Supplemental Material

A. Physics Simulation

In the following, we recapitulate the physical model and provide more details on the derivations and the particular discretization scheme used.

Strong Form Let the reference configuration be given in Ω^r. The displacement at time t is given by $u : \Omega^r \times t \to \mathbb{R}^3$. Then the linear *Green strain tensor* is given by

$$E(u) := \frac{1}{2}\left(\nabla u + (\nabla u)^T\right) \in \mathbb{R}^{3\times 3} \tag{1}$$

and the *Piola-Kirchoff stress tensor*

$$P(u) := 2\mu E(u) + \lambda \operatorname{tr}(E(u))\mathbf{1} \tag{2}$$

with the *Lamé coefficients* μ and λ derived from the Young's modulus k and the Poisson ratio ρ. The dynamic elasticity problem in strong form is then defined as

$$\begin{aligned}
m\ddot{u} - \operatorname{div} P(u) &= f_B \quad \text{in } \Omega^r \times \mathbb{R}_0^+ && \text{(3a)}\\
u &= u_D \quad \text{on } \Gamma_D^r \times \mathbb{R}_0^+ && \text{(3b)}\\
P(u)\cdot \mathbf{n} &= f_S \quad \text{in } \Gamma_N^r \times \mathbb{R}_0^+ && \text{(3c)}\\
u &= u^0 \quad \text{in } \Omega^r \times \{0\} && \text{(3d)}\\
\dot{u} &= \dot{u}^0 \quad \text{in } \Omega^r \times \{0\}\,, && \text{(3e)}
\end{aligned}$$

with the mass m, Dirichlet boundaries Γ_D^r and Neumann boundaries Γ_N^r. In order to arrive at linear system of equations, we follow the classical FEM theory and first transform the strong form into the weak form. Then we plug in the discretization scheme in Supp. A.1 and the matrices emerge.

Weak Form To obtain the weak form, let $V := H^1(\bar{\Omega}^r \to \mathbb{R}^d)$ be the space of test and trial functions. Because Neumann and Dirichlet boundaries are enforced weakly, the space of test and trial functions coincide. Starting from the right hand side of Eq. (3a), the generalized divergence theorem yields:

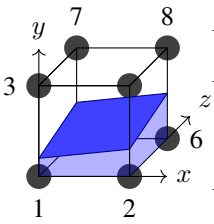
$$\begin{aligned}
&\int_\Omega f_B \cdot v \,\mathrm{dx} = \\
&= \int_\Omega -\operatorname{div} P(u)\cdot v \,\mathrm{dx} + \int_\Omega m\ddot{u}\cdot v \,\mathrm{dx}\\
&= \int_\Omega \sum_{j=1}^{d} \left(\mu\left(\nabla u_j + \frac{\partial}{\partial x_j}u\right) + \lambda\sum_{i=1}^{d}\frac{\partial u_i}{\partial u_j}\mathbf{1}_j\right)\cdot \nabla v_j \,\mathrm{dx}\\
&\quad - \int_{\partial\Omega}\sum_{j=1}^{d}\left(\left(\mu\left(\nabla u_j + \frac{\partial}{\partial x_j}u\right) + \lambda\sum_{i=1}^{d}\frac{\partial u_i}{\partial u_j}\mathbf{1}_j\right)\cdot \mathbf{n}\right)v_j \,\mathrm{ds}\\
&\quad + \int_\Omega m\ddot{u}\cdot v \,\mathrm{dx}.
\end{aligned} \tag{4}$$

A.1. Hexahedral Finite Element Discretization

The weak form is now discretized using hexahedral elements. This splits the domain Ω into the disjoint union of each cell $\Omega = \bigcup_e \Omega^e$, which allows us to decompose the full equation (4) into per-element expressions. For example,

$$\int_\Omega m\ddot{u}\cdot v \,\mathrm{dx} = \sum_e \int_{\Omega^e} m\ddot{u}\cdot v \,\mathrm{dx}. \tag{5}$$

We use standard trilinear shape functions as basis functions for the finite hexahedral elements (see Fig. 1).



$$\begin{aligned}
N_1(\mathbf{x}) &:= \left(1-\frac{x}{h}\right)\left(1-\frac{y}{h}\right)\left(1-\frac{z}{h}\right)\\
N_2(\mathbf{x}) &:= \frac{x}{h}\left(1-\frac{y}{h}\right)\left(1-\frac{z}{h}\right)\\
&\vdots\\
N_8(\mathbf{x}) &:= \frac{x}{h}\frac{y}{h}\frac{z}{h}
\end{aligned}$$

Figure 1: Trilinear hexahedral element and embedded surface

Thus, inside a cell, any function f can be approximated by trilinear interpolation of its values at the eight corners $\mathbf{v}_i$ as

$$f(\mathbf{x}) \approx \sum_{i=1}^{8} f(\mathbf{v}_i) N_i(\mathbf{x}). \tag{6}$$

Partially Filled Cells We further embed the object boundary into the simulation grid, and consider cells that are partly filled with material. For cells of size $[0,1]^3$, and the object given as signed distance function ϕ, we define the part of a cell that is contained in the object as

$$\Omega^e := \{\mathbf{x} \in [0,1]^3 : \phi(\mathbf{x}) \le 0\}. \tag{7}$$

For a trilinear interpolation within cells, integrals of arbitrary functions over cells can be approximated as

$$\int_{\Omega^e} f(x)\,\mathrm{dx} \approx \int_{\Omega^e}\sum_{i=1}^{8} f(\mathbf{v}_i)N_i(x)\,\mathrm{dx} = \sum_{i=1}^{8} f(\mathbf{v}_i)\underbrace{\int_{\Omega^e} N_i(x)\,\mathrm{dx}}_{=:w_v(e,i)} \tag{8}$$

where the integrals of the basis functions w_v are precomputed and stored per cell. As cell vertices do not necessarily lie on the object surface, we use Nitsche's method [1, 4] to incorporate Dirichlet boundaries.

Discrete Equations With the basis functions from Fig. 1 we arrive at the following per-element expressions:

$$\begin{aligned}\int_{\Omega^e} m\ddot{u}\cdot v\,\mathrm{dx} &= (\mathbf{v^e})^T \underbrace{\int_{\Omega^e} \Phi^e(x)^T m \Phi^e(x)\,\mathrm{dx}}_{=:M^e\in\mathbb{R}^{24\times 24}} \ddot{\mathbf{u}}^\mathbf{e}\\ \int_{\Omega^e} \mu\ldots+\lambda\ldots\,\mathrm{dx} &= (\mathbf{v^e})^T \underbrace{\int_{\Omega^e} B^e(x)^T C B^e(x)\,\mathrm{dx}}_{=:K^e\in\mathbb{R}^{24\times 24}} \mathbf{u}^\mathbf{e}\\ \int_{\Gamma_N^e} f_S\cdot v\,\mathrm{ds} &= (\mathbf{v^e})^T \underbrace{\int_{\Gamma_N^e} \Phi^e(x)^T\,\mathrm{dx}\,\mathbf{f_S}}_{=:\mathbf{f^e}\in\mathbb{R}^8}\end{aligned} \tag{9}$$

Here, $\Phi^e(x) \in \mathbb{R}^{3\times 24}$ and $B^e(x) \in \mathbb{R}^{6\times 24}$, respectively, store for each coordinate the values of the basis functions N_i and the derivatives.

$$B^e(x) := \left(\begin{array}{ccc|ccc|c|ccc} \frac{\partial N_1(x)}{\partial x_1} & & & & & & \ldots & \frac{\partial N_8(x)}{\partial x_1} & & \\ & \frac{\partial N_1(x)}{\partial x_2} & & & & & & & \frac{\partial N_8(x)}{\partial x_2} & \\ & & \frac{\partial N_1(x)}{\partial x_3} & & & & & & & \frac{\partial N_8(x)}{\partial x_3} \\ \frac{\partial N_1(x)}{\partial x_2} & \frac{\partial N_1(x)}{\partial x_1} & & & & & & \frac{\partial N_8(x)}{\partial x_2} & \frac{\partial N_8(x)}{\partial x_1} & \\ \frac{\partial N_1(x)}{\partial x_3} & & \frac{\partial N_1(x)}{\partial x_1} & & & & & \frac{\partial N_8(x)}{\partial x_3} & & \frac{\partial N_8(x)}{\partial x_1} \\ & \frac{\partial N_1(x)}{\partial x_3} & \frac{\partial N_1(x)}{\partial x_2} & & & & & & \frac{\partial N_8(x)}{\partial x_3} & \frac{\partial N_8(x)}{\partial x_2} \end{array}\right) \tag{10}$$

$C \in \mathbb{R}^{6\times 6}$ is the regular material matrix for the chosen Lamé coefficients.

After these per-element expressions are combined to large matrices spanning the whole domain, the linear system $M\ddot{u} + Ku = f$ emerges.

Corotation To handle large rotations, we utilize the corotation formulation [7, 3]. First, the rotational part R^e of the deformation of cell e is extracted. Let the average deformation gradient F^e be computed as

$$F^e = \mathbf{1}_3 + \frac{1}{4h}\sum_{i=1}^{8} \mathbf{u}_{s(e,i)} \begin{pmatrix} (-1)^i \\ (-1)^{\lceil i/2\rceil} \\ (-1)1^{\lceil i/4\rceil} \end{pmatrix}^T. \tag{11}$$

The rotational component R^e is then given by the polar decomposition $F^e = R^e S^e$ and can be computed with iterative procedures [3, 6], or a more robust, but also more expensive Analytic Polar Decomposition [5]. Second, given R^e, the per-element term $K^e\mathbf{u}^\mathbf{e}$ is replaced by

$$R^e K^e((R^e)^T(\mathbf{x^e}+\mathbf{u^e}) - \mathbf{x^e}). \tag{12}$$

A.2. Blocked Expressions for Matrix Assembly

One advantage of our chosen discretization is that the per-element stiffness matrix K^e (Eq. (9)) has a 3×3 block structure that is highly amenable to optimized implementations. Each block describes the interactions between the coordinates of the two corresponding cell nodes. Modulo index variations, the computation of these 3x3 blocks is identical for all 64 blocks. Furthermore, the corotational strain formulation and Nitsche Dirichlet boundaries can be incorporated into the block-wise decomposition in a straight forward way.

The regular block structure facilitates storing the stiffness matrix K in a blocked compressed-sparse-row (CSR) format. During assembly, work groups can process a single cell in parallel. Reductions within the cell, as needed, e.g., for the computation of corotations and gradients, can be performed efficiently with warp-reductions. By including analytic simplifications of the basis function evaluations (and their derivatives), we arrive at a highly GPU-friendly algorithm that yields good performance.

The evaluation of the per-element stiffness matrix K^e is performed blockwise. Each of the 8×8 blocks $K^e_{i,j} \in \mathbb{R}^{3\times 3}$ are computed in parallel. Let $\frac{\partial N_i(v_c)}{\partial \mathbf{x}_i}$ be the derivatives of the basis functions at the eight cell corner. This $8\times 8\times 3$ table only contains the entries $-\frac{1}{h}, 0$ or $\frac{1}{h}$. It is stored in constant memory on the GPU and hence allows fast access via the cache. The per-block expression for the stiffness matrix K^e, see Eq. (9) for the definition, is given by:

$$K_{i,j}(x) = \begin{bmatrix} (2\mu+\lambda)\frac{\partial N_i(x)}{\partial x_1}\frac{\partial N_j(x)}{\partial x_1} + \mu\left(\frac{\partial N_i(x)}{\partial x_2}\frac{\partial N_j(x)}{\partial x_2} + \frac{\partial N_i(x)}{\partial x_3}\frac{\partial N_j(x)}{\partial x_3}\right) & \mu\frac{\partial N_i(x)}{\partial x_1}\frac{\partial N_j(x)}{\partial x_2} + \lambda\frac{\partial N_i(x)}{\partial x_2}\frac{\partial N_j(x)}{\partial x_1} & \mu\frac{\partial N_i(x)}{\partial x_1}\frac{\partial N_j(x)}{\partial x_3} + \lambda\frac{\partial N_i(x)}{\partial x_3}\frac{\partial N_j(x)}{\partial x_1} \\ \mu\frac{\partial N_i(x)}{\partial x_2}\frac{\partial N_j(x)}{\partial x_1} + \lambda\frac{\partial N_i(x)}{\partial x_1}\frac{\partial N_j(x)}{\partial x_2} & (2\mu+\lambda)\frac{\partial N_i(x)}{\partial x_2}\frac{\partial N_j(x)}{\partial x_2} + \mu\left(\frac{\partial N_i(x)}{\partial x_1}\frac{\partial N_j(x)}{\partial x_1} + \frac{\partial N_i(x)}{\partial x_3}\frac{\partial N_j(x)}{\partial x_3}\right) & \mu\frac{\partial N_i(x)}{\partial x_2}\frac{\partial N_j(x)}{\partial x_3} + \lambda\frac{\partial N_i(x)}{\partial x_3}\frac{\partial N_j(x)}{\partial x_2} \\ \mu\frac{\partial N_i(x)}{\partial x_3}\frac{\partial N_j(x)}{\partial x_1} + \lambda\frac{\partial N_i(x)}{\partial x_1}\frac{\partial N_j(x)}{\partial x_3} & \mu\frac{\partial N_i(x)}{\partial x_3}\frac{\partial N_j(x)}{\partial x_2} + \lambda\frac{\partial N_i(x)}{\partial x_2}\frac{\partial N_j(x)}{\partial x_3} & (2\mu+\lambda)\frac{\partial N_i(x)}{\partial x_3}\frac{\partial N_j(x)}{\partial x_3} + \mu\left(\frac{\partial N_i(x)}{\partial x_1}\frac{\partial N_j(x)}{\partial x_1} + \frac{\partial N_i(x)}{\partial x_2}\frac{\partial N_j(x)}{\partial x_2}\right) \end{bmatrix}. \tag{13}$$

A.3. Nitsche Dirichlet Boundaries

To incorporate Dirichlet boundaries with Nitsche's method, the weak form from Eq. 9 is extended using productive zeros $u - u_0$:

$$-\int_{\Gamma_N^r} P(u)\cdot\mathbf{n}\cdot v\,\mathrm{ds} - \int_{\Gamma_N^r} P(v)\cdot\mathbf{n}\cdot(u-u_0)\,\mathrm{ds} - \eta\int_{\Gamma_N^r}(u-u_0)\cdot v\,\mathrm{ds}. \tag{14}$$

The first term makes the resulting linear system symmetric. The second term enforces the Dirichlet boundaries. The third term acts as a regularizer and the parameter η has to be chosen as $\eta \geq ch^{-1}$ with h being the grid size and c a sufficient large constant. In our experiments, we chose 10^8 for stable results.

These boundary conditions can also be formulated in an efficient, blocked matrix form. First, the definition of $P(u)$

(Eq. (2)) is used, leading to the following terms that are added to the weak form (Eq. (4)):

$$\begin{aligned}
&\underbrace{-\int_{\Gamma_D^e} \sum_{j=1}^{3} \left(\mu \left(\nabla u_j \cdot n + \frac{\partial u}{\partial x_j} \cdot n \right) + \lambda n_j \sum_{i=1}^{3} \frac{\partial u_i}{\partial x_i} \right) v_j \, \mathrm{ds}}_{\text{(I)}} \\
&\underbrace{-\int_{\Gamma_D^e} \sum_{j=1}^{3} \left(\mu \left(\nabla v_j \cdot n + \frac{\partial v}{\partial x_j} \cdot n \right) + \lambda n_j \sum_{i=1}^{3} \frac{\partial v_i}{\partial x_i} \right) u_j \, \mathrm{ds}}_{\text{(I')}} \\
&\underbrace{-\eta \int_{\Gamma_D^e} u \cdot v \, \mathrm{ds}}_{\text{(II)}}
\end{aligned} \tag{15}$$

Expressing u and v using the basis functions and their derivatives, combined in Φ^e and B^e, we obtain

$$K^e \mathrel{-}= \int_{\Gamma_D^e} \Phi^e(x)^T D^e B^e(x) \, \mathrm{ds} \tag{16}$$

for (15.I) with D^e given by

$$D^e := \begin{pmatrix} 2\mu n_1 + \lambda n_1 & \lambda n_1 & \lambda n_1 \\ \lambda n_2 & 2\mu n_2 + \lambda n_2 & \lambda n_2 \\ \lambda n_3 & \lambda n_3 & 2\mu n_3 + \lambda n_3 \end{pmatrix}. \tag{17}$$

Here, the equation for (15.I') is the above equation, just transposed. This integral is evaluated by computing the sum of the values at the eight corners weighted by $w_b(e, c)$. The value of Φ_i evaluated at vertex c has the special property that $\Phi_i = \mathbf{1}_3 \mathbb{1}_{i=c}$. This allows us to derive the following simplification and solution of Eq. (16), combining both (15.I) and (15.I'):

$$\begin{aligned}
\mathrm{KD}_{j,c} := &\begin{bmatrix} B_1(\lambda n_1 + 2\mu n_1) + B_2 \mu n_2 + B_3 \mu n_3 \\ B_2 \mu n_1 + B_1 \lambda n_2 \\ B_3 \mu n_1 + B_1 \lambda n_3 \end{bmatrix}, \\
&\begin{bmatrix} B_2 \lambda n_1 + B_1 \mu n_2 \\ B_1 \mu n_1 + B_2(\lambda n_2 + 2\mu n_2) + B_3 \mu n_3 \\ B_3 \mu n_2 + B_1 \lambda n_3 \end{bmatrix}, \\
&\begin{bmatrix} B_3 \lambda n_1 + B_1 \mu n_3 \\ B_3 \lambda n_2 + B_2 \mu n_3 \\ B_1 \mu n_1 + B_2 \mu n_2 + B_3(\lambda n_3 + 2\mu n_3) \end{bmatrix} \in \mathbb{R}^{3 \times 3}
\end{aligned} \tag{18}$$

$$\text{with } B_i := \frac{\partial N_j(v_c)}{\partial x_i}$$

$$K_{i,j} \mathrel{-}= w_b(e, i) \mathrm{KD}_{j,i} + w_b(e, j) \mathrm{KD}_{i,j}^T. \tag{19}$$

Part (15.II) even simplifies to the following expression:

$$K_{i,j} \mathrel{-}= 1_{i=j} \eta w_b(e, i) \mathbf{1}_3. \tag{20}$$

A.4. Time integration

Once the stiffness matrix K, the mass matrix M and the force vector f are assembled from the per-element matrices, we introduce Raleigh damping via a matrix $D = \alpha_1 M + \alpha_2 K$, where α_1 specifies the mass damping and α_2 the stiffness damping. The resulting linear system solve the resulting linear system

$$M\ddot{u} + D\dot{u} + Ku = f \tag{21}$$

is then solved per timestep t with a Newmark scheme that takes the following form [2]:

$$\begin{aligned}
&\left(\frac{1}{\theta \Delta t} M + D + \theta \Delta t K \right) u^{(t)} \\
&= \left(\frac{1}{\theta \Delta t} M + D + (1 - \theta) \Delta t K \right) u^{(t-1)} + \frac{1}{\theta} M \dot{u}^{(t-1)} + \Delta t f
\end{aligned} \tag{22}$$

with $\frac{1}{2} \leq \theta < 1$ and

$$\dot{u}^{(t)} = \frac{1}{\theta \Delta t} (u^{(t)} - u^{(t-1)}) - \frac{1 - \theta}{\theta} \dot{u}^{(t-1)}. \tag{23}$$

If f is time-dependent, we perform a time-splitting of the forces in Eq. (24) to improve the numerical stability of the collisions. This splitting scheme is given by

$$f = \theta f^{(t)} + (1 - \theta) f^{(t-1)}. \tag{24}$$

The hyper-parameter θ in the Newmark time integration scheme was set to 0.6 in all of our experiments. As confirmed by our tests, values of θ between 0.5 and 0.75 do not result in noticeable differences. Only for large timesteps and low Rayleigh damping, values of $\theta = 0.99$ and above can introduce undesirable damping. A visual comparison of different values of θ can be found in the supplemental video.

B. Newton Solver for Tri-linear Interpolation

In Sect. 4, a Newton solver for finding the interpolation weights given the interpolated values is used. Its details are described below:

Recall that the standard tri-linear interpolation is given as

$$\begin{aligned}
f(\alpha, \beta, \gamma) &= (1 - \alpha)(1 - \beta)(1 - \gamma)\mathbf{x}_1 + \alpha(1 - \beta)(1 - \gamma)\mathbf{x}_2 \\
&\quad + \cdots + \alpha\beta\gamma\mathbf{x}_8 \\
&= \mathbf{z}_1 + \alpha\mathbf{z}_2 + \beta\mathbf{z}_3 + \gamma\mathbf{z}_4 + \alpha\beta\mathbf{z}_5 + \alpha\gamma\mathbf{z}_6 + \beta\gamma\mathbf{z}_7 \\
&\quad + \alpha\beta\gamma\mathbf{z}_8.
\end{aligned} \tag{25}$$

Then, a Newton iteration is computed as

$$\alpha\beta\gamma^{t+1} = \alpha\beta\gamma - J^{-1} f(\alpha\beta\gamma^t), \tag{26}$$

with

$$J = \begin{pmatrix} | & & \\ \mathbf{z}_2 + \beta\mathbf{z}_5 + \gamma\mathbf{z}_6 + \beta\gamma\mathbf{z}_8 & \cdots \\ | & & \end{pmatrix}. \tag{27}$$

being the Jacobian.

C. Adjoint Code Examples

In this section we present the adjoint code of selected operations. The derivatives for every operation are omitted for space constraints. They can be found in the source code[2].

C.1. Adjoint Code of the SSC cost function

To compute gradients for the SSC cost function within the adjoint framework, we need to compute the adjoint of the inverse of the cell-wise tri-linear interpolation. Therefore, we define the tri-linear interpolant

$$\mathbf{E}(\mathbf{u} = \{\alpha, \beta, \gamma\}, p = \{\mathbf{z}_1, ..., \mathbf{z}_8\}) := \mathbf{z}_1 + \alpha\mathbf{z}_2 + \beta\mathbf{z}_3 + \gamma\mathbf{z}_4 + \alpha\beta\mathbf{z}_5 + \alpha\gamma\mathbf{z}_6 + \beta\gamma\mathbf{z}_7 + \alpha\beta\gamma\mathbf{z}_8 - \mathbf{x} = \mathbf{0}. \tag{28}$$

and compute its derivative with respect to the control points $p = (\mathbf{z}_1, ..., \mathbf{z}_8)$ as

$$F = \frac{\partial \mathbf{E}}{\partial \mathbf{p}} = -\begin{pmatrix} | & | & | & | & | & | & | & | \\ 1 & \alpha & \beta & \gamma & \alpha\beta & \alpha\gamma & \beta\gamma & \alpha\beta\gamma \\ | & | & | & | & | & | & | & | \end{pmatrix} \in \mathbb{R}^{3\times 8}. \tag{29}$$

With this matrix, the adjoint variables of $\mathbf{z}_1$ to $\mathbf{z}_8$ are computed as $(\hat{\mathbf{z_1}}, ..., \hat{\mathbf{z_8}}) = (\alpha', \beta', \gamma')F$. Finally, the adjoint values $\hat{z_1}, ..., \hat{z_8}$ are added to the adjoint values of the per-vertex displacements $\hat{\mathbf{u}}_{i,j,k}, ..., \hat{\mathbf{u}}_{i+1,j+1,k+1}$, by taking the adjoint of the mapping from $\mathbf{x}_1, ..., \mathbf{x}_8$ to $\mathbf{z}_1, ..., \mathbf{z}_8$ that is given by the tri-linear interpolation.

The gradients, given on the whole computational domain, are transferred back to the active nodes with the adjoint of the Eulerian extensions of the displacements. If the current simulation shows large differences to the observations, some points are typically matched with cells that lie outside the object. The adjoint of the extension step then "pulls back" the gradients from these points onto the object's surface and connects them to the Lagrangian simulation. Next, we discuss how to compute the gradients of the damping parameters.

C.2. Adjoint Code of the damping parameters

The adjoint code of the damping parameters is presented here as an example. For a variable x of the forward step, $\hat{x}$ denotes the adjoint/gradient of that variable. One simulation timestep is split as follows:

1. Computation of stiffness matrix K.
2. Computation of Rayleigh damping matrix $D = \alpha_1 M + \alpha_2 K$ where α_1 and α_2, respectively, are the damping on mass and stiffness.
3. Newmark time integration Eq. (22).

In the adjoint code, these steps are performed in reverse order. One adjoint simulation timestep is split as follows:

1. Adjoint of Newmark time integration $\rightarrow \hat{D}$
2. Adjoint of Rayleigh damping:

$$\hat{M} = \alpha_1 \hat{D}\ ,\ \hat{K} = \alpha_2 \hat{D} \tag{30a}$$

$$\hat{\alpha}_1 = \text{vec}(M) \bullet \hat{D}\ ,\ \hat{\alpha}_2 = \text{vec}(K) \bullet \hat{D}. \tag{30b}$$

3. Adjoint of stiffness matrix using $\hat{K}$ among others.

The adjoint variables $\hat{\alpha}_1$ and $\hat{\alpha}_2$ are summed up for every timestep, giving rise to the final gradients for these parameters.

D. Physical Units

The material estimates are performed in a virtual, unitless coordinate system. To convert the estimated values to physical units, the simulated gravity needs to be scaled to relate the object's size in the virtual space to its size in the real world. Furthermore, the object's mass (the Young's modulus depends on it) and the time step are required. Since the object's mass cannot be recovered from the observations, we weigh the objects beforehand. The time step is given by the camera framerate, i.e. 60 fps.

More specifically, we measure the size of of the object S' in virtual meters m′ via the signed distance function of the input ϕ_0 that defines the object in reference configuration Ω_r. Given the size of the object S in meters, we can compute the first scaling factor $f_{\text{size}} = {}^{S}/_{S'}$ in ${}^{\text{m}}/_{\text{m}'}$. Next, let M in kg be the mass of the real object. The parameter m in the physical model Eq. (3) specifies the mass density. Hence the virtual mass M' in kg' is given by $M' = mV$ where V is the object's volume computed as $V = f_{\text{size}}^3 \int_{\Omega_r} 1\,\text{dx}$. The scaling factor for the mass is then given as $f_{\text{mass}} = {}^{M}/_{M'}$ in ${}^{\text{kg}}/_{\text{kg}'}$. Last, we parameterize the simulation with the real-world time step, hence $f_{\text{time}} = {}^{1\text{s}}/_{1\text{s}'}$. With these scaling factors, we can convert the value of the virtual Young's modulus k (see Sect. 3) into real-world units. The unit of the Young's modulus is Pascal (${}^{\text{kg m}}/_{\text{s}^2}$), hence the real-world value is given as $k \cdot {}^{f_{\text{mass}} f_{\text{size}}}/_{f_{\text{time}}^2}$. Similarly, let g be the reconstructed gravitational acceleration in virtual units. The real-world gravity is then given by $g \cdot {}^{f_{\text{size}}}/_{f_{\text{time}}^2}$.

E. Extended Stability Analysis

In the following we demonstrate the robustness of our solver by comparing reconstructions to synthetic ground truth values.

E.1. Gradient Stability for Varying Number of Timesteps and Noise

We first investigate the robustness of the gradients computed by our differentiable solver for the SSCs when varying the number of steps over time, and when introducing noise. The following tests are performed with the torus data set shown in Fig. 2, and show gradient evaluations when varying the Young's modulus estimate around a ground

[2] https://github.com/shamanDevel/SparseSurfaceConstraints

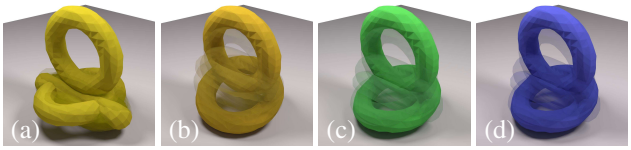


Figure 2: For different noise and camera settings, our optimizer was used to estimate the torus Young's modulus. From left to right, images show simulations using the minimal, maximal, reconstructed and ground truth values. Our reconstruction (c) closely matches the ground truth in (d).

truth value of 5000 along the x-axis. I.e. we expect the cost (shown in blue in the following graphs) to have a minimum at 5000, while the gradient (shown in red in terms of it's magnitude) should be negative to the left of 5000, and positive on the right side.

The insights we gain from the plots in Fig. 3 are twofold: First, the more timesteps are simulated, the more do numerical errors in the adjoint pass accumulate and the noisier the gradients become. Note, however, that the torus a rather difficult test case as it exhibits strong deformations. Despite this effect, the gradients retain the correct sign, i.e., overall direction of the gradient, visible in Fig. 3 from the fraction of the red curves above and below the dashed red line. The negative parts lie on the left side of the ground truth value of 5000, while positive gradients lie on the right side. Second, the simulation becomes more stable with increasing noise magnitude. The increase in noise leads to a smoothing of the point assignments, reducing outliers, and hence smoothing the cost function. Furthermore, note that the absolute value of the cost function is larger for higher noise values than for lower ones. This is because even in the optimal case, the observations can be quite far from the surface due to the noise.

E.2. Finite difference based Gradient Estimation

Next, we compare our proposed gradient estimation to a finite-difference based estimation. In particular, we shed light on the influence of finite-difference based gradient estimation using the hyper-parameter Δx in the gradient approximation $f'(x) \approx \frac{f(x+\Delta x)-f(x)}{\Delta x}$ on reconstruction accuracy.

Fig. 4 plots the cost function and corresponding gradient estimates for the torus test case when varying the Young's modulus. Especially for a small Δx of 5 units, the resulting gradients exhibit strong noise and a large number of sign flips. This behavior is alleviated for $\Delta x = 100$ in the negative part of the gradient, but remains critical on the positive side (above the ground truth value of 5000). In none of the full optimization runs performed in this work (as presented in the next section, Supp. E.4) we were able to reach convergence with the finite difference approach. The adjoint method, in contrast, equipped with our proposed cost function produces numerically stable and smooth results. The corresponding case is shown in the middle column of Fig. 4.

E.3. Influence of Diffusion Distance

The diffusion distance ϕ_{max} (Sect. 4) specifies the size of the narrow band in the diffusion step, i.e., the maximal distance an observed point can have to the surface to be considered in the optimization. In the following, we investigate its influence on the gradient estimation.

Fig. 5 shows the effect of ϕ_{max} on the cost function and gradient estimates, as well as the reconstructed Young's modulus. ϕ_{max} is given in terms of voxels. For small values below 1, only simulation points very close to the observation are used. This introduces additional local minima and leads to gradients with a wrong sign when the current simulation is far from the ground truth (first two plots, left side). Larger values of ϕ_{max} yield correct gradients for the full range of values, although the quality can deteriorate once values become too large and matching becomes ambiguous. In all of our experiments, we found a value of ϕ_{max} between 2 and 5 to produce stable and accurate results.

Note that if a small value for ϕ_{max} is chosen, it can happen that few or no points are matched, especially for later time steps when a simulation deviates from the observations. This typically does not pose a problem for our optimization as long as early time steps tie the simulation to the observations. As the optimizer converges towards a tighter match between observations and cells, more and more points from later timesteps are included in the cost function and improve the results.

E.4. Optimization Stability Analysis

Next, we consider our full optimization, and compare how the factors previously discussed for gradient estimation influence our full algorithm. We sample 20 random start values for the Young's modulus and let the optimization find the optimal value.

We first evaluate the influence of the number of different views on the optimization process. The camera locations are randomly sampled on the hemisphere above the ground and focus on the center of the torus. As Fig. 6 confirms, almost all runs (95% on average, i.e. 19 out of 20) converge and the number of cameras does not have an influence on stability, even a single camera provides enough observations, and multiple views do not negatively affect the reconstruction quality of our algorithm.

Note that in many case, the Young's modulus estimates exhibit slight deviations from the ground truth value of 5000. This is caused by the fact that our data generation step relies on triangle surfaces generated with Marching Cubes. The SSC however, directly matches tri-linearly interpolated SDF values, which hence cannot be matched exactly in most cases.

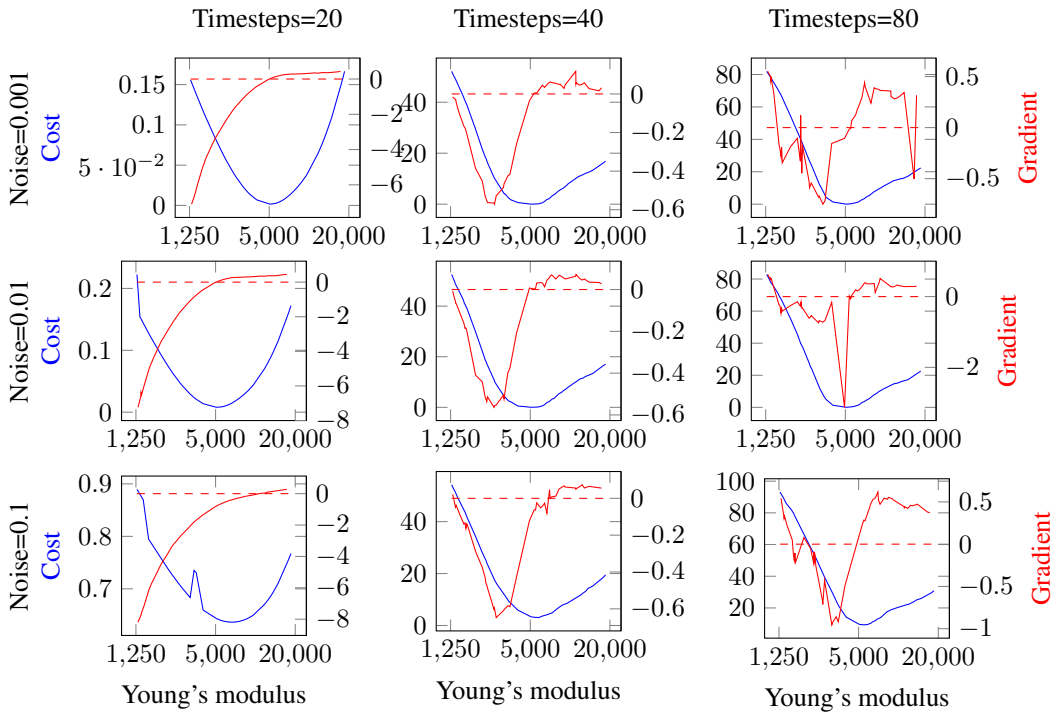


Figure 3: Influence of number of timesteps and camera noise on the cost function and gradient estimates. Test were performed on the torus data set with $\phi_{\text{max}} = 5$.

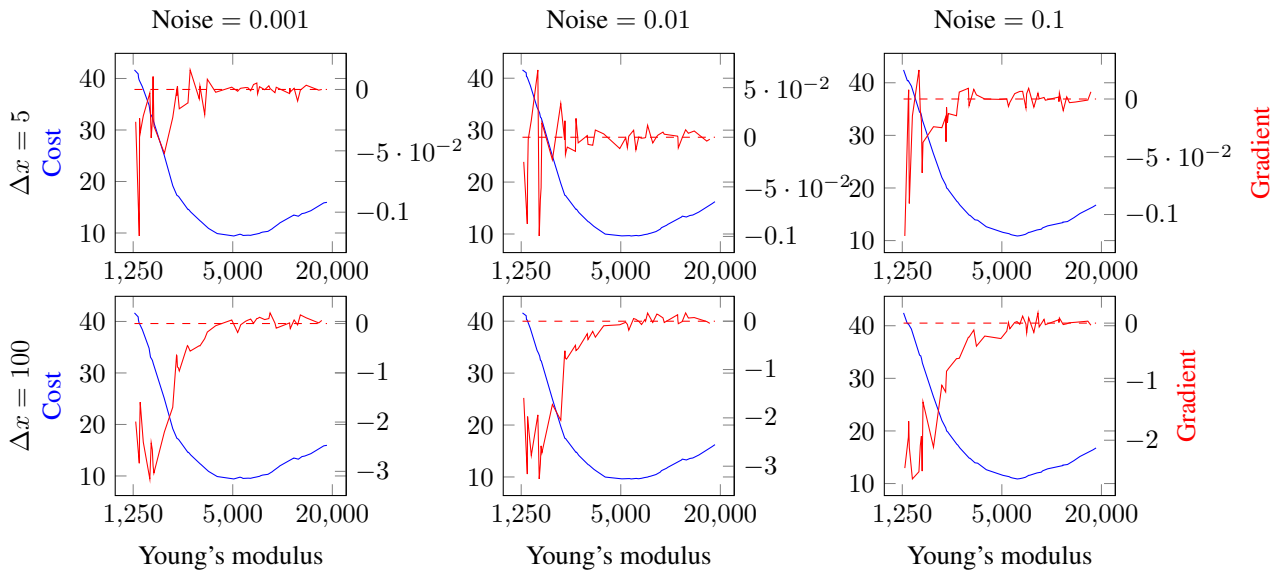


Figure 4: Evaluation of the gradient estimation using finite differences. The test was performed on the torus test case with $\phi_{\text{max}} = 5$ and 40 timesteps. The jaggedness of the red curves, and especially the large number of sign flips visually indicate that the finite-difference approach is not suitable for optimization.

The previous results are consistent with those we found when analyzing the influence of the camera resolution on optimization convergence (see Fig. 7). For one camera, increasing the resolution, and hence using more points in the cost function, does not improve the optimization.

We also vary the camera noise and diffusion distance, results for which are shown in Fig. 8. For a low value for ϕ_{max}

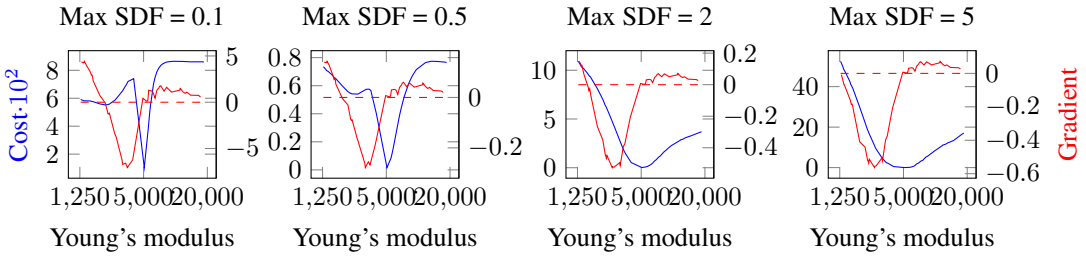


Figure 5: Influence of diffusion distance ϕ_{max} on cost function and gradient estimates for the torus data set (Young's modulus = 5000) with camera noise 0.01 and 40 timesteps.

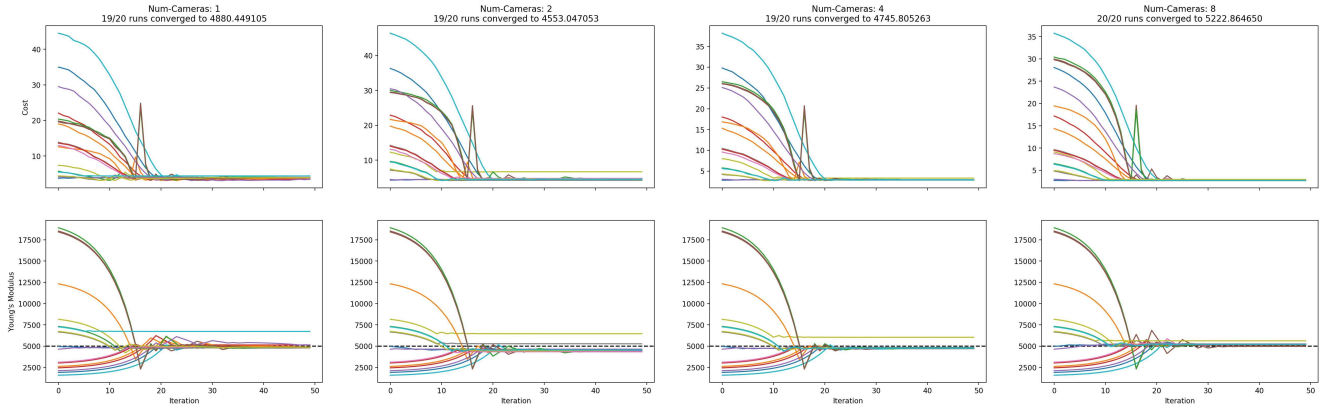


Figure 6: Influence of number of views on optimization convergence for the torus test case, using 40 timesteps, noise=0.1, maxSdf=5.0, and 20 runs with varying initial Young's modulus. The average relative error of the converged runs to the ground truth value is 2.39 %, 8.94 %, 5.08 %, 4.46 % for $1, 2, 4, 8$ cameras, respectively.

of only one, only 10 of 14 runs converge, the other runs start with a very low Young's modulus and get stuck. For a value of $\phi_{\text{max}} = 5$, 13 of 14 runs converge to the ground truth. This shows that the convergence rate decreases if the width of the narrow band is chosen too small. In accordance to previous results from the gradient evaluations in Supp. E.1, the amount of noise has little influence on the stability of the optimization. The results get slightly better with increased noise (more regularization).

Last, we compare the stability of the algorithm for gradients computed with the adjoint method and for gradients computed with the Finite Different Method and varying Δx, see Fig. 9. None of the runs that use the Finite Different Method converge to the Ground Truth value for the Young's modulus, but rather converge to different local minima away from the ground truth value. This is in accordance to the gradient analysis in Supp. E.2 that show a very noisy behaviour for the gradients. In terms of average relative error, this manifests itself as an error of over 35 % for the runs with the finite difference method as compared to 1.41 % for the adjoint method

E.5. Stability for Varying Boundary Conditions

To analyze the dependency of the optimization process on the boundary conditions, we use a scene where a ball bounces on the ground plane. Six different settings are used, with randomly sampled orientation and position of the ground plane, and initial linear velocity of the ball, see Fig. 10. This leads to strongly differing behavior, i.e., the ball rolling with very different speeds in different directions. For each setting, 20 initial values for the Young's modulus between 0.1x and 10x the ground truth value of 2000 are randomly sampled. (Same 20 values for each of the six settings). The simulation is performed over 20 timesteps (one bounce), recorded with one virtual camera of resolution 50x50 and a Gaussian noise with variance of 0.07 voxels. The optimization is performed with a maxSDF value of 1 over 30 iterations.

The results are shown in Fig. 11. Between 18 and 20 of the 20 initial values converge to the ground truth. Interestingly, for Young's Moduli smaller than the ground truth, the cost function increases first before converging towards zero. This seems to indicate that the gradients from the adjoint method point into the right direction, even though this is not directly reflected in the cost function value. Despite this effect, these tests show that our method also robustly

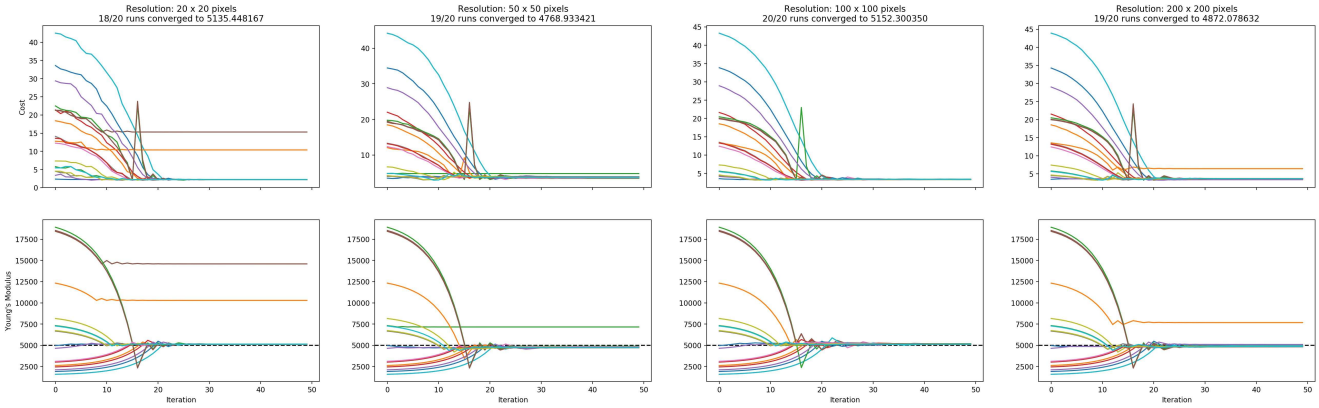


Figure 7: Influence of camera resolution on optimization convergence for the torus test case, using 40 timesteps, noise=0.1, $\phi_{\max} = 5.0$, 20 runs with varying initial Young's modulus. The average relative error of the converged runs to the ground truth value of 5000 is $2.71\,\%, 4.62\,\%, 3.05\,\%, 2.56\,\%$ for a resolution of $20^2, 50^2, 100^2, 200^2$, respectively.

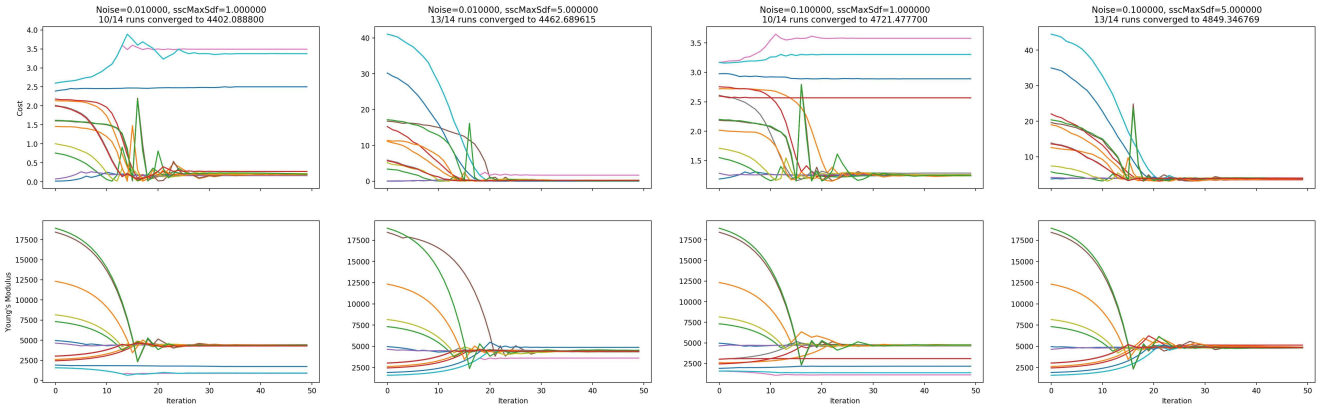


Figure 8: Influence of $\phi_{\max}$ and camera noise on optimization convergence for the torus test case, using 40 timesteps and 14 runs with varying initial Young's modulus. The average relative error of the converged runs to the ground truth for the tests from left to right are: $11.96\,\%, 10.75\,\%, 5.57\,\%, 3.01\,\%$.

recovers the synthetic ground truth under varying boundary conditionns.

E.6. Recovering Multiple Parameters

To analyze the stability of the optimization for multiple parameters, we used the bouncing ball test case again and this time optimize for gravity, the Young's modulus and stiffness damping. We sampled 60 different initial configurations randomly and let the optimizer run for 20 iterations. The ground truth simulation is shown in Fig. 12 on the left. The convergence plots for all 60 runs are shown in Fig. 13. As one can see, all runs converge to a solution that has almost zero cost. The reconstructed value for the gravity is quite uniform, but a strong inter-dependency between Young's modulus and stiffness damping is clearly seen. Despite differing values, the reconstructions very closely match the ground truth.

E.7. Different Optimizers

Throughout our work, we use the R-Prop optimizer for the reconstruction. Here we compare it to a simple Gradient Descent optimizer and the L-BFGS optimizer. While R-Prop only uses the sign of the gradient to determine the next search direction, Gradient Descent also uses the norm of the gradient with an additional adaptive step size, so we would expect faster convergence. L-BFGS approximates the Hessian matrix and as a second-order method should converge even faster.

We compared the three optimizers on three different test cases, the fixed Stanford dragon (see Fig. 14, the bouncing ball from Supp. E.5 and the pillow-ramp test case Fig. 8d. To clearly see the behaviour of the different optimizers when started from the same initial configurations, we only used one or two runs per setting. The results are shown in Fig. 15 and Fig. 16.

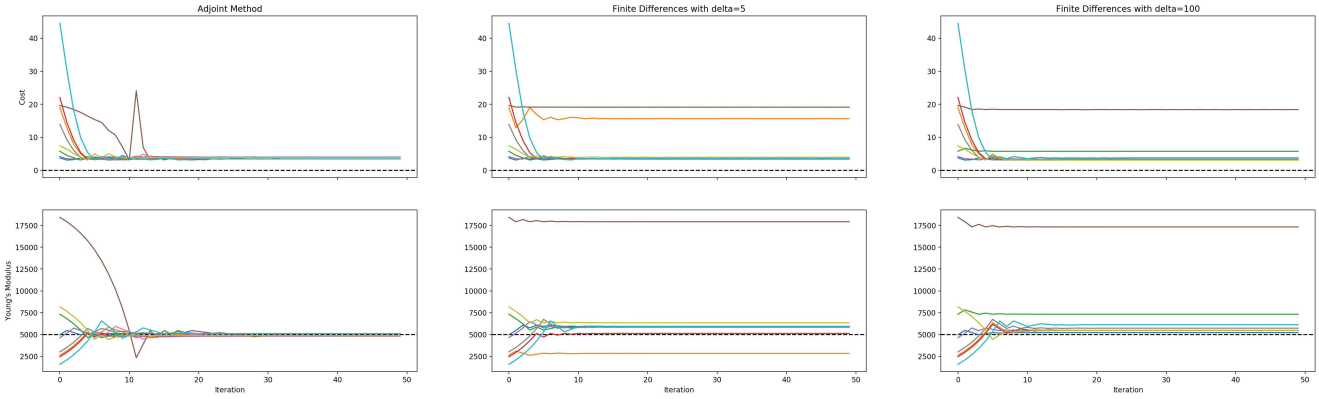


Figure 9: Comparison of the optimization when the gradients are computed with the adjoint method (left) or with finite differences with different values for Δx. For clarity, only seven runs are shown. Note that none of the runs with finite differences comes close to the ground truth solution. The average relative error of all runs to the ground truth value of 5000 is $1.41\,\%, 35.14\,\%, 39.33\,\%$ for the adjoint method and finite differences with $\Delta x = 5, 100$, respectively. Especially, runs with initial values that are far away from the ground truth don't converge with the finite differences.

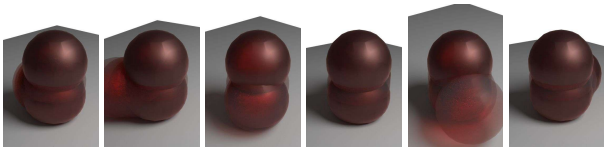


Figure 10: The six ground orientations used to test the independence of the optimization from boundary conditions. The images show the substantially different collision behavior for each case.

For the first two test cases, the Dragon and Bouncing Ball, both the R-Prop and the Gradient Descent algorithm converge to a solution (see Fig. 14c) that is indistinguishable from the ground truth. Gradient Descent converges faster as expected. The L-BFGS algorithm is often too aggressive in choosing the step size, which leads to the optimization getting stuck in sub-optimal local minima (see Figure 15b. Furthermore, it uses the values of the cost function itself to determine the step size, which can lead to instabilities if the cost function only increases when the solution improves. This is the case for the bouncing ball test case when started from a low value for the Young's modulus, see Supp. E.5. These two sources of instability lead to a divergent optimization with the L-BFGS algorithm for the bouncing ball, see Figure 15c.

In the multi-parameters optimization, the Gradient Descent algorithm has severe problems. See e.g. Fig. 16 for an example of the Pillow-Ramp test case where the Gradient Descent algorithm completely diverges and leads to a state where the simulation collapses. We believe that this is because Gradient Descent uses the same step size for all parameters. R-Prop and L-BFGS both use a different step size for each parameter. In the test case using the pillow, the L-BFGS starts to oscillate and gets stuck in a sub-optimal local minimum. R-Prop behaves much smoother and finds a good solution.

F. Real-World Test Cases

Here we analyze the convergence of our algorithm for real-world scenarios, as presented in Sect. 6.2. Reconstructed parameter values for the best five runs are given in Table 1 for the Teddy, Table 2 for the Pillow-Ramp and Table 3 for the Pillow-Flat data set. Furthermore, Fig. 17 shows selected frames and the convergence plots for the Pillow-Flat example. The statistics and timings are depicted in Table 4.

References

[1] J. Benk, M. Ulbrich, and M. Mehl. The Nitsche method of the Navier-Stokes Equations for Immersed Boundaries. In *Seventh International Conference on Computation Fluid Dynamics (ICCFD7)*, 2012.

[2] Chris Greenough. Newmark's method of direct integration. HTTP://WWW.SOFTENG.RL.AC.UK/ST/PROJECTS/FELIB3/DOCS/HTML/INTRO/INTRO-NODE52.HTML, 2001. Accessed: 05/17/2018.

[3] Michael Hauth and Wolfgang Strasser. Corotational simulation of deformable solids. *Journal of WSCG*, 12(1-3), 2003.

[4] Mika Juntunen and Rolf Stenberg. Nitsche's method for general boundary conditions. *Mathematics of Computation*, 78(267):1353–1374, 2009.

[5] Tassilo Kugelstadt, Dan Koschier, and Jan Bender. Fast corotated fem using operator splitting. *Computer Graphics Forum (SCA)*, 37(8), 2018.

[6] Ken Shoemake and Tom Duff. Matrix animation and polar decomposition. In *Proceedings of the confer-*

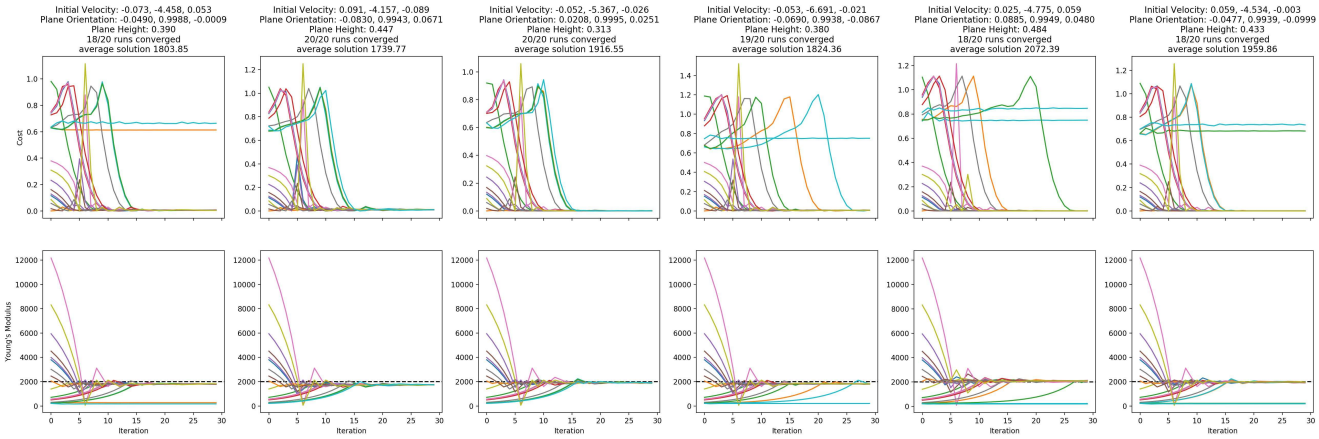


Figure 11: Same scene, a ball bounces on the floor, with different boundary conditions. Six random settings for the ground plane configuration and initial linear velocity.

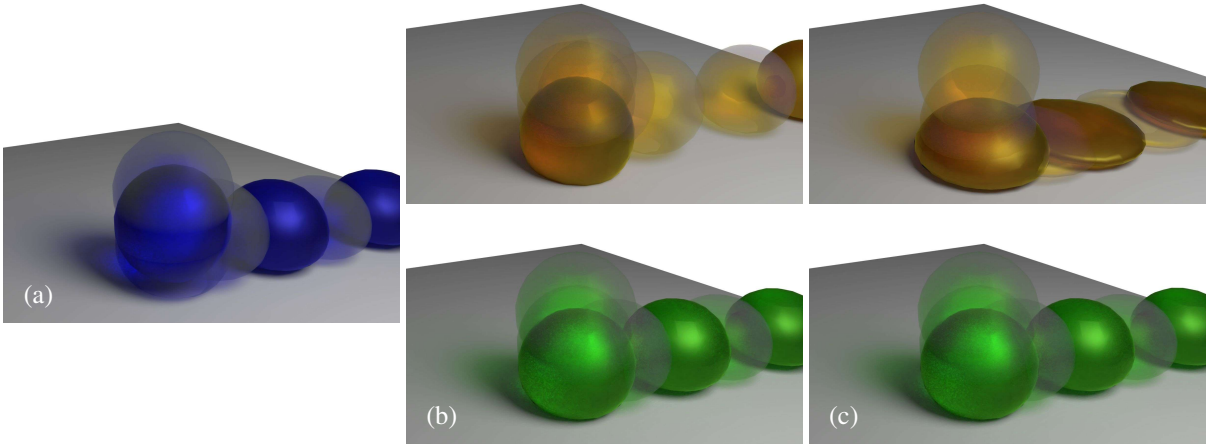


Figure 12: Multi-parameter optimization for the bouncing ball test case. Ground truth (a) and the initial and reconstruction configuration for the two runs with the highest initial cost (b, c). Even though the reconstructed values are different from the ground truth, the output visually matches the ground truth.

ence on Graphics interface, volume 92, pages 258–264, 1992.

[7] Eftychois D. Sifakis. Fem simulation of 3d deformable solids: A practitioner's guide to theory, discretization and model reduction: Part one: The classical fem method and discretization methodology, 2012.

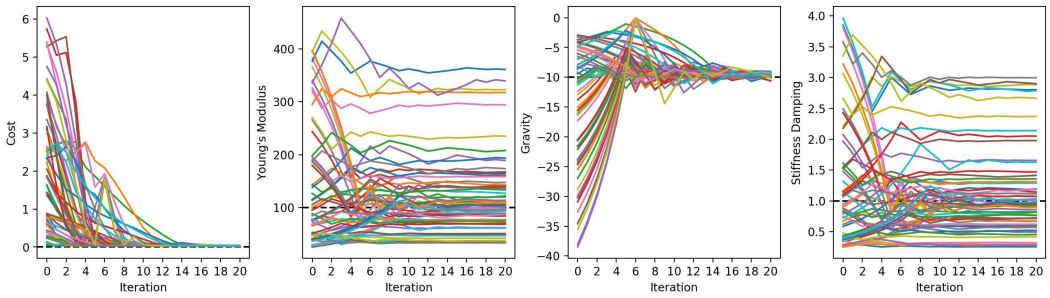


Figure 13: Bouncing ball, optimized for gravity, Young's modulus and stiffness damping. All 60 initial samples converge to a solution that has almost zero cost and visually same behaviour (see Fig. 12). The values for the Young's modulus and stiffness damping, however, strongly differ, showing their inter-dependency.

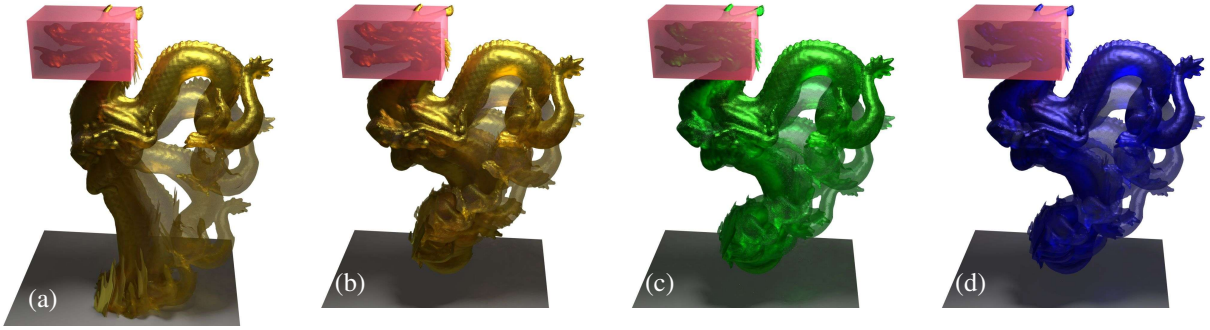


Figure 14: Optimization of the Young's modulus on the Dragon. From left to right: minimal and maximal initial value, reconstruction and ground truth.

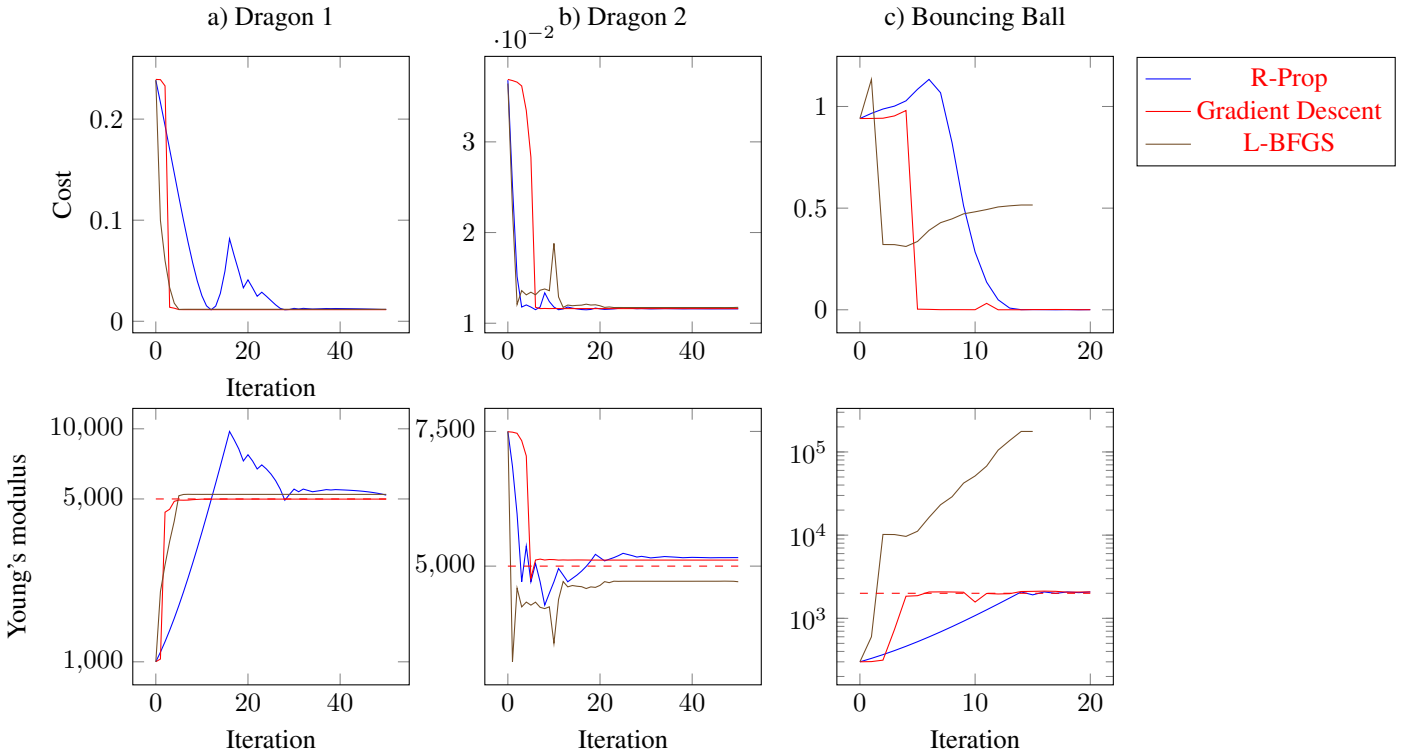


Figure 15: Comparison of the different optimizers for different scenes.

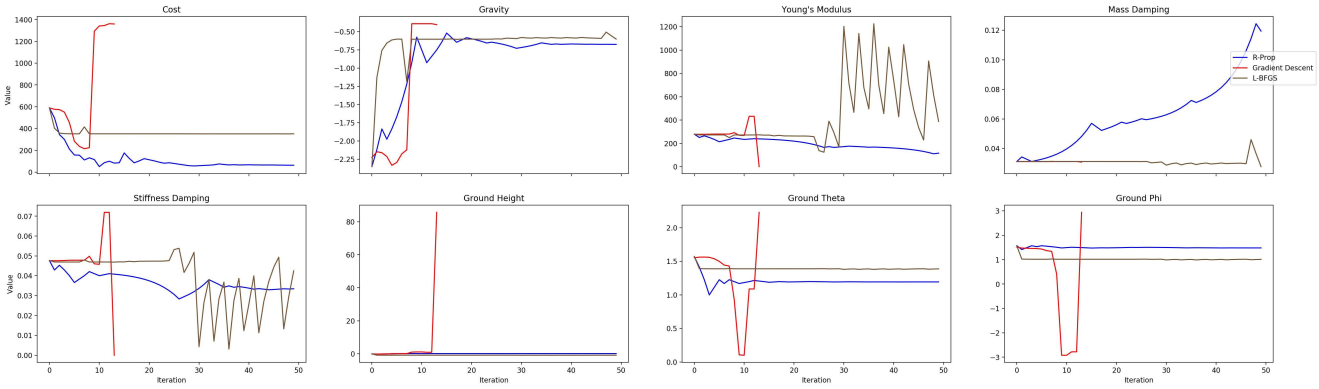


Figure 16: Comparison of the different optimizers for the Pillow-Ramp example

Run	14	15	18	3	1
Initial Cost	143.924	109.387	134.841	136.685	136.551
Recon. Cost	8.461	8.506	8.693	8.747	8.810
Gravity	-1.536	-1.526	-1.538	-1.521	-1.530
Young's Modulus	7.817	8.126	8.690	7.840	7.383
Mass Damping	0.240	0.086	0.080	0.122	0.102
Stiffness Damping	0.027	0.033	0.018	0.040	0.043

Table 1: Reconstructed Parameter values from the teddy data set. Only the five best runs are shown.

Run	7	12	4	18	9
Initial Cost	589.252	605.694	550.197	402.254	383.658
Recon. Cost	59.816	91.852	125.231	164.708	170.164
Gravity	-0.710	-1.430	-1.681	-0.805	-1.573
Young's Modulus	209.421	142.042	914.497	839.310	1143.883
Mass Damping	0.068	0.064	0.071	0.176	0.044
Stiffness Damping	0.044	0.082	0.864	0.176	0.184
Ground Height	0.127	0.179	0.181	-0.009	0.009
Ground Theta	1.188	1.229	1.034	1.288	1.222
Ground Phi	1.480	1.226	1.622	1.502	1.466

Table 2: Reconstructed Parameter values from the Pillow-Ramp data set. Only the five best runs are shown.

Run	8	7	3	5	2
Initial Cost	249.871	154.268	456.128	587.173	116.719
Recon. Cost	21.126	29.303	30.233	34.395	48.241
Gravity	-0.872	-0.883	-1.031	-1.017	-1.038
Young's Modulus	9.200	37.206	233.827	9.191	18.835
Mass Damping	0.078	0.051	0.119	0.425	0.046
Stiffness Damping	0.015	0.008	0.011	0.014	0.011

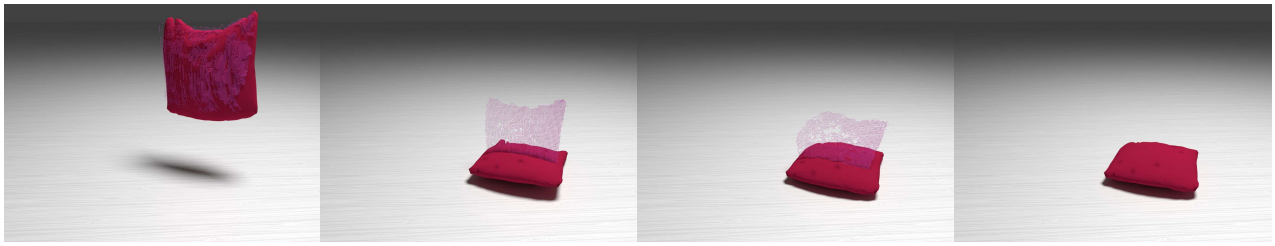
Table 3: Reconstructed Parameter values from the Pillow-Flat data set. Only the five best runs are shown.

	Ball	Teddy	Pillow-Fl.	Pillow-R.
# active nodes	770	2440	3836	4390
# elements	516	1716	2848	3284
# diffused nodes	2605	14612	19006	21530
# timesteps	20	100	450	175
# cameras	1	1	1	1
Camera res.	50x50	320x240	320x240	320x240
Obs. n'th	1	5	5	5
ϕ_{max}	2	10	10	10
# ini. cond.	20-60	18	10	15
Forw. sim. (s)	0.077	0.803	0.240	0.264
Cost eval. (s)	0.199	0.165	0.085	0.153
Adj. sim. (s)	0.058	0.764	0.213	0.292
# iter. steps	30	50	50	50

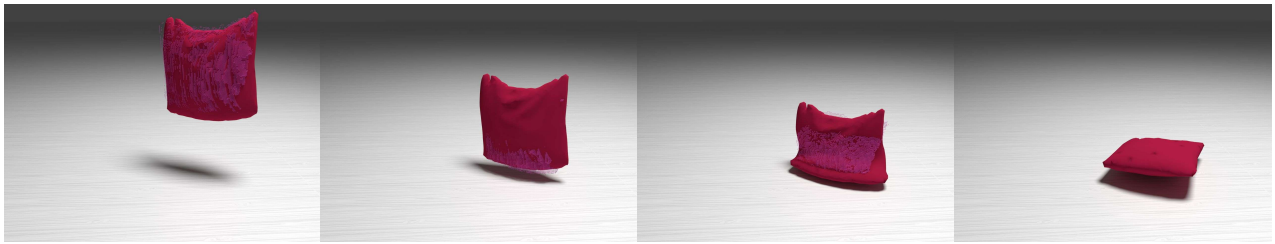
Table 4: Timings (per timestep) and model statistics. "Obs. n'th" denotes the interval in simulation steps between observations, "Ini. cond." denotes the number of randomly perturbed initial conditions.

(a) Color Observation

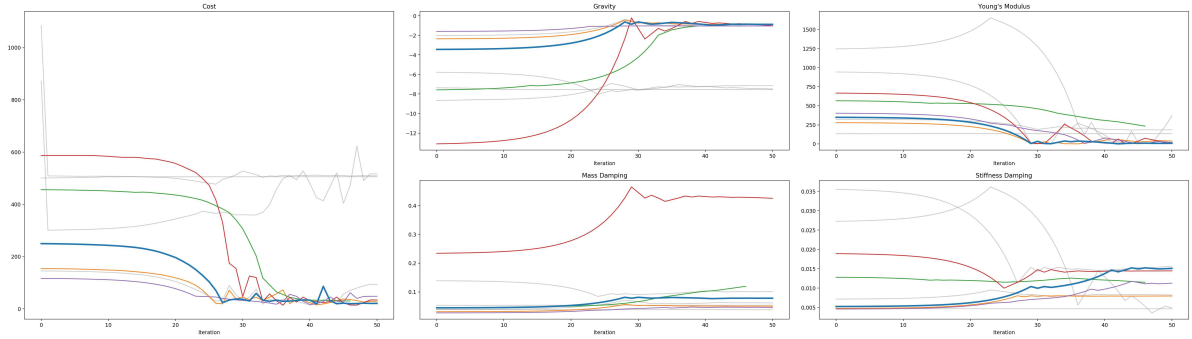
(b) Depth Observation



(c) Initial configuration for the optimization



(d) Reconstructed solution



(e) Optimization started from 10 different initial values. The best five runs are drawn in color, the very best run is drawn in thick lines and displayed in the renderings above.

Figure 17: Selected frames and plots of the optimization process for the Pillow-Flat test case. The rows of (a-d) each show a sequence of steps over time.