Real-Time Rigging and Secondary Motion for Sketch-Based 3D Characters

Chau Thi Ma and Tam Minh Le(✉)

Faculty of Information Technology, VNU University of Engineering and Technology, Hanoi, Vietnam
letam166204@gmail.com

Abstract. Sketch-based modeling systems simplify the creation of 3D characters, but animating them remains a challenging task due to the complexity of rigging and secondary motion control. Traditional animation workflows require extensive manual effort to define skeletal structures and dynamic motion, making the process time-consuming and technically demanding. This creates a gap between creative expression and technical challenges. To address the problem, we propose an interactive framework that combines real-time skeletonization with secondary motion generation using a mass-spring system. Our system includes three key functions: (i) the user sketches a character, (ii) the system automatically generates a rigged skeleton, and (iii) the system applies dynamic animations without the need for manual keyframing. The system combines the best aspects of two methods: the simplicity and accessibility of sketch-based modeling, and the expressive, physics-driven secondary motion provided by the mass-spring system. Beginner animators can create expressive, physics-driven animations effortlessly without tedious adjustments.

Keywords: Sketch-based modeling · Mass-spring system · Real-time animation

1 Introduction

Computer graphics (CG) play a fundamental role in various industries, including video games, virtual and augmented reality (VR/AR), scientific visualization, material simulation, and education [18]. In the gaming industry, animations enhance player immersion and bring characters to life. VR and AR applications rely on dynamic 3D content to create interactive experiences, while material simulations use physics-based animations to study real-world behavior in engineering and scientific fields. As CG continues to expand into new domains, the demand for accessible and efficient 3D modeling and animation tools has grown significantly.

However, creating 3D animations remains a challenging and labor-intensive process that requires expertise in multiple disciplines. 3D modeling, the first

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15898, pp. 213–228, 2026.
https://doi.org/10.1007/978-3-031-97657-5_13

step in animation, involves constructing detailed objects and characters, often requiring years of practice with tools like Blender[1] 3ds Max[2] and Maya[3] To simplify this process, sketch-based modeling systems such as Teddy [12], MonsterMash [8], and FiberMesh [20] have been developed. These systems allow users to generate plausible 3D models from 2D sketches, significantly lowering the barrier to entry for beginners.

While sketch-based modeling helps create static 3D models, rigging—the process of defining an internal skeleton and binding it to a mesh—remains a major hurdle. Rigging requires technical proficiency and manual adjustments to ensure proper articulation for animation. To address this, automatic skeletonization systems such as RealSkel [16] and RigMesh [5] have been introduced, reducing the need for manual rigging and streamlining the animation workflow.

Even with rigged models, animation is still complex, especially when incorporating secondary motion effects like swaying hair, bouncing cloth, or dynamic character movement. Various physics-based animation techniques have been developed to assist animators, including Fast Mass-spring Systems [15], spring bone dynamics in tools like Cartoon Animator 5[4] These techniques automate secondary motion, enhancing realism without requiring frame-by-frame keyframing.

As seen in this workflow, animation is a multi-step process that includes modeling, rigging, and animation, not to mention additional elements like texturing and lighting [9]. Even though many tools exist, beginner animators may still struggle to integrate all these processes seamlessly. The steep learning curve and technical complexity can hinder creativity and slow down production. To bridge this gap, we introduce a sketch-based 3D modeling system with real-time skeletonization and built-in spring bone dynamics. Our system leverages the real-time skeletonization technique RealSkel [16] by integrating the Fast Mass-spring system [15]. This allows for automatic rigging while introducing real-time dynamic, bouncy effects, eliminating the need for manual setup. Our method is heavily inspired by Fast Mass-spring Systems [15] but differs in two key aspects. While their study focuses on physically accurate motion, our framework emphasizes secondary motion to enhance expressiveness, with less focus on achieving perfect realism. Secondly, whereas existing approaches treat mass-spring systems as mesh-based physics simulations for cloth and hair, we simplify the process by using spring bones for object animation. This makes our system more intuitive and accessible for novice users in animation.

2 Background and Notation

2.1 Related Works

Sketch-Based Modeling. The pioneering sketch-based system, Teddy [12], significantly simplifies shape modeling by allowing users to create 3D forms with

[1] Blender2023: https://www.blender.org.
[2] Max2023: https://www.autodesk.com/products/3ds-max.
[3] Maya2023: https://www.autodesk.com/products/maya.
[4] Cartoon Animator 5: https://www.reallusion.com/cartoon-animator/.

just a few strokes. Over time, numerous follow-up works [8,14,20,25,26] have expanded on this approach, enabling the creation of more complex shapes and making sketch-based modeling more comprehensive. These systems offer a fun and intuitive experience for novice users, making 3D modeling more accessible. However, their simplicity can be a drawback for professionals who require precise control over model accuracy.

Rigging and Skeletonization. In modern character animation pipelines, a rig consists of a skeleton, a cycle-free graph where nodes represent joints and edges represent bones. Skin weights define how the model's surface deforms in response to skeletal movement [5]. Because of the complexity of the manual process, automatic skeletonization and rigging techniques have been extensively researched. Given a static character mesh and a generic skeleton, Pinocchio [3] adapts the skeleton to the character and attaches it to the surface, allowing skeletal motion data to animate the character. RigMesh [6] extends this concept by creating a rig for each sketched part in real-time, and update the rig as parts are merged or cut. More recent approaches have leveraged deep learning for automation. RigNet [29] introduces a neural network that predicts a skeleton that matches the animator expectations in joint placement and topology. For single-image rigging, Object Wake-up [30] reconstructs 3D objects and generates skeletons from 2D images using deep implicit functions. Specifically for sketch-based modeling, RealSkel [16] provides a real-time skeletonization technique. Despite these advancements, automatic skeletonization and rigging alone is not sufficient for high-quality animation. Proper rigging still requires refinement, such as adjusting joint positions, fine-tuning skin weights, and incorporating inverse kinematics (IK).

Mass-Spring System. Mass-spring systems have long been used in both scientific computing and physics-based animation to simulate movement and deformation [4]. These systems represent objects as networks of interconnected masses and springs, enabling realistic simulations of elasticity, bending, and other physical behaviors. Early models for cloth simulation [7,22] demonstrated how simple physics-based structures could produce visually convincing motion. They have also been applied to approximate the deformation of volumetric solids [27] and linear elements such as hair [23,24]. Spring bone dynamics, as seen in tools like Cartoon Animator 5[5] and VRM[6] provide simplified secondary motion effects, making them valuable for game development and AR/VR applications. Researchers have also improved these methods for greater stability and efficiency, with the Fast Mass-Spring method [15] introducing a more efficient solver for real-time deformable object simulation.

After reviewing related works, we find that RealSkel [16] and the Fast Mass-Spring method [15] align well with our research objectives. These methods fulfill key requirements, including real-time skeletonization to quickly generate an ani-

[5] Cartoon Animator 5: https://www.reallusion.com/cartoon-animator/spring-animation.html.

[6] Vrm Springbone: https://vrm.dev/en/univrm/springbone/univrm_secondary/.

matable skeleton from sketches, and efficient mass-spring simulations for responsive motion. Therefore, in the following sections, we examine the fundamental concepts behind these two influential works. A clear understanding of their methodologies is essential for implementing our system and effectively integrating their strengths.

2.2 Real-Time Skeletonization for Sketch-Based Modeling

The method consists of three main steps: local sub-skeleton extraction, sub-skeleton connection, and global skeleton refinement. First, the local skeleton is extracted from the processed polygon stroke, forming a subpart along with the corresponding sub-mesh. Next, individual sub-skeletons are connected based on their intersections and the sequential order in which the subparts were created. Finally, a global refinement process allows users to adjust the skeleton structure with coarse-to-fine control, ensuring an optimized and animation-ready hierarchy.

2.3 Fast Simulation of Mass-Spring Systems

The fast mass-spring system technique provides an efficient way to solve mass-spring systems governed by Hooke's Law through a rapid implicit method [15]. Instead of directly solving the system's equations, this technique reformulates the problem as an optimization task. Based on Newton's second law, we derive:

$$\mathbf{p}_{n+1} = \mathbf{p}_n + h\mathbf{v}_{n+1} \tag{1}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{M}^{-1}\mathbf{f}(\mathbf{p}_{n+1}) \tag{2}$$

Using the position update equation from implicit Euler integration, we can express velocity in terms of discrete differences.

Backward Difference Approximation. The velocity at time step n is given by:

$$h\mathbf{v}_n = \mathbf{p}_n - \mathbf{p}_{n-1} \tag{3}$$

Forward Difference Approximation. Similarly, the velocity at the next time step $n+1$ can be written as:

$$h\mathbf{v}_{n+1} = \mathbf{p}_{n+1} - \mathbf{p}_n \tag{4}$$

To eliminate velocity from the implicit Euler update, we multiply Eq. (2) by h and substitute Eqs. (3) and (4). To simplify notation, define $\mathbf{x} := \mathbf{p}_{n+1}$ and $\mathbf{y} := 2\mathbf{p}_n - \mathbf{p}_{n-1}$. Multiplying both sides of this equation by the mass matrix $\mathbf{M}$, we obtain a more compact form:

$$\mathbf{M}(\mathbf{x} - \mathbf{y}) = h^2\mathbf{f}(\mathbf{x}) \tag{5}$$

Reformulating the Energy Potential. As stated in the paper by [15], the main idea of their technique is to reformulate the energy potential E in a way that will allow us to employ a block coordinate descent method. In a mass-spring system, the elastic potential energy represents the energy stored in a deformed spring. According to Hooke's Law, the potential energy function is given by:

$$E(\mathbf{p}_i, \mathbf{p}_j) = \frac{1}{2}k(\|\mathbf{p}_i - \mathbf{p}_j\| - r_{ij})^2 \tag{6}$$

Because of the $\|\mathbf{p}_i - \mathbf{p}_j\|$ term, calculating E is a non-linear problem. The presence of this square root function means that the energy function is not a simple quadratic function. An interesting discovery has been made to simplify and accelerate the optimization algorithm. Specifically, for each spring, the nonlinear energy function can be reformulated as a smaller, independent optimization problem. This means, we need to find the vector of length that is as close as possible to the current spring vector.

$$(\|\mathbf{p}_i - \mathbf{p}_j\| - r_{ij})^2 = \min_{\mathbf{d}_{ij} \in \mathbb{R}^3, \|\mathbf{d}\| = r_{ij}} \|(\mathbf{p}_i - \mathbf{p}_j) - \mathbf{d}_{ij}\|^2$$

Since $\mathbf{f}(\mathbf{x})$ represents the force, and force is the negative gradient of potential energy [10], we have:

$$\mathbf{f}(\mathbf{x}) = -\nabla E(\mathbf{x}) \tag{7}$$

Substituting this into Eq. (5) gives:

$$\mathbf{M}(\mathbf{x} - \mathbf{y}) = -h^2 \nabla E(\mathbf{x}) \tag{8}$$

This matches the critical point condition of the function:

$$g(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{y})^T \mathbf{M}(\mathbf{x} - \mathbf{y}) + h^2 E(\mathbf{x}) \tag{9}$$

since setting $\nabla g(\mathbf{x}) = 0$ results in:

$$\mathbf{M}(\mathbf{x} - \mathbf{y}) + h^2 \nabla E(\mathbf{x}) = 0 \tag{10}$$

Thus, the solutions to Eq. (5) correspond to the critical points of the function given in (9). This leads to an optimization formulation $min_{\mathbf{x}} g(\mathbf{x})$. This formulation, known as variational implicit Euler [19], allows solving the motion equations using optimization techniques instead of direct integration. This approach avoids numerical instability and large time-step restrictions that often arise in explicit integration methods [2].

From the method described in the paper [15], we know that the potential energy of the system can be written as:

$$E(\mathbf{x}) = \min_{\mathbf{d} \in U} \left(\frac{1}{2}\mathbf{x}^T \mathbf{L}\mathbf{x} - \mathbf{x}^T \mathbf{J}\mathbf{d} + \mathbf{x}^T \mathbf{f}_{\text{ext}} \right) \tag{11}$$

where $U = \left\{(\mathbf{d}_1, \ldots, \mathbf{d}_s) \in \mathbb{R}^{2s} \mid \|\mathbf{d}_i\| = r_i\right\}$ represents the set of rest-length spring directions.

Now, substituting Eq. (11) into (9), we obtain:

$$\begin{aligned} g(\mathbf{x}) &= \frac{1}{2}(\mathbf{x}-\mathbf{y})^T\mathbf{M}(\mathbf{x}-\mathbf{y}) + h^2 \min_{\mathbf{d}\in U}\left(\frac{1}{2}\mathbf{x}^T\mathbf{L}\ \mathbf{x} - \mathbf{x}^T\mathbf{J}\mathbf{d} + \mathbf{x}^T\mathbf{f}_{\text{ext}}\right) \\ &= \min_{\mathbf{x}\in\mathbb{R}^{3\,m},\mathbf{d}\in U}\left[\frac{1}{2}\mathbf{x}^T\mathbf{M}\ \mathbf{x} - \mathbf{x}^T\mathbf{M}\ \mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{M}\ \mathbf{y} + h^2\left(\frac{1}{2}\mathbf{x}^T\mathbf{L}\ \mathbf{x} - \mathbf{x}^T\mathbf{J}\mathbf{d} + \mathbf{x}^T\mathbf{f}_{\text{ext}}\right)\right] \end{aligned} \tag{12}$$

Since we are minimizing with respect to $\mathbf{x}$, the constant term $\mathbf{y}^T\mathbf{M}\ \mathbf{y}$ can be ignored. Also, for simplification we introduce $\mathbf{b}$ as $\mathbf{b} = h^2\mathbf{f}_{ext} - \mathbf{M}\mathbf{y}$. We obtain the following:

$$g(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^{3\,m},\mathbf{d}\in U}\left(\frac{1}{2}\mathbf{x}^T(\mathbf{M}+h^2\mathbf{L})\mathbf{x} - h^2\mathbf{x}^T\mathbf{J}\mathbf{d} + \mathbf{x}^T\mathbf{b}\right) \tag{13}$$

Thus, Eq. (13) represents the final optimization problem, where the minimizer x^* provides the exact solution to the implicit Euler timestep.

3 Method

3.1 System Overview

Figure 1 highlights the key differences between the original RealSkel [16] system (left) and our proposed approach (right). Both frameworks follow a similar initial workflow: beginning with a hand-drawn 2D silhouette, performing automatic skeleton extraction, and optionally allowing users to add subparts.

In RealSkel, once the skeleton is generated and the shape undergoes skinning, users can directly interact with joints to produce real-time rigid animations. By contrast, our approach extends this workflow by integrating a Mass-Spring System for Skeletons after the skinning stage. This addition introduces secondary motion effects, allowing for real-time dynamic animation. In comparison, the original system applies strictly rigid transformations, lacking the dynamic movement achieved in our method.

We divide our system into three procedures:

1. **Automatic Skeleton Extraction:** The system first applies constrained Delaunay triangulation [21] to process the input stroke. Here, we define the user's hand-drawn input as the silhouette of the model. The spine of the polygon, which also serves as the model's skeleton, is then automatically determined using the method described by [1].
2. **3D Shape Generation & Skinning:** A 3D shape is then generated from the 2D silhouette using the approach outlined in [12]. Initially, at this stage, we elevate the vertices of the spine by an amount proportional to their distance from the polygon. The user, however, can adjust the model's inflation by modifying the thickness parameter afterward. The system constructs a polygonal mesh that wraps around the spine and polygon, ensuring that sections

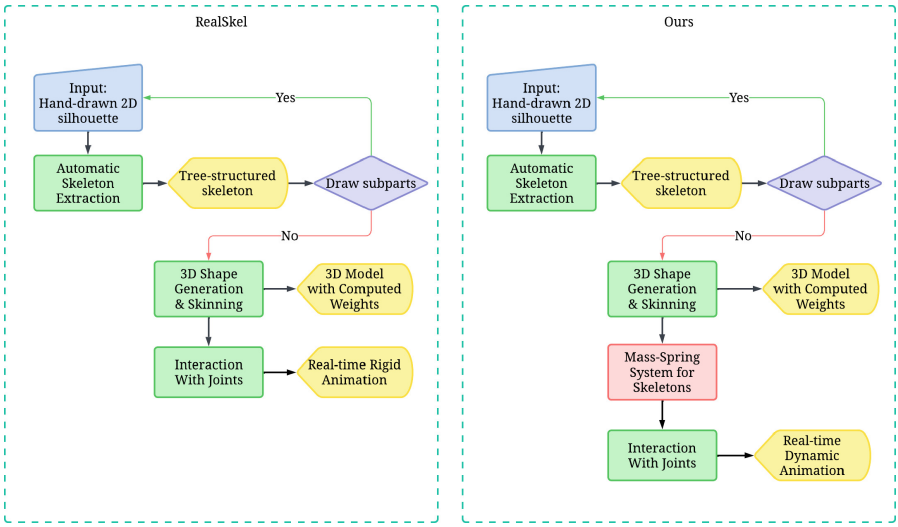

Fig. 1. Comparison of RealSkel and Our Approach. The original RealSkel system (left) directly applies rigid animation after skeleton extraction and skinning. Our approach (right) introduces a Mass-Spring System for Skeletons (highlighted in pink), adding secondary motion effects for real-time dynamic animation. (Color figure online)

form oval shapes and calculate. Skinning weights are then automatically computed using Bounded Biharmonic Weights (BBW) [13], preparing the model for skinning transformation via Linear Blend Skinning (LBS).

3. **Mass-spring System for Skeletons** To implement a mass-spring system for skeletal dynamics, we adapt the method proposed by [15], originally designed for cloth simulation, to a hierarchical skeletal structure. Unlike cloth, which is typically modeled as a network graph with interconnected nodes, a skeleton is inherently tree-structured, where each joint has a single parent and potentially multiple children. We treat each joint as a mass node and each bone as a spring to simulate skeletal dynamics. Although there are three types of springs: structural, bend and shear [11], in this study, we have not yet implemented shear and bend springs, which also play a crucial role in simulating more flexible deformations. Instead, we focus solely on structural springs to maintain bone length while introducing secondary motion effects.

The user can then draw additional subparts to expand the model, triggering a new cycle of skeletonization, meshing, and skinning.

3.2 Functions

In this section, we present the system's functionality from the user's perspective (see Fig. 2).

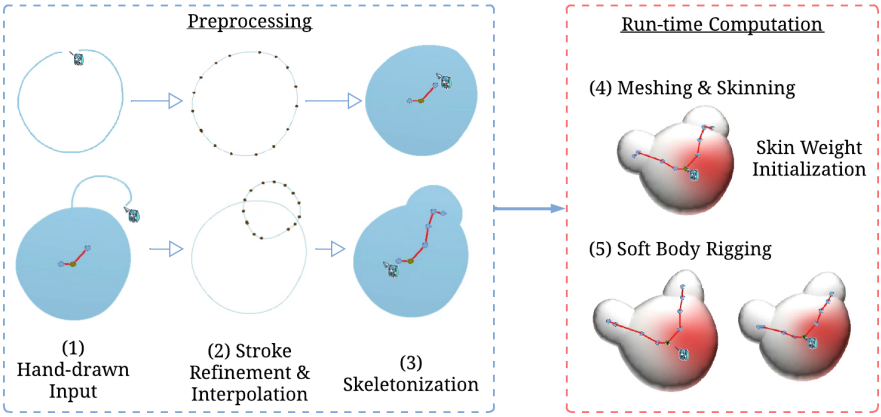

Fig. 2. Overview of the sketch-based soft body rigging system. (1) The user provides a hand-drawn input to define the contour of the shape. (2) Stroke refinement and interpolation are performed to generate a cleaner representation. (3) A skeleton is extracted from the refined strokes to define the deformation structure. The first row illustrates the process when drawing a single part of the 3D model, while the second row shows the process of merging multiple sub-parts to create a larger model. (4) A meshing and skinning process assigns weights to the structure, generating a simulation mesh. (5) The system applies soft body rigging using a mass-spring system to enable physics-driven deformation. The final animation utilizes a combination of LBS for primary deformation and a Spring Bone system for secondary dynamic motion.

First, the user begins by providing a hand-drawn sketch, which serves as the initial input for the modeling process. The system then refines and interpolates the strokes to generate a smooth and consistent shape. Next, a skeleton is extracted from the refined strokes, defining the structural framework of the model. Once the skeleton is established, the user can press 'D' to initiate the meshing and skinning stage. The system automatically computes weight initialization based on the mesh structure. Soft-body rigging is applied using a mass-spring system, enabling the model to exhibit dynamic deformations. The user can then select individual joints in the skeleton to manipulate, causing the entire structure to move dynamically as if connected by springs. Notably, when the user grabs a joint, it is considered fixed within the mass-spring system and is fully controlled by user movement for ease of use and convenience.

3.3 Implementation

Thanks to the work by [15], computing the mass-spring system state at each time step is simplified into a straightforward iterative process based on a local-global approach:

– **Step 1 (Local Update)**: Given the current positions $\mathbf{p}$, determine d_{ij} for each spring.

- **Step 2 (Global Update)**: Using all computed vectors d_{ij}, update $\mathbf{x}$ by minimizing the quadratic energy function described in Eq. (13).
- **Step 3 (Convergence Check)**: If the solution has not yet converged, return to Step 1. If the maximum number of iterations is exceeded, terminate the algorithm.

We implemented our method using *C++* and the *Qt* framework[7] *CMake*[8] is also required for build configuration and project management. The linear system was solved using the *Eigen* library[9].

The source code for the original RealSkel system is publicly available at: https://github.com/jingma-git/RealSkel.

Our implementation is available at: https://github.com/maple1606/Sketch-to-Model

Building Matrices. Before solving for the new positions of nodes, we construct three key matrices $\mathbf{L}$, $\mathbf{J}$ and $\mathbf{M}$ described in the original paper [15]. The process is detailed in Algorithm 1.

Algorithm 1. Construction of $\mathbf{L}$, $\mathbf{J}$, and $\mathbf{M}$

Require: Spring system with n points and s springs.
Initialize empty sparse matrices $\mathbf{L}$, $\mathbf{J}$, and $\mathbf{M}$.
$h^2 \leftarrow$ time step squared.
for each spring k connecting nodes (i, j) **do**
  for each coordinate $d \in \{x, y, z\}$ **do**
    $l_{3i+d,3i+d}$ += k_k, $l_{3j+d,3j+d}$ += k_k
    $l_{3i+d,3j+d}$ −= k_k, $l_{3j+d,3i+d}$ −= k_k
  end for
end for
for each spring k connecting nodes (i, j) **do**
  for each coordinate $d \in \{x, y, z\}$ **do**
    $j_{3i+d,3k+d}$ += k_k, $j_{3j+d,3k+d}$ −= k_k
  end for
end for
for each node i **do**
  for each coordinate $d \in \{x, y, z\}$ **do**
    $m_{3i+d,3i+d} \leftarrow m_i$
  end for
end for
Precompute $\mathbf{A} = \mathbf{M} + h^2\mathbf{L}$ and factorize for solving.

[7] Qt2023: https://www.qt.io.
[8] CMake2025: https://cmake.org/.
[9] eigenweb: https://eigen.tuxfamily.org.

Local Update. The local update relies on local information for each spring. Specifically, it computes the spring direction vectors for each spring in the system, ensuring that they maintain their rest length constraint. Algorithm 2 details the local update step.

Algorithm 2. Local Update for Mass-spring System

Require: Current positions $\mathbf{p}$, rest lengths r_{ij} of springs
Ensure: Updated spring direction vectors $\mathbf{d}_{ij}$
1: $j \leftarrow 0$
2: **for** each spring $(i, j) \in$ system's spring list **do**
3: Compute displacement vector: $\mathbf{p}_{12} \leftarrow \mathbf{p}_i - \mathbf{p}_j$
4: Normalize the displacement: $\hat{\mathbf{p}}_{12} \leftarrow \frac{\mathbf{p}_{12}}{\|\mathbf{p}_{12}\|}$
5: Compute updated spring direction: $\mathbf{d}_{ij} \leftarrow r_{ij} \cdot \hat{\mathbf{p}}_{12}$
6: Store $\mathbf{d}_{ij}$ in spring direction list
7: $j \leftarrow j + 1$
8: **end for**

Global Update. The global update, on the other hand, involves all springs simultaneously. In this step, we solve for $\mathbf{x}$ while keeping $\mathbf{d}$ fixed. We start from the quadratic minimization problem described in Eq. (13). To find the optimal $\mathbf{x}$, we take the gradient with respect to $\mathbf{x}$ and set it to zero:

$$\frac{\partial}{\partial \mathbf{x}} \left(\frac{1}{2} \mathbf{x}^T (\mathbf{M} + h^2 \mathbf{L}) \mathbf{x} - h^2 \mathbf{x}^T \mathbf{J} \mathbf{d} + \mathbf{x}^T \mathbf{b} \right) = 0 \tag{14}$$

Since $\mathbf{L}$ is symmetric and positive semi-definite, the system matrix $\mathbf{M} + h^2 \mathbf{L}$ is symmetric positive definite, this simplifies to:

$$(\mathbf{M} + h^2 \mathbf{L}) \mathbf{x} - h^2 \mathbf{J} \mathbf{d} + \mathbf{b} = 0 \tag{15}$$

Rearranging, we obtain:

$$(\mathbf{M} + h^2 \mathbf{L}) \mathbf{x} = h^2 \mathbf{J} \mathbf{d} - \mathbf{b}. \tag{16}$$

This is a convex quadratic system where $\mathbf{M}$ represents the mass matrix, $h^2 \mathbf{L}$ accounts for stiffness from the implicit Euler integration, $h^2 \mathbf{J} \mathbf{d}$ encodes local deformation constraints, and $\mathbf{b}$ contains external forces and previous state dependencies.

Since the system matrix $\mathbf{M} + h^2 \mathbf{L}$ is symmetric positive definite, this guarantees the existence of a unique solution, which we solve efficiently using Cholesky factorization. By precomputing this factorization (since the system matrix remains constant when mass, timestep, and connectivity do not change), we significantly speed up the computation in each iteration. Algorithm 3 details the global update step.

Algorithm 3. Global Update Step

Require: Precomputed matrices $\mathbf{M}, \mathbf{L}, \mathbf{J}$ and external forces $\mathbf{f}_{\text{ext}}$.

1: Compute the right-hand side:

$$\mathbf{b} = h^2 \mathbf{f}_{\text{ext}} - \mathbf{M}\mathbf{y}.$$

2: Compute $\mathbf{x}$:

$$\mathbf{x} = (\mathbf{M} + h^2 \mathbf{L})^{-1} (h^2 \mathbf{J}\mathbf{d} - \mathbf{b}).$$

3: Update the system state with $\mathbf{x}$.

Convergence Check. For ease of implementation, our simulation assumes convergence after a fixed number of iterations (e.g., 100). However, more advanced stopping criteria could be employed to adaptively determine convergence based on error thresholds or energy minimization.

Algorithm 4. Mass-Spring Solver with Fixed Iterations

Require: Maximum iterations n

1: Store previous state: $\mathbf{p}_{n-1} \leftarrow \mathbf{p}_n$
2: **for** $i = 1$ to n **do**
3: **Local Step:** Update spring directions $\mathbf{d}_{ij}$ by normalizing displacement vectors
4: **Global Step:** Solve for $\mathbf{x}$ using the precomputed Cholesky factorization
5: **end for**

4 Results and Discussion

4.1 Experimental Setup

For performance evaluation, all experiments were conducted on an AMD Ryzen 7 7840H processor with Radeon 780M Graphics. To ensure consistency, auto-parallelization was disabled, and all computations were performed on a single thread. Following the sub-stepping strategy [17], we set the time step to $h = 1/60$ s, corresponding to a 60 FPS simulation rate. As for external forces, we only consider gravitational acceleration $g = 9.8\,\text{m/s}^2$ [10] and user-applied forces, which arise from direct interactions such as pulling, pushing, or manipulating the system.

As for user interaction, the object responds to physics forces only when the user moves a joint with the mouse, allowing for precise control over movement. However, a limitation of this approach is that forces are not applied continuously, meaning the object lacks ongoing secondary motion unless actively manipulated. As a result, animation techniques like follow-through, slow-in, slow-out [28] may appear less pronounced.

4.2 Dynamic Motion Effects

Figure 3 illustrates the dynamic deformation response of our method under different external forces. The leftmost objects represent the initial state before any deformation is applied. The green dot marks the central joint, which, when moved, causes the entire object to follow its position, exhibiting dynamic effects.

When subjected to stretching forces (moving the mouse in one direction), the skeleton expands, causing the structure to elongate. Conversely, applying compression forces (moving the mouse in the opposite direction) causes the skeleton to contract. The stress distribution, visualized in red, highlights regions experiencing higher deformation. In contrast, without incorporating a mass-spring system into the bones, the object remains rigid and unchanged regardless of how the central joint is moved.

However, a significant limitation of our method stems from the automatic BBW computation within the framework by [16]. The weight distribution is uneven, as evidenced by the red stress area being concentrated in a specific region rather than being evenly spread across the entire model. Moreover, users cannot manually adjust the weights or change the structure of the skeleton, which further restricts control over the deformation behavior. As a result, moving a blue joint does not propagate its influence effectively to other joints, reducing the clarity of localized deformations. While re-implementing BBW could enhance

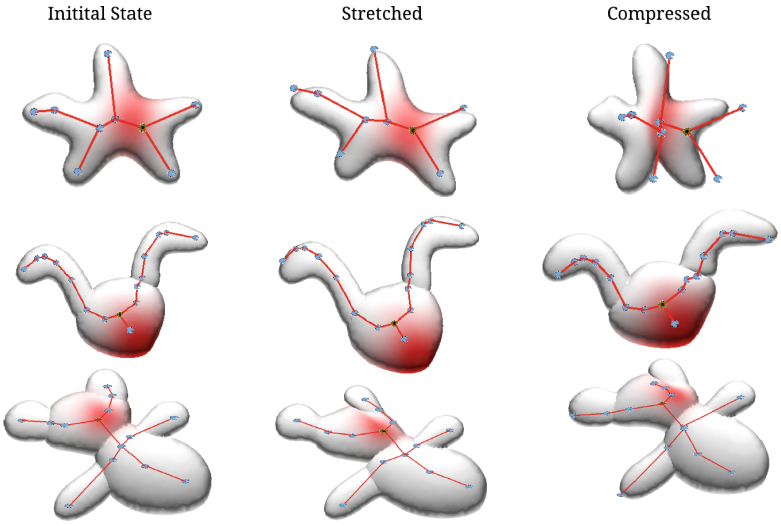

Fig. 3. Dynamic deformation effects under different external forces. The leftmost column represents the initial state before deformation. The stretched column shows the effect of applying external forces that elongate the structure, while the compressed column illustrates the deformation when forces are applied in the opposite direction. The internal skeleton (red lines) adapts dynamically, and the stress distribution (highlighted in red) indicates regions experiencing high deformation. (Color figure online)

weight distribution and deformation accuracy, this is beyond the scope of our study. This limitation remains a key challenge in our approach.

Another limitation is that while we successfully introduce secondary animation, the resulting motion is not as bouncy as observed in well-known frameworks such as Cartoon Animator 5[10] and VRM[11] This can be attributed to the absence of shear and bending springs in our implementation, as discussed in Sect. 3.1.

4.3 Effects of Stiffness Parameter

Stiffness determines how resistant a spring is to deformation, controlling how quickly it returns to its original shape after being stretched or compressed. The unit of stiffness is Newtons per meter (N/m) [10], meaning a higher stiffness value results in less deformation under the same force, leading to more rigid motion [4]. Therefore, adjusting the stiffness parameter is key to getting the best dynamic effects for each character. Since no pre-computation is needed, users can tweak stiffness in real time to find the most natural movement. As shown in Table 1, choosing the right stiffness is important. Because our system adds spring effects to bones for simple object animation (not for cloth or hair), lower stiffness values generally work better.

Table 1. Effects of different stiffness levels on dynamic motion.

Stiffness Level (N/m)	Effect
Too Low (<5)	Unstable, vibrating, or numerical errors in calculation
Low (5–10)	Fast but may overshoot, slight jittering
Medium (10–100)	Balanced, elastic, natural-looking motion
High (>100)	Slow, bouncy motion is barely noticeable

4.4 Complexity

The local update function, as described in Sect. 3.3, iterates over all springs in the system and updates their directions. The complexity for this step is therefore $\mathcal{O}$(n).

The global update function, as described in Sect. 3.3, involves solving a linear system. Since we use Cholesky decomposition, the factorization of the $n \times n$ system matrix requires $\mathcal{O}(n^3)$ operations. Once factorized, solving for x using forward and backward substitution takes $\mathcal{O}(n^2)$ time. The overall complexity of the global update step is therefore $\mathcal{O}(n^3)$. Although this complexity may seem large, our system remains computationally feasible as it is applied to skeletal

[10] Cartoon Animator 5: https://www.reallusion.com/cartoon-animator/spring-animation.html.

[11] VRM Springbone: https://vrm.dev/en/univrm/springbone/univrm_secondary/.

structures rather than dense meshes. In typical cases, the number of joints does not exceed $n = 1000$, leading to $n^3 = 1000^3 = 1,000,000,000$. Empirical tests show that for typical character skeletons ($n \leq 30$), the update process completes within milliseconds, ensuring smooth interactive performance. Thus, the computational cost is acceptable within our application domain.

5 Conclusion

In this work, we introduced a sketch-based 3D modeling system that integrates real-time skeletonization with mass-spring dynamics, enabling users to easily create soft-body effects for 3D characters. By leveraging the RealSkel [16] framework for automatic skeleton extraction and mass-spring physics [15] for skeletal dynamics, our system allows for the generation of dynamic secondary motion without the need for complex rigging or extensive animation expertise.

While the system successfully bridges sketch-based modeling and dynamic animation, there are several avenues for future work. One important direction is to refine the weight distribution process to achieve smoother and more consistent deformations across the model. Another key area is enabling continuous secondary motion without requiring user input, allowing the model to exhibit persistent dynamic effects. Additionally, expanding the system to include bend and shear springs would further enhance the flexibility and realism of the character's movements. To ensure that the system remains efficient and scalable, future work will also focus on improving computational performance, particularly for more complex models or detailed animations.

By addressing these challenges, we aim to enhance the system's capabilities, making it a more powerful and user-friendly tool, not only for animators but also for educational purposes in simulating materials.

References

1. Aichholzer, O., Aurenhammer, F.: Straight skeletons for general polygonal figures in the plane. In: Cai, J.-Y., Wong, C.K. (eds.) COCOON 1996. LNCS, vol. 1090, pp. 117–126. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61332-3_144
2. Baraff, D., Witkin, A.: Large steps in cloth simulation. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, pp. 43–54. Association for Computing Machinery, New York (1998). https://doi.org/10.1145/280814.280821
3. Baran, I., Popović, J.: Pinocchio: automatic skeleton generation for 3D models. ACM Trans. Graph. (TOG) **26**(3), 33 (2007)
4. Bargteil, A.W., Shinar, T.: An introduction to physics-based animation. In: ACM SIGGRAPH 2019 Courses. SIGGRAPH 2019. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3305366.3328050
5. Borosán, P., Jin, M., DeCarlo, D., Gingold, Y., Nealen, A.: Rigmesh: automatic rigging for part-based shape modeling and deformation. ACM Trans. Graph. **31**(6), 1–9 (2012). https://doi.org/10.1145/2366145.2366217

6. Borosán, P., Auzinger, T., Wimmer, M.: Rigmesh: automatic rigging for articulated characters. Comput. Graph. Forum **31**(2pt3), 381–390 (2012)
7. Bridson, R., Fedkiw, R., Anderson, J.: Robust treatment of collisions, contact and friction for cloth animation. ACM Trans. Graph. **21**(3), 594–603 (2002). https://doi.org/10.1145/566654.566623
8. Dvoroznak, M., Sykora, D., Curless, B., Curtis, C.J., Sorkine-Hornung, O., Salesin, D.H.: Monster mash: a single-view approach to casual 3D modeling and animation, pp. 1–12. New York, NY, USA (2020)
9. Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F.: Computer Graphics: Principles and Practice, 2nd edn. Addison-Wesley Longman Publishing Co., Inc, USA (1990)
10. Halliday, D., Resnick, R., Walker, J.: Fundamentals of Physics. Halliday & Resnick Fundamentals of Physics. John Wiley & Sons Canada, Limited (2010)
11. Hsiao, S.W., Chen, R.Q.: A method of drawing cloth patterns with fabric behavior. In: Proceedings of the 5th WSEAS International Conference on Applied Computer Science, ACOS 2006, pp. 635–640. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA (2006)
12. Igarashi, T., Matsuoka, S., Tanaka, H.: Teddy: A Sketching Interface for 3D Freeform Design, 1st edn. Association for Computing Machinery, New York (1999)
13. Jacobson, A., Baran, I., Popović, J., Sorkine, O.: Bounded biharmonic weights for real-time deformation. ACM Trans. Graph. **30**(4) (2011). https://doi.org/10.1145/2010324.1964973
14. Karpenko, O.A., Hughes, J.F.: Smoothsketch: 3D free-form shapes from complex sketches. ACM Trans. Graph. **25**(3), 589–598 (2006)
15. Liu, T., Bargteil, A.W., O'Brien, J.F., Kavan, L.: Fast simulation of mass-spring systems. ACM Trans. Graph. **32**(6) (2013). https://doi.org/10.1145/2508363.2508406
16. Ma, X., Xie, S., Takayama, K.: Realskel: real-time skeletonization for sketch-based modeling. Comput. Graph. Forum **40**(7), 123–133 (2021)
17. Macklin, M., et al.: Small steps in physics simulation. In: Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation. SCA 2019. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3309486.3340247
18. Marschner, S., Shirley, P.: Fundamentals of Computer Graphics, 4th edn. A. K. Peters Ltd, USA (2016)
19. Martin, S., Thomaszewski, B., Grinspun, E., Gross, M.: Example-based elastic materials. ACM Trans. Graph. **30**(4) (2011). https://doi.org/10.1145/2010324.1964967
20. Nealen, A., Igarashi, T., Sorkine, O., Alexa, M.: Fibermesh: designing freeform surfaces with 3D curves. ACM Trans. Graph. **26**(3), 41-es (2007). https://doi.org/10.1145/1276377.1276429
21. Paul Chew, L.: Constrained delaunay triangulations. Algorithmica **4**(1–4), 97–108 (1989). https://doi.org/10.1007/BF01553881
22. Provot, X.: Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In: Proceedings of Graphics Interface 1995, GI 1995, pp. 147–154. Canadian Human-Computer Communications Society, Toronto, Ontario, Canada (1995)
23. Rosenblum, R.E., Carlson, W.E., Tripp, E., III.: Simulating the structure and dynamics of human hair: modelling, rendering and animation. J. Vis. Comput. Animat. **2**(4), 141–148 (1991)
24. Selle, A., Lentine, M., Fedkiw, R.: A mass spring model for hair simulation. ACM Trans. Graph. **27**(3), 1–11 (2008). https://doi.org/10.1145/1360612.1360663

25. Sugihara, M., De Groot, E., Wyvill, B., Schmidt, R.: A sketch-based method to control deformation in a skeletal implicit surface modeler. In: Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling, SBM 2008, pp. 65–72. Eurographics Association, Goslar, DEU (2008)
26. Tai, C.L., Zhang, H., Fong, J.: Prototype modeling from sketched silhouettes based on convolution surfaces. Comput. Graph. Forum **23**, 71–84 (2004). https://doi.org/10.1111/j.1467-8659.2004.00006.x
27. Teschner, M., Heidelberger, B., Müller, M., Gross, M.: A versatile and robust model for geometrically complex deformable solids, pp. 312–319 (2004). https://doi.org/10.1109/CGI.2004.1309227
28. Thomas, F., Johnston, O.: Disney Animation: The Illusion of Life. Abbeville Press, New York (1981)
29. Xu, Z., Zhou, Y., Kalogerakis, E., Landreth, C., Singh, K.: Rignet: neural rigging for articulated characters. In: ACM Transactions on Graphics (TOG), vol. 39, p. 67. ACM (2020)
30. Yang, J., et al.: Object wake-up: 3D object rigging from a single image (2021). https://arxiv.org/abs/2108.02708