Dynamo Bone

Technical Documentation

Author: Zeuk Games

Date: July 11, 2025

1. Overview

The Dynamo Bone script is a lightweight implementation for Unity that drives secondary motion on hierarchies such as tails, hair, cloth strips, cables, and similar appendages. It computes per-frame physics using spring-mass principles with four key tunable parameters—Damping, Elasticity, Stiffness, and Inertia—while omitting advanced features like per-bone distribution curves and collision handling to keep the footprint small.

2. Quick Start

1. Add the `DynamoBone` component to the *root bone* (the highest Transform) of the chain you want to animate.
   • Examples: the base joint of a tail, the hair root on a character's head, the first link of a swinging chain.
2. Ensure the child hierarchy is correctly parented so the script can traverse it at runtime.
3. Play the scene. The bones will automatically simulate.
4. Tweak `Damping`, `Elasticity`, `Stiffness`, and `Inertia` for the desired feel.
5. Use `Update Rate` to trade accuracy for performance (higher = less CPU, slightly more lag).

3. Public Fields

• Root (Transform) – The root joint of the dynamic chain. If left null, the GameObject the script is attached to is used.
• UpdateRate (float, Hz) – Maximum solver updates per second. A value of 60 runs every frame at 60 FPS; 30 updates every other frame, and so on.
• Damping (0–1) – Fraction of velocity removed each step. 0 = no damping (loose), 1 = fully critically damped (frozen).
• Elasticity (0–1) – Strength of the spring pulling the bone back toward the rest pose.
• Stiffness (0–1) – How rigidly the bone maintains its local orientation relative to its parent.
• Inertia (0–1) – How much world-space motion of the object influences the bones (higher = bones lag more behind the object’s movement).
• EndOffset (Vector3) – Optional vector added to the tip of the last bone to simulate soft flesh beyond the skeleton (for example, ears or hats).

4. Runtime Cycle

Each frame the component performs the following steps:
1. Early Exit – If the object is not visible or the distance to the camera exceeds a culling threshold, simulation is skipped to save CPU.
2. DeltaTime Clamping – If the frame time exceeds 1 / UpdateRate, multiple sub-steps are executed for stability.
3. External Forces – Gravity and user-supplied `ExternalForce` are applied to each virtual point.
4. Inertia – Object movement since the previous frame is subtracted to allow bones to 'lag'.
5. Verlet Integration – Bone tips are moved according to spring and damping forces.
6. Stiffness & Elasticity – Bones are pulled back toward their rest orientation and length.
7. Constraint Enforcement – Optional axis freeze prevents rotation on selected axes.
8. Pose Application – The final world positions are converted back to bone rotations.

5. Implementation Details

• SetupBones() – Recursively walks the hierarchy starting at `Root`, caching default local rotations, bone lengths, and endpoints.
• Update() – Drives the time-step accumulator so that physics runs at the target `UpdateRate`.
• UpdateDynamics(float dt) – Core solver integrating forces and re-applying constraints.
• ApplyToBones() – Writes computed positions back to the Transform rotations each frame.

Note: All math is done in world space to avoid scaling issues, then converted back to local space for animation blending. The solver operates on virtual 'particles' placed at each bone tip.

6. Limitations & Design Trade-offs

• No Collision – Bones may clip through geometry unless other solutions (e.g., Unity Physics) are layered on top.
• Uniform Parameters – Without distribution curves, every bone uses the same damping, elasticity, stiffness, and inertia values.
• Non-Iterative Constraints – The solver runs a single pass per sub-step, which is sufficient for modest chains (<32 bones) but may stretch under heavy dynamics.

7. Licensing & Support

For questions, feature requests, or contributions, contact therealzeuk@gmail.com.