]

\[ x_{-1} = 2 \tan^{-1} \frac{\overrightarrow{q_{xyz}} \cdot \vec{x}_0}{q_w} \]

Twist of \( q_{-1} \) around \( \vec{x}_0 \)

Fiber Bundle Twist Reduction
Graphics Gems IV, 230 – 236
Inertializing Quaternions

\[ q_0 = \begin{bmatrix} 10000 \\ -1 \end{bmatrix} \]

\[ \tilde{x}_0 = \text{Axis}(q_0) \quad x_0 = \text{Angle}(q_0) \]

\[ v_0 = \frac{x_0 - x_{-1}}{\Delta t} \]

\[ q_{-1} = \begin{bmatrix} 10000 \\ -2 \end{bmatrix} \]

\[ x_{-1} = 2 \tan^{-1} \left( \frac{\overrightarrow{q_{xyz}} \cdot \tilde{x}_0}{q_w} \right) \]

Twist of \( q_{-1} \) around \( \tilde{x}_0 \)

Fiber Bundle Twist Reduction
Graphics Gems IV, 230 – 236
Inertializing Quaternions

\[ q_0 = \begin{pmatrix} 1 \end{pmatrix} \] \[ q_{-1} = \begin{pmatrix} -1 \end{pmatrix} \]

\[ \tilde{x}_0 = \text{Axis}(q_0) \quad x_0 = \text{Angle}(q_0) \]

\[ v_0 = \frac{x_0 - x_{-1}}{\Delta t} \]

\[ q_t = \begin{pmatrix} \text{Axis: } & \tilde{x}_0 \\ \text{Angle: } & x_t \end{pmatrix} \]

\[ x_{-1} = 2 \tan^{-1} \frac{\overrightarrow{q_{xyz}} \cdot \tilde{x}_0}{q_w} \]

Twist of \( q_{-1} \) around \( \tilde{x}_0 \)

BLENDING VS INERTIALIZATION
**Blending vs Inertialization**

**Blending**
- Evaluate both Source & Target during transition
- Variable anim frame cost

**Inertialization**
- Only evaluate Target during transition
- Fixed anim frame cost
**BLENDING VS INERTIALIZATION**

**Blending**
- Manage multiple sets of state during transitions
- Adds complexity

**Inertialization**
- Only maintain one set of state during transitions
- Fire and forget
BLENDING VS INERTIALIZATION

Blending

Inertialization
BLENDING VS INERTIALIZATION
INERTIALIZATION IN A GAME ENGINE
INERTIALIZATION IN A GAME ENGINE

- Inertialization Node / Filter
- Animation System Hooks
- Code Hooks
**Inertialization Node**

- Evaluated after the main animation graph
- Input is discontinuous pose stream
- Output is inertialized pose stream
Inertialization Node

- When a new inertialization is requested:
  - Compute and store $x_0, v_0$ for all joints
  - Store $t_1$ and set $t = 0$
- Every frame:
  - Update $t$ with delta time
  - Evaluate and apply $x(t)$ for all joints
  - Store the OUTPUT pose in the pose history buffer
ANIMATION SYSTEM HOOKS

- Add “inertialization” as a new blend curve type
- When a blend is requested with “inertialization” type:
  - Inertialize with the supplied blend time
  - Zero the blend time to bypass regular blending
CODE HOOKS

- Expose “Request Inertialization” to code
- Eliminate other types of discontinuities
- And other tricks...
TIPS AND TRICKS
SMOOTHING OTHER DISCONTINUITIES

- Gears of War 3:
  - Snap character rotation when switching to sprint
- Gears of War 4:
  - Snap character rotation when switching to sprint
  - Inertialize away the discontinuity
LOCOMOTION FILTERING

- Gears controls are very responsive (twitchy)
- Filter inputs to locomotion blend spaces
- If filtered values are too far from actual values...
  - Snap to actual values
  - Inertialize
- Fluid pose even with twitchy inputs
**FIRE & FORGET – MOTION WARping**

- Don’t need to maintain warp point data across transitions
- Only 1 active warp at a time
- Simplifies bookkeeping
- Simplifies replication

S. Dickinson. Motion Warping in ‘Gears of War 4’: Doing More with Less. GDC 2017
THANK YOU

DAVID BOLLO
dbollo@microsoft.com