Eurographics Symposium on Geometry Processing 2025
M. Attene and S. Sellán
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 5

Real-Time Secondary Animation with Spring Decomposed Skinning

B. Akyürek and Y. Sahillioğlu

Dept. of Computer Engineering, Middle East Technical University, Turkey
bartu.akyurek@metu.edu.tr, ys@ceng.metu.edu.tr

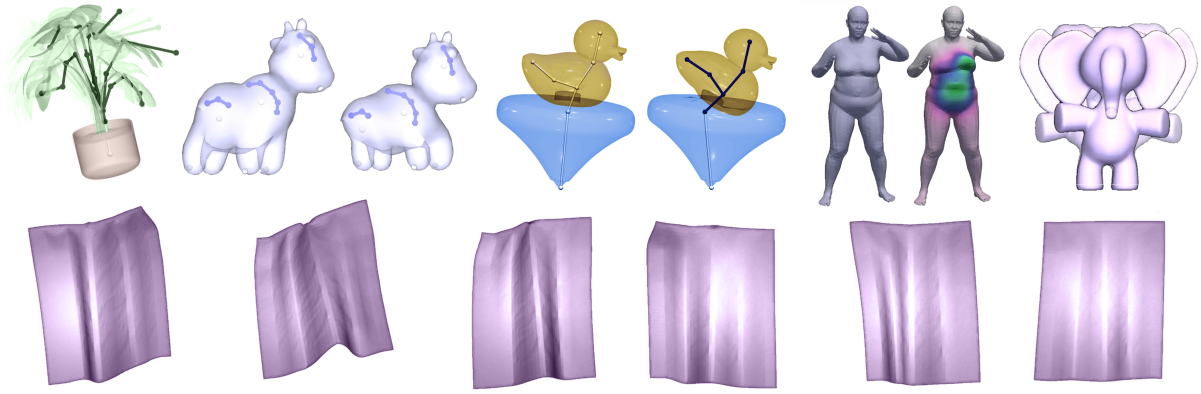


Figure 1: *Our method extents to a wide range of dynamics including secondary motion on composite rigid bodies (e.g. plant leaves), stretch/squish on elastic bodies (cow, rubber duck), jiggling of soft bodies (fat tissues), global secondary motion (vibrating elephant), and cloth animation.*

Abstract

We present a framework to integrate secondary motion into the existing animation pipelines. Skinning provides fast computation for real-time animation and intuitive control over the deformation. Despite the benefits, traditional skinning methods lack secondary dynamics such as the jiggling of fat tissues. We address the rigidity of skinning methods by physically simulating the deformation handles with spring forces. Most studies introduce secondary motion into skinning by employing FEM simulation on volumetric mesh vertices, coupling their computational complexity with mesh resolution. Unlike these approaches, we do not require any volumetric mesh input. Our method scales to higher mesh resolutions by directly simulating deformation handles. The simulated handles, namely the spring bones, enrich rigid skinning deformation with a diverse range of secondary animation for subjects including rigid bodies, elastic bodies, soft tissues, and cloth simulation. In essence, we leverage the benefits of physical simulations in the scope of deformation handles to achieve controllable real-time dynamics on a wide range of subjects while remaining compatible with existing skinning pipelines. Our method avoids tetrahedral remeshing and it is significantly faster compared to FEM-based volumetric mesh simulations.

CCS Concepts

• ***Computing methodologies*** → ***Animation;*** *Physical simulation;*

1. Introduction

Rigid bodies in nature often exhibit oscillatory motion, referred to as secondary dynamics, arising as a response to initial primary motion. Soft bodies also have a similar motion as they jiggle under force. Despite being widely used in animation, skinning methods remain too stiff to exhibit such dynamics on both soft and rigid bodies. To emulate dynamic deformation, researchers have been developing physical simulation techniques. The lack of realism in skinning methods has been compensated by physically simulating the material through mesh vertices; however, incorporating physics

into mesh deformation might involve complex implementations. Additionally, many physical deformation schemes are dependent on the mesh resolution. Tetrahedral meshes are also commonly required for FEM simulations as in [HMT*12; IKNP16; AF15; ZZCB21; CGC*02; HTC*13; RF14; MZS*11; KB18; KBB*17]. As a result, these approaches significantly increase the computational costs for finer resolution. Another drawback is that controlling the results of physical simulation is limited; hence, physical simulation deformation methods remain mostly unintuitive compared to skinning.

Introducing plausible dynamic effects while preserving the primary intuition behind skinning remains an open area of research. Spring Decomposed Skinning (SDS) tackles this problem by introducing physics in the scope of skinning bones. With this approach, SDS combines fast computation and intuitive user control advantages of skinning with the plausible dynamics of physical simulations.

We use spring forces to simulate dynamic motion in skeletal bones, referred to as *spring bones*. These forces are modeled using Hookean springs due to their sinusoidal behavior (for the jiggling secondary motion) and computational efficiency. However, despite their simplicity, Hookean springs can suffer from numerical instability, caused by a time-stepping scheme, particularly under large time steps [WB01]. In addition, they are only conditionally stable and require carefully chosen initial parameters [MHTG05]. To address these stability issues, we adopt a Position-Based Dynamics (PBD) framework [MHHR07] and we further impose kinematic constraints to ensure intuitive skeletal movement.

Our method takes an articulated character animation as input and allows the user to determine which bones will be simulated as spring bones. Optionally, we utilize helper bones as an addition to the primary skeleton, allowing the user to localize the dynamics on smaller surfaces. With this pipeline, we introduce dynamics on existing animation, introducing a small overhead to the traditional skinning pipeline. Our method is simple yet capable of capturing a diverse range of dynamic motion. Skinning at its heart is a decomposition of complex surface deformation to a limited set of bone transformations. Following this intuition, we decompose dynamic mesh surface deformation into a linear combination of spring motions. We also provide the source code at `https://github.com/bartuakyurek/Spring-Decomposed-Skinning`.

1.1. Contributions

Spring bones are dynamic as their motion depends on the current time and configuration. On the other hand, the existing geometric skinning methods remain static as they require additional manual keyframe input to produce the same motion. Despite the additional keyframes, our computed dynamics can be quite cumbersome to replicate with manual labor. In essence, our method provides a controllable physical simulation scheme through rigging.

Unlike common approaches, our framework does not require volumetric mesh input or anatomical primitives. As a result, SDS is highly compatible with existing animation pipelines. Volumetric mesh simulations are generally dependent on the mesh resolution. In contrast, our framework scales to higher mesh resolutions by limiting the physical simulations to rig handles, providing fast computation of dynamic motion.

The main contributions of our proposed framework Spring Decomposed Skinning are as follows:

- Provide artistic control over physical simulations through deformation handles. These handles achieve both **global** and **local** secondary motion over diverse subjects including rigid bodies, soft tissues, and cloth surfaces.
- Decompose dynamic surface deformation into a linear combination of simple spring oscillations, which aligns with the core intuition behind skinning.
- Achieve real-time dynamic deformation on a broad range of 3D models without requiring computationally complex tetrahedralization.
- A method to compute Inverse Kinematics-like transformations of the dynamically posed skeleton bones.
- An easy-to-integrate framework into existing animation pipelines, compatible with traditional skinning input and output.
- Formalize a framework around the "jiggle bones" construct, closing the gap between industry applications and academic research.

2. Related Work

Skinning Decomposition is a term introduced in [KSO10] to find bone transformations and bone-vertex binding weights from example poses. [LD12] decompose a set of example poses into rigid transformations. Even though this study is not focused on example-based decomposition, we are still interested in decomposing *dynamic* poses into a set of bone transformations.

Geometric skinning approaches such as LBS commonly suffer from rigidity in character motion and commonly lack dynamic effects such as jiggling, swaying, muscle bulging, or elastic deformations [RF16; WLP*17; Muk15]. Researchers have been developing solutions to bring more life into skinning frameworks and increase the expressiveness of animations.

Blendshapes deformation studied in various studies such as [KBB*17], [LMR*23], [MWF*12], is useful when the expressiveness of skinning methods falls short, e.g. for facial expressions where the surface is highly complex. In addition to skinning pipelines, our framework can be also integrated with blendshapes deformation.

Anatomical models, including [KBB*17; LST09; RRC*18; AS07], utilize the underlying anatomy of bones and muscles to enable highly realistic deformation. However, these methods are complex to implement and sophisticated anatomical knowledge might not be available for a given mesh.

Example-based methods are employed in numerous studies such as [SZT*08; MG03; Muk15] to introduce realistic dynamics. A significant number of papers such as [ZZCB21; MR22; YWM*21; SGOC20; CO18; PRMB15; JHG*22] also extend the example based approach by making use of neural networks. Several studies including [SGX*21; SGXT20; BRPB17], utilize motion capture technology. Despite their plausible dynamic results,

motion capture technology remains an expensive solution. In addition, the scope of example-based approaches is limited to the dataset and it is another challenge to generalize the dynamics. In contrast, our framework proposes a general framework that can be applied to versatile subjects without additional dataset costs.

FEM-based methods such as [CBC*05; CGC*02; HMT*12] employ physical simulation to compute dynamic motion occurring under external forces such as wind, gravity, and collisions. Many papers, including [HMT*12; IKNP16; AF15; ZZCB21; CGC*02; HTC*13; RF14; MZS*11; KB18; KBB*17; TRPO21], incorporates FEM simulation into skinning pipeline through a volumetric tetrahedral mesh. Despite the achieved dynamic effects, tetrahedralization is computationally expensive and is not trivial for some geometry. Time complexity is also a bottleneck especially for the higher mesh resolutions, making high-resolution tetrahedral FEM not suitable for real-time applications. In this work, we avoid complex tetrahedralization to achieve both scalable and controllable dynamics.

Complementary Dynamics [**fastcompdyn**; ZBLJ20] introduce secondary motion that does not undo the deformation produced by the geometric methods. To this end, an orthogonality constraint between the rigid motion and the computed secondary motion is injected. Similarly, several papers have also utilized an orthogonality constraint for dynamic deformation as in [CBC*05] and [WU23]. [TRPO21] models soft-tissue dynamics as deviations from parametrized skeleton motion. On the downside, complementary dynamics do not fully cover the range of secondary motion as some objects have additional jiggling in the same direction of their primary motion. Shaking a plant pot in Figure 9 results both jiggling in complementary and primary spaces. In this sense our secondary animation can be utilized to both produce complementary motion and to exaggerate the primary motion; we call them local and global dynamics respectively.

Mass-spring systems are used in rope, cloth and garment animation [LBOK13], hair [SLF08], and soft tissue simulation [Gol18], and mesh unfolding [SK16]. Several works [NT98; LST09; KHS01; TT93; MHYH17] utilize mass-spring networks on muscular anatomy or in between muscle and skin tissues for realistic deformations. [KB18] use spring constraints on tetrahedral edges for a physics based character skinning, while [LTW95] use layered mass-spring networks to simulate soft tissues for realistic facial expressions. [JHG*22] augment a quasistatic neural network with analytically integrated zero-restlength springs to capture dynamic deformation, especially for human soft tissue dynamics. Other methods include [vFTS07], which define a limited number of mass-spring sets to achieve elastic deformations, and [WLP*17], who use mass-spring networks to deform 2D illustrations. Although not directly a mass-spring system, [DBT10] proposes physically simulated dynamic curves with mass and stiffness parameters for jiggling 2D animation. [TN19] uses the notion of spring rigs by adding helper rigs between primary rig and mesh vertices. They statically correct skinning artifacts in post-processing via spring forces. In contrast, we utilize spring rigs for dynamic motion in skeletal subspace.

Helper bones are utilized in [CM24; Muk15; MK16; Muk18; MG03] to emulate nonlinear deformation such as muscle bulging and tissue jiggling, that aren't achievable by an intuitive rig. Inspired by these studies, we utilize helper bones to widen the range of dynamics we achieve. As an extension to helper bones studies focusing on helper bones individually, we allow chains of helper bones to achieve local soft tissue jiggling and cloth dynamics. We also show that helper bones are not only useful for tissue deformations but also for a wider range of dynamics.

Rig-level physics methods provide a deformation framework driven by rig parameters. One of the pioneering works that handles physical forces on rig-level is Velocity Skinning by [RTK*21]. In Velocity Skinning, the bones are associated with an additional set of weights based on their velocities, to be used in the underlying skinning pipeline. These extra weights allow dynamic floppy and squashy deformation. [GBFP11] also combine the accuracy of physical deformation with skinning handles. Building on Velocity Skinning, [SRKZ24] inject a time-varying signal into skinning space to capture oscillatory secondary motion. This signal, modeled as a damped sinusoidal, effectively represents a single one-dimensional spring. In our approach, we directly simulate skinning bones as a spring that enables chains of springs to produce more complex secondary motion. Another study unifying physics with skinning deformation was recently conducted by [WU23] on 2D triangulated surfaces or 3D tetrahedral meshes. Our method is also driven by the rig parameters, unifying physical simulation with skinning pipelines. SDS distinguishes itself by directly simulating the bones, providing artistic control over the physical deformation simply through the skeleton. more realistic behavior.

Several works with volumetric mesh simulations also parametrize skeletal motion in their physical deformation pipeline. [CGC*02] utilize a coarse volumetric mesh to introduce elastic deformation driven by skeletal animation. In [HTC*13], rig parameters are used as a degree of freedom in physical simulations on a volumetric mesh. Similarly, in Rig-Space Physics [HMT*12], the rig parameters determine the stiffness parameters of the underlying FEM-based elastic deformation, augmenting the physical simulation with skeletal animation. As a disadvantage, these frameworks might require extra cumbersome weight painting and texture mapping work to convert an existing animated model into a volumetric input. Our method avoids such remeshing steps and can readily be used to add dynamics to an existing animated character.

Jiggle bones are known in the industry to some extent, with one of the most recent uses of jiggle bones is by the game God of War Ragnarök where the developers have used helper bones to produce tissue jiggling and muscle bulge by the help of anatomic heuristics [Wan23]. However, to the best of our knowledge jiggle bones are not directly introduced in skinning literature although similar works such as [MK16] exist. In this work, we propose a basic framework of spring bones skinning to leverage both the realism of physical simulations and the speed of traditional skinning, which has been a gap between industry applications and academic research. Albeit simple, the method is effective and popular, and has a positive impact on game development.

3. Method

Given a rigged model and its animation keyframes, Spring Decomposed Skinning (SDS) defines a simple framework to simulate the rig bones. Initially, the user selects specific rig bones to be treated as *spring bones*. These bones include primary bones and helper bones. After an initial configuration, our method automatically computes the dynamic bone locations throughout the animation sequence.

Primary bones are typically part of the original rig that drive the main motion, such as the arms and legs of a human skeleton. In contrast, *helper bones* are auxiliary bones introduced by the user, intended to localize secondary motion to smaller regions, typically used for soft tissue dynamics. As a rule of thumb, primary bones are converted to spring bones when global secondary motion is desired, such as oscillations resulting from the rig's main motion. Helper spring bones, on the other hand, are added when local secondary motion is needed, such as to simulate the jiggling of belly fat.

We define a spring bone with a spring attached to two masses located at both tips of the bone as in Figure 2. The first mass, namely *fixed mass*, is located at the bone head and its position remains unchanged throughout the mass-spring simulation. Only kinematic constraints and user-defined keyframes update these fixed mass locations. The second mass, i.e. *free mass*, is placed at the bone tail and the simulation updates its position dynamically at every frame.

Spring rest length can be adjusted by the user via positioning the fixed mass along the bone, from bone head to tail. At its extreme, both the fixed mass and free mass are located at the bone tail, creating a *point spring bone*. Converting primary bones into point spring bones enables squash and stretch deformations, where secondary motion enhances the expressiveness of primary motion as in Figure 15.

We introduce the notion of fixed and free masses to emulate skeletal dynamics in a way that aligns with the intuition of skeletal animation. Unlike classical mass-spring networks, where all particles are updated during simulation, we maintain the hierarchical structure of a skeleton by fixing the first mass of each spring. This fixed mass preserves its orientation relative to its parent and is not updated during the mass-spring simulation. Instead, its position is driven by rigid skinning transformations and kinematic constraints, which are discussed in the following sections. This design preserves the parent-child relationship of skeletal trees, where each bone (except the root) is oriented relative to its parent.

At each frame, we simulate the free masses and update the rig bone orientations accordingly. Let $B^0 : \{B_i^0\}$ be the set of bone orientations $B_i^0 \in R^{2\times 3}$ in the rest pose, including locations of bone head and tail. Similarly, let B^R and B^D be the set of rigid and dynamic bone orientations respectively. Rigid bone transformations computed by forward kinematics $M^R : B^0 \rightarrow B^R$ maps the rest pose to the rigid pose. Based on the initial settings and the rig pose at a given frame, our framework dynamically updates the rig pose and computes the dynamic bone transformations $M^D : B^R \rightarrow B^D$ to be used in the underlying skinning method.

3.1. Simulation

We simulate spring bones as simple Hookean springs with a Position Based Dynamics (PBD) framework [MHHR07]. Initially,

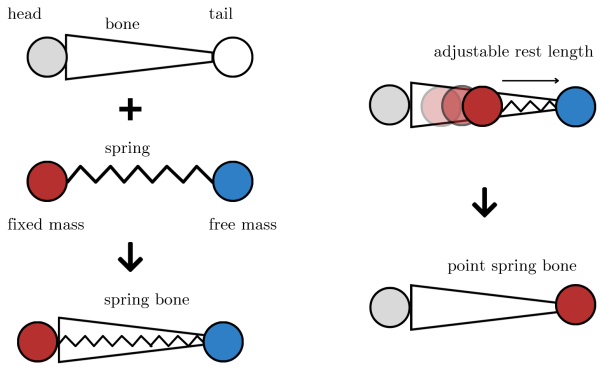


Figure 2: *Placement of a mass-spring system along a skeletal bone. The fixed mass is driven by rigid animation keyframes, while the free mass is updated via dynamic simulation at each timestep. The spring rest length is determined by the fixed mass position; translating the fixed mass along the bone axis reduces the rest length. In the extreme configuration, both the fixed and free masses are positioned at the bone's tail, connected via a zero rest length spring, which forms a point spring bone.*

the user determines system parameters including spring stiffness $k_s \in R$, damping $k_d \in R$, and mass $m \in R$ to be used in mass-spring simulation. Particle locations for each particle i to $\mathbf{x}_i^0$ are initialized with respect to rest pose, and their velocities to $\mathbf{v}_i^0 = 0$. For PBD algorithm, we also save the inverse masses as $w_i = 1/m_i$.

At each frame, all masses are first displaced by forward kinematics as in traditional skinning pipelines. Then, our mass-spring simulator updates free mass positions based on Hookean spring forces and mass velocities (steps 3-4 in Figure 5).

Let $\mathbf{p}_1, \mathbf{p}_2 \in R^3$ be the particle positions attached to both ends of a spring, $\mathbf{v}_1, \mathbf{v}_2 \in R^3$ their velocities respectively, and l_0 be the spring rest length. We have the spring vector $\mathbf{p}_{1,2} = \mathbf{p}_2 - \mathbf{p}_1$ and normalized spring vector $\mathbf{n} = \frac{\mathbf{p}_{1,2}}{\|\mathbf{p}_{1,2}\|}$. Hookean spring force $\mathbf{f}_s$ and damping force $\mathbf{f}_d$ acting on a p_1 is given by:

$$\mathbf{f}_s = k_s(\|\mathbf{p}_{1,2}\| - l_0)\mathbf{n} \quad (1) \qquad \mathbf{f}_d = -k_d[\mathbf{n}\cdot(\mathbf{v}_2 + \mathbf{v}_1)]\mathbf{n} \quad (2)$$

For the other particle p_2, the spring force is taken as $-\mathbf{f}_s$. We take the total external forces as $\mathbf{f} = \mathbf{f}_s + \mathbf{f}_d$ to be used in PBD framework. Note that in Equations 1 and 2 additional bending of a spring is not represented; that is, spring bones move in a linear trajectory with only affine transformations. However, bending phenomena emerge when using spring chains, where parent-child relationships allow the chain to bend. This occurs because child bones inherit both the rotations of their ancestors and their own rotations, which are inferred after the free masses are relocated via simulation.

Algorithm 1 takes all the mass positions X in a rig and updates the free mass locations for a single iteration with time step Δt. In line 2 we skip the simulation if a particle is fixed in the system, i.e. its mass is zero. In line 4, parallel to PBD algorithm, we provide parameter d_s to further damp the particle velocities $\mathbf{v} \leftarrow d_s\mathbf{v}$ that is

Algorithm 1 PBD-based Simulation

Require: $X : \{x_i \in R^3\}$
1: **for all** particles i **do**
2: **if** m_i is 0: continue
3: $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_i$
4: $\mathbf{v}_i \leftarrow d_s \mathbf{v}_i$
5: **end for**
6: **forall** particles i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
7: **if** `stretch_constraints`
8: $\tilde{\mathbf{p}}_i \leftarrow$ projectStretchConstraints$(\mathbf{p}_1, ..., \mathbf{p}_N)$
9: **endif**
10: **for all** particles i **do**
11: $\mathbf{v}_i \leftarrow (\tilde{\mathbf{p}}_i - \mathbf{x}_i)$
12: $\mathbf{x}_i \leftarrow \tilde{\mathbf{p}}_i$
13: **end for**

a parameter for 3D artist to adjust for desired effects. In our experiments, we used velocity damping in range $0 < d_s < 1$ for plausible dynamics.

Note that the external forces can include gravity and other contact forces. However, in our experiments we typically omit gravitational acceleration or contact forces and focus only on the effects of spring forces. For instance in Figure 3, lower stiffness k_s causes slower motion whereas higher stiffness abruptly vibrates the spring bone and returns faster to rest length in a stable system.

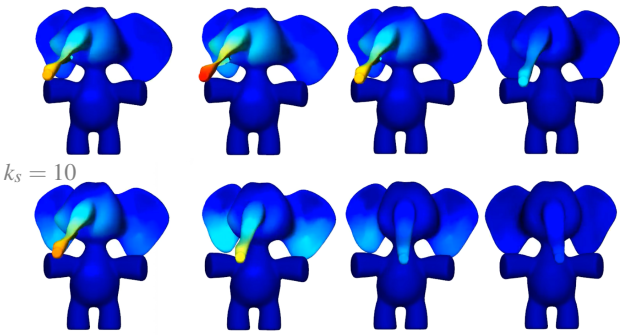


Figure 3: *Effect of spring stiffness on convergence behavior. Increasing spring stiffness k_s leads to faster convergence toward the rest length.*

Since we limit our scope to spring forces and avoid other external forces, we skip the collision constraints in PBD algorithm. Instead in line 8, we provide optional stretching constraints (see [BMOT13]) to be projected after the initial velocity and particle update. The constraints in line 8 are projected as $\tilde{\mathbf{p}}_i \leftarrow \mathbf{p}_i + \Delta\mathbf{p}_i$ where $\Delta\mathbf{p}_i$ is the correction vector. For a single spring, we use stretching correction vectors $\Delta\mathbf{p}_1$ and $\Delta\mathbf{p}_2$ as given in [BMOT13]. We typically iterate once to optional project stretch constraints in our experiments.

In lines 7-9, we optionally provide stretching constraints given in [BMOT13] to project spring bones back to their original length. Let $\mathbf{p}$ be the mass positions are updated with explicit Euler integration. Following PBD algorithm, the first set of constraints are projected as $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$ where $\Delta\mathbf{p}$ is the correction vector in line 8. For each particle at $\mathbf{p}_i$ connected to $\mathbf{p}_j$ with a spring, the correction vector is defined as:

$$\Delta\mathbf{p_i} = \frac{w_i}{w_i + w_j}(\|\mathbf{p_{i,j}}\| - l_0)\frac{\mathbf{p_{i,j}}}{\|\mathbf{p_{i,j}}\|} \tag{3}$$

Despite PBD framework being unconditionally stable, projecting stretching constraints often does not produce plausible jiggling bones. To improve the resulting dynamics, we further project constraints after a PBD-based simulation that is explained in Section 3.2.

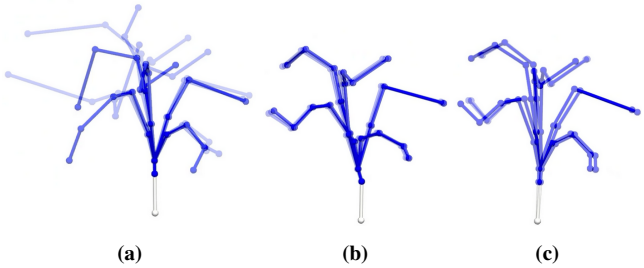


Figure 4: *Effects of constraints on Monstera plant rig motion. Blue represents spring bones. A rigid white bone is rotated side by side to jiggle the bones. (a) Stretch constraint causes a circular motion. (b) Our kinematic constraints projected inside PBD framework result in vibrations and fail to produce smooth jiggling motion. (c) Kinematic constraints projected outside of PBD framework produce more plausible jiggling.*

3.2. Kinematic Constraints

Once our simulator in Algorithm 1 updates the free mass locations, the connection between a bone tail with its children bone heads are broken since fixed masses are not simulated (step 4 in Figure 5). Hence, to ensure bone connectivity, we project additional constraints based on the rig kinematic tree.

Our kinematic constraints projected inside a classical PBD framework result in abrupt vibrations (Figure 4b). In contrast, when applied after the simulation loop, these constraints produce more plausible jiggling effects (Figure 4c). Furthermore, using solely the stretch constraint results in an extra circular motion that might not be desirable (Figure 4a). Therefore, we provide another user option `fixed_scale` to preserve rest bone lengths after simulation. Projecting these additional constraints after PBD-based simulation avoids unintuitive vibrations and results in smoother rig dynamics.

Steps we follow to restore bone connectivity and rest bone lengths are illustrated in Figure 5. In Steps 1-4 the rig is posed with forward kinematics and mass-spring systems are simulated. From Step 4→5, if the user enables `fixed_scale` option, the spring bone is scaled back to its rest pose length (Algorithm 2 lines $2-6$). Then, the descendant bones are translated to their rest pose orientations (Algorithm 2 lines $7-11$); i.e. if they were fully connected

Algorithm 2 Kinematic Constraints

Require: B^0, K_B, B
1: **for all** bones i **do**
2: **if** fixed_scale **then**
3: $l = \|B_{1i} - B_{2i}\|$
4: $\mathbf{d} = (B_{1i} - B_{2i})/l$
5: $B_{2i} \leftarrow (l - l_0)\mathbf{d}$
6: **end if**
7: **for all** children $j \in K_B(i)$ **do**
8: $B_j = B_{2j} - B_{1j}$
9: $B_{1j} \leftarrow B_{2i} + t_j$
10: $B_{2j} \leftarrow B_{1j} + B_j$
11: **end for**
12: **end for**

the children bones are carried to their parent's tail, or if they were unconnected, they are carried with an offset (line 9).

In Algorithm 2, B^0 is the rest bone locations and $B_i = (B_{1i}, B_{2i})$ is the bone at index i with its head and tail located $B_{1i}, B_{2i} \in R^3$ respectively. K_B is the kinematic relations $K_B : \{(B_i, B_j)\}$ where in every tuple B_i is the parent bone of B_j. Lastly, B denotes the bones in the posed space. In our case, B is the bone orientations after the simulation.

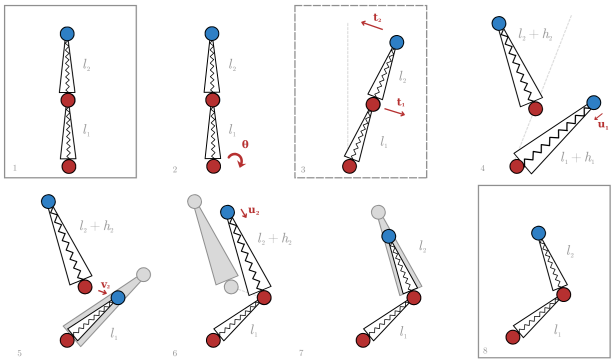
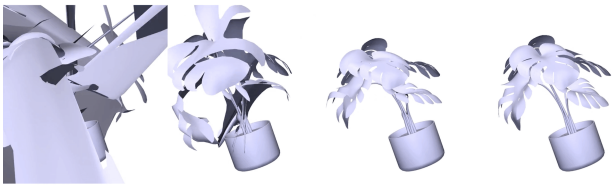


Figure 5: *Steps to preserve rest bone lengths and connectivity in a toy case. First box is the rest pose. User applied rotation rotates the bone chain through forward kinematics in the dotted box. Last box is the dynamically posed bone chain where original length and connectivity are preserved.*



(a) (b) (c) (d)

Figure 6: *Comparison of simulation constraints on skinning deformation. (a) Unconstrained. (b) Only stretch constraints. (c) Our constraints. (d) Rigid LBS.*

3.3. Skinning Transformations

In skinning, underlying bone transformations determine the mesh deformation. Once we simulate the spring bones, we need to infer the necessary bone transformations that are given to skinning. In this work, we adopt the most widely used skinning method, linear blend skinning (LBS), as our underlying skinning method. Let $v_j^0 \in R^{4\times 1}$ be rest pose vertex in homogeneous coordinates, $\mathbf{M}_i \in R^{4x4}$ be the bone i's transformation matrix and w_{ij} be the binding weight between bone i and vertex j. Deformed vertex location v_j in homogeneous coordinates can be defined as:

$$v_j = \sum_{i=0}^{N} w_{ij} \mathbf{M}_i v_j^0 \tag{4}$$

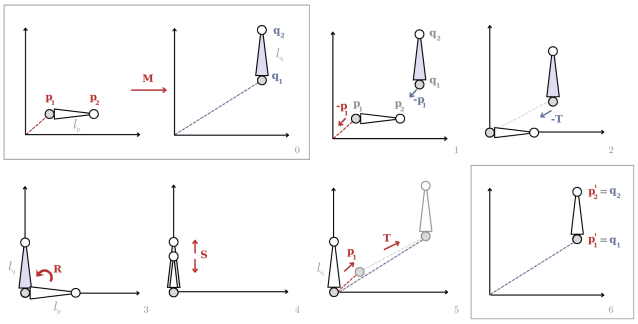


Figure 7: *The procedure to compute chained affine transformations. Rest pose bone (source) and posed bone (target) is given in the first box. Our goal is to find a transformation **M** that maps source line segment to target line segment. At the last box the source line segment is aligned with the target segment after Rotate-Scale-Translate affine transformations are applied.*

In traditional skinning pipelines, bone transformations M_i are directly computed via underlying kinematics, either by forward or inverse kinematics, after the user poses the skeleton bones. In our case, the user input transforms are altered once the spring bones are simulated. Therefore, our problem is to find a new transformation $\mathbf{M}_i^* : B_i^0 \rightarrow B_i^D$ to pose the rest pose bones into their dynamically posed locations.

To compute the dynamic bone transformations, we decompose a matrix M_i^* into its individual affine transformations: **R**otate, **S**cale, **T**ranslate (RST). In Figure 7 steps to find affine transformations are illustrated. In the end, necessary transformation matrices $\mathbf{M}_i$ of each bone i are computed for Equation 4. Intermediate steps of the computation are provided in Appendix A.

3.4. Implementation

Our method takes traditional skinning input: a mesh, a rig, and keyframe poses. The user decides which bones are spring bones and specifies their mass-spring parameters. Alternatively, similar to [MK16], we also use helper bones to extend the primary rig to simulate local dynamics. On the computational side, these helper

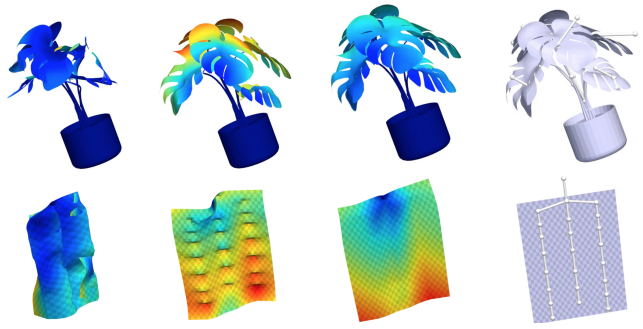


Figure 8: *Comparison of different bone transformations. (a) SVD-based Least-Squares Rigid Motion [SR17] is used to map rest pose to dynamic pose. Note that this algorithm requires at least 3 points to produce the optimal rigid motion; therefore, we used bone endpoints as well as its midpoint to match 3 points of a rest pose bone to a dynamically posed bone. (b) Only bone tail translations are fed in the skinning. Notice that this approach can be plausible for irregular shapes (top) but fails to produce smooth deformation for a flat surface (bottom). (c) Our proposed RST transformation. (d) Rigid LBS.*

bones are the same as regular bones, except their rest pose orientations differ in intuition.

Following the intuition of orthogonal forces represented in Complementary Dynamics [ZBLJ20], the helper spring bones are mainly built complementary to the existing rigid rig. The majority of the helper bones are nearly orthogonal to the bones responsible for primary motion.

Algorithm 3 Overall flow of our pipeline

Require: $V^0, W, B^0, K_B, \theta, D, \gamma$
1: initialize_simulator(B^0, D, γ)
2: **for all** frames t **do**
3: $\theta^t \leftarrow \theta(t)$
4: $B^R, M^R \leftarrow$ forward_kinematics(B^0, K_B, θ^t)
5: $B^* \leftarrow$ simulate(B^R)
6: $B^D \leftarrow$ kinematic_constraints(B^*, K_B)
7: $M^D \leftarrow$ get_bone_transforms(B^D)
8: $M \leftarrow$ compose_transforms(M^D, M^R, D)
9: $V \leftarrow$ skinning(V^0, W, M)
10: **end for**

In Algorithm 3, spring bone indices D and parent-bone relations K_B, mass-spring parameters $\gamma : (k_s, k_d, d_s, m)$ are given by the user together with rest shape V^0, binding weights W, rest pose bones B^0 and the animation keyframes θ at the initialization stage. In line 3, the bone poses θ^t for the current frame is taken and fed into forward kinematics in line 4. After simulating and projecting kinematic constraints, we compute RST transformations in line 7. Once we obtain dynamic bone transformations, in line 8 we construct final bone transformations to be fed into the underlying skinning method. We keep the bone transformations M^R obtained in forward kinematics for rigid bones and update bone transformations with M^D only for spring bones. In the end, after the initial rig and mass-spring parameters setup, our pipeline automatically computes the dynamic bone orientations B^D and vertex locations V.

4. Results

Spring bones achieve plausible dynamic motion over a diverse set of subjects, including rigid bodies, elastic bodies, soft tissues, and garment animation. For instance, a plant pot is a composition of rigid bodies. Under the motion, the plant leaves can rotate and translate; however, scaling is not allowed. To achieve such dynamics, we simulate the bones attached to leaves with spring bones under our `fixed_scale` option to preserve the original bone lengths.

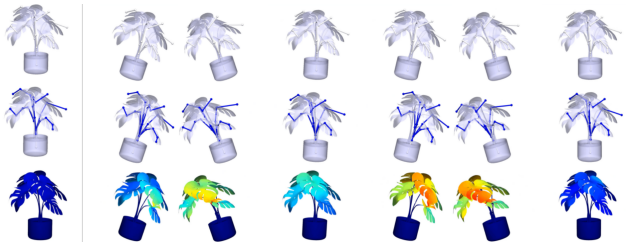


Figure 9: *Skinning of a shaking Monstera plant pot. Top: LBS deformation. White bones indicate rigid bones. Middle: Our dynamic deformation achieved with spring bones colored as blue. Single rigid white bone is left on the root that is responsible for shaking the pot. Bottom: Our deformation. Notice that the leaves are shaking side to side whereas the pot remains rigid. This is achieved via weight painting the pot area with 1.0 weights for the rigid root node.*

Note that for the plant, duck, and cloth animations in Figures 9, 10, and 11, only the root bone is rotated by the user. Due to the forward kinematics, the same rotation is inherited by all of the descendant bones. This single rotation produces a stiff motion in LBS as expected. Given this input, our method automatically infers the dynamic motion in a single shot. Regardless of the underlying geometric skinning pipeline, our produced dynamics are cumbersome to replicate manually because the 3D artist would have to keyframe the rotations one by one for numerous frames. We generally use similar stiffness and damping parameters for our main results. For parameter details and to further inspect our secondary animation contributions, please refer to the supplemental video.

We color-coded our results with the Euclidean distance difference between the rigid deformation vertices and our deformation. Red regions indicate the largest Euclidean distance between LBS and ours, while blue regions represent zero distance where the deformation exactly matches with LBS. Rest poses are given at the left of the figure, where rigid bones are colored white and spring bones are colored blue.

4.1. Helper bones

Converting existing rig bones to spring bones allows the user to impose global dynamics throughout the 3D character. Alternatively,

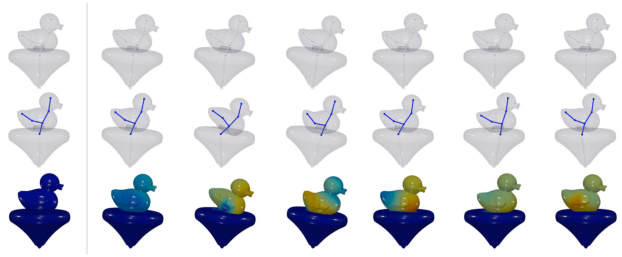


Figure 10: *An elastic body example is demonstrated on the rubber duck model rotated back and forth. Top: LBS. Middle: Ours. Bottom: Ours color-coded in comparison to LBS. Our method achieves smooth dynamic motion as if the rubber duck is floating on water without actually simulating any contact forces.*

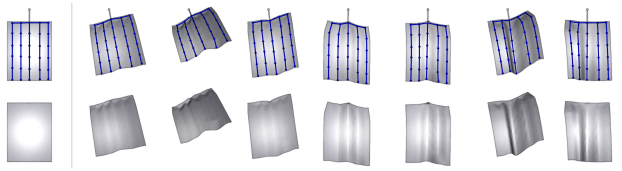


Figure 11: *SDS applied on a piece of cloth rotated back and forth. Parallel chains of spring bones form an indented surface when the garment is deformed.*

adding spring helper bones to the original rig allows imposing local dynamics such as jiggling of fatty belly area, as a complementary motion.

We test our method on the popular SMPL model for fat tissue jiggling [LMR*23]. We use pose sequences provided in the DFAUST dataset [BRPB17] and obtain rigid deformation of these poses with the SMPL. Then, we set up helper bones extending the SMPL skeleton. Once our framework simulates the helper bones, we add the simulated motion on SMPL deformation to achieve soft tissue jiggling. (Figure 13-14). Unlike classical skinning, the SMPL model is defined on blendshapes. Therefore, for SMPL demonstrations we directly add the bone tail translations on the blendshaped mesh between each current frame t and the previous frame $t-1$ such that bone i transformation is given by $B^D_{2i}(t) - B^D_{2i}(t-1)$ for $t \in [1,...,N]$ where N is the number of animation frames and the first frame is taken as initial pose $B^D_{2i}(0) = B^0_{2i}$. We observe that the bone tail translations are enough to produce plausible jiggling in local areas; however, as seen in Figure 8, this approach can create artifacts on large flat surfaces.

Helper bones are also handy for garment animation. We observe that adding parallel spring chains can approximate cloth dynamics as in Figure 11. A failure case is demonstrated in Figure 12 where a single spring chain fails to approximate the cloth surface. Both cloth rigs are bound to the cloth surface automatically with heat diffusion [BP07] weights.

As an advantage, spring rigs enable more user control over the cloth deformation, as opposed to traditional cloth simulation where the user has limited control over the garment deformation. Traditionally, the user can control the material properties by changing the mass and stiffness parameters; however, the computed result highly depends on the physical simulation. In our pipeline, the user has control over the physical simulation parameters (mass and stiffness parameters), and the rig parameters (placement and binding weights of primary or helper bones). These controls enable the user to achieve versatile dynamics and improve the degrees of freedom.

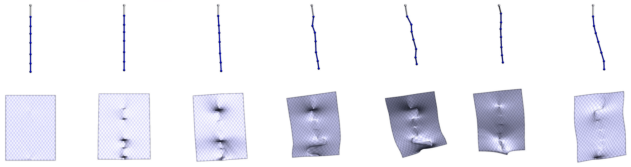


Figure 12: *SDS with a single spring chain fails to produce cloth dynamics.*

4.2. Comparison

We provide a comparison with Fast Complementary Dynamics in Figure 18. We also compare our results with a recent paper by Wu and Umetani [WU23] which provides a framework for controllable skinning dynamics. In their work, Wu and Umetani simulate a tetrahedral mesh with Position Based Dynamics (PBD) constrained in a complementary subspace, inspired by [ZBLJ20]. Then, they infer the dynamic handle locations with inverse kinematics. Similar to our approach, Wu and Umetani aim to unify physical simulation with the skinning pipeline. However, their method is complementary to ours: they first simulate the mesh and then infer the handle locations, whereas we first simulate the rig and subsequently infer dynamically deformed vertices based on simulated bone transformations.

In Figures 15 and 16, user input is applied to the yellow handles in LBS, and the same input is given to all the skinning methods. Green points indicate the free handles whose dynamic locations are computed via inverse kinematics and reds are the fixed handles in Wu and Umetani's method. In our case, we use blue spring bones, e.g. to simulate ears and trunk in Figure 16.

Note that for the comparisons with [WU23], deformation is based on point handles. On the other hand, classical skinning applications feature a skeleton hierarchy. Our RST algorithm is designed for the skeleton bones, assuming the bones have a positive length. Therefore, in these comparisons, we only use translations of the point handles for the skinning transformations. We observe that translations can be sufficient in local dynamics of non-flat surfaces as also demonstrated on SMPL in Figure 13. We convert these point handles to point spring bones, creating a zero-length spring (Figure 2). Comparison in Figure 15 demonstrates how the point spring handles result in a global secondary motion. Additional spring helper bones are required for localized soft tissue jiggling, e.g. around the elephant ears and trunk in Figure 16.

Simulating the tetrahedral mesh can introduce an unpredicted motion on the mesh (Figure 15). In comparison, our method remains controllable as it doesn't depend on the mesh vertices. Furthermore, inspired by Complementary Dynamics [ZBLJ20], Wu

Figure 13: *Helper spring bones (blue bones) are used as an addition to primary bones to simulate fat tissue dynamics. Top: animation with SMPL blendshapes. Bottom: our simulated bones' motion is added on top of blendshapes deformation.*

Figure 14: *The same helper spring bones in Figure 13 are used to simulate fat tissue dynamics for a different animation sequence. Top: animation with SMPL blendshapes. Bottom: our simulated bones' motion is added on top of blendshapes deformation. Color coding indicate the highest to lowest change in vertex Euclidean distances with respect to top row.*

and Umetani (2023) introduce a complementary motion that is orthogonal to the rigid motion. However, the resulting motion might not match the desired dynamics. For instance, in Figure 15 Spot the cow rocks back and forth; however, the complementary motion produces rotation of the cow head from side to side, producing an entirely different motion. In comparison, our point springs enhance the rocking motion by exaggerating the movement in the same direction.

Another problem with Wu and Umetani's tetrahedral simulation arises when detailed rigs are used (Figure 16, second and fifth row). In volumetric simulation, collapsing skinning artifacts often appear as cracks between neighboring tetrahedra. With detailed rigs, these artifacts become more frequent around closely positioned bones, resulting in self-intersections and rapid volume loss across multiple springs. Under a time-stepping scheme, this condition can cause abrupt changes in spring forces around the neighborhood and ultimately amplify spring forces. In Wu and Umetani's framework, PBD simulation and inverse kinematics have a feedback system, prone to severe instabilities under these conditions. Additionally, detailed rig input increases stiffness in complementary dynamics (Figure 18, which opposes the intuition of deformation control through rigging. Since our method is based on rig simulation, our dynamics remain mostly independent from mesh resolution and controllable through handles. Lastly, tetrahedralization is not a trivial task to deploy on existing animation as it requires the re-assembly of binding weights and texture, prone to excessive manual labor. As an advantage, our method allows working on shell meshes that can be easily adopted on existing rigged characters. In short, spring bones avoid tetrahedral simulation disadvantages and remain controllable and scalable for different mesh and rig resolutions.

4.3. Timing

Similar to LBS, our time complexity is mainly dependent on the number of bones in the rig rather than the number of vertices. For

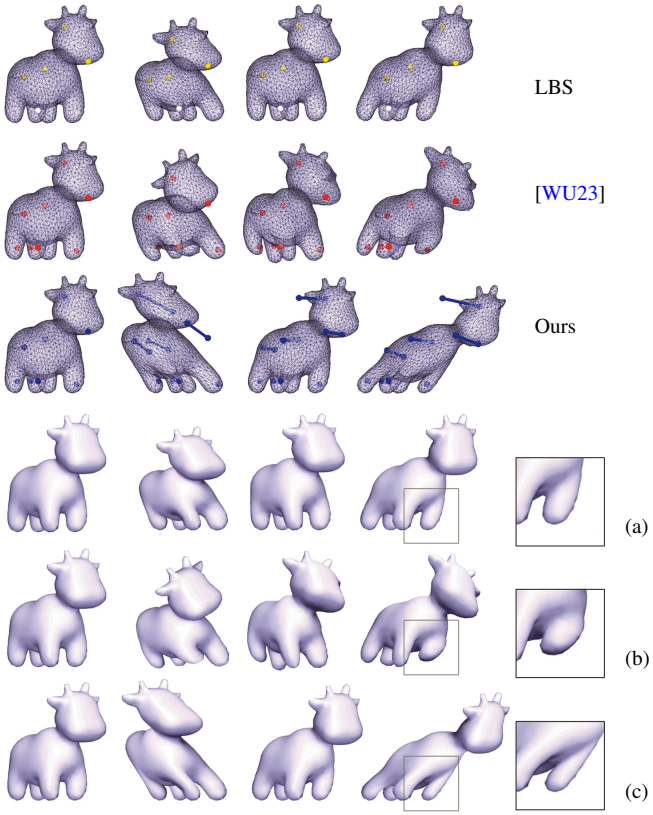


Figure 15: *Comparison for rocking motion. (a) LBS. (b) In [WU23], the fixed handles (red dots in the second row) are intended to limit the PBD simulation on the bounded vertices. Despite all the handles are being fixed and the leg handles are not moved during the animation, notice that the cow legs are still affected by the tetrahedral mesh simulation (detail box), which reduces user controllability over the animation. (c) Spring bones used in primary handles result in global stretch/squish dynamics, specifically squishy and stretchy dynamics.*

example, deformation on Duck model is faster than Cloth and Monstera models (Table 1) despite having a significantly larger number of vertices, due to its lower number of rig bones. Note that in Table 1, SMPL deformation is faster because its demonstration doesn't include skinning computations, i.e. simulated bone tail translations are directly added on top of blendshapes. For the rest of the models in Table 1, we use our RST algorithm to compute bone transformations.

Our method is around 10 times faster compared to [WU23] as our implementation mainly depends on the number of rig bones and avoids simulating the tetrahedral mesh vertices. For comparison, we also use additional helper bones for all the models in Table 2, doubling the number of total bones.

The experiments are conducted on a 2.5 GHz Quad-Core Intel Core i7 processor, 16 GB of RAM. For both our results and [WU23] comparisons we use Python 3.9.18. Furthermore, our implementation is dependent on Numpy 1.26.4 while Wu and Umetani's implementation is dependent on Taichi 1.7.1.

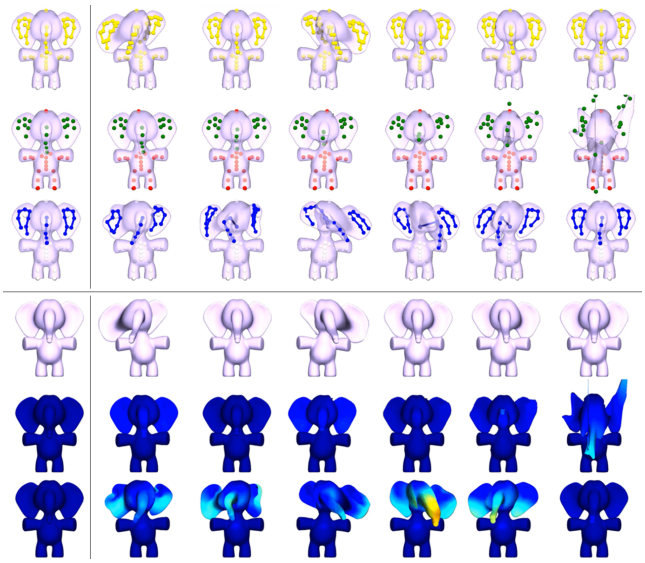


Figure 16: *Comparison with a helper bone rig. Top 3 rows show how the rig moves during the animation and bottom 3 rows are their final mesh rendering results of LBS, Wu and Umetani (2023), and ours respectively. Rest poses are given on the left side.*

Model	$\|V\|$	$\|B_s\|/\|B\|$	LBS (ms)	Ours (ms)
Cloth	2025	22/23	5.26	12.21
Monstera	4971	23/24	6.19	14.11
Duck	12932	5/6	4.51	9.50
SMPL	8541	9/25	N/A	4.67

Table 1: *Per frame timings on average. $|V|$ is the number of mesh vertices. $|B_s|$ is the number of spring bones and $|B|$ is the total number of bones in the rig. Note that SMPL model is based on blendshapes skinning so that LBS is omitted in the table. The time required for rendering is not included.*

5. Discussions

5.1. Methods in the Industry

Industrial methods use jiggle bones to simulate secondary motions, specifically focused on animal tails, ponytail hair, or muscle jiggling. [Wan23] constrain the spring motion within a cone, creating a controlled range for muscle jiggling. Unreal Engine also provides dynamic bones similar to those described by [Wan23].

To the best of our knowledge, existing applications do not consider bone scaling. In contrast, we also provide the option to scale the bones which enables secondary motion globally, such as squash or stretch effects similar to Velocity Skinning, as shown in Figure 15. The introduction of free and fixed masses and their kinematic

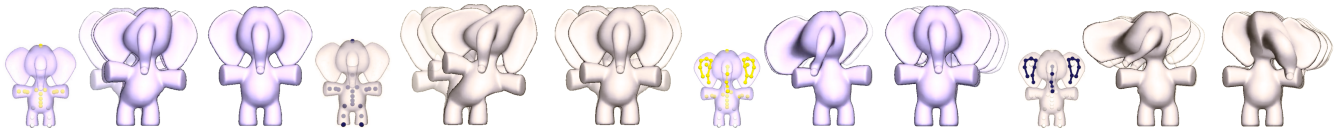


Figure 17: *In LBS rigs, yellow handles are translated throughout the animation, dark blue handles are our spring bones simulated during the animation. (a) LBS. (b) Ours. Spring bones used as primary bones impose global dynamics such as floppy and squishy deformation. (c) LBS with helper bones around ears and trunk. (d) Ours with helper bones. Spring bone chains in helper bones impose local dynamics.*

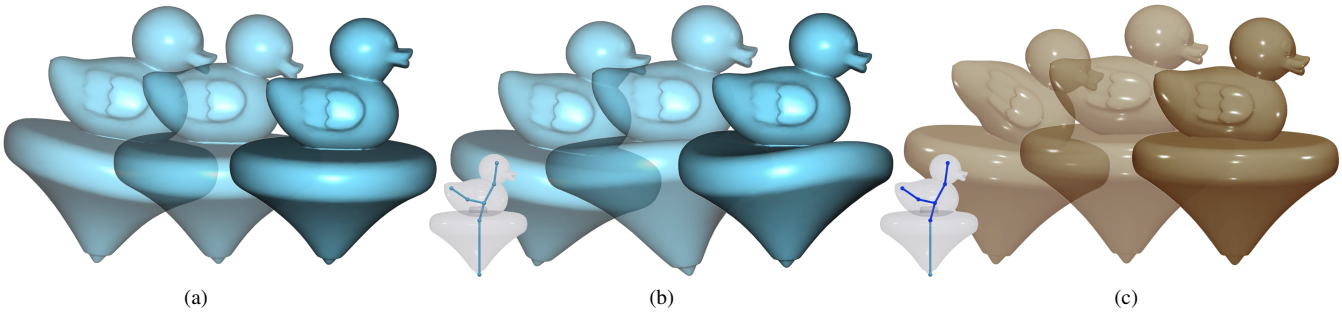


Figure 18: *Comparison with Fast Complementary Dynamics [***fastcompdyn***] (a) with a single handle (b) with a full rig (light blue bones). (c) Ours with a full rig where spring bones are colored blue. Notice that adding a detailed rig in [***fastcompdyn***] severely reduces dynamics in the duck body attached to multiple bones, with secondary motion concentrated in the bottom plate attached to a single bone. Our method controls secondary motion via handles, i.e. keeping the bottom plate rigid through a single rigid bone, while the rest of the duck moves dynamically following the spring bones (see supplemental video).*

Model	$\|V_{tet}\|$	$\|B\|$	LBS (ms)	Ours (ms)	[WU23] (ms)
Spot	837	8	1.15	1.98	20.73
	837	15	1.86	3.45	27.10
Spot (HQ)	1762	8	1.29	2.40	22.97
	1762	15	2.13	4.05	31.23
Elephant	8541	25	4.14	7.87	90.41
	8541	47	6.16	11.91	114.79

Table 2: *Timing measurements of comparison with volumetric mesh input in milliseconds. Note that the time required for rendering is not included. $|V_{tet}|$ denotes the tetrahedral vertex count. $|B|$ is the number of bones in the rig. Timings are averaged over 500 frames. For consistency, all bones are taken as spring bones in ours, and fixed bones in Wu and Umetani's method.*

constraints allows us to define scaling or optionally preserve original bone lengths.

On top of semi-implicit Euler method which results in similar motion in industrial applications, we utilize PBD with optional constraints, which was useful for local dynamics such as belly fat jiggle in Figures 13 and 14. Overall this framework allows us to define local and global secondary dynamics that generalize to a broad range of applications.

While tools like Unity and Blender also offer cloth dynamics for mass-spring simulation, these typically operate as classical simulations applied directly to mesh vertices. In contrast, our work introduces a skinning-based framework that generalizes secondary motion by representing dynamic mesh deformation as a linear combination of a small set of springs, rather than simulating each vertex position individually.

Although it is possible to create jiggle-bone effects in existing tools by assigning parent-child relationships to vertex groups, Spring Decomposed Skinning formalizes this process into a unified framework, providing a groundwork for future enhancements to secondary motion systems.

5.2. Creative Workflow

Our proposed framework is designed to be easily integrated into existing skinning pipelines and can be rapidly applied within common 3D animation software. In particular, a rigged animation sequence created in software such as Blender can be exported into our framework. Once the user selects the spring bones and sets the simulation parameters, they can achieve dynamic deformation in a plug-and-play fashion. Parameter settings given in the supplemental video are mostly similar across various models.

If necessary, users can add helper bones to drive local secondary motion. While this requires recomputation of binding weights, in most cases, automatic methods for vertex-bone binding weights are sufficient. Minimal weight painting may be needed to impose rigid material constraints, such as binding the pot of a plant to a rigid bone, as in Figure 9, or the bottom plate of the duck in Figure 18. The presented results are primarily bounded by automatic heat dif-

fusion weights. In specific models, including plant pot and duck, minimal weight painting is applied to control rigid material behavior; however, we avoid extensive weight painting and primarily rely on automatic weight computations to provide fast prototyping.

5.3. Limitations and Future Work

So far we have studied Hookean spring bones for their sinusoidal nature and simplicity. Even though the results are promising, our system inherits the primary limitations of Hookean spring forces. In [MHTG05], limitations of mass-spring systems are discussed. Essentially, mass-spring systems are sensitive to chosen parameters and incorrect parameter settings can cause instability. Even with an unconditionally stable framework such as PBD, the exploding forces still exist. This is also because our system is position-based, rather than force-based. That is, the user directly determines the positions of the bones, in contrast to applying forces to determine the positions. Therefore, the force required to move the rig from one orientation to another can explode depending on the system parameters. As a future work, more robust spring systems can be adapted for better stability.

Our spring bones are limited to 3 Degrees of Freedom (DoF). Specifically, the springs are translated along *xyz* axes in 3D space without twisting or bending. To allow a larger DoF and achieve twisting motion, angular velocities can also be considered as in [GBFP11].

In our RST algorithm, we directly use matrices to rotate the vectors; however, for DQS and other skinning methods, a quaternion-based rotation can also be employed that can also be used to avoid the potential volume collapse of rotation matrices. Furthermore, the scope of our proposed RST algorithm can be further studied to evaluate its effectiveness.

We also observe that the longer helper bone chains are more difficult to control to obtain a desirable secondary motion. This is also because self-intersections can occur without collision detection. We leave collision detection and integrating other external forces as future work. Finally, we manually set up simulation parameters to achieve the desired secondary animation. Although the parameters are often shared across different examples, automatizing the SDS framework is also an open area of research.

6. Conclusion

Spring Decomposed Skinning proposes a simple yet effective framework to emulate a real-time and diverse range of secondary animation on existing geometric skinning pipelines. The scope of SDS dynamics covers a wide range, including secondary motion on rigid bodies, elastic bodies, soft tissues, and garment animation. Unlike the common FEM-based approaches, SDS does not require mesh tetrahedralization which significantly improves time complexity and is easy to integrate with existing animation pipelines.

Furthermore, SDS directly simulates the bones, providing artistic control over the physical deformation. With this approach, as opposed to volumetric mesh simulations directly simulating mesh vertices, SDS aligns with the core intuition behind skinning by decomposing the complex surface dynamics into a linear combination of simple spring motions. Ultimately, SDS shows a promising direction to achieve real-time, controllable, and versatile dynamics.

7. Acknowledgments

We thank the anonymous reviewers for their constructive comments. This work was supported by TUBITAK under the project EEEAG-124E183.

Appendix A: Computation of RST

Initially, we take both rest pose bone and deformed pose bone line segments to their bone space where their head is at the origin. At the precomputation stage, for every bone i an offset matrix $\mathbf{B}_o = \begin{bmatrix} I & -B_{1i}^0 \\ 0 & 1 \end{bmatrix}$ and its inverse $\mathbf{B}_o^{-1} = \begin{bmatrix} I & B_{1i}^0 \\ 0 & 1 \end{bmatrix}$ is stored where $\mathbf{I} \in R^{3\times 3}$ is the identity matrix.

Then, both the rest pose bone i and simulated bone i is translated to bone space by offset $\mathbf{B}_o$ (step 1 in Figure 7). In the bone space, the amount of translation matrix $\mathbf{T}$ for RST matrix is saved as:

$$\mathbf{t}_i^* = B_{1i}^D - B_{1i}^0 \qquad (5) \qquad\qquad \mathbf{T} = \begin{bmatrix} I & \mathbf{t}_i^* \\ 0 & 1 \end{bmatrix} \qquad (6)$$

and the target bone is translated to origin such that its tail location is computed as $B_{2i}^{D'}$ and in bone space, the rest bone's tail is located at $B_{2i}^{0'}$:

$$B_{2i}^{D'} = B_{2i}^D - B_{1i}^0 - \mathbf{t}_i^* \qquad (7) \qquad\qquad B_{2i}^{0'} = B_{2i}^0 - B_{1i}^0 \qquad (8)$$

At this point, our problem is to find the rotation transformation to align $R : B_{2i}^{0'} \rightarrow B_{2i}^{D'}$. Let θ be the angle between two normalized vectors $p, q \in R^3$ where $\|p\| = \|q\| = 1$, then $c = cos\theta = p \cdot q$ and $s = sin\theta = \|p \times q\|$. Let the rotation be around axis $r = [r_x, r_y, r_z]$ where $r = p \times q / \|p \times q\|$ and let $\kappa = 1 - c$. Rotation matrix R^* that aligns q to p can be computed by Rodrigues' formula in matrix form:

$$R^* = \begin{bmatrix} \kappa r_x^2 + c & \kappa r_x r_y - r_z s & \kappa r_x r_z + r_y s \\ \kappa r_x r_y + r_z s & \kappa r_y^2 + c & \kappa r_y r_z - r_x s \\ \kappa r_x r_z - r_y s & \kappa r_y r_z + r_x s & \kappa r_z^2 + c \end{bmatrix} \qquad (9)$$

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}^* & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \qquad (10)$$

We also know spring simulation scales the spring bones. To match the scaling between rest pose bone (source) and dynamically posed bone (target), we rotate the rest bone's tail by applying rotation matrix $\mathbf{R}$ to the rest bone tail $B_{2i}^{0'}$. The scaling between two aligned vectors is given by:

$$s = \frac{\left\| B_{2i}^{D'} \right\|}{\left\| \mathbf{R}^* B_{2i}^{0'} \right\|} \quad (11) \qquad \mathbf{S} = \begin{bmatrix} sI & \mathbf{0} \\ 0 & 1 \end{bmatrix} \quad (12)$$

We combine these **R**otate, **S**cale, **T**ranslate (RST) transformation matrices in Equations 10, 12, and 6 in

$$\mathbf{M}^* = \mathbf{TSR} \quad (13)$$

Lastly, we transform the bones from bone space to world space back. The overall affine transformations can be chained into a single transformation as:

$$\mathbf{M}_i^D = \mathbf{B_0^{-1}} \mathbf{M}^* \mathbf{B_0} \quad (14)$$

that is used to transform the rest pose bone to its posed space inside the underlying skinning method. In our experiments we use dynamic bone transformations $\mathbf{M}_i^D$ for transformation matrices $\mathbf{M}_i$ in Equation 4.

egbibsample