Technical Supplement to "GIPC: Fast and stable Gauss-Newton optimization of IPC barrier energy"

KEMENG HUANG, The University of Hong Kong, Hong Kong and TransGP, Hong Kong
FLOYD M. CHITALU*, The University of Hong Kong, Hong Kong
HUANCHENG LIN, The University of Hong Kong, Hong Kong and TransGP, Hong Kong
TAKU KOMURA, The University of Hong Kong, Hong Kong and TransGP, Hong Kong

This document provides supplementary details about our derivations and implementation of the main paper. § 1 describes the reference (explicit) constructions of the constraint Jacobian, together with the normal vector. We then use these descriptions to detail our derivations (and simplifications) of the change-of-basis tensor in § 2. We also prove in § 3 that the first eigenvalue (Eq. (18) in the paper) is always positive, and that remaining eigenvalues (Eq. (19) in the paper) are always negative. § 4 describes the intermediate steps we use to arrive at the eigenpairs of our mollified barrier Hessian. In § 5, we provide a summary of the analysis conducted on the friction Hessian, which is followed by supplementary implementation details of our global matrix-free PCG solver in § 6. Supplementary unit test results are provided in § 7. We also provide a comparison of our barrier method and the original formulation on the same hardware (CPU) in § 8. Further supplemental results concerning different linear solvers and the impact of friction-Hessian projection on overall performance are provided in § 9 and § 10, respectively.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; **Collision detection**; **Massively parallel algorithms**.

Additional Key Words and Phrases: IPC, Barrier Hessian, Eigen Analysis, GPU

1 EXPLICITLY CONSTRUCTING JACOBIANS

In this section, we describe the reference method for explicitly computing the constraint Jacobian $\mathbf{J}(\mathbf{x}, \hat{d})$ and the normal vector $\mathbf{n}(\mathbf{x})$ for a given contact pair (see also Fig. 1). Several of the constructions we will describe can be found throughout literature (see *e.g.* [Kane et al. 1999; Müller et al. 2015; Sifakis and Barbic 2012]) but we provide them here for self-containment.

Vertices of a contact pair form a simplex, with which we measure distance d via a matrix $\mathbf{F}(\mathbf{x}, \hat{d}) = \mathbf{E}(\mathbf{x})\bar{\mathbf{E}}(\mathbf{x}, \hat{d})^{-1}$, where the terms defining this $\mathbf{F}$ denote the ideal- $\bar{\mathbf{E}}$ and current-shape $\mathbf{E}$ matrices of the simplex, respectively. In what follows, we will show how to compute $\mathbf{E}$ and $\bar{\mathbf{E}}$ for each possible simplex/contact-pair.

1.1 Point-triangle

The *point-triangle* case represents a tetrahedron, where the variables constituting the matrix $\mathbf{F}$ are

$$\mathbf{E} = \left[\mathbf{x}_2 - \mathbf{x}_1 | \mathbf{x}_3 - \mathbf{x}_1 | \mathbf{x}_4 - \mathbf{x}_1\right] \in \mathbb{R}^{3\times 3}$$
$$\bar{\mathbf{E}} = \left[\bar{\mathbf{x}}_2 - \bar{\mathbf{x}}_1 | \bar{\mathbf{x}}_3 - \bar{\mathbf{x}}_1 | \bar{\mathbf{x}}_4 - \bar{\mathbf{x}}_1\right] \in \mathbb{R}^{3\times 3}.$$

We construct $\bar{\mathbf{E}}$ with

$$\bar{\mathbf{x}}_1 = \mathbf{x}_1 + \left(\hat{d} - d\right)\mathbf{n}, \quad \bar{\mathbf{x}}_i = \mathbf{x}_i, \quad i = 2, 3, 4$$

where $\mathbf{n}$ is the triangle normal

$$\mathbf{n} = \frac{(\mathbf{x}_3 - \mathbf{x}_2) \times (\mathbf{x}_4 - \mathbf{x}_2)}{\|(\mathbf{x}_3 - \mathbf{x}_2) \times (\mathbf{x}_4 - \mathbf{x}_2)\|} \in \mathbb{R}^{3\times 1}, \tag{1}$$

and $d = \mathbf{v} \cdot \mathbf{n}$ is the distance between the point $\mathbf{x}_1$ and triangle $\mathbf{x}_2$, $\mathbf{x}_3$, $\mathbf{x}_4$ using $\mathbf{v} = \mathbf{x}_1 - \mathbf{x}_2$.

1.2 Edge-edge

The *edge-edge* case also represents a tetrahedron, where $\mathbf{E}$ and $\bar{\mathbf{E}}$ are constructed like the *point-triangle* case. However, the ideal shape positions are now

$$\bar{\mathbf{x}}_1 = \mathbf{x}_1, \quad \bar{\mathbf{x}}_2 = \mathbf{x}_2,$$
$$\bar{\mathbf{x}}_3 = \mathbf{x}_3 + \left(\hat{d} - d\right)\mathbf{n}, \quad \bar{\mathbf{x}}_4 = \mathbf{x}_4 + \left(\hat{d} - d\right)\mathbf{n},$$

where the normal is computed as in Eq. (1) but using the edge vectors, and with $d = \mathbf{v} \cdot \mathbf{n}$ representing the distance between these edges using $\mathbf{v} = \mathbf{x}_3 - \mathbf{x}_1$.

1.3 Point-edge

The *point-edge* case represents a triangle, with a non-square $\mathbf{F}$ constructed with

$$\mathbf{E} = \left[\mathbf{x}_2 - \mathbf{x}_1 | \mathbf{x}_3 - \mathbf{x}_1\right] \in \mathbb{R}^{3\times 2}$$
$$\bar{\mathbf{E}} = \left[\bar{\mathbf{x}}_2 - \bar{\mathbf{x}}_1 | \bar{\mathbf{x}}_3 - \bar{\mathbf{x}}_1\right] \in \mathbb{R}^{2\times 2},$$

We can compute $\bar{\mathbf{E}}$ by projecting vertex components to the plane

$$\bar{\mathbf{x}}_1 = \mathbf{K}\mathbf{H}\left(\mathbf{x}_1 + \left(\hat{d} - d\right)\mathbf{n}_e\right), \quad \bar{\mathbf{x}}_2 = \mathbf{K}\mathbf{H}\mathbf{x}_2, \quad \bar{\mathbf{x}}_3 = \mathbf{K}\mathbf{H}\mathbf{x}_3,$$

where the in-plane edge normal $\mathbf{n}_e$ ($\mathbf{n}_e \perp \mathbf{x}_2\mathbf{x}_3$) is given by

$$\mathbf{n}_e = \frac{(\mathbf{x}_2 - \mathbf{x}_3) \times \mathbf{n}_t}{\|(\mathbf{x}_1 - \mathbf{x}_2) \times \mathbf{n}_t\|} \in \mathbb{R}^{3\times 1},$$

with $\mathbf{n}_t$ denoting triangle normal computed as in Eq. (1) but using $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$. The matrix $\mathbf{H} \in \mathbb{R}^{3\times 3}$ represents a rotation to align the triangle to a canonical axis plane, *e.g.* with normal $\mathbf{n}_t = (0, 1, 0)$.

*Currently unaffiliated. Was at The University of Hong Kong while contributing to this work.

Authors' addresses: Kemeng Huang, kmhuang@connect.hku.hk, kmhuang819@gmail.com, The University of Hong Kong, Pokfulam Road, Hong Kong and TransGP, Building 19W SPX1, Hong Kong; Floyd M. Chitalu, floyd.m.chitalu@gmail.com, The University of Hong Kong, Pokfulam Road, Hong Kong; Huancheng Lin, lamws@connect.hku.hk, The University of Hong Kong, Pokfulam Road, Hong Kong and TransGP, Building 19W SPX1, Hong Kong; Taku Komura, taku@cs.hku.hk, The University of Hong Kong, Pokfulam Road, Hong Kong and TransGP, Building 19W SPX1, Hong Kong.

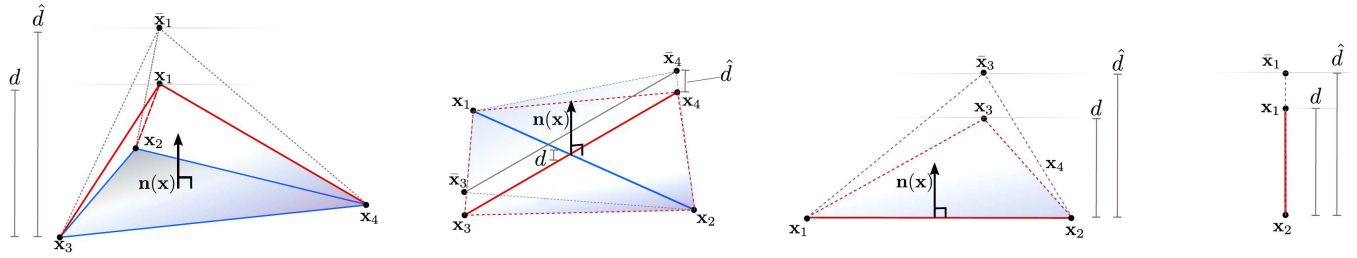


Fig. 1. *The primitive contact pairs that are tested for intersection. From left to right, we have point-triangle (tetrahedron); edge-edge (tetrahedron); point-edge (triangle); and point-point (line-segment). In each case, the vertices can be seen as forming a simplex with which we formulate an impermeable barrier energy using a Jacobian (matrix) constructed from these vertices.*

Several methods exist to determine this rotation with one example being Rodrigues' formula [Murray et al. 1994] from which we get

$$\mathbf{H} = \begin{cases} 2\frac{(\mathbf{p}+\mathbf{b})(\mathbf{p}+\mathbf{b})^T}{(\mathbf{p}+\mathbf{b})^T(\mathbf{p}+\mathbf{b})} - \mathbf{I} & \mathbf{p} \neq -\mathbf{b} \\ \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \text{otherwise,} \end{cases} \tag{2}$$

where $\mathbf{p} := \mathbf{n}_t$, $\mathbf{b} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$ and $\mathbf{K} \in \mathbb{R}^{2\times 3}$ is a matrix (subject to our choice of $\mathbf{b}$) to extract the xz components of a 3D vector

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

Finally, $d = \mathbf{v} \cdot \mathbf{n}_e$ is the distance between point $\mathbf{x}_1$ and edge $\mathbf{x}_2\mathbf{x}_3$ using $\mathbf{v} = \mathbf{x}_1 - \mathbf{x}_2$.

1.4 Point-Point

The *point-point* case reduces to a line segment and where the Jacobian is a vector. We have

$$\mathbf{E} = \mathbf{v} \in \mathbb{R}^{3\times 1} \quad \text{and} \quad \bar{\mathbf{E}} = \begin{bmatrix} \bar{\mathbf{x}}_2 - \bar{\mathbf{x}}_1 \end{bmatrix} \in \mathbb{R}, \tag{3}$$

where $\mathbf{v} = \mathbf{x}_2 - \mathbf{x}_1$, and

$$\bar{\mathbf{x}}_1 = \mathbf{K}\mathbf{H}\left(\mathbf{x}_1 + \left(\hat{d} - d\right)\mathbf{n}\right) \in \mathbb{R} \quad \text{and} \quad \bar{\mathbf{x}}_2 = \mathbf{K}\mathbf{H}\mathbf{x}_2 \in \mathbb{R},$$

where $\mathbf{n} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$ is the normal, and the rotation $\mathbf{H}$ is constructed similarly to Eq. (2) *i.e.* as one which aligns $\mathbf{n}$ to the canonical y-axis of $(0, 1, 0)$. Assuming the same choice for the vector $\mathbf{b}$ as above, the matrix $\mathbf{K} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$ now extracts the y-component of a 3D vector. Thus, $d = \mathbf{v} \cdot \mathbf{n} = \|\mathbf{n}\| = \|\mathbf{v}\|$.

In all cases above, we have $\mathbf{KHn}$ as the normal vector used to evaluate the gap function, using $\mathbf{n}_e$ for the point-edge case.

2 CHANGE-OF-BASIS TENSOR

This section outlines how we compute the change-of-basis tensor $\frac{\partial \mathbf{F}}{\partial \mathbf{x}} \in \mathbb{R}^{\dim(\mathbf{F})\times 3s}$ from a contact pair comprised of s vertices. This tensor can be understood as a column vector with block entries of dimensions $\dim(\mathbf{F}) \equiv \mathbb{R}^{3\times m}$ for an m-dimensional simplex.

Definition. The general expression defining the change-of-basis tensor is given by

$$\frac{\partial \mathbf{F}}{\partial \mathbf{x}} = \frac{\partial \mathbf{E}}{\partial \mathbf{x}}\bar{\mathbf{E}}^{-1} + \mathbf{E}\frac{\partial \bar{\mathbf{E}}^{-1}}{\partial \mathbf{x}}, \tag{4}$$

where $\bar{\mathbf{E}}(\mathbf{x}(t))$ is our so-called 'ideal' configuration matrix which varies unlike the case of hyperelastic materials where the analogous reference shape matrix is constant[1]. This variation is evaluated with the identity

$$\frac{\partial \bar{\mathbf{E}}^{-1}}{\partial \mathbf{x}_i} = -\bar{\mathbf{E}}^{-1}\frac{\partial \bar{\mathbf{E}}}{\partial \mathbf{x}_i}\bar{\mathbf{E}}^{-1},$$

and is computed w.r.t the m vertices in the stencil of a contact pair. It is worth noting that the second term $\mathbf{E}\frac{\partial \bar{\mathbf{E}}^{-1}}{\partial \mathbf{x}_i}$ of Eq. (4) has been observed to be negligible during the conducted empirical trials.

Higher order derivatives. The second order derivative $\partial^2\left(\bar{\mathbf{E}}^{-1}\right)/\partial \mathbf{x}^2$ exists too since the analytic expressions of some entries in $\bar{\mathbf{E}}$ are dependent on the normal vector $\mathbf{n}(\mathbf{x})$ that is computed from the cross-product between edge-vectors. This is also the reason why our local force Jacobian (Eq. (15) in the paper) is dependent on $\partial^2\mathbf{F}/\partial \mathbf{x}^2$ ($\partial^2\mathbf{J}/\partial \mathbf{x}^2$ in the paper), which intuitively captures higher-order variations in $\bar{\mathbf{E}}$ due to a changing normal vector $\mathbf{n}(\mathbf{x})$ w.r.t positions.

2.1 Relationship between F and J

The contact constraint Jacobian $\mathbf{J} \equiv \mathbf{S}$ that we use in the paper can be viewed as corresponding to the stretch factor of the matrix $\mathbf{F} = \mathbf{RS}$ from polar decomposition[2]. This decomposition yields a rotation $\mathbf{R}$ and symmetric stretch $\mathbf{S} = \mathbf{V}\Sigma\mathbf{V}^T$, both of which will be diagonal in our method with $\mathbf{R} = \mathbf{U} = \mathbf{V} = \mathbf{I}$ due to the compressive distortion assumption of the simplex along the vector $-\mathbf{n}(\mathbf{x})$ as outlined in Figure 2 of the paper. In general, either $\mathbf{F}$ or $\mathbf{J}$ can be used to define the barrier function b, its gradient $\partial b/\partial \mathbf{x}$ and force Jacobian $\partial^2 b/\partial \mathbf{x}^2$: The dimensions of the local force vector and force Jacobian matrix will be equal in either case, which can be demonstrated by applying tensor vectorization (*e.g.* on either Eq. (13) or Eq. (15) in the paper) to show that tensor contraction in the derivatives will cancel out differences in the dimensions between $\mathbf{F}$ and $\mathbf{J}$ for a given m-dimensional simplex with $\mathbf{x} \in \mathbb{R}^{3s\times 1}$. The change-of-basis tensor $\partial \mathbf{J}/\partial \mathbf{x}$ is computed w.r.t the explicit entries in the diagonal structure of $\mathbf{J}$.

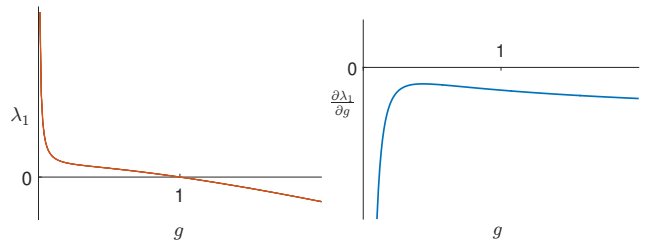
[1] See also the element rehabilitation scheme of Kim et al. [2019].
[2] It is likewise possible to take the perspective of singular value decomposition $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^T$ from which $\mathbf{J} = \Sigma$ since the method outlined in the paper may be viewed as working in the local principal stretch space of the deforming simplex along $-\mathbf{n}$.

ALGORITHM 1: Warp reduction for matrix-vector multiplication.

```
for each thread, thread_id do
    /* Variable initialization                          */
    shared offset;
    H_dims ← m² // Total entries per Hessian ∈ ℝ^{m×m}
    H_id ← thread_id / H_dims // Hessian index
    R_id ← (thread_id%H_dims) / m // Hessian row index
    C_id ← (thread_id%H_dims)%m // Hessian col index
    v_id ← C_id / 3 // Vertex index in Hessian
    vc_id ← C_id % 3 // vertex component index
    e_id ← thread_id%m // Entry index in Hessian row
    h ← load(H_id, R_id, C_id, ...) // Hessian entry value
    // component of vector multiplied with Hessian
    c ← load(v_id, vc_id, ...) ;
    r ← c · h // scalar multiplication result
    if thread_id == 0 then
        offset ← (m − e_id);
    end
    /* Wait for 'offset' initialization                 */
    barrier();
    B_id ← (thread_id−offset+m) / m // Hess row index in block
    // Re-calibrated offset of 1st full Hessian row
    l_id ← (thread_id − offset)%m ;
    if B_id == 0 then
        l_id ← thread_id;
    end
    w_id ← thread_id%32 // Warp index
    // is 1st thread after boundary
    is_boundary ← (l_id == 0)||(w_id == 0) ;
    // set the bit value of mark according to boundary
    mark ← cudaBrev(cudaBallot(is_boundary)) ;
    // length of reduction range in warp
    interval ← cudaClz(mark«(w_id + 1)) ;
    // clamp to warp size
    interval ← min(interval, 31 − w_id) ;
    /* warp reduction (accumulate 'r')                  */
    iter ← 1;
    while iter < m do
        // read value 'r' of neighbour(s) in warp
        tmp ← cudaShfldown(r, iter) ;
        if interval ≥ iter then
            r += tmp
        end
        iter «= 1 // bitshift
    end
    // only 1st thread after boundary will write
    if is_boundary then
        addToOutputArray(r) // write using atomics
    end
end
```

3 EIGENVALUE PROPERTIES

This section provides the proof showing that only the first eigenvalue λ_1 (*cf.* Eq. (18) and Eq. (19) in the paper) is ever positive for the standard contact pairs *i.e.* those not representing the nearly-parallel



(a) *Eigenvalue* **(b)** *First derivative*

Fig. 2. *A plot of Eq. (5) in the paper and its derivatives, where* $\hat{d} = 1$.

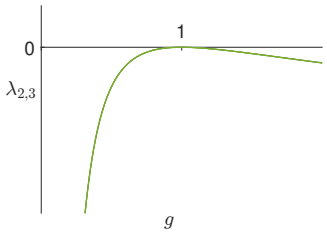


Fig. 3. *A plot of Eq. (6) in the paper, where* $\hat{d} = 1$.

case. We have used Eq. (12) in the main paper to obtain

$$\lambda_1 = \frac{2\hat{d}^4(6g + 2g\ln(g) - 7g^2 - 6g^2\ln(g) + 1)}{g}, \tag{5}$$

$$\lambda_{2,3} = \frac{-2\hat{d}^4\,(g-1)\,(g + 2g\ln\,(g) - 1)}{g^2}, \tag{6}$$

as the expanded expressions that we use for our proof. We would arrive at different expressions for $\lambda_{1,2,3}$ if we instead used Eq. (24) in the paper but the behaviour (plots) remains the same.

λ_1 is always positive. Since our gap function $g \in (0, 1)$ has limited range, we have

$$\lim_{g\to 0^+} \lambda_1 = +\infty \tag{7}$$

$$\lim_{g\to 1} \lambda_1 = 0, \tag{8}$$

where the condition $\lambda_1 \geq 0$ is always true (Fig. 2a).

Proof. The first eigenvalue λ_1 is monotonically decreasing, which we demonstrate using the 1st derivative of Eq. (5)

$$\frac{\partial \lambda_1}{\partial g} = \frac{-2\hat{d}^4(13g^2 - 2g + 6g^2\ln(g) + 1)}{g^2}. \tag{9}$$

Eq. (9) is *always* negative (Fig. 2b) which is inline with our observation in Eq. (8) that λ_1 is ever decreasing from positive to zero as $g \to 1$. Therefore, $\frac{\partial \lambda_1}{\partial g} < 0$ is always true, which means $\lambda_1 > 0$ is also true when $g \in (0, 1)$. □

$\lambda_{2,3}$ are always negative. The second eigenvalue λ_2 and third eigenvalue λ_3 are always less than zero when $g \in (0, 1)$.

Proof. Eq. (6) has an odd number of terms which are less-than or equal-to zero

$$\begin{aligned} g - 1 &\leq 0, \\ g + 2g\ln(g) - 1 &\leq 0, \\ -2\hat{d}^4 &\leq 0. \end{aligned} \tag{10}$$

Thus, we have $\lambda_{2,3} \leq 0$, which is also shown in Fig. 3.

□

Eq. (7)-Eq. (10) summarise why exactly one eigenpair is sufficient to guarantee positive semi-definiteness for minimizing our barrier energy.

4 APPROXIMATE MOLLIFIED HESSIAN TERMS

This section summarises of the steps to derive Eq. (32) in the paper (§ 4.1), and the steps we follow to arrive at the eigensystem of the last two terms of our mollified barrier Hessian (§ 4.2).

4.1 Eigenvectors of the first two terms

We follow the approach outlined in [Kim et al. 2019], to arrive at the eigenpairs

$$\lambda_{\gamma 1} = 2\left(\frac{\partial \tilde{b}}{\partial \gamma} + 2\gamma\frac{\partial^2 \tilde{b}}{\partial \gamma^2}\right), \quad \mathbf{Q}_{\gamma 1} = \frac{1}{\sqrt{\gamma}}\mathbf{J}\mathbf{n}_\gamma\mathbf{n}_\gamma^T.$$

where

$$\mathbf{Q}_{\gamma 1} = \frac{1}{\sqrt{c}}\begin{bmatrix}1 & 0 & 0\\ 0 & \sqrt{c} & 0\\ 0 & 0 & f\end{bmatrix}\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix} = \mathbf{n}_\gamma\mathbf{n}_\gamma^T$$

is the expansion from which we get the expression used in the paper. Similar steps are followed to obtain λ_{g1} and $\mathbf{Q}_{g1} = \mathbf{n}_g\mathbf{n}_g^T$.

For the eigenpairs

$$\lambda_{\gamma 2,\gamma 3} = 2\frac{\partial \tilde{b}}{\partial \gamma}, \quad \mathbf{Q}_{\gamma 2}, \mathbf{Q}_{\gamma 3},$$

we use the twist matrices from Smith et al. [2019]

$$\mathbf{T}_x = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 0 & -1 & 0\end{bmatrix}, \quad \mathbf{T}_y = \begin{bmatrix}0 & 0 & -1\\ 0 & 0 & 0\\ 1 & 0 & 0\end{bmatrix}, \quad \mathbf{T}_z = \begin{bmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 0\end{bmatrix},$$

to arrive at

$$\mathbf{Q}_{\gamma 2} = normalize(\mathbf{U}\mathbf{T}_x\Sigma\mathbf{V}^T\mathbf{n}_\gamma\mathbf{n}_\gamma^T), \tag{11}$$

$$\mathbf{Q}_{\gamma 3} = normalize((\sigma_y\hat{a}_y\mathbf{U}\mathbf{T}_z - \sigma_z\hat{a}_z\mathbf{U}\mathbf{T}_y)\Sigma\mathbf{V}^T\mathbf{n}_\gamma\mathbf{n}_\gamma^T), \tag{12}$$

using SVD $\mathbf{J} = \mathbf{U}\Sigma\mathbf{V}^T$, where $\sigma_{y,z}$ are the second and third singular values from Σ, and $\hat{a}_{y,z}$ are the entries of the vector $\mathbf{V}^T\mathbf{n}_\gamma$. We also have $\mathbf{U} = \mathbf{V} = \mathbf{I}$ due to the special diagonal structure of $\mathbf{J}$ from which $\mathbf{Q}_{\gamma 2}$ and $\mathbf{Q}_{\gamma 3}$ can be simplified as

$$\mathbf{Q}_{\gamma 2} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 0 & -1 & 0\end{bmatrix}\mathbf{n}_\gamma\mathbf{n}_\gamma^T, \quad \mathbf{Q}_{\gamma 3} = \begin{bmatrix}0 & 1 & 0\\ -1 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\mathbf{n}_\gamma\mathbf{n}_\gamma^T.$$

Similarly, we can get $\mathbf{Q}_{g2}$ and $\mathbf{Q}_{g3}$ by

$$\mathbf{Q}_{g2} = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 0 & -1 & 0\end{bmatrix}\mathbf{n}_g\mathbf{n}_g^T, \quad \mathbf{Q}_{g3} = \begin{bmatrix}0 & 0 & 1\\ 0 & 0 & 0\\ -1 & 0 & 0\end{bmatrix}\mathbf{n}_g\mathbf{n}_g^T.$$

4.2 Eigenpairs of the last two terms

We arrive at $\lambda_7\mathbf{Q}_7$ and $\lambda_8\mathbf{Q}_8$ in the paper with

$$\begin{aligned} \frac{\partial^2 \tilde{b}}{\partial \gamma \partial g}\mathrm{vec}\left(\frac{\partial g}{\partial \mathbf{J}} \otimes \frac{\partial \gamma}{\partial \mathbf{J}}\right) + \frac{\partial^2 \tilde{b}}{\partial g \partial \gamma}\mathrm{vec}\left(\frac{\partial \gamma}{\partial \mathbf{J}} \otimes \frac{\partial g}{\partial \mathbf{J}}\right) = \\ \frac{\partial^2 \tilde{b}}{\partial \gamma \partial g}\mathbf{g}_g\mathbf{g}_\gamma^T + \frac{\partial^2 \tilde{b}}{\partial g \partial \gamma}\mathbf{g}_\gamma\mathbf{g}_g^T = \\ \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4t\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 4t & 0 & 0 & 0 & 0\end{bmatrix}, \end{aligned} \tag{13}$$

which is an expansion of the last two terms of the mollified barrier Hessian, where $\mathbf{g}_* = \mathrm{vec}(\partial(*)/\partial\mathbf{J})$ with $*$ being a placeholder for γ or g. We obtain the closed-form expressions of the eigenpairs by analysing this matrix in Eq. (13) as an eigenproblem.

We arrive at our expressions for $\lambda_7'\mathbf{Q}_7'$ and $\lambda_8'\mathbf{Q}_8'$ in the paper by solving an eigenproblem on the auxiliary matrix

$$\begin{aligned} \mathbf{M} &= \lambda_{\gamma 1}\mathrm{vec}\left(\mathbf{Q}_{\gamma 1}\right)\mathrm{vec}\left(\mathbf{Q}_{\gamma 1}\right)^T + \lambda_{g1}\mathrm{vec}\left(\mathbf{Q}_{g1}\right)\mathrm{vec}\left(\mathbf{Q}_{g1}\right)^T \\ &\quad + \lambda_7\mathrm{vec}\left(\mathbf{Q}_7\right)\mathrm{vec}\left(\mathbf{Q}_7\right)^T + \lambda_8\mathrm{vec}\left(\mathbf{Q}_8\right)\mathrm{vec}\left(\mathbf{Q}_8\right)^T \\ &= \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & \lambda_{\gamma 1} & 0 & 0 & 0 & 4t\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 4t & 0 & 0 & 0 & \lambda_{g1}\end{bmatrix}, \end{aligned} \tag{14}$$

based on the fact that the mollified Hessian is defined as

$$\mathrm{vec}\left(\frac{\partial^2 \tilde{b}}{\partial \mathbf{J}^2}\right) \equiv \mathbf{M} + \sum_{i=\gamma 2,\gamma 3,g2,g3}\lambda_i\mathrm{vec}\left(\mathbf{Q}_i\right)\mathrm{vec}\left(\mathbf{Q}_i\right)^T, \tag{15}$$

The non-orthogonal eigenmatrices $\mathbf{Q}_{\gamma 1}$, $\mathbf{Q}_{g1}$, $\mathbf{Q}_7$, $\mathbf{Q}_8$ lie in the subspace represented by $\mathbf{Q}_7'$ and $\mathbf{Q}_8'$, which we determine by solving an eigenproblem on $\mathbf{M}$.

5 ANALYSIS OF FRICTION HESSIAN

This section provides a summary of the analysis conducted on the Hessian of the smooth friction model, as proposed by [Li et al. 2020]. The model uses a 'lagged' sliding basis, where contact force is computed using some quantities that are computed at the last time step. Specifically, the formulation of the local friction force is

$$F_k(\mathbf{x}, \lambda_k^n, \boldsymbol{T}_k^n, \mu) = -\mu\lambda_k^n\boldsymbol{T}_k^n f_1(\|\mathbf{u}_k\|)\frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}, \tag{16}$$

where μ denotes the friction coefficient, k here denotes the collision pair index, $\mathbf{u}_k = \boldsymbol{T}_k^{nT}\mathbf{x}_k^r \in \mathbb{R}^{2\times 1}$, $\mathbf{x}_k^r$ is the relative displacement of k-th collision pair, while λ_k^n and $\boldsymbol{T}_k^n \in \mathbb{R}^{3s\times 2}$ represent the sliding

basis and contact normal force derived from the previous time step that we refer to by the superscript n here. The function denoted here by f_1 will provide a smooth and monotonic transition from 0 to 1 over a finite range

$$f_1(\|\mathbf{u}_k\|) = \begin{cases} -\frac{\|\mathbf{u}_k\|^2}{\epsilon_v^2 \Delta t^2} + \frac{2\|\mathbf{u}_k\|}{\epsilon_v \Delta t}, & \|\mathbf{u}_k\| \in (0, \Delta t \epsilon_v) \\ 1, & \|\mathbf{u}_k\| > \Delta t \epsilon_v \end{cases}, \tag{17}$$

which is also detailed in [Li et al. 2020]. This leads to the corresponding friction potential

$$D_k(\mathbf{x}) = \mu \lambda_k^n f_0(\|\mathbf{u}_k\|). \tag{18}$$

Here, f_0 is defined such that $f_0' = f_1$ and $f_0(\epsilon_v h) = \epsilon_v \Delta t$ so that $F_k(\mathbf{x}) = -\nabla_{\mathbf{x}} D_k(x)$. The Hessian of $D_k(\mathbf{x})$ can be simply given by:

$$\nabla_{\mathbf{x}}^2 D_k(\mathbf{x}) = \mu \lambda_k^n \boldsymbol{T}_k^n \left(\frac{f_1'(\|\mathbf{u}_k\|)\|\mathbf{u}_k\| - f_1(\|\mathbf{u}_k\|)}{\|\mathbf{u}_k\|^3} \mathbf{u}_k \mathbf{u}_k^T + \frac{f_1(\|\mathbf{u}_k\|)}{\|\mathbf{u}_k\|} \mathbf{I}_2 \right) \boldsymbol{T}_k^{nT} \in \mathbb{R}^{3s \times 3s}. \tag{19}$$

where $\mathbf{I}_2$ is a 2×2 identity matrix. Eigenanalysis of the Hessian in Eq. (19) essentially boils down to the analysis of a simple 2×2 matrix,

$$\frac{f_1'(\|\mathbf{u}_k\|)\|\mathbf{u}_k\| - f_1(\|\mathbf{u}_k\|)}{\|\mathbf{u}_k\|^3} \mathbf{u}_k \mathbf{u}_k^T + \frac{f_1(\|\mathbf{u}_k\|)}{\|\mathbf{u}_k\|} \mathbf{I}_2, \tag{20}$$

which can be accomplished by analytically solving a quadratic characteristic polynomial.

6 DATA-PARALLEL SOLVER

This section provides the high level information (and algorithm) that we use to parallelise our GPU implementation of preconditioned conjugate gradients (PCG) [Shewchuk 1994] without constructing the full system matrix. This scheme has been adopted previously, *e.g.* by Gao et al. [2018] and Wang et al. [2020], where our novelty lies in the parallel optimization of local matrix-vector multiplications. The remaining vector-vector multiplications of PCG is optimized with the global reduction methods of Zhao et al. [2020].

The traditional approach to solving the global linear system requires prior construction/assembly of a (sparse) system matrix, which involves incrementally *adding* contributions into its block entries from the local Hessians [Baraff and Witkin 1998; Tamstorf et al. 2015]. This is especially suitable for elastic deformation problems with finite elements or mass-spring systems where mesh topology is fixed to allow pre-computation of the non-zero entries/indices in the global matrix. In our case the time-dependent nature of the number of IPC barrier energy terms precludes such pre-computation to affect the cost of booking-keeping for tracking dynamically-changing sparse matrix indexing offsets.

Decomposing system matrix-vector multiplication. It is possible to avoid such global construction by decomposing system matrix back to the local form using the distributive property of addition, which we use and can be demonstrated using a simple 2D mass-spring system with three nodes and two springs. Fig. 4 shows two springs

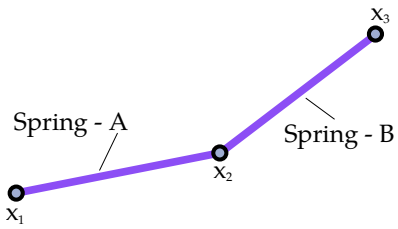


Fig. 4. *Mass spring system with two springs*

endowed with a (local system) matrix each

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \equiv \frac{\partial^2 \Psi_A}{\partial \mathbf{x}^2}, \tag{21}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} \equiv \frac{\partial^2 \Psi_B}{\partial \mathbf{x}^2}, \tag{22}$$

which correspond to their energies Ψ_A and Ψ_B, respectively, with block entries $\mathbf{A}_{ij}, \mathbf{B}_{ij} \in \mathbb{R}^{2\times 2}$. The global system matrix may be constructed as follows:

$$\underbrace{\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & 0 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\mathbb{A}} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{B}_{11} & \mathbf{B}_{12} \\ 0 & \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix}}_{\mathbb{B}} = \underbrace{\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & 0 \\ \mathbf{A}_{21} & \mathbf{A}_{22} + \mathbf{B}_{11} & \\ 0 & \mathbf{B}_{12} & \mathbf{B}_{22} \end{bmatrix}}_{\mathbb{H}},$$

with dimensions $\mathbb{H} \in \mathbb{R}^{6\times 6}$ which is based on the connectivity of the system.

Two springs sharing one vertex will result in a summation of two specific block entries from $\mathbf{A}$ and $\mathbf{B}$, which are determined by a local-to-global index map. This map, which is constant, is based on the connectivity of the mass spring system, where 'local' refers to node indices in a spring versus the 'global' indices in the mass-spring system[3]. Thus, we have

$$\begin{aligned} \mathbb{H}\mathbf{c} &= (\mathbb{A} + \mathbb{B})\mathbf{c}, \\ &= \mathbb{A}\mathbf{c} + \mathbb{B}\mathbf{c} \in \mathbb{R}^{6\times 1}, \end{aligned} \tag{23}$$

with $\mathbf{c} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 \end{bmatrix}^T$ and $\mathbf{c}_i \in \mathbb{R}^{2\times 1}$ to give

$$\begin{bmatrix} \mathbf{A}_{11}\mathbf{c}_1 + \mathbf{A}_{12}\mathbf{c}_2 \\ \mathbf{A}_{21}\mathbf{c}_1 + (\mathbf{A}_{22} + \mathbf{B}_{22})\,\mathbf{c}_2 + \mathbf{B}_{23}\mathbf{c}_3 \\ \mathbf{B}_{32}\mathbf{c}_2 + \mathbf{B}_{33}\mathbf{c}_3 \end{bmatrix} = \mathbb{H}\mathbf{c} = \mathbf{c}'.$$

where the vector $\mathbf{c}$ is synonymous with the variable defined in Line (5) of the 'modified-pcg' Algorithm of Baraff and Witkin [1998]. The global solution $\mathbf{c}'$ can be decomposed into local solution vectors

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11}\mathbf{c}_1 + \mathbf{A}_{12}\mathbf{c}_2 \\ \mathbf{A}_{21}\mathbf{c}_1 + \mathbf{A}_{22}\mathbf{c}_2 \end{bmatrix} = \mathbf{c}'_{\mathbf{a}}, \in \mathbb{R}^{2\times 1} \tag{24}$$

and

$$\begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{c}_2 \\ \mathbf{c}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{11}\mathbf{c}_2 + \mathbf{B}_{12}\mathbf{c}_3 \\ \mathbf{B}_{21}\mathbf{c}_2 + \mathbf{B}_{22}\mathbf{c}_3 \end{bmatrix} = \mathbf{c}'_{\mathbf{b}} \in \mathbb{R}^{2\times 1}. \tag{25}$$

[3]Note that the same analogy carries through when dealing with contact using our IPC formulation: a collision between two surface boundary elements (*e.g.* a point and an triangle) will form simplex from which we construct our barrier function, its gradient and Hessian. A vertex will have a local index in the simplex and a global index in the respective mesh(es) in contact.

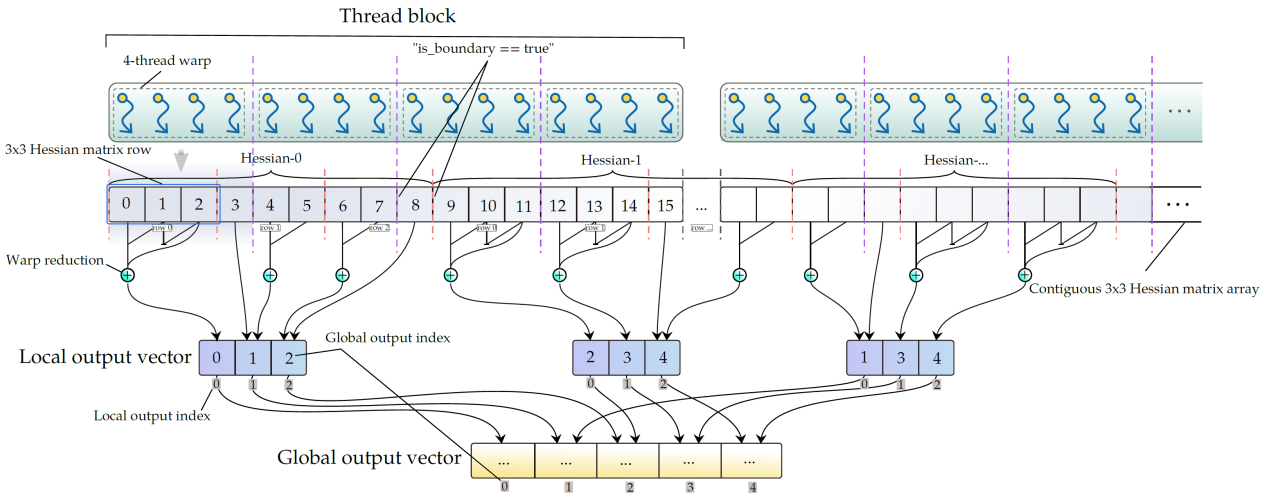


Fig. 5. *Warp reduction for parallel dense matrix-vector multiplication (see also Algo. (1)).*

to give $\mathbf{c}' = \begin{bmatrix} \mathbf{c}'_{a1} & \left(\mathbf{c}'_{a2} + \mathbf{c}'_{b1}\right) & \mathbf{c}'_{b2} \end{bmatrix}^T$ as the basis upon which we build our implementation of the local system matrix-vector multiplications during PCG. So if the local solution vectors $\mathbf{c}'_a$ and $\mathbf{c}'_b$ are computed independently, then it is possible to later merge results into the 'global' vector $\mathbf{c}'$. We use this property to solve the linear system $\mathbb{H}\mathbf{d} = \mathbf{g}$ without ever forming the global system $\mathbb{H}\mathbf{c}$ at each iteration of PCG.

Parallel local matrix-vector multiplication. We assign one thread per scalar entry in a local system matrix (*i.e.* in $\mathbf{A}$ or in $\mathbf{B}$), and apply a CUDA-warp level reduction scheme (Algo. (1)) to compute the components of the local solution vector (*i.e.* $\mathbf{c}'_a$, or $\mathbf{c}'_b$) in parallel. Briefly, threads in a warp are each mapped to one scalar entry in a local system matrix; the thread will then multiply the matrix entry to the corresponding component of the input vector $\mathbf{c}$. The multiplication result is then stored in private register memory, which (using CUDA builtins `clz`, `shfldown` *etc.*) is then accumulated within each warp before being added to the global output vector $\mathbf{c}'$.

However, the inconsistency between the CUDA thread block size (can only be 2^n, where n > 4) and the local Hessian dimension ($3n \times 3n$, where $n = 4, 3, 2$), as well as the warp size (32), unavoidably results in a challenge for executing the process of multiplying the matrix-vector elements and summing them in parallel. Ideally, the elements of the same row of Hessian are managed by the same thread warp, as then the summation of the matrix-vector elements can be done by warp reduction, which is significantly faster than summing the product individually by atomic operations. As shown in Fig. 5, the inconsistent sizes of the thread block/thread warp/matrix dimensions easily cause each row/matrix to be managed across different thread blocks/warps.

To maximize parallelism, we define a data structure that represents the size and range of reduction in each row of the matrix, which consists of the start position and the length of the reduction. For example, in Fig. 5, in the first row of Hessian-0, the start position is 0, and the length is 3, while in the second row the start position is 4 and the length is 2. We can maximize the parallelism and minimize the branching, such that approximately one thread per row (of any local system matrix) will actually commit data to the global memory (see Algo. (1) for the details).

The local system matrices from our IPC formulation will at-most have dimensions $\mathbb{R}^{12\times12}$ but allocating such space for all of our contact pairs (potentially millions) is wasteful since some pairs require much less memory (*e.g. point-point* with $\mathbb{R}^{6\times6}$) . Thus, we also arrange these matrices into four storage buffers based on the *type* of contact pairs to minimize storage costs and improve parallelism by grouping workloads with same Hessian dimensions. In addition, we also design the warp reduction scheme (Algo. (1)) to assume that all local system matrices assigned to each block/group of CUDA threads have the same local system matrix dimensions to reduce booking-keeping and minimize divergent execution flow.

7 SUPPLEMENTARY UNIT TESTS

This section provides supplementary results, which show additional units tests that where used to evaluate our implementation. These results are shown in Fig. 6 and Fig. 7. Readers are also referred to our supplementary video.

8 COMPARISON OF BARRIER METHODS ON CPU

Different aspects of implementation such as hardware choice (*e.g.* GPU or CPU) and the choice of linear solver will inevitably influence the performance enhancements observed in our barrier method. Thus, to validate the accuracy of the improvements described in the paper, we conduct additional evaluations of our barrier method within the CPU-IPC framework. This involved implementing our barrier method in CPU-IPC while using the same direct solver, CHOLMOD, to measure performance improvement. The purpose of this evaluation was to demonstrate improvements across various time steps and material stiffness parameter settings. The detailed experimental settings and the corresponding evaluation results are provided in Tab. 5, which shows the improvements of our method as described

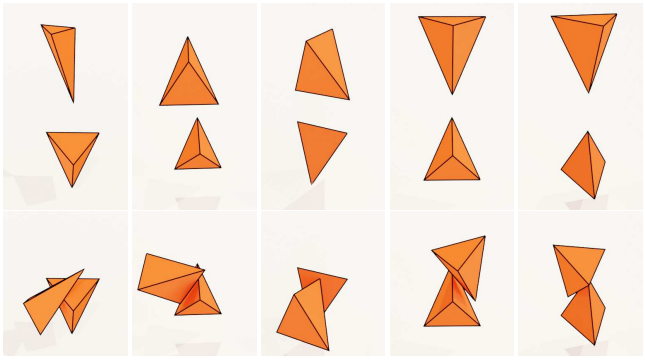


Fig. 6. *Unit tests: From left to right we have the point-triangle, edge-edge, parallel edge-edge, point-edge and point-point case. We robustly pass all these tests.*

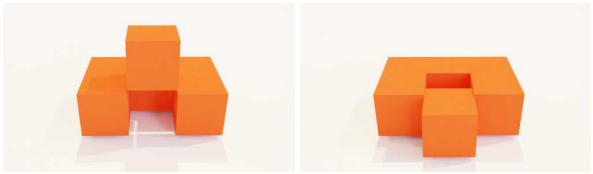


Fig. 7. *Aligned, close and nonsmooth contact test: conforming collisions are also accurately and stably resolved.*

in the paper (see column #i). Our method leads to a consistently lower number of Newton iterations on average. The advantage of our approach is most evident when simulating with highly stiff materials, reducing the number of iterations by up-to 3×.

Table 1. *Solution tolerance versus the number of Newton iterations, and the gradient-norm at convergence.*

	1e-2	1e-3	1e-4	1e-5	1e-6	1e-7
Iterations	3159	5772	9300	11123	14474	16631
Gradient-norm	3e−2	3e−3	3e−4	3e−5	3e−6	3e−7

Table 2. *Newton Iterations w.r.t the gradient-norm tolerance in standard CPU-IPC [Li et al. 2020].*

	3e−2	3e−3	3e−4	3e−5	3e−6	3e−7
Iterations	3582	6292	9765	11910	15105	16962

9 COMPARISON BETWEEN DIRECT SOLVER AND PCG

In this section, we present and discuss results based on an apple-to-apple comparison between the direct solver and PCG method in terms of Newton solver iterations, tested with the scene shown in Fig. (10 (a)) in the paper. This experiment is based on the setup

Table 3. *Newton iterations w.r.t PCG tolerance and Newton tolerance (based on gradient-norm). The data is plotted in Fig. 8.*

PCG Tolerance	Newton solver tolerance (gradient-norm)					
	3e−2	3e−3	3e−3	3e−5	3e−6	3e−7
1e−1	3981	7058	10418	13726	14970	19021
1e−2	3894	6585	10573	13257	15213	18534
1e−3	3698	6381	10340	12747	15915	18651
1e−4	3557	6116	10051	12163	14508	17367
1e−5	3846	6344	9896	12075	15320	17454
1e−6	3766	6288	9068	12695	14414	17491
1e−7	3666	6235	9921	12059	15120	17222

Table 4. *Newton Iterations w.r.t the gradient-norm tolerance, using our barrier method (with mollification) implemented the CPU-IPC framework.*

	3e−2	3e−3	3e−4	3e−5	3e−6	3e−7
Iterations	2322	5425	8280	11231	14407	16148

described in § 8, using our PCG method within the CPU-IPC framework [Li et al. 2020]. The objective is to illustrate that the PCG tolerance employed in our experiments (*i.e.* to give $\delta_{new} <$ 1e-4δ_0 as the termination condition of PCG), as mentioned in the paper, is sufficient for generating simulations of comparable accuracy to those achieved with direct solver in CPU-IPC.

In Tab. 1 and Tab. 2, we provide reference data showing the number of Newton iterations w.r.t tolerance, where this tolerance is defined based on the solution and the gradient, respectively. We obtain this data by running standard CPU-IPC [Li et al. 2020] with a direct solver (CHOLMOD) in two phases. In the first phase, we run with six distinct settings of the solution tolerance and record the number of Newton iterations and gradient-norm at convergence. We then repeat the simulation in the second phase using the recorded gradient-norm as the solver tolerance to analyse the number of iterations required to converge. The solution threshold with the highest accuracy in our experimental setup is approximately 1e−7 m/s. The gathered data in Tab. 1 and Tab. 2 is used as reference to compare against the results shown in Tab. 3. It is important to note that the convergence condition is determined by $\frac{\|\mathbf{d}\|^\infty}{l\Delta t} \leq \varepsilon_d$ for the solution tolerance and $\frac{\|\mathbf{g}\|^\infty}{l\Delta t^2} \leq \varepsilon_g$ for the gradient tolerance.

In Tab. 3, we provide the data we obtain by running standard CPU-IPC with PCG instead of a direct solver, which we use to demonstrate that CPU-IPC converges to the same accuracy with either of these two linear solvers. The gradient-norm is used for determining whether the Newton solver has converged in this experiment, which we do in order to ensure a fair comparison because using the solution for validation may be less reliable given that it varies with different linear solvers (potentially impacting convergence/Newton iterations). Crucially, the data of Tab. 3 and Fig. 8 provide further evidence that our choice of solution tolerance (1e−4) for PCG in

	v,t,f	ρ,E,υ	$\hat{d}$, ϵ_d	μ, ϵ_v	Δt, #Δt	buildCP		buildGH		solve		CCD		#i		misc		timeTot		**speedup**
						cpu-Li	cpu-Our	cpu-Li	cpu-Our	cpu-Li	cpu-Our	cpu-Li	cpu-Our	cpu-Li	cpu-Our	cpu-Li	cpu-Our	cpu-Li	cpu-Our	(*cpu-Our vs. cpu-Li*)
Fig. (10)(paper)	32k, 135k, 38k	1e3,1e4,0.49	1e-3, 1e-2	-	0.01, 95	6.33e2	4.39e2	2.97e2	1.95e2	3.39e3	2.68e3	5.41e2	3.75e2	33.2	24.4	1.93e1	1.61e1	4.88e3	3.71e3	**1.32**×
		1e3,1e5,0.49			0.01, 80	4.09e2	2.34e2	2.05e2	1.08e2	2.32e3	1.51e3	3.58e2	2.07e2	26.5	16.5	1.81e1	1.30e1	3.31e3	2.07e3	**1.60**×
		1e3,1e6,0.49			0.01, 100	4.66e2	2.27e2	2.39e2	1.01e2	2.74e3	1.42e3	4.04e2	2.27e2	24.8	12.7	2.14e1	1.25e1	3.87e3	1.99e3	**1.94**×
		1e3,1e7,0.49			0.01, 81	4.62e2	1.44e2	2.33e2	6.28e1	2.62e3	8.76e2	3.94e2	1.23e2	29.8	9.93	3.11e1	1.10e1	3.74e3	1.22e3	**3.06**×
Fig. (14)(paper)	8k, 36k, 10k(dolphin) 30k, -, 60k(funnel)	1e3,1e4,0.40	1e-3, 1e-2	-	0.01, 300	2.64e3	2.35e3	5.47e2	2.16e2	3.46e3	3.10e3	2.06e3	1.75e3	22.7	20.1	1.84e2	1.67e2	8.89e3	7.58e3	**1.15**×
					0.02, 150	1.62e3	1.25e3	4.27e2	1.52e2	2.55e3	1.92e3	1.36e3	1.03e3	29.6	23.2	1.03e2	8.68e1	6.06e3	4.45e3	**1.36**×
					0.03, 100	1.24e3	9.50e2	3.62e2	1.44e2	2.13e3	1.69e3	1.09e3	8.47e2	36.8	28.4	6.82e1	5.68e1	4.89e3	3.69e3	**1.32**×
					0.04, 75	9.69e2	7.18e2	3.28e2	1.16e2	1.72e3	1.29e3	8.93e2	6.96e2	38.4	28.6	5.38e1	4.41e1	3.97e3	2.87e3	**1.38**×
					0.05, 60	8.86e2	6.55e2	3.13e2	1.21e2	1.59e3	1.24e3	8.28e2	6.88e2	43.6	32.7	4.47e1	3.63e1	3.66e3	2.74e3	**1.33**×

Table 5. *Performance summary using comparison Li et al. [2020]. The columns are as follows: number of vertices, including the interior for tetrahedral meshes (*v*); number of tetrahedra (*t*); number of surface triangles (*f*); time step size in seconds (Δt); material density(ρ), Young's modulus (E) in units of pascals Pa, and Poisson's ratio (υ); computational accuracy target in meters ($\hat{d}$) which is set w.r.t. to the scene bounding box diagonal length l; Newton Solver tolerance threshold (ϵ_d); friction coefficient (μ) and velocity magnitude bound (ϵ_v);Total number of time steps (#Δt); Total time to build/find contact pairs (*buildCP*); Total time to build energy gradients and Hessians for all types (*buildGH*); Total linear solver time (*solve*); Total CCD time (*CCD*); Average number of Newton iterations per time step (*#i*); Total time for remaining miscellaneous tasks (*misc*); Total simulation compute time (*timeTot*); We replace the barrier method of CPU-IPC (Li et al. [2020]) with ours to estimate the speedup of our method. All time measurements are presented in seconds.*

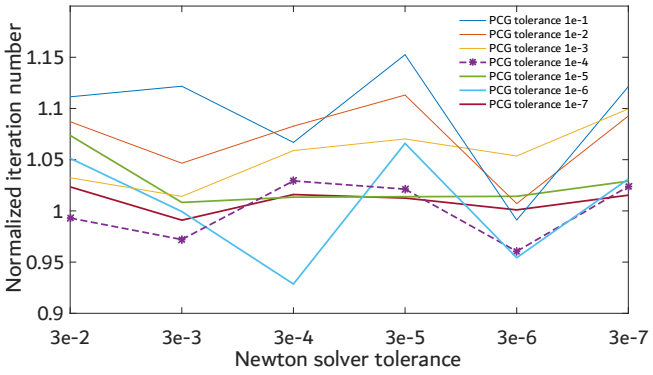


Fig. 8. *Normalized plot of the rows in Tab. 3: Newton-iteration number when using PCG w.r.t tolerance. We normalize by the corresponding iteration number for the direct solver (*cf. *Tab. 2). The figure demonstrates that the difference in Newton-iteration number between using PCG and the direct solver is typically within a margin of approximately* 5% *when PCG tolerance is smaller than* 1e−4. *Using a higher accuracy tolerance for PCG does not necessarily lower the Newton-iteration count, as locally accurate solutions (*i.e.*per Newton iteration) do not necessarily translate into a globally optimal convergence rate. Our data further indicates that even with a relatively low-accuracy PCG tolerance, the Newton solver can still converge. This follows the fact that PCG provides an optimal search direction and step size within the current Krylov subspace, which corresponds to the direction of energy descent in that subspace, even with a relatively low-accuracy tolerance.*

the paper yields an equivalent number of Newton iterations when compared to higher-accuracy thresholds.

In order to demonstrate that our barrier method, using the PCG solver with 1e−4 tolerance, maintains the accuracy of the IPC method, we conducted additional tests. We replaced the barrier method with our approach and continued to use the gradient-norm threshold to assess the convergence capability of our method across different simulation accuracies. The results, presented in Tab. 4, indicate that our barrier method is capable of performing high-accuracy simulations as well.

10 IMPACT OF FRICTION HESSIAN PROJECTION

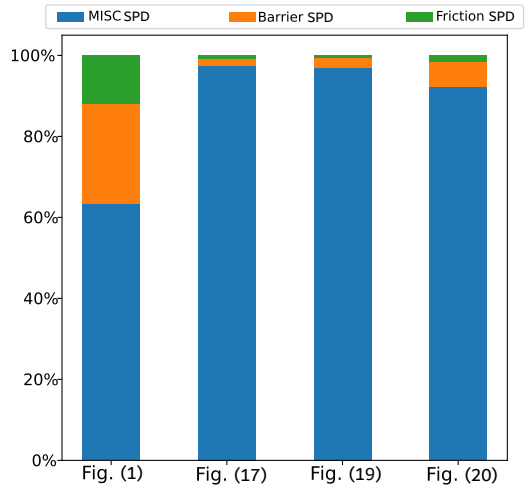


Fig. 9. *Percentage of time on Hessian projection: Time breakdown considering the local Hessian projections in our simulator, which include those of the elastic energy. Here we particularly focus on projections for friction- and barrier Hessians. The x-axis refers to the figures/demos provided in the paper, which are the examples involving friction (full simulation).*

Fig. 9 provides a comprehensive breakdown of compute time between local friction- and barrier Hessian projection alongside the remaining miscellaneous local projections that take place in our simulator (e.g. elastic energy Hessian). The results show that projection of the friction hessian has negligible overhead when compared to our approximated analytic barrier Hessian projection. We have found friction Hessian projection to merely require one-third to one-half the time needed to compute barrier Hessian projection.

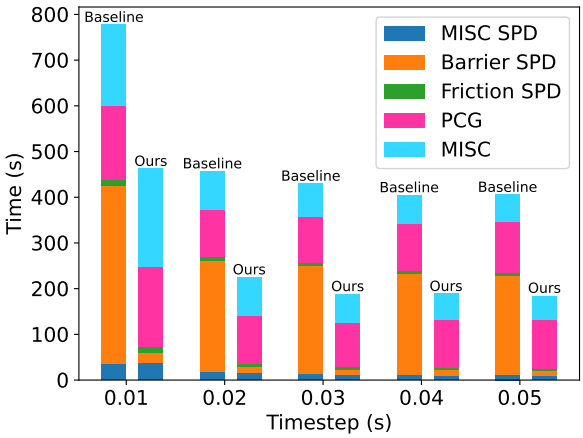


Fig. 10. *By employing the Funnel test showcased in Fig. (14) and conducting the comparison illustrated in Fig. (9) of the main paper with friction added, we include a time breakdown of the friction- and barrier Hessian projections, along with the remaining miscellaneous components of our simulator.*

To provide a deeper understanding of the role friction plays within large-scale contact simulations, we also run a duplicate experiment of the demo shown in Fig. (14) of the paper, setting the friction coefficient to $1e{-}2$ (*i.e.* μ parameter in Tab. 5 or Tab. (5) in the paper). A breakdown w.r.t total simulation time is presented in Fig. 10, where it can be observed that the time expenditure associated with friction Hessian projection has overall minimal footprint.

REFERENCES

David Baraff and Andrew P. Witkin. 1998. Large Steps in Cloth Simulation. In *Proceedings of SIGGRAPH 1998*, Steve Cunningham, Walt Bransford, and Michael F. Cohen (Eds.). ACM, 43–54.

Ming Gao, Xinlei Wang, Kui Wu, Andre Pradhana, Eftychios Sifakis, Cem Yuksel, and Chenfanfu Jiang. 2018. GPU Optimization of Material Point Methods. 37, 6 (2018).

C. Kane, E.A. Repetto, M. Ortiz, and J.E. Marsden. 1999. Finite element analysis of nonsmooth contact. *Computer Methods in Applied Mechanics and Engineering* 180, 1 (1999), 1–26.

Theodore Kim, Fernando De Goes, and Hayley Iben. 2019. Anisotropic Elasticity for Inversion-Safety and Element Rehabilitation. *ACM Trans. Graph.* 38, 4, Article 69 (jul 2019), 15 pages.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental Potential Contact: Intersection-and Inversion-Free, Large-Deformation Dynamics. *ACM Trans. Graph.* 39, 4, Article 49 (2020).

Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. 2015. Air meshes for robust collision handling. *ACM Trans. Graph.* 34, 4 (2015), 1–9.

Richard M. Murray, S. Shankar Sastry, and Li Zexiang. 1994. *A Mathematical Introduction to Robotic Manipulation* (1st ed.). CRC Press, Inc., USA.

Jonathan R Shewchuk. 1994. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.* Technical Report. USA.

Eftychios Sifakis and Jernej Barbic. 2012. FEM Simulation of 3D Deformable Solids: A Practitioner’s Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses*. Article 20, 50 pages.

Breannan Smith, Fernando De Goes, and Theodore Kim. 2019. Analytic Eigensystems for Isotropic Distortion Energies. *ACM Trans. Graph.* 38, 1, Article 3 (feb 2019), 15 pages.

Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed Aggregation Multigrid for Cloth Simulation. *ACM Trans. Graph.* 34, 6, Article 245 (oct 2015), 13 pages.

Xinlei Wang, Minchen Li, Yu Fang, Xinxin Zhang, Ming Gao, Min Tang, Danny M. Kaufman, and Chenfanfu Jiang. 2020. Hierarchical Optimization Time Integration for CFL-Rate MPM Stepping. *ACM Trans. Graph.* 39, 3, Article 21 (apr 2020), 16 pages.

Zipeng Zhao, Kemeng Huang, Chen Li, Changbo Wang, and Hong Qin. 2020. A Novel Plastic Phase-Field Method for Ductile Fracture with GPU Optimization. *Comput. Graph. Forum* 39, 7 (2020), 105–117.