II Simposio CEA de los GT: Ingeniería de Control - Modelado, Simulación y Optimización - Educación en Automática (2026)

Elastic Deformable Material Simulation in Unity based on Mass-Spring-Damper Models

Sellart, K. D.*, Oña, E. D., Łukawski, B., Jardón, A.

RoboticsLab, University Carlos III of Madrid, Avda. de la Universidad, 30, 28911 Leganés, Madrid, España.

To cite this article: Sellart, K. D, Oña, E. D., Łukawski, B., Jardón, A. 2026. Elastic Deformable Material Simulation in Unity based on Mass-Spring-Damper Models. Actas del II Simposio CEA de los GT: Ingeniería de Control - Modelado, Simulación y Optimización - Educación en Automática. Zaragoza, España.

Resumen

La simulación de materiales elásticos deformables es relevante para el desarrollo de aplicaciones robóticas que involucren contacto utilizando motores de videojuegos como Unity. Este trabajo aborda el desarrollo de un modelo de material deformable basado íntegramente en las primitivas físicas nativas de Unity (*Rigidbodies* y *Joints*). El modelo propuesto está basado en sistemas Masa-Muelle-Amortiguador (MSD) para mallas planas y multicapa generadas proceduralmente. Este modelo proporciona un método simplificado y computacionalmente más ligero, que permite una simulación adecuada con menos recursos y alto realismo en la deformación. Los resultados demuestran la viabilidad de generar y manipular materiales con propiedades mecánicas configurables en tiempo real, ofreciendo una herramienta eficiente y transparente para la investigación en robótica.

Palabras clave: Deformación elástica, Unity, Sistemas Masa-Muelle-Amortiguador, Robótica, Modelado 3D

Abstract

The simulation of deformable elastic materials is relevant for the development of robotic applications involving contact using game engines like Unity. This paper addresses the development of a deformable material model based entirely on Unity's native physics primitives (Rigidbodies and Joints). To achieve this, a deformable elastic material model based on Mass-Spring-Damper (MSD) systems is implemented for procedurally generated planar and multilayer meshes. This model provides a simplified and computationally lighter method, enabling adequate simulation with fewer resources and high realism in deformation. The results demonstrate the feasibility of generating and manipulating materials with configurable mechanical properties in real time, offering an efficient and transparent tool for robotics research.

Keywords: Elastic deformation, Unity, Mass-Spring-Damper Systems, Robotics, 3D Modeling

1. Introduction

The simulation of soft bodies and deformable materials has acquired critical importance across various engineering and computer science disciplines, particularly in the fields of robotics and medical simulation (Qin et al., 2024). The ability to accurately predict a deformable material's response to external forces is fundamental for developing complex robotic applications, such as force-control systems.

Although there is specific software designed for soft material simulation, most are costly or possess a very steep learning curve. An example is the case of SOFA (SOFA, 2006), an open-source framework renowned for complex physical simulations, particularly within the medical field. It supports both Mass-Spring-Damper (MSD) models for faster, real-time simulations, and Finite Element Method (FEM) for high-fidelity, complex scenarios. Additionally, SOFA includes plugins to interact with game engines such as Unity, running the SOFA simulation in the background to compute deformation. However, this solution reduces Unity's simulation performance due to two-way communication with the external tool.

In this regard, simulation of material deformation in the native environment of Unity would improve the real-time simulation performance. Thus, various packages such as *Obi* (Obi, 2019) or *DefKit* (Defkit, 2016) are available in the game en-

*Corresponding author: kenjisellart@gmail.com

gine. The Obi package consists of several modules including cloth, fluid, and soft body simulations, among others. It is a high-fidelity tool with a highly successful real-time simulation, however it is a commercial (paid) solution. This tool operates using Position Based Dynamics (PBD), which, while allowing for very stable simulations (especially in highly rigid materials and fluids) and superior quality, has a steeper initial learning curve due to its more complex mathematical foundations.

On the other hand, the DefKit tool utilizes an MSD model based on native primitives (Rigidbody, Collider, and Spring Joint), but relies on tetrahedral formations. Although it integrates native Unity components, it exhibits deficiencies in simulations involving 2D and 3D objects with large flat surfaces.

In this context, this paper focuses on creating an elastic deformable material model based on a procedural mesh generation method for Unity game engine. Physics of the proposed model will be managed by the native physics engine (NVIDIA PhysX, integrated into Unity) since its core is based on MSD links built with native primitives such as *Rigidbodies* and *Joints*. Besides, the MSD model offers a favorable balance between physical accuracy and computational efficiency (Nealen et al., 2006). Therefore, this approach aims to replicate as realistically as possible the behavior of elastic deformable materials in Unity. However, the goal is not exact material fidelity, but operational realism for interactive simulation of robotic applications that involve physical contact or textile garment manipulation.

2. Architecture for procedural mesh generation

The mesh model presented in this article is fundamentally based on the use of native Unity *GameObjects* to generate a grid structure using the MSD model as its operational logic. Due to the high density of elements required to form the mesh, this process has been entirely automated through a procedural script. Consequently, Unity's physics engine assumes the primary role in the dynamic simulation, while the code performs an auxiliary function focused on element generation and data acquisition. Furthermore, it is important to highlight that the code[1] is responsible for generating and updating the visual representation of the mesh in real-time.

2.1. Mesh unit architecture

The mesh unit architecture consists of four fundamental components: nodes, *Rigidbody* (mass), *Spring Joint* (linkage), and *Sphere Collider* (collision volume). Nodes constitute the basic unit of the mesh and are generated in an ordered quadrangular or rectangular layout. To maintain a structured hierarchy, they are grouped under parent objects termed *Mesh*. Each node is assigned a *Rigidbody* to grant it mass attributes and gravity response. Finally, *Sphere Colliders* are added to nodes to provide them with a physical body for environmental interaction.

To link these nodes and provide the mesh with elastic behavior, the *Spring Joint* (SJ) component is utilized. This element integrates both a spring and a damper within a single component. By joining two nodes through this linkage, the basic modular architecture of the mesh is established, as shown in Figure 1(a).

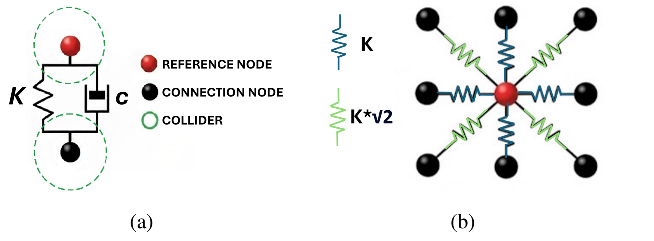


Figure 1: Architecture composition of the mesh. a) Basic connection joint. b) Possible connections from reference node to surrounding nodes.

The union of these units forms the complete structure, where each node connects with all its contiguous neighbors, resulting in a maximum 8-connectivity (Moore neighborhood). Given the orthogonal arrangement of the nodes, connectivity could be restricted to 4 or extended to 8; the latter was chosen as it allows for a more uniform force distribution and higher deformation precision, as illustrated in Figure 1(b).

2.2. Mesh generation procedure

Mesh generation begins by defining the base and height of the mesh, as well as the spacing between nodes (scale), to calculate the total number of nodes. In the mesh, each node is an independent object; therefore, a script vector is initialized to store node references for easier access. During this first step of node generation, *Rigidbody* and *Sphere Collider* components are added to each node, while simultaneously setting it as a child of an *Empty Object* for better organization.

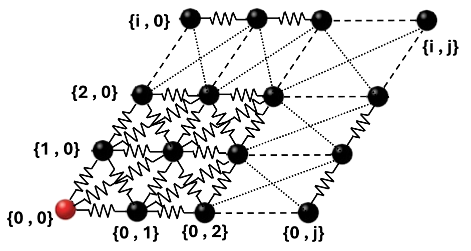


Figure 2: Architecture of the single-layer mesh structure.

Once all nodes have been processed, a second iteration is performed to add the SJ components between all the adjacent nodes, which provide the mesh with its elastic properties. To avoid adding redundant springs and compromising the mesh's isotropy, the nodes in the vector are traversed in order following a specific pattern during spring instantiation. This ensures that all necessary springs are generated to maintain the 8-connectivity mentioned in the previous section, allowing the resulting mesh to exhibit an isotropic load distribution, as illustrated in Figure 2. Note that nodes located at the edges and corners have fewer adjacent neighbors, resulting in a reduced number of connections for these specific elements.

The final step involves the visual rendering of the mesh. Although this does not influence the physical behavior of the

[1] The code developed in this paper is available in `https://github.com/KenjiDavid/unity_deformable_material_simulation`.

model, it allows for a more intuitive visualization of its changes. This procedure relies on Unity's *Mesh* class, which utilizes triangle-based topology to generate and model various geometries, ranging from terrains to character meshes. Since the material in this work is also based on triangles, this system is used to generate the visual representation of the mesh. This is the base procedure that repeats in all different versions of this work, which are attained through modification of this template, as shown in Algorithm 1.

Algorithm 1: MSD-based Mesh Generation

```
Data:
  1. Mesh Base ($B$) and Height ($H$)
  2. Grid Scale ($S$) (distance between nodes)
  3. Layer count ($L$) and separation ($D$) for multilayer meshes
Result: Instantiated Unity Mesh structure
Calculate total number of nodes and initialize vector $V$ to
  store references;
for each node in $V$ do
    Create Empty Object named "Node_x";
    Set Mesh GameObject as parent at assigned grid
      coordinates;
    Assign Rigidbody and Collider components to the node;
end
for each node in $V$ do
    if neighboring node exists then
        Add Spring Joint component between current node
          and neighbor;
        if system is multilayer (Based on algorithm) then
            Add Spring Joint between nodes;
            Add Spring Joint between nodes in adjacent
              layers (Multi-layer);
        end
    end
end
Execute Mesh Generation update (layer by layer for
  multilayer structures);
```

2.3. Multi-layer models for volumetric simulation (3D)

To extend the MSD model to the 3D domain and simulate materials with volume, multi-layer architectures were developed to introduce effective resistance along the vertical axis. The adopted approach consists of stacking multiple 2D meshes and interconnecting them via SJ (similar to Figure 1(a)) using a 1:1 connection where each node in the lower layer is linked to its corresponding node in the upper layer.

This initial model faced two primary drawbacks: penetration resistance and the effect of shear forces. On the one hand, utilizing colliders only on the top layer with 1:1 vertical connections proved insufficient, as the lower layers failed to interact with external objects, allowing them to penetrate the mesh more easily. This issue was resolved by adding colliders to every layer; although this increases computational cost, it significantly enhances the mesh's penetration resistance.

On the other hand, applying forces with a component parallel to the mesh makes it susceptible to shear forces, as shown in Figure 3(a), which weakens the isotropic component of the structure. To correct this defect, strategic diagonal springs were incorporated between layers, moving beyond a simple 1:1 connection to a multiple-link arrangement. Regarding the diagonal springs, the decision again arose between using a 4-connectivity or 8-connectivity (excluding the direct vertical node). In this case, 8-connectivity was deemed unsuitable, as its implementation represents a substantial increase in computational cost compared to its marginal effect on mesh behavior. Therefore, a 5-connectivity was chosen, consisting of one vertical and four diagonal links (to nodes on the perpendicular axes of the plane), as illustrated in Figure 3(b).

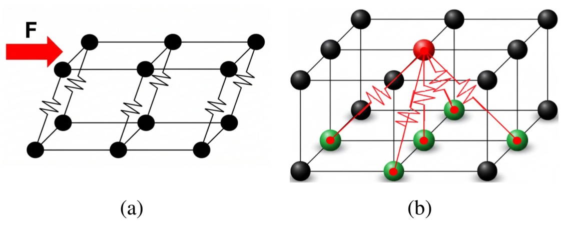


Figure 3: Multiple layer Mesh. a) Schematic of shear force effect on the multi-layer mesh, b) Generated *Spring Joints* on multi-layer mesh

Ultimately, this yields a 3D model with solidity and penetration resistance consistent with 3D objects, while maintaining the elastic behavior. Although this feature is beyond the current scope of study, it represents a functional optimization intended for future research on complex 3D interactions.

3. Possible optimizations to baseline model

3.1. Introducing angular stiffness

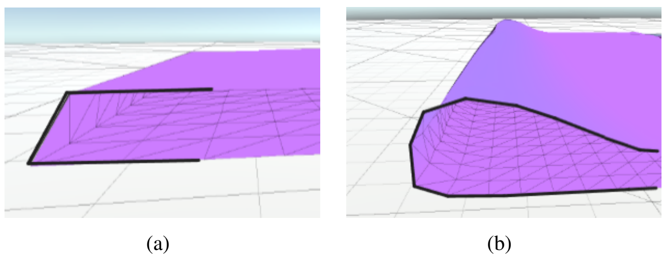


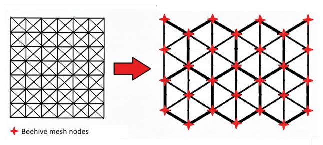
Figure 4: Behavioral difference between joints. a) Using SJ: the material collapses due to lack of angular stiffness b) Using CJ: the material doesn't collapse thanks to CJ torque.

The previous MSD model has relied on the SJ component for physical interaction. However, this element only applies an axial force (tension/compression) proportional to the distance between the two connected *Rigidbodies*. When applied to planar and volumetric meshes, this stiffness proves insufficient: while the material resists stretching, it lacks bending resistance. This technical deficiency leads to unrealistic physical behavior in certain cases where the structure collapses immediately under gravity or manipulation, similar to a sheet of paper lacking structural integrity (Figure 4(a)).

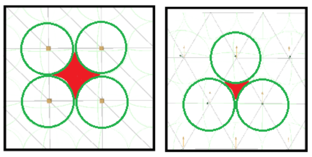
To mitigate this effect, the SJ can be replaced by the *Configurable Joint* (CJ) component during the execution of Algorithm 1. This component accurately models the bending stiffness required for a realistic physical representation of elastic materials (Figure 4(b)).

3.2. Structural optimization: the beehive mesh

As the mesh grows in size, the computational cost of MSD models scales rapidly with the number of nodes and springs; therefore, structural topology optimization is essential for high-resolution simulations. While a traditional quadrangular grid requires eight connections per internal node, the proposed beehive mesh —inspired by honeycomb structures— utilizes a hexagonal topology where each central node connects to only its six nearest neighbors (Figure 5(a)).



(a)



(b)

Figure 5: Structural comparison of mesh types. a) Traditional quadrangular mesh (8-connectivity) to beehive mesh (6-connectivity). b) Space between colliders in beehive mesh.

The mesh generation process for the beehive mesh is adapted by generating triangles from rhombuses instead of squares. Although the 6-connectivity topology introduces a slight increase in anisotropy compared to the eight-connection model, this trade-off is negligible in high-resolution meshes, where the collective behavior of nodes dilutes individual connection effects. Consequently, the beehive mesh not only increases computational efficiency, but also organizes the nodes more effectively to maximize spatial coverage (Figure 5(b)).

This hexagonal optimization enhances performance by significantly reducing the number of components required for a given resolution, as illustrated in Table 1. This reduction directly alleviates the load on Unity's physics engine, enabling the simulation of high-density meshes that would be infeasible with a traditional grid structure.

Table 1: Comparison of employed resources in beehive model.

Metric	**Grid (6x7)**	**Beehive (6x7)**	**Δ (%)**
Rigidbodies (*Physics*)	56	28	~50%
Total Springs (*Joints*)	181	63	~65%
Connectivity × Node	8	6	~25%

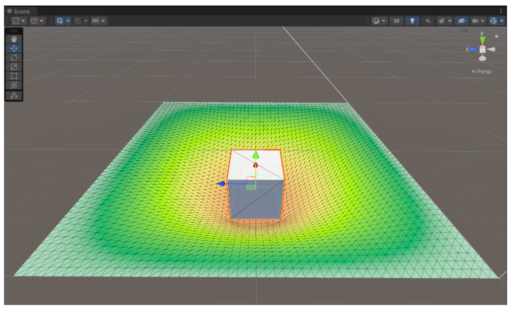
4. Preliminarily analysis of mesh elastic response

Several experiments have been performed in order to preliminarily evaluate the elastic capabilities of the proposed model. Note that the goal at this stage is not to analyze the performance with absolute mathematical rigor nor material properties. Thus, the elastic response of the mesh is analyzed using an equivalent simplified model of vertical spring deformation. A specific load will be dropped onto the generated mesh. Once stabilized, deformation data will be collected by calculating the mean vertical displacement of all nodes. This experiment will be replicated using three different loads and four distinct volumes, as well as varying mesh resolutions for both the SJ and CJ models.

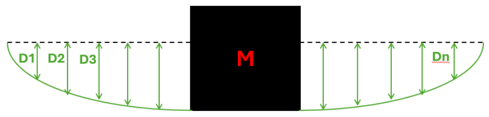
4.1. Experimental protocol

The experimental procedure has consisted of placing an external object (a cube) on the generated mesh (Figure 6(a)). Once it has reached a state of rest, the deformations of each node relative to a zero reference line have been extracted. Mean deformation (D_{mean}) has been calculated as the average vertical position of all nodes, omitting the fixed nodes at the perimeter of the structure (Figure 6(b)).

For the various simulation conditions, the base mesh has been fixed at a size of 20×20 units, with grid resolutions varying from 5×5 to 50×50 nodes. In these experiments, the *Spring* property in the generated mesh has been defined with a stiffness of 10 N/m and $\sqrt{2}\cdot 10$ N/m for the diagonal springs. Static loads have been applied using cubes with controlled masses of 1 kg, 5 kg, and 10 kg; simultaneously, their volumes have been varied from $1 \times 1 \times 1$ to $8 \times 8 \times 8$ to analyze how load distribution affects mesh deformation.



(a)



(b)

Figure 6: Experimental protocol: a) Simulation of elastic response; b) Estimated mean deformation.

For this preliminary study, the mesh has been analyzed at rest; thus, only the equivalent stiffness (K_{eq}) is considered, avoiding more complex calculations such as damping effects or elastic coefficients used in more rigorous frameworks (in materials sciences such as Poisson Coefficient). It should be noted that K_{eq} is not an intrinsic physical property of the material, but rather a simplified parameter introduced to enable an internal comparative analysis. Although the load is not a point load, we analyzed it as such; consequently, the analysis of the mesh elastic response is based on the following Equation 1:

$$\sum F = 0 \Rightarrow m \cdot g - K_{eq} \cdot D_{mean} = 0 \Rightarrow K_{eq} = \frac{m \cdot g}{D_{mean}} \quad (1)$$

In other words, the system is analyzed in the static case when the load and mesh are balanced according to Equation 1. In accordance with this protocol, two independent series of measurements have been performed on the single-layer mesh: the first using the SJ component and the second employing the CJ.

4.2. Results using the SJ-based model

Table 2 presents a sample of the raw data for the averaged deformation (D_{mean}) for each tested mesh and the K_{eq} estimated using the simplified Spring model, namely, we calculate the K_{eq} through Equation 1. It can be observed that as the mesh resolution increases, the D_{mean} value tends to stabilize. However, the values obtained from the lowest resolution mesh (5×5) are notably inconsistent with the average results of the remaining cases.

Table 2: Elastic response for Spring Joint and 10 kg load

Grid	Param.	Cube-1	Cube-2	Cube-4	Cube-8
5x5	D_{mean} [m]	1.399	1.384	1.383	1.368
	K_{eq} [N/m]	70.115	70.860	70.932	71.700
10x10	D_{mean} [m]	0.834	0.823	0.811	0.764
	K_{eq} [N/m]	117.630	119.167	120.904	128.403
20x20	D_{mean} [m]	0.603	0.605	0.595	0.552
	K_{eq} [N/m]	162.557	162.143	164.897	177.800
40x40	D_{mean} [m]	0.553	0.551	0.543	0.506
	K_{eq} [N/m]	177.439	177.879	150.582	193.769
50x50	D_{mean} [m]	0.563	0.560	0.554	0.516
	K_{eq} [N/m]	174.290	175.118	177.199	190.004

Table 3 is created by further simplifying all data from the test by calculating the mean of all K_{eq} from every load mass excluding (5×5) mesh resolution for further analysis. It reveals an interesting fact: the K_{eq} of the global system increases drastically with the applied load. When increasing the mass from 1 kg to 10 kg, an increase in K_{eq} of approximately 203% has been measured. This suggests that the simplified model might not work in this case because K_{eq} does not remain constant.

Table 3: Analysis of equivalent stiffness dependence on load

Load (Kg)	Fc (N)	D_{mean} (m)	K_{mean} (N/m)	Δ (%)
1	9.81	0.199	53.57	Base (0%)
5	49.05	0.395	125.11	~ 133%
10	98.10	0.621	162.49	~ 203%

4.3. Results using the CJ-based model

By repeating the loading protocol using the CJ component, Table 4 presents a sample of the raw data for the D_{mean} and K_{eq}. In this case, as the mesh resolution increases, the D_{mean} also increases, which translates into a decrease in the K_{eq}. However these values also tend to stabilize. Similar to the previous section, the values obtained from the lowest resolution mesh (5×5) are notably inconsistent.

Table 5 is also created by simplifying data from the tests of the mesh with CJ, and reveals similar results as those for SJ: the K_{eq} of the global system increases with the applied load. However, the increments are not as drastic as in the one obtained for SJ. This indicates, as in the previous subsection, that the simplified model is not accurate for this approach, although the variation of K_{eq} is lower due to the higher accuracy of CJ in physical fidelity simulations as CJs have more degrees of freedom.

Table 4: Elastic response for Configurable Joint and 10 kg load

Grid	Param.	Cube-1	Cube-2	Cube-4	Cube-8
5x5	D_{mean} [m]	1.092	1.087	1.087	0.993
	K_{eq} [N/m]	89.834	90.218	90.276	98.758
10x10	D_{mean} [m]	1.079	1.042	1.006	0.816
	K_{eq} [N/m]	90.935	94.175	97.521	120.201
20x20	D_{mean} [m]	1.442	1.458	1.422	1.200
	K_{eq} [N/m]	68.041	67.303	69.001	81.726
40x40	D_{mean} [m]	1.667	1.692	1.708	1.664
	K_{eq} [N/m]	58.860	57.963	57.421	58.960
50x50	D_{mean} [m]	1.719	1.736	1.744	1.693
	K_{eq} [N/m]	57.055	56.508	56.238	57.953

Table 5: Analysis of equivalent stiffness dependence on load

Load (Kg)	Fc (N)	D_{mean} (m)	K_{mean} (N/m)	Δ (%)
1	9.81	0.324	47.62	Base (0%)
5	49.05	0.94	60.18	~ 26%
10	98.10	1.44	71.87	~ 51%

5. Discussion

This article analyzes the feasibility of Unity as a simulation environment for elastic materials using the elemental MSD model and native Unity primitives. Some preliminary experiments were conducted to compare the performance in mesh deformation during interaction with a static load.

The results presented in this study suggest that the Unity engine's capabilities are adequate for simulating deformable objects in real-time, similar to another internal package such as Obi or related work (Mohan et al., 2024). This approach may improve the simulation performance compared with using a plugin with an external tool such as SOFA to manage the material deformation.

Given the K_{eq}, it can be concluded that increasing the mesh resolution leads to a stabilization of the results, as observed in Tables 2 and 4. However, the results of the two tests exhibit a marked difference: the mesh utilizing CJ tends to undergo significantly greater deformation than the SJ mesh. This can be attributed to the additional degrees of freedom (DoF) provided by the CJ. Consequently, the applied force is distributed across the mesh more effectively, resulting in higher overall deformation. This leads to K_{eq} for the CJ that are considerably lower than those of its counterpart, which must be interpreted as a structural characteristic rather than an error.

Another conclusion is that it is not accurate to approximate the mesh with a simplified vertical MSD system. A more detailed analysis of the results reveals an increase in the K_{eq} across different resolutions and volumes as a function of the object's mass; a higher mass results in a higher K_{eq} as demonstrated in Table 3 and Table 5. This confirms the existence of geometric nonlinearity within the model, a desirable feature in such simulations (though not an intended outcome in this paper) due to its closer approximation of real world physics, yet a significant challenge for the analytical formalization of the model due to the difficulty of extrapolating constituent properties. However, this challenge is a way to scale this work to a higher step. In this aspect, the choice between an SJ or CJ model based on test results depends on the application. While SJ is more suitable for general-purpose simulations focusing on simplicity, CJ is preferable for scenarios requiring higher structural rigor.

It is worth noting that existing models within the Unity platform for practical applications are diverse and functional. For instance, the Va et al. (2021) study demonstrates the efficacy of MSD systems applied to cloth, achieving optimizations via GPU computation. Other approaches, such as the Va et al. (2023b) study, employ tetrahedral methods for soft body generation, presenting a structural alternative to the procedural triangular/hexagonal mesh generation used in this study. Both these works and prior literature confirm that implementing MSD models in Unity allows for the development of fully functional elastic materials. This article aligns with this research line by providing a connectivity architecture (quadrangular and hexagonal) that balances physical stability with simplicity of implementation, facilitating integration into general-purpose systems without relying on complex external libraries.

Although the result of this work is a fully functional mesh with broad prospects for potential applications, the system is not without limitations such as: the difficulty of mathematical formalization, computational cost, and micro-scale applicability. The first aspect arises from the experimental analysis, which concludes that the mesh design cannot be reduced to a vertical MSD system. The critical point is the geometric nonlinearity under different loads; while desirable for realism, this characteristic complicates the translation of real material properties into a mathematical model (Nedel and Thalmann, 1998), adding structural complexity. This aspect will be included in future work towards simulating physically-based materials.

The second limitation lies within Unity's internal processing. Although the MSD model is more comprehensible than other theoretical approaches, increasing the resolution significantly increases the number of elements the engine must manage. Since Unity defaults to CPU-based physics processing, the system reaches a saturation point that slows down the simulation once certain node thresholds are exceeded. This is exacerbated by the fact that, to ensure numerical stability, this study uses a reduced fixed timestep of 0.01s, effectively doubling the physics calculations per second compared to the Unity default. A potential solution involves off-loading the computation of the *Game Objects* from the CPU to the GPU. This aspect could be relevant in the development of VR-based telepresence applications involving force-control (Lukawski et al., 2025).

Finally, the precision analysis of the SJ component at microscales (0.1 and 0.01) showed unacceptable relative errors. This limitation is numerical in nature, stemming from PhysX's management of minuscule displacements and high parameter contrasts. For high-fidelity applications, the integration of dynamics solvers decoupled from Unity, such as PBD, is suggested. PBD offers unconditional stability and robust time-step management, overcoming the numerical instabilities inherent in native joints in boundary scenarios (Obi, 2019; Va et al., 2023a; Zhang et al., 2022) and is a different approach to model improvement.

6. Conclusions

This paper has presented an architecture for the simulation of deformable elastic materials using the Mass-Spring-Damper (MSD) system within the Unity engine. The proposed model can enable simulations of soft materials in robotic applications; for example, deformable object manipulation by robotic arms, as well as cloth simulations. However, further testing is required in order to model the inherent material properties for more fidelity force-control simulations. The model proposed in this work has the potential to scale into a more comprehensive framework while maintaining the objectives of open accessibility and user simplicity.

Acknowledgments

This research has been partially financed by "iREHAB: AI-powered Robotic Personalized Rehabilitation" project DTS22/00105, financed by Instituto de Salud Carlos III (ISCIII) and EU; and partially financed by FotoArt5.0-CM project (TEC-2024/TEC-308) via the program of I+D activities "Tecnologías 2024" from the Comunidad de Madrid.

References

Defkit, 2016. Defkit – Deformable (Soft) Bodies Toolkit. Unity Comunity, online; Accessed: Apr. 26, 2025.
URL: https://discussions.unity.com/t/defkit-deformable-soft-bodies-toolkit-released/630100

Lukawski, B., Montesino, I., Oña, E. D., Victores, J. G., Balaguer, C., Jardón, A., 2025. Towards the development of telepresence applications with TIAGo and TIAGo++ using a virtual reality headset. In: 2025 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). IEEE, pp. 192–197.
DOI: 10.1109/ICARSC65809.2025.10970173

Mohan, D. M., Zhong, Y., Smith, J., Ehrampoosh, A., Shirinzadeh, B., 2024. Soft-tissue deformation model for virtual reality-based surgery training using Unity3D. In: 2024 IEEE 18th International Conference on Advanced Motion Control (AMC). IEEE, pp. 1–6.
DOI: 10.1109/AMC58169.2024.10505703

Nealen, A., Müller, M., Keiser, R., Boxerman, E., Carlson, M., 2006. Physically based deformable models in computer graphics. In: Computer graphics forum. Vol. 25. Wiley Online Library, pp. 809–836.
DOI: 10.1111/j.1467-8659.2006.01000.x

Nedel, L. P., Thalmann, D., 1998. Real time muscle deformations using mass-spring systems. In: Proceedings. computer graphics international (cat. no. 98ex149). IEEE, pp. 156–165.
DOI: 10.1109/CGI.1998.694263

Obi, 2019. Obi – Unified Particle Physics for Unity. Virtual Method Studio, online; Accessed: Apr. 26, 2025.
URL: https://obi.virtualmethodstudio.com/

Qin, L., Peng, H., Huang, X., Liu, M., Huang, W., 2024. Modeling and simulation of dynamics in soft robotics: A review of numerical approaches. Current Robotics Reports 5 (1), 1–13.
DOI: 10.1007/s43154-023-00105-z

SOFA, 2006. SOFA framework, simulation open-framework architecture. GitHub, online; Accessed: Apr. 26, 2025.
URL: https://github.com/sofa-framework/sofa

Va, H., Choi, M.-H., Hong, M., 2021. Real-time cloth simulation using compute shader in Unity3D for AR/VR contents. Applied Sciences 11 (17), 8255.
DOI: 10.3390/app11178255

Va, H., Choi, M.-H., Hong, M., 2023a. Efficient simulation of volumetric deformable objects in Unity3D: Gpu-accelerated position-based dynamics. Electronics 12 (10), 2229.
DOI: 10.3390/electronics12102229

Va, H., Choi, M.-H., Hong, M., 2023b. Real-time surface-based volume constraints on mass-spring model in Unity3D. IEEE Access 11, 17857–17869.
DOI: 10.1109/ACCESS.2023.3245130

Zhang, F., Sun, Z., Wang, T., 2022. Brain modeling for surgical training on the basis of Unity 3D. In: International Symposium on Artificial Intelligence and Robotics. Springer, pp. 1–8.
DOI: 10.1007/978-981-19-7943-9_1