

FreeForm: Reduced-Order Deformable Simulation from Particle-Based Skinning Eigenmodes

Supplementary Material

6. Further Discussion on MPM and SPH

As mentioned in Sec. 2, many particle-based physics simulation methods have been proposed. MPM and SPH are particularly attractive, as they can handle a wide variety of material models, including plasticity effects, topology and phase changes, and, as is our focus here, elastodynamics.

However, this versatility regarding topology changes is also what makes MPM and SPH sensitive to spatial discretization, as the interaction stencils change over time. For MPM, two particles will interact if and only if they share at least one common grid node; for SPH, only if the support of their kernels overlap. Inevitably, as the material is increasingly stretched, particles will eventually get further apart than this critical distance, and numerical fracture will happen, as illustrated in Fig. 6. For SPH, particles becoming locally co-dimensional can also lead to numerical conditioning issues, with the velocity gradient becoming singular and requiring special care [44]. For MPM elastic bodies, deformation gradient estimation can be made more robust by leveraging a rest pose mesh [21], or even by rasterizing forces from a Lagrangian model [14], when such a representation is available. However, integration accuracy will still suffer when the number of particles per cell is not sufficient, while a too coarse grid will exhibit locking; this makes picking the grid resolution difficult for uneven particle distributions.

The same sampling criteria apply to our RKPM kernel centers; however, our method only needs to worry about the rest pose, for which it is easier to control the sample distribution and kernel width, while MPM and SPH need to have the particles remain well distributed at each timestep. Resampling particles over time can avoid those issues [49]; however, this is not really practical when simulating a pre-defined number of Gaussian splats, for instance.

Moreover, while so-called implicit variants of MPM and SPH have been proposed, most still treat advection as an explicit step and are therefore subject to the Courant–Friedrichs–Lewy (CFL) condition. In contrast, our total-Lagrangian approach, with shape functions remaining fixed over time and implicit time stepping, does not have a constraint on the size of time step.

7. Implementation Details

7.1. Our Method

RKPM construction. Given an input object, our method first constructs a set of RKPM kernels around the object

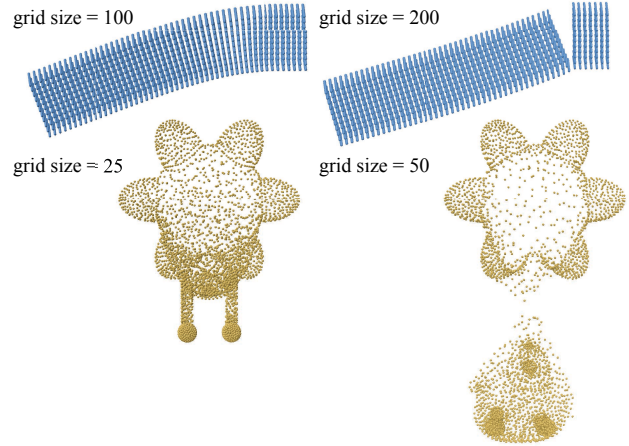


Figure 6. The Material Point Method (MPM) is versatile, but presents challenges for deformable body simulation due to the occurrence of unintended numerical fracture depending on the grid resolution (shown in top-left corner).

shape. We start by dense sampling integration points around the object. Unless otherwise stated, we sample points on a uniform grid inside the object bounding box, and then reject points outside the object for shapes with well-defined an inside/outside test functions, e.g., a watertight mesh. For shapes where the inside/outside test cannot be easily applied, e.g. 3DGS, we directly use the given points as integration points (after downsampling if necessary to avoid out-of-memory error). With the integration points determined, we then apply Farthest Point Sampling (FPS) to select around 1k points as RKPM kernel centers to ensure that kernels are equidistantly distributed. We set each Gaussian kernel radius r to be the minimal distance to reach two other centers, so that the space around the kernels is well-covered by RKPM.

Eigenanalysis. After RKPM kernels are constructed, we assemble the Hessian matrix \mathbf{H}_w according to Prop. 1 and \mathcal{M} according to Eq. (14). We perform generalized eigen-decomposition using `torch.lobpcg` on CUDA in double precision. We take the eigenvectors associated with m smallest eigenvalues as the nodal values for our skinning weight estimation. Notice that the solutions of the eigen-decomposition always includes a constant mode associated with zero eigenvalue. Simplicitis does not find this constant mode in its optimization process, and thus manually append

a constant mode that corresponds to a global affine transformation. To ensure a fair comparison with Simplicits, when reporting the number m of skinning weights, we do not include the constant mode for both methods throughout the paper.

Simulation. Once the skinning weights are determined, we run simulation of deformable objects in the same approach as Simplicits [33]. Our implementation is built on top of the open-source Kaolin library based on the Warp language and PyTorch, and we further boost the runtime performance with more efficient kernel launching via CUDA graph captured in Warp and PyTorch while maintaining the same simulation results. In the paper, we test both our method and Simplicits with the same enhanced implementation for fairness. We use Newton’s method with line search based on Wolfe conditions to solve implicit time-stepping in Eq. (2), allowing up to 20 updates per time step with a convergence tolerance of 10^{-8} . To solve the linear system in Newton’s method, we use the direct solver `torch.linalg.solve` in PyTorch.

7.2. Baseline Methods

Simplicits. We use the recommended implementation in Kaolin. The neural field for skinning weights is a 6-layer MLP (excluding the input and output layers) of layer width 64, trained for 10k iterations using the Adam optimizer, with a learning rate of 10^{-3} . The elasticity and orthogonality loss weights are set to 0.1 and 10^6 , respectively. At run time, we adopt the same simulation implementation as our method for Simplicits.

MPM and SPH. For MPM, we use the GPU-based `warpm`¹ implementation. For the standard beam test, we use $dt = 10^{-4}$ s, whereas $dt = 10^{-3}$ s leads to numerical explosion. For SPH, we use the elasticity model in Kugelstadt et al. [24] implemented in `SPlisHSPlasH`², and $dt = 0.01$ s for the beam test.

Finite Element Methods. Finite Element Methods are widely adopted and regarded as the standard approach for simulating elasticity. In this work, we use converged FEM simulation results as the gold standard reference for evaluating various methods. Our full-DoF FEM simulation is implemented based on `warp.fem`³. The simulation uses the same Neo-Hookean elasticity model in Eq. (16) on tetrahedral meshes. The solver adopts the backward Euler method for implicit time stepping, solved by Newton’s method.

¹<https://github.com/zeshunzong/warp-mpm/>

²<https://github.com/InteractiveComputerGraphics/SPlisHSPlasH>

³<https://nvidia.github.io/warp/modules/fem.html>

m	Simplicits	Ours	FEM	MPM	SPH
6	5.01	3.01			
9	3.95	3.71			
16	5.09	5.42	427.2	23.1	37.8
32	10.0	10.7			

Table 5. Comparison of runtime in milliseconds (ms) for a simulation step of $dt = 0.01$ s on average. The timing results are reported for the beam-bending experiment.

7.3. Runtime Comparison

We report the time used to simulate a period of 0.01s in the beam-bending experiment by various methods in Table 5. Our method and Simplicits adopt the same solver and therefore reach similar runtime performance. FEM is a full-DoF simulation that yields accurate results but takes orders of magnitude longer to run. MPM and SPH also achieve competitive runtime performance, but are still slower than our reduced-order formulation.

8. Proof of Proposition 1

In this section, we provide a proof of Proposition 1. Consider the deformation map $\Phi(\mathbf{X}, \mathbf{d}) = \mathbf{u}(\mathbf{X}, \mathbf{d}) + \mathbf{X}$, where the displacement $\mathbf{u}(\mathbf{X}, \mathbf{d})$ is parameterized by RKPM in Eq. (6) and the DoFs are the nodal displacements $\mathbf{d} = \{\mathbf{d}_k \in \mathbb{R}^3\}_{k=1}^K = \{(\mathbf{d}_k^x, \mathbf{d}_k^y, \mathbf{d}_k^z)^T\}_{k=1}^K$.

$$\begin{aligned} \Phi(\mathbf{X}, \mathbf{d}) &= \begin{bmatrix} \Phi^x(\mathbf{X}, \mathbf{d}) \\ \Phi^y(\mathbf{X}, \mathbf{d}) \\ \Phi^z(\mathbf{X}, \mathbf{d}) \end{bmatrix} \\ &= \sum_{k=1}^K \phi_k(\mathbf{X}) \mathbf{d}_k + \mathbf{X}, \end{aligned} \quad (18)$$

The deformation gradient $\mathbf{F}(\mathbf{X}, \mathbf{d}) \in \mathbb{R}^{3 \times 3}$ is the spatial derivative of Φ at each point $\mathbf{X} \in \Omega$, given by

$$\begin{aligned} \mathbf{F}(\mathbf{X}, \mathbf{d}) &= \nabla \Phi(\mathbf{X}, \mathbf{d}) = \begin{bmatrix} \nabla \Phi^x(\mathbf{X}, \mathbf{d})^T \\ \nabla \Phi^y(\mathbf{X}, \mathbf{d})^T \\ \nabla \Phi^z(\mathbf{X}, \mathbf{d})^T \end{bmatrix} \\ &= \sum_{k=1}^K \mathbf{d}_k \nabla \phi_k(\mathbf{X})^T + \mathbf{I} \\ &= \underbrace{\begin{bmatrix} \mathbf{d}_1^T & \dots & \mathbf{d}_K^T \end{bmatrix}}_{\mathbf{d}^T \in \mathbb{R}^{3 \times K}} \overbrace{\begin{bmatrix} \nabla \phi_1(\mathbf{X})^T \\ \vdots \\ \nabla \phi_K(\mathbf{X})^T \end{bmatrix}}^{\nabla \phi(\mathbf{X}) \in \mathbb{R}^{K \times 3}} + \mathbf{I}. \end{aligned} \quad (19)$$

where ∇ denotes the derivative with respect to the point \mathbf{X} . With a slight abuse of notation, we write

$$\begin{aligned}\phi(\mathbf{X}) &= \begin{bmatrix} \phi_1(\mathbf{X}) \\ \dots \\ \phi_K(\mathbf{X}) \end{bmatrix} \in \mathbb{R}^K, \\ \mathbf{d} &= \begin{bmatrix} \mathbf{d}_1^T \\ \vdots \\ \mathbf{d}_K^T \end{bmatrix} = [\mathbf{d}^x, \mathbf{d}^y, \mathbf{d}^z] \in \mathbb{R}^{K \times 3},\end{aligned}\quad (20)$$

where $\mathbf{d}^x, \mathbf{d}^y, \mathbf{d}^z \in \mathbb{R}^K$ are the nodal displacements in the x, y, z directions, respectively. Then we have

$$\begin{aligned}\mathbf{F}(\mathbf{X}, \mathbf{d}) &= \begin{bmatrix} \mathbf{F}_{11} & \mathbf{F}_{12} & \mathbf{F}_{13} \\ \mathbf{F}_{21} & \mathbf{F}_{22} & \mathbf{F}_{23} \\ \mathbf{F}_{31} & \mathbf{F}_{32} & \mathbf{F}_{33} \end{bmatrix} \\ &= \begin{bmatrix} (\mathbf{d}^x)^T \\ (\mathbf{d}^y)^T \\ (\mathbf{d}^z)^T \end{bmatrix} [\partial_x \phi(\mathbf{X}) \quad \partial_y \phi(\mathbf{X}) \quad \partial_z \phi(\mathbf{X})] + \mathbf{I} \\ &= \mathbf{d}^T \nabla \phi(\mathbf{X}) + \mathbf{I},\end{aligned}\quad (21)$$

where $\partial_x \phi(\mathbf{X}) = [\partial \phi_1(\mathbf{X})/\partial x \quad \dots \quad \partial \phi_K(\mathbf{X})/\partial x]^T$, and similarly for $\partial_y \phi(\mathbf{X})$ and $\partial_z \phi(\mathbf{X})$.

Applying the strain energy density and integrating over the domain Ω by the Monte Carlo method, we obtain the total elastic potential energy as

$$E_{\text{pot}}(\mathbf{d}) = \int_{\Omega} \Psi(\mathbf{F}(\mathbf{X}, \mathbf{d})) d\mathbf{X} \approx \sum_i v_i \Psi(\mathbf{F}(\mathbf{X}_i, \mathbf{d})), \quad (22)$$

where \mathbf{X}_i is the i -th sample point, and v_i is the weight of the i -th sample point. Then its weight-space Hessian matrix

$$\mathbf{H}_w = \mathbf{H}_{xx} + \mathbf{H}_{yy} + \mathbf{H}_{zz} \quad (23)$$

contains the Hessian of E_{pot} with respect to the nodal displacements $\mathbf{d}^x, \mathbf{d}^y, \mathbf{d}^z$ around the rest position $\mathbf{d} = \mathbf{0}$, respectively. Take \mathbf{H}_{xx} as an example, and denote $\text{Hess}(\cdot, \cdot)$ as the Hessian of the first argument with respect to the second argument.

$$\begin{aligned}\mathbf{H}_{xx} &= \text{Hess}(E_{\text{pot}}, \mathbf{d}^x) = \text{Hess}\left(\sum_i v_i \Psi(\mathbf{F}_i), \mathbf{d}^x\right) \\ &= \sum_i v_i \text{Hess}(\Psi(\mathbf{F}_i), \mathbf{d}^x),\end{aligned}\quad (24)$$

where $\mathbf{F}_i = \mathbf{F}(\mathbf{X}_i, \mathbf{d})$ is a shorthand for the deformation gradient at the i -th sample point. Notice that \mathbf{F} is an affine transformation of \mathbf{d} , so the chain rule for Hessian matrix gives

$$\text{Hess}(\Psi(\mathbf{F}_i), \mathbf{d}^x) = \mathbf{J}_i^T \text{Hess}(\Psi(\mathbf{F}_i), \mathbf{F}_i) \mathbf{J}_i, \quad (25)$$

where \mathbf{J}_i is the Jacobian matrix of $\text{vec}(\mathbf{F}_i)$ with respect to \mathbf{d}^x around $\mathbf{d} = \mathbf{0}$ and $\text{Hess}(\Psi(\mathbf{F}_i), \mathbf{F}_i)$ is the Hessian of strain energy density with respect to the flattened

$\text{vec}(\mathbf{F}_i) \in \mathbb{R}^9$ around $\mathbf{F}_i = \mathbf{I}$. For the Neo-Hookean energy in Eq. (16), we have the gradient

$$\begin{aligned}\text{grad}(\Psi(\mathbf{F}_i), \mathbf{F}_i) &= \mu \mathbf{F}_i + (\lambda + \mu)(J - \gamma) J \mathbf{F}_i^{-T}, \\ J &= \det(\mathbf{F}_i)\end{aligned}\quad (26)$$

and the Hessian

$$\begin{aligned}\text{Hess}(\Psi(\mathbf{F}_i), \mathbf{F}_i) &= \mu \mathbf{I} \\ &+ (\lambda + \mu)(2J - \gamma) \text{vec}(J \mathbf{F}_i^{-T})^T \text{vec}(\mathbf{F}_i^{-T}) \\ &- (\lambda + \mu)(J - \gamma) J (\mathbf{F}_i^{-1} \otimes \mathbf{F}_i^{-T}) \mathbf{K},\end{aligned}\quad (27)$$

$$\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

where \otimes denotes the Kronecker product. When $\mathbf{F}_i = \mathbf{I}$, the Hessian simplifies to

$$\begin{aligned}\text{Hess}(\Psi, \mathbf{F}_i) &= \mu \mathbf{I} + (\lambda + \mu)(2 - \gamma) \mathbf{K}_1 \\ &- (\lambda + \mu)(1 - \gamma) \mathbf{K},\end{aligned}\quad (28)$$

$$\mathbf{K}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Here λ and μ are Lamé coefficients depending on the sample point \mathbf{X}_i , which we omit for simplicity. On the other hand, from Eq. (21) we know that

$$\text{vec}(\mathbf{F}_i) = \begin{bmatrix} \mathbf{F}_{11} \\ \mathbf{F}_{12} \\ \mathbf{F}_{13} \\ \mathbf{F}_{21} \\ \mathbf{F}_{22} \\ \mathbf{F}_{23} \\ \mathbf{F}_{31} \\ \mathbf{F}_{32} \\ \mathbf{F}_{33} \end{bmatrix} = \begin{bmatrix} \partial_x \phi(\mathbf{X}_i)^T \mathbf{d}^x + 1 \\ \partial_y \phi(\mathbf{X}_i)^T \mathbf{d}^x \\ \partial_z \phi(\mathbf{X}_i)^T \mathbf{d}^x \\ \partial_x \phi(\mathbf{X}_i)^T \mathbf{d}^y \\ \partial_y \phi(\mathbf{X}_i)^T \mathbf{d}^y + 1 \\ \partial_z \phi(\mathbf{X}_i)^T \mathbf{d}^y \\ \partial_x \phi(\mathbf{X}_i)^T \mathbf{d}^z \\ \partial_y \phi(\mathbf{X}_i)^T \mathbf{d}^z \\ \partial_z \phi(\mathbf{X}_i)^T \mathbf{d}^z + 1 \end{bmatrix}. \quad (29)$$

Thus the Jacobian matrix \mathbf{J}_i is given by

$$\mathbf{J}_i = \frac{\partial \text{vec}(\mathbf{F}_i)}{\partial \mathbf{d}^x} = \begin{bmatrix} \partial_x \phi(\mathbf{X}_i)^T \\ \partial_y \phi(\mathbf{X}_i)^T \\ \partial_z \phi(\mathbf{X}_i)^T \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^{9 \times K}. \quad (30)$$

Substituting Eq. (28) and Eq. (30) into Eq. (24), we get

$$\begin{aligned} \text{Hess}(\Psi(\mathbf{F}_i), \mathbf{d}^x) &= \mu \sum_{s=x,y,z} \partial_s \phi(\mathbf{X}_i) \partial_s \phi(\mathbf{X}_i)^T \\ &\quad + (\lambda + \mu)(2 - \gamma) \partial_x \phi(\mathbf{X}_i) \partial_x \phi(\mathbf{X}_i)^T \\ &\quad + (\lambda + \mu)(\gamma - 1) \partial_x \phi(\mathbf{X}_i) \partial_x \phi(\mathbf{X}_i)^T \end{aligned} \quad (31)$$

With $\gamma = 1 + \mu/(\lambda + \mu)$, we have

$$\begin{aligned} \text{Hess}(\Psi(\mathbf{F}_i), \mathbf{d}^x) &= (\lambda + 2\mu) \partial_x \phi(\mathbf{X}_i) \partial_x \phi(\mathbf{X}_i)^T \\ &\quad + \mu \partial_y \phi(\mathbf{X}_i) \partial_y \phi(\mathbf{X}_i)^T + \mu \partial_z \phi(\mathbf{X}_i) \partial_z \phi(\mathbf{X}_i)^T. \end{aligned} \quad (32)$$

Similarly,

$$\begin{aligned} \text{Hess}(\Psi(\mathbf{F}_i), \mathbf{d}^y) &= (\lambda + 2\mu) \partial_y \phi(\mathbf{X}_i) \partial_y \phi(\mathbf{X}_i)^T \\ &\quad + \mu \partial_x \phi(\mathbf{X}_i) \partial_x \phi(\mathbf{X}_i)^T + \mu \partial_z \phi(\mathbf{X}_i) \partial_z \phi(\mathbf{X}_i)^T, \\ \text{Hess}(\Psi(\mathbf{F}_i), \mathbf{d}^z) &= (\lambda + 2\mu) \partial_z \phi(\mathbf{X}_i) \partial_z \phi(\mathbf{X}_i)^T \\ &\quad + \mu \partial_x \phi(\mathbf{X}_i) \partial_x \phi(\mathbf{X}_i)^T + \mu \partial_y \phi(\mathbf{X}_i) \partial_y \phi(\mathbf{X}_i)^T. \end{aligned} \quad (33)$$

In conclusion,

$$\begin{aligned} \mathbf{H}_w &= \mathbf{H}_{xx} + \mathbf{H}_{yy} + \mathbf{H}_{zz} \\ &= \sum_i \sum_{s=x,y,z} v_i \text{Hess}(\Psi(\mathbf{F}_i), \mathbf{d}^s) \\ &= \sum_i \sum_{s=x,y,z} v_i (\lambda + 4\mu) \partial_s \phi(\mathbf{X}_i) \partial_s \phi(\mathbf{X}_i)^T \\ &= \sum_i v_i (\lambda + 4\mu) \nabla \phi(\mathbf{X}_i) \nabla \phi(\mathbf{X}_i)^T \\ &\approx \int_{\Omega} (\lambda(\mathbf{X}) + 4\mu(\mathbf{X})) \nabla \phi(\mathbf{X}) \nabla \phi(\mathbf{X})^T d\mathbf{X}, \end{aligned} \quad (34)$$

The (i, j) -th element of \mathbf{H}_w is

$$(\mathbf{H}_w)_{ij} = \int_{\Omega} (\lambda(\mathbf{X}) + 4\mu(\mathbf{X})) \nabla \phi_i(\mathbf{X})^T \nabla \phi_j(\mathbf{X}) d\mathbf{X}. \quad (35)$$

This completes the proof of Proposition 1.

9. Additional Evaluation and Analysis

9.1. Basis Fitting Residual

For reduced-order methods including Simplicits and ours, we also perform a least square fitting of the FEM simulation

Test	m	Simplicits	Ours
Bend	6	4.54e-06	6.67e-07
	9	1.19e-06	4.33e-07
	16	5.34e-07	1.40e-07
	32	1.93e-07	5.60e-08
Twist	6	9.41e-05	2.20e-05
	9	1.52e-05	5.94e-06
	16	5.19e-06	5.83e-07
	32	7.56e-07	3.29e-07

Table 6. Comparison of basis fitting residual for reduced order methods (Simplicits and ours) for the standard beam test.

results using the predicted skinning weights, and report the fitting residual on the standard beam test in Table 6 and on the Thingi10K/Simready datasets in Table 7.

Concretely, given N rest-post vertices of the FEM mesh $\{\mathbf{X}_i\}_{i=1}^N$ and the corresponding deformed positions $\mathbf{x}_{i,t}^{\text{FEM}}$ at time t from the full-order FEM simulation, we fit the reduced-order DoFs \mathbf{z} (the m affine transformations in Eq. (1)) by minimizing the squared displacement error:

$$\mathbf{z}_t^* = \arg \min_{\mathbf{z}} \frac{1}{N} \sum_{i=1}^N \|\Phi(\mathbf{X}_i, \mathbf{z}) - \mathbf{x}_{i,t}^{\text{FEM}}\|^2, \quad (36)$$

where $\Phi(\mathbf{X}, \mathbf{z})$ is the skinning deformation map in Eq. (1). Notice that in Simplicits and our method, the deformation map Φ is linear with respect to \mathbf{z} , so Eq. (36) is a linear least square problem and can be solved analytically. With the optimal fitted deformation \mathbf{z}_t^* , we report the average residual

$$\frac{1}{NT} \sum_{i=1}^N \sum_{t=1}^T \|\Phi(\mathbf{X}_i, \mathbf{z}_t^*) - \mathbf{x}_{i,t}^{\text{FEM}}\|$$

over the N vertices and T time steps, with the same normalization by bounding box size as described in Sec. 4.

Compared with the mean squared error measured on the simulated deformation in Sec. 4, this metric factors out the difference in dynamic behavior accumulated during the simulation, and only evaluates the expressiveness of the predicted skinning weights. The results in Table 6 and 7 demonstrate that our skinning weights, or simulation basis, can better explain the FEM simulation results than the Simplicits baseline.

9.2. Multi-Layer Material Simulation

In Fig. 7, we compare our method with Simplicits on a heterogeneous sphere consisting of four layers of stiff-soft-stiff-soft materials. During simulation, the sphere is dropped to the ground. This case is challenging because it requires the global simulation basis to express the distinct dynamic patterns across different stiffness regions. Our

Method	Fix Side		Pull Farthest		Pull Boundary	
	MSE	Max	MSE	Max	MSE	Max
Simplicits	1.64e-05	3.60e-05	1.15e-04	2.86e-04	4.62e-05	9.81e-05
Ours	4.55e-06	1.00e-05	3.52e-05	8.84e-05	1.30e-05	2.63e-05
Improvement	72.3%	72.1%	69.5%	69.1%	71.9%	73.2%
Simplicits	5.71e-12	6.77e-12	4.74e-05	1.35e-04	7.31e-05	2.25e-04
Ours	1.90e-12	2.51e-12	7.51e-06	2.18e-05	9.98e-06	2.92e-05
Improvement	66.8%	62.9%	84.2%	83.9%	86.3%	87.0%

Table 7. Basis fitting residual error on the Thingi10K and Simready Datasets. We compute the least square fitting of the FEM simulation results using the predicted skinning weights from Simplicits and our method, and report the fitting residual to quantify the capability of those skinning weights to express the full-order FEM deformation. Our results show consistent improvement over the Simplicits baseline.

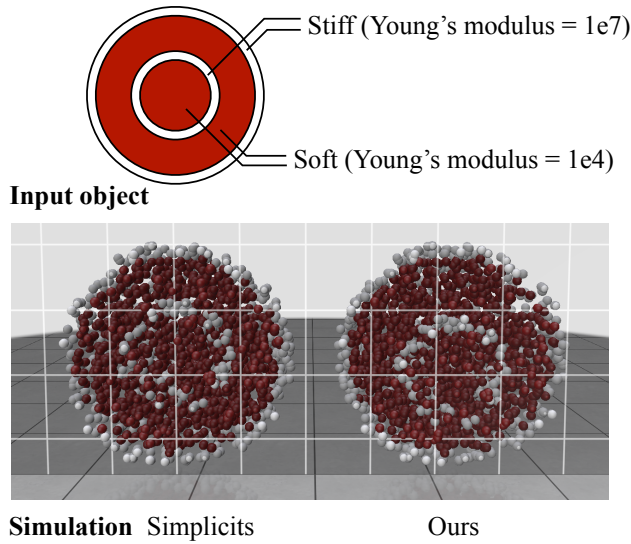


Figure 7. We compare our method with Simplicits on a heterogeneous sphere consisting of four layers of stiff-soft-stiff-soft materials. The hard layers are shown in white and the soft layers in red. We visualize the internal deformation of the spheres using a slice plane that passes through the sphere centers.

method captures realistic and distinct deformation behaviors across the varying layers. In contrast, the Simplicits simulation only produces stiff, global deformation. This result is best viewed in the supplementary video.

9.3. Stress Visualization

In Fig. 8, we visualize the stress distribution on the beam in the twist test. Concretely, we convert the first Piola-Kirchhoff stress tensor $\mathbf{P} = \frac{\partial \Psi}{\partial \mathbf{F}}$ defined in the reference configuration into Cauchy stress tensor in the deformed configuration using the relation

$$\boldsymbol{\sigma} = \frac{1}{\det \mathbf{F}} \mathbf{P} \mathbf{F}^T,$$

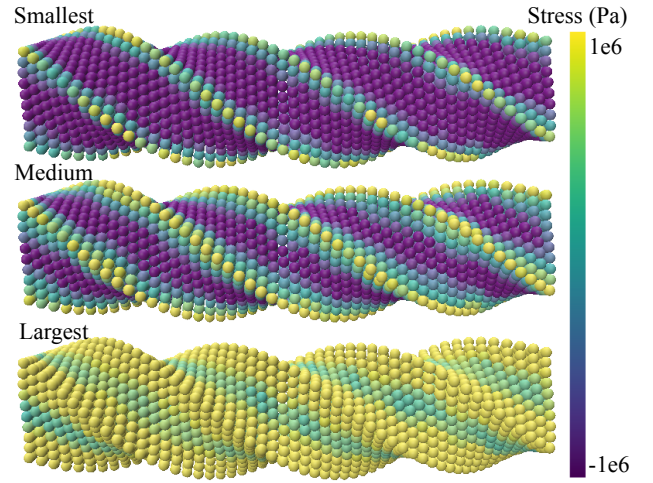


Figure 8. The Cauchy stress distribution on the twisted beam test. We plot the three principal stresses components of the Cauchy stress tensor capped within the range of $[-1 \times 10^6, 1 \times 10^6]$ Pa.

which is a symmetric 3×3 tensor at each point over the deformed beam. We plot the three principal stress fields (the eigenvalues of $\boldsymbol{\sigma}$) from lowest to highest in Fig. 8. This result allows us to visualize the distribution of stress within the beam, and identify the regions of high stress concentration. We hope this will inspire more results on mechanical analysis on top of reduced-order simulation in the future.