

Smoothly Differentiable and Efficiently Vectorizable Contact Manifold Generation

Onur Beker¹, A. René Geist¹, Anselm Paulus¹, Nico Gürtler¹, Ji Shi¹, Sylvain Calinon^{2,3}, Georg Martius¹
¹University of Tübingen ²Idiap Research Institute ³EPFL

Abstract—Simulating rigid-body dynamics with contact in a fast, massively vectorizable, and smoothly differentiable manner is highly desirable in robotics. An important bottleneck faced by existing differentiable simulation frameworks is contact manifold generation: representing the volume of intersection between two colliding geometries via a discrete set of properly distributed contact points. A major factor contributing to this bottleneck is that the related routines of commonly used robotics simulators were not designed with vectorization and differentiability as a primary concern, and thus rely on logic and control flow that hinder these goals. We instead propose a framework designed from the ground up with these goals in mind, by trying to strike a middle ground between: i) convex primitive based approaches used by common robotics simulators (efficient but not differentiable), and ii) mollified vertex-face and edge-edge unsigned distance-based approaches used by barrier methods (differentiable but inefficient). Concretely, we propose: i) a representative set of smooth analytical signed distance primitives to implement vertex-face collisions, and ii) a novel differentiable edge-edge collision routine that can provide signed distances and signed contact normals. The proposed framework is evaluated via a set of didactic experiments and benchmarked against the collision detection routine of the well-established Mujoco XLA framework, where we observe a significant speedup. Supplementary videos can be found at <https://github.com/bekeronur/contax>, where a reference implementation in JAX will also be made available at the conclusion of the review process.

I. INTRODUCTION

What characteristics define a good physics simulator for robotics? It is appropriate to split such desiderata into two groups based on their underlying motivation: “physics for prediction”, and “physics for control”. When simulating physics for prediction, there already exists an intelligent specification (e.g., a design or a behavior) driving the simulation, and one mainly wants to observe an approximation of its consequences in a way that achieves a notion and degree of realism dictated by the downstream application. For example, in the context of physically-based animation [56, 3], it is the animator who sets up a scene or a character and specifies its behavior. In the context of computer-aided engineering software [13], it is the responsibility of a mechanical engineer to specify an intelligent design to be tested. In the context of physics engines for computer games [21], the burden of intelligent conduct falls on the player. When simulating physics for control, however, the desired consequences are determined first, and one instead wants to infer an intelligent specification (e.g., system parameters to be identified or actuator control signals to be commanded) that will drive the simulation to incite said consequences.

While physics for prediction purposes should be formulated

with realism as the primary goal, formulations of physics for control purposes should instead actively try to identify the right simplifications to reality in order to reduce the complexity of searching for the desired intelligent specification. Particularly within the context of differentiable rigid body dynamics simulation with contact, this involves making the appropriate relaxations to the discontinuous reality of contact interactions in order to create a well-behaved optimization landscape. But what are these appropriate relaxations, and where should they happen in the overall simulation process?

The numerous computations that make up any typical rigid-body dynamics simulation routine can be grouped into two main subroutines [56, 3]: i) *collision detection* to obtain geometric measurements about relative configurations of the simulated surfaces that is useful for transcribing contact constraints, and ii) *contact dynamics* to timestep the simulation in a way that respects said contact constraints. Naturally, relaxations have to be made on both of these fronts to ensure the differentiability of the whole simulation process. This paper focuses on the collision detection part of the overall problem.

Looking at the collision detection routines of commonly used simulation frameworks in robotics [49, 34, 14, 47], one can observe that most of the underlying algorithms [44, 17, 52, 20] were conceived from a physics for prediction perspective (often within a computer graphics, video game design, or computational science and engineering context), and are optimized for speed and minimal memory footprint. This creates complications when they are employed for generating intelligent robot behavior. For example, such algorithms often involve significant branching and control flow, which result in pathological search landscapes with many discontinuities that throw off numerical optimization algorithms [33]. Similarly, they often tend to contain logic that imposes restrictions for vectorized simulation and domain randomization of simulation geometry, which are crucial in robotics applications such as sampling-based model predictive control [24, 28] or sim-to-real transfer of reinforcement learning policies [2].

Collision detection routines of the state-of-the-art simulation approaches in computer graphics, such as barrier-based methods [29], pursue a different approach. These methods operate by explicitly considering contacts between vertex-face (V-F) and edge-edge (E-E) pairs in a smoothly differentiable manner, with the downside of being slower and less memory efficient. They also only generate unsigned penetration metrics and unsigned contact normals, whereas complementary-based contact dynamics formulations [5, 4, 48, 11] commonly used

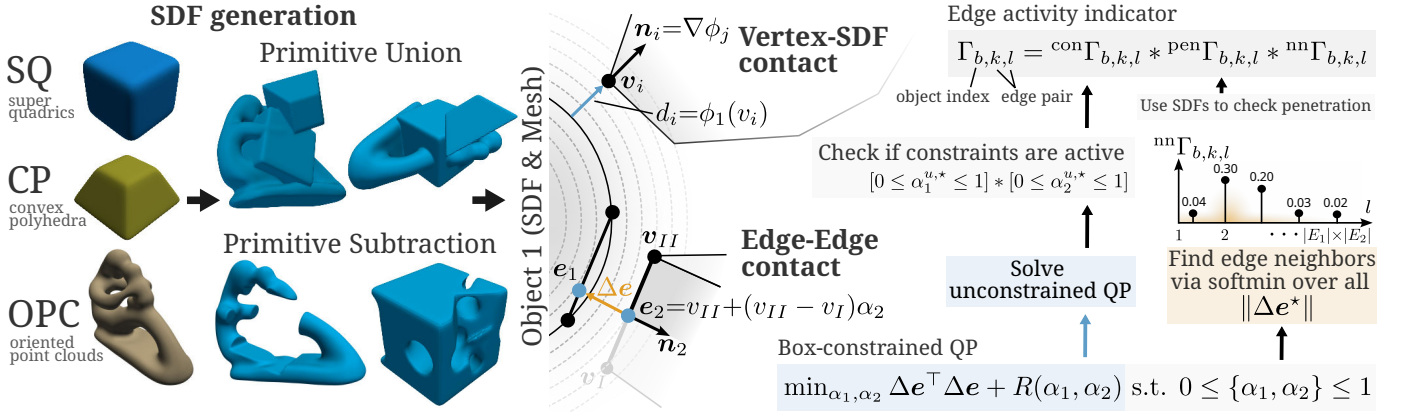


Fig. 1: We introduce a contact manifold generation framework that is smoothly differentiable, fast, and well-suited for vectorization. This is enabled by two contributions. The first is a pipeline for constructing signed distance fields (SDF) for arbitrary surfaces from a representative set of analytical primitives. These SDFs then serve two roles: (i) acting as a differentiable acceleration structure to efficiently select vertex–face (V–F) and edge–edge (E–E) contact candidates, and (ii) providing signed distances, normals, and activity indicators for V–F and E–E interactions in a way that retains curvature information (since by default, they happen between flat, planar facets). The second contribution is a method to make witness point computations for V–F and E–E pairs smoothly differentiable by solving the associated quadratic programs via an analytical active-set approach.

in robotics explicitly require signed versions.

The main contribution of this paper is to adapt the collision detection concepts of barrier-based methods for robotics by proposing a novel contact manifold generation method that: i) is smoothly differentiable to facilitate optimization, ii) is fast and memory efficient to facilitate vectorization, iii) generates signed penetration metrics and contact normals to be compatible with the complementarity-based methods for contact dynamics.

II. RELATED WORK

The standard process employed by common simulators used in robotics [49, 47, 14] to generate a contact manifold (i.e., a properly distributed set of contact points) between two arbitrary surfaces can be abstracted into the following steps [16, 20]:

- (i) Converting each non-convex surface into a mesh with well-distributed facets via a computational geometry algorithm [26, 12, 35], and decomposing it into a set of convex meshes [53]. All of the remaining steps operate on pairs of these convex meshes.
- (ii) Running standard collision detection routines like GJK+EPA [17, 31, 52] or SAT [19] to obtain information such as a pair of “witness points” (the end-points of the largest penetration distance) or closest mesh facets.
- (iii) Building a contact manifold that sufficiently represents the entire volume/area of intersection between surfaces.

Alternative ways of implementing the last step are [20]:

- Incrementally building it across simulation time-steps by maintaining a buffer of witness points based on distance heuristics about when to add or remove from the buffer.
- Building a contact manifold from scratch every time-step (i.e. “one-shot”) by: i) running GJK+EPA to identify the penetration vector and tangential axes to it, and ii) inducing small rotations on the two geometries multiple times around the tangential axes and re-running GJK+EPA each time.

- Building a contact manifold one-shot by applying polygon clipping algorithms [44] to the closest mesh-facets.

As expected, these routines involve discontinuities due to significant branching, which means simply applying automatic differentiation to them results in uninformative gradients, and it is necessary to devise further smoothing schemes. The work by Montaut et al. [32] proposes a modification to the standard GJK+EPA routine that uses randomized smoothing to obtain useful gradients via the score function estimator [55]. The main downsides of this approach are that: i) it is sampling-based (and hence inefficient), ii) it is not entirely transparent how exactly the parameters of the sampling distribution alter the overall dynamics, iii) it only addresses witness point computation. The work of Tracy et al. [51] in turn takes a different approach and solves for the growth distance [36] between pairs of a representative set of convex primitives (including convex meshes) by formulating them as cone-constrained convex programs. Gradients through this convex program are then obtained via implicit differentiation and smoothed via modulating the log-barrier coefficient of the underlying interior-point solver [50]. Its main downsides are: i) it does not scale (in terms of speed and memory footprint) to meshes with a large number of faces, and ii) it again only addresses witness point computation. In contrast, this paper proposes a deterministic method for generating a complete contact manifold that scales to complex meshes, while ensuring that each smoothing parameter influences the dynamics through a closed-form expression, thereby making its impact explicit.

Barrier-based methods for contact simulation [29] take a different approach from the previously described standard routines. Instead of producing a contact manifold consisting of a relatively small number of points with signed distances and normals, they solely compute unsigned distances between V–F and E–E pairs. These are then used within a log-barrier term that enforces non-penetration constraints, added to the

usual optimization objective for the implicit time integration of unconstrained rigid-body systems. While technically differentiable, there are two aspects of this approach incompatible with common simulation routines employed in robotics: i) they do not produce signed distances and normals, ii) they need acceleration structures (e.g. kd-trees or bounding volume hierarchies) to avoid the $O(|V| * |F| + |E| * |E|)$ cost for all V-F and E-E pair checks, which hinder the vectorization and JIT compilation utilities of common array computing frameworks [8]. This paper aims to address these bottlenecks.

It is also beneficial to examine the collision detection routines of existing differentiable simulators. The Nimble simulator [54] implements a custom mesh-mesh collision routine that computes gradients analytically via a branching algorithm in the backward pass, which introduces discontinuities [46]. The Dojo framework [25] in turn only supports collisions between a sphere and a plane/sphere/capsule/box [45], which can be resolved via a single contact point and differentiated analytically. The MJX framework [18] supports mesh-mesh collisions via the following routine [20]: i) run SAT to identify edges and faces closest to contact, ii) apply a clipping algorithm to build a polygonal contact manifold, iii) use a heuristic to select the four points on the polygon with approximately maximal area. While the implementation replaces all branching control flow with equivalent branchless functionality provided by the `jax.lax` module (to enable vectorization and JIT compilation), the discontinuities remain. The DiffMJX simulator [37] in turn implements a version of the MJX collision routine with smoothed gradients via the `softjax` library [38], but the smooth selection operations that enable this also cause a significant slowdown for gradient computation. These observations suggest that contact manifold construction presents a significant bottleneck in the differentiable simulation literature, a problem that this paper addresses.

III. BACKGROUND: SMOOTH APPROXIMATIONS TO NON-SMOOTH OPERATORS

The proposed framework makes extensive use of existing literature around smooth approximations of operators such as comparisons, clipping, and top-K selection. The goal of this section is to provide a compressed summary of the related background and notation for ease of reference. While more detailed surveys can be referred to if needed [38], the main point to emphasize is that whenever any non-smooth operation of a type covered in this section is employed later in the exposition, a smooth and differentiable counterpart exists (and is utilized by the software implementation). Throughout the section, τ denotes a positive smoothing factor (and approximations get more accurate but less smooth as $\tau \rightarrow 0$). The sigmoid function $\sigma(x) = (1 + \exp(-x))^{-1}$ can be used to smoothly approximate a comparison operator $\llbracket x > a \rrbracket : \mathbb{R} \rightarrow [0, 1]$ (using Iverson-Bracket) via $\sigma((x - a)/\tau)$. The hyperbolic tangent function $\tanh(x) = \sigma(2x) - 1$ can in turn be used to approximate the sign operator as $\tanh(x/\tau)$. The $\text{relu}(x)$ function can be smoothly approximated using a softplus function $s_+(x) = \tau \log(1 + \exp(x/\tau))$. The clipping operation $f_{\text{clip}}(x, x_{\min}, x_{\max}) = x_{\min} +$

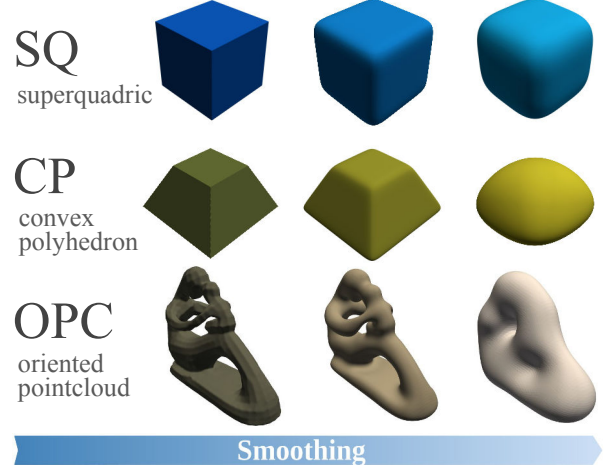


Fig. 2: Visualizations of how the proposed SDF primitives behave with increasing levels of smoothing.

$\text{relu}(x - x_{\min}) - \text{relu}(x - x_{\max})$ can in turn be softened into a softclip operation $s_{\text{clip}}(x, x_{\min}, x_{\max})$ by replacing relu operations with s_+ . The scalar sigmoid and softplus functions can be generalized to operate on vectors $\mathbf{x} \in \mathbb{R}^D$ to obtain the softmax and realsoftmax (i.e. logsumexp) functions respectively, notated as $s_{\text{argmax}}(\mathbf{x})_i = \exp(x_i/\tau) / \sum_{j=1}^D \exp(x_j/\tau)$ and $\text{LSE}(\mathbf{x}) = \tau \log[\sum_{i=1}^D \exp(x_i/\tau)]$. Note that $s_{\text{argmax}}(\mathbf{x})$ is a vector of probability masses that approximates an argmax operation, whereas $\text{LSE}(\mathbf{x})$ is a scalar that corresponds to a soft selection of the maximum component of \mathbf{x} . For any vector $\mathbf{x} \in \mathbb{R}^D$, the top-K operator can be smoothed to obtain a matrix-valued soft top-K operation $\Gamma_K(\mathbf{x}) \in [0, 1]^{K \times D}$, where the entry $\Gamma_K(\mathbf{x})_{ki} \approx 1$ if x_i is the k -th largest entry of \mathbf{x} , and $\Gamma_K(\mathbf{x})_{ki} \approx 0$ otherwise. There exists a variety of such soft top-K operators [39, 57, 22, 38], and this paper employs [39]. The matrix $\Gamma_K(\mathbf{x})$ is a probabilistic representation of the *indices* that correspond to the top-K entries, and these entries *themselves* can be selected simply via a matrix product $\Gamma_K(\mathbf{x})\mathbf{x} \in \mathbb{R}^K$.

IV. METHODS

A. A Birds-Eye View of the Most Essential Steps

The presentation of the method follows a top-down exposition, with this section providing an overview of how all contributions fit together and the subsequent sections elaborating on each. Let $(\mathcal{S}_1, \mathcal{S}_2)$ be two (not necessarily convex) closed surfaces in 3D, with $SE(3)$ poses $\mathbf{T}_1, \mathbf{T}_2 \in \mathbb{R}^{4 \times 4}$ that can be smoothly parametrized via 6D $SE(3)$ vectors $\mathbf{s}_1 = \log(\mathbf{T}_1)$, $\mathbf{s}_2 = \log(\mathbf{T}_2)$. The goal of contact manifold generation is to implement a function $g_{1,2} : \mathbb{R}^{6+6} \rightarrow \mathbb{R}^{|K| \times (3+1+3)}$; $g_{1,2}(\mathbf{s}_1, \mathbf{s}_2) = \{(\mathbf{p}_k, d_k, \mathbf{n}_k)\}_{k=1}^{|K|}$ that outputs $|K|$ contact points $\mathbf{p}_k \in \mathbb{R}^3$, signed distances $d_k \in \mathbb{R}$, and contact normals $\mathbf{n}_k \in \mathbb{R}^3$. The task at hand is to define $g_{1,2}$ such that it is smoothly differentiable and efficiently vectorizable. The

proposed framework achieves this as follows:¹

- Every surface has two representations associated with it: i) a mesh (with vertices $V = \{\mathbf{v}_i \in \mathbb{R}^3\}$ and edges $E = \{\mathbf{e}_j \in \mathbb{Z}^2\}$), and ii) a smoothly differentiable SDF $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}$.
- All vertices (on both surfaces) are assigned penetration metrics using the SDF of the opposing surface. The proposed framework provides methods to build such SDFs efficiently from a set of analytical primitives L (where $|L|$ is much smaller than $|V|$ and $|F|$), making this a relatively cheap $O(|V| \times |L|)$ operation. All edges are then assigned penetration metrics based on a weighted sum of the penetration metrics of their corner vertices, which is again relatively cheap ($O(|E| \times 2)$). Then $(|V_1|, |V_2|)$ vertices and $(|E_1|, |E_2|)$ edges on each surface that are in the deepest penetration are selected via soft top-K operations. The numbers $|V_b|$ and $|E_b|$ for $b \in \{1, 2\}$ are set as part of the simulation description and kept fixed throughout the lifetime of the simulation. This design choice, same as MJX [18], is because JIT compilation and vectorization require the number of contact points to remain constant. This step essentially implements a differentiable acceleration structure.
- As discussed in Li et al. [29], considering V-F and E-E contact types is sufficient to transcribe constraints that prevent penetrations between two surfaces in arbitrary relative poses. In the proposed framework, V-F contacts are replaced with vertex-SDF (V-S) contacts for efficiency. For each of the $|E_1| \times |E_2|$ edge-edge pairs, two contact points (one on each edge) are generated together with the associated d_k and \mathbf{n}_k (as well as a scalar “activity indicator” γ that will be introduced later) via a novel E-E contact routine.

B. Building Smooth SDF Approximations of Arbitrary Surfaces for Implementing Vertex-SDF Contacts

As shown in Fig. 1 and 2, smoothly differentiable SDFs are built from combining analytical primitives $\phi_l(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}$ via smooth union and subtraction operations [41, 42] implemented using the logsumexp (real-softmax) function. In particular:

- The smooth union of $|L|$ primitives $\{\phi_l(\mathbf{x})\}_{l=1}^{|L|}$ is given by $\phi(\mathbf{x}) = -\text{LSE}([-\phi_1(\mathbf{x}), \dots, -\phi_{|L|}(\mathbf{x})])$.
- The smooth subtraction of $\phi_-(\mathbf{x})$ from $\phi_+(\mathbf{x})$ is given by $\phi(\mathbf{x}) = \text{LSE}(\phi_+(\mathbf{x}), -\phi_-(\mathbf{x}))$.

This approach of fusing analytical primitives to obtain implicit surface descriptions has long been used by the computer graphics community to model complex surfaces in great detail [40, 43]. As one cannot simply code new implicit surface descriptions from scratch for every new robotics simulation, the proposed framework provides three families of primitives. These are (in the order of most to least efficient in terms of memory and compute requirements): i) superquadrics, ii) convex polyhedra, and iii) oriented pointclouds.

¹Since forward kinematics is already smoothly differentiable and is not the focus of this paper, the rest of the exposition omits the dependencies on $\mathbf{s}_1, \mathbf{s}_2$ for brevity and all quantities will be assumed to be expressed in world coordinates after forward kinematics.

1) *Superquadric (SQ) Primitives*: An SQ in its canonical form (i.e. centered at the origin, axis-aligned, with unit scaling) is defined by an *inside–outside* implicit function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ such that the surface corresponds to the unit level set $f(\mathbf{x}) = 1$ (exterior: $f(\mathbf{x}) > 1$). The standard definition of f and the corresponding approximate signed distance ϕ are [30]:

$$\mathbf{x} = [x, y, z] \quad (1)$$

$$f(\mathbf{x}) = \left[(x^2)^{1/\varepsilon_2} + (y^2)^{1/\varepsilon_2} \right]^{\varepsilon_2/\varepsilon_1} + (z^2)^{1/\varepsilon_1} \quad (2)$$

$$\phi(\mathbf{x}) = (1 - f(\mathbf{x})^{-\varepsilon_1/2}) / \|\mathbf{x}\|, \quad (3)$$

where $\varepsilon_{\{1,2\}}$ control the “boxiness”. Shape scaling can be obtained by normalizing the coordinates $\tilde{\mathbf{x}} = [x/a, y/b, z/c]$ with axis lengths (a, b, c) . Even though $\phi(\mathbf{x}) = 0$ holds on the SQ surface and $\nabla\phi(\mathbf{x})$ points in the correct direction everywhere, the approximation does not satisfy the Eikonal property $|\nabla\phi(\mathbf{x})| = 1$ far away from the SQ surface, meaning the value $\phi(\mathbf{x})$ represents a metrically accurate distance only near the SQ surface. This does not pose a problem for simulation for two reasons: i) contact points always remain very close to the SQ surface (since contact forces prevent further penetration), ii) the complementarity-based formulations of contact dynamics can accommodate nonmetrical d_i as long as its zero crossing happens on the surface (since the non-metric scaling simply acts as a virtual contact impedance). Signed distances and normals are therefore obtained from $d_i = \phi(\mathbf{x})$ and $\mathbf{n}_i = \nabla f(\mathbf{x}) / \sqrt{\tau + \|\nabla f(\mathbf{x})\|^2}$.

2) *Convex Polyhedron (CP) Primitives*: A convex polyhedron is a collection of planes $\{(\mathbf{n}_i, \mathbf{p}_i)\}_{i=1}^N$, where \mathbf{n}_i and \mathbf{p}_i denote the normal of the plane and a point on it. The exact SDF and its continuously differentiable approximation are [58]:

$$\phi_{\text{exact}}(\mathbf{x}) = \max_i \mathbf{n}_i^\top (\mathbf{x} - \mathbf{p}_i) \quad (4)$$

$$\phi(\mathbf{x}) = \text{LSE}([\mathbf{n}_1^\top (\mathbf{x} - \mathbf{p}_1), \dots, \mathbf{n}_N^\top (\mathbf{x} - \mathbf{p}_N)]) \quad (5)$$

Contact normals are then obtained as $\nabla\phi(\mathbf{x}) / \sqrt{\tau + \|\nabla\phi(\mathbf{x})\|^2}$.

3) *Oriented Pointcloud (OPC) Primitives*: If one has access to an oriented pointcloud $\{(\mathbf{p}_i, \mathbf{n}_i)\}_{i=1}^N$ on any non-convex surface, a radial basis interpolation of the following form can be used to obtain a smooth SDF approximation [6]:

$$\phi(\mathbf{x}) = \frac{\sum_{i=1}^N \mathbf{B}_i(\mathbf{x} - \mathbf{p}_i; \boldsymbol{\theta}_i) \mathbf{n}_i^\top (\mathbf{x} - \mathbf{p}_i)}{\sum_i \mathbf{B}_i(\mathbf{x} - \mathbf{p}_i; \boldsymbol{\theta}_i)} \quad (6)$$

Contact normals are again obtained as $\nabla\phi(\mathbf{x}) / \sqrt{\tau + \|\nabla\phi(\mathbf{x})\|^2}$.

In general, SQ primitives are very compute- and memory-efficient and can model a wide-range of complex geometries [1]. Therefore, the default option in the proposed framework is to use the algorithm of Liu et al. [30] to decompose collision geometries into SQ primitives. CP primitives are less efficient, but can model sharp corners better and can generally be added to the decomposition whenever an SQ primitive would be insufficient to model a feature. OPC primitives are the most expressive and can model arbitrary nonconvex surfaces, but are also the least efficient. They should only be used when a feature to be modeled is inherently non-convex, and SQ or CP primitives would be insufficient.

C. Generating Vertex-SDF (V-S) Contacts

For every selected vertex \mathbf{v}_i , a V-S contact is generated by utilizing the smooth SDF approximation ϕ of the opposing surface to compute the associated signed distance $d_i = \phi(\mathbf{v}_i)$ and contact normal $\mathbf{n}_i = \nabla\phi(\mathbf{v}_i)$.² Since ϕ is smoothly differentiable, so are d_i and \mathbf{n}_i .

D. Generating Edge-Edge (E-E) Contacts

1) *Witness Point Computation (General Case)*: Points \mathbf{p} on a convex polygon with N corners $\mathbf{v}_1, \dots, \mathbf{v}_N$ can be parameterized with a linear term $\mathbf{p}(\boldsymbol{\alpha}) = \sum_i \mathbf{v}_i \alpha_i$ and an equality constraint $\sum_i \alpha_i = 1$. Therefore witness points $\mathbf{p}_1(\boldsymbol{\alpha}_1^*), \mathbf{p}_2(\boldsymbol{\alpha}_2^*)$ between two arbitrary polygons (1,2) can be found via a QP:

$$\min_{\boldsymbol{\alpha}_1 \in \mathbb{R}^{N_1}, \boldsymbol{\alpha}_2 \in \mathbb{R}^{N_2}} (\mathbf{p}_1(\boldsymbol{\alpha}_1) - \mathbf{p}_2(\boldsymbol{\alpha}_2))^\top (\mathbf{p}_1(\boldsymbol{\alpha}_1) - \mathbf{p}_2(\boldsymbol{\alpha}_2)) \quad (7)$$

$$\text{s.t.} \quad \sum_{1 \leq i \leq N_1} \alpha_{1,i} = 1, \boldsymbol{\alpha}_1 \succeq \mathbf{0} \quad (8)$$

$$\sum_{1 \leq i \leq N_2} \alpha_{2,i} = 1, \boldsymbol{\alpha}_2 \succeq \mathbf{0}. \quad (9)$$

Since mesh primitives such as vertices ($N=1$), edges ($N=2$), and triangles ($N=3$) are all convex polygons, witness point computations between them can all be solved as QPs.

Jumps in the solution of such a QP can happen due to two reasons: i) the quadratic term loses rank (i.e., becomes only positive semi-definite) and there are infinitely many solutions, or ii) inequality constraints are switching on and off. The most common way to smooth (i) is to add L_2 (i.e., Tikhonov) regularization [33], and (ii) is to use an interior point method and increase the weight of the log barrier function [50]. The problem with this log barrier approach is that it involves an iterative numerical scheme (e.g., primal-dual Newton iterations on the KKT system) to find the *exact* minimum on the log barrier, which is an overkill for small QPs (e.g., E-E and V-F collisions), considering that the non-smoothed versions can be solved via active set methods [33] much faster and with a fraction of the effort. Moreover, such iterative procedures are inefficient for vectorization, because a vectorized iterative solver applied to a batch of QPs will only finish as quickly as the slowest-converging QP in the batch. Therefore, for the particular class of QPs we want to solve (i.e. witness point computations between V-E-F primitives), it is more promising to search for ways to alter active set methods to make them smoothly differentiable and efficiently vectorizable.

The main bottleneck with employing the active set method is that it involves: i) branching control flow (i.e., when identifying which subset of the constraints are active), and ii) non-smooth logic operators. The proposed framework overcomes these by implementing smoothed variants of the following most trivial, analytical (i.e., non-iterative), branchless active set method:

- (i) Analytically compute the unconstrained minimizer \mathbf{x}^{u*} .
- (ii) Analytically compute the constrained minimizers $\{\mathbf{x}_i^{c*}\}_{i=1}^{2^N}$ for all 2^N possible ways of turning the N

inequality constraints $g(\mathbf{x})_i \geq 0$ into equality constraints by switching them on/off, and pick the member from $\{\mathbf{x}_i^{c*}\}_{i=1}^{2^N}$ that minimizes the objective as \mathbf{x}^{c*} .

- (iii) Compute a binary indicator $\gamma = \prod_i [g(\mathbf{x}^{u*})_i \geq 0]$ about whether \mathbf{x}^{u*} is within the constraint set.
- (iv) Return $\mathbf{x}^* = \gamma \mathbf{x}^{u*} + (1 - \gamma) \mathbf{x}^{c*}$.

Clearly, this is only scalable for very small QPs, and as the QP size increases, the interior-point method will quickly become more efficient. Yet, for simple witness point routines (e.g., V-F, E-E), we have found this to be significantly faster, especially under massive vectorization. It can also be easily smoothed using approximations from S.III for operations such as inequality checks and argmin evaluations. The proposed framework implements variants of this approach for both V-F and E-E contacts, although as previously discussed in S.IV-A, the V-F routine is never actually used since V-S contacts are more efficient. The supplementary document contains exact descriptions for both V-F and E-E routines in pseudo-code.

2) *Creating E-E Witness Points*: Points \mathbf{e} on an edge between corners $\mathbf{v}_I, \mathbf{v}_{II} \in \mathbb{R}^3$ can be parameterized with a linear term $\mathbf{e}(\alpha) = \mathbf{v}_I + (\mathbf{v}_{II} - \mathbf{v}_I)\alpha$ and a box constraint $\alpha \in [0, 1]$. This means for E-E contacts, the unsigned distance QP (7) can be expressed using only box-constraints:

$$\begin{aligned} \boldsymbol{\alpha}^* = \arg \min_{\alpha_1, \alpha_2 \in \mathbb{R}} (\mathbf{e}_1(\alpha_1) - \mathbf{e}_2(\alpha_2))^\top (\mathbf{e}_1(\alpha_1) - \mathbf{e}_2(\alpha_2)) \\ \text{s.t. } 0 \leq \alpha_1, \alpha_2 \leq 1, \end{aligned} \quad (10)$$

with $\boldsymbol{\alpha}^* = [\alpha_1^*, \alpha_2^*]$.

L_2 regularization: As shown in Fig.3, when the tangential directions $\mathbf{t} = \mathbf{v}_{II} - \mathbf{v}_I$ of two edges point in the exact same 3D direction, the quadratic term in the QP above is rank deficient, and the minimum is not unique, which means the witness points exhibit a jump around this pathological configuration. As discussed previously, an L_2 regularization term $R(\alpha_1, \alpha_2) = \lambda [(\alpha_1 - 0.5)^2 + (\alpha_2 - 0.5)^2]$ is added to the objective of (10) to counteract this. This biases the solution towards the center of the edge, with increasing λ resulting in more smoothing (Fig. 3 - middle). Then the QP is solved with the following variant of the general analytical active set approach from S.IV-D1:

- (i) Compute the unconstrained minimizer $\boldsymbol{\alpha}^{u*} = \mathbf{Q}^{-1} \mathbf{c}$.
- (ii) Compute the minimizers $\{\boldsymbol{\alpha}_1^{c*}, \dots, \boldsymbol{\alpha}_4^{c*}\}$ along all four edges of the box constraints. For example along the edge $[\alpha_1 = 1, \alpha_2]$ the minimizer is $[1, f_{\text{clip}}(-\frac{Q_{12}-c_2}{Q_{22}}, 0, 1)]$, smoothed via a softclip operation. Then select the minimizer of the objective from $\{\boldsymbol{\alpha}_1^{c*}, \dots, \boldsymbol{\alpha}_4^{c*}\}$ as $\boldsymbol{\alpha}^{c*}$, smoothed via a softmax operation.
- (iii) Compute the indicator $\gamma = [0 \leq \alpha_1^{u*} \leq 1] * [0 \leq \alpha_2^{u*} \leq 1]$, smoothed using soft comparison operators.
- (iv) Return $\boldsymbol{\alpha}^* = \gamma \boldsymbol{\alpha}^{u*} + (1 - \gamma) \boldsymbol{\alpha}^{c*}$.

Note that an E-E contact is “colliding” (i.e., moving the edges along $\mathbf{t}_1 \times \mathbf{t}_2$ can create a contact) only when $\boldsymbol{\alpha}^{u*}$ lies within the constraint set. Therefore, γ can be used as a soft indicator to identify if a given E-E contact is colliding.

3) *Assigning Signed Distances and Normals*: After obtaining $\boldsymbol{\alpha}^*$, the corresponding witness points are obtained as $\mathbf{e}_1^* =$

²Note that for SQ primitives, normals are actually obtained from ∇f and not $\nabla \phi$, since the latter is non-monotonic and can cause inverted normals.

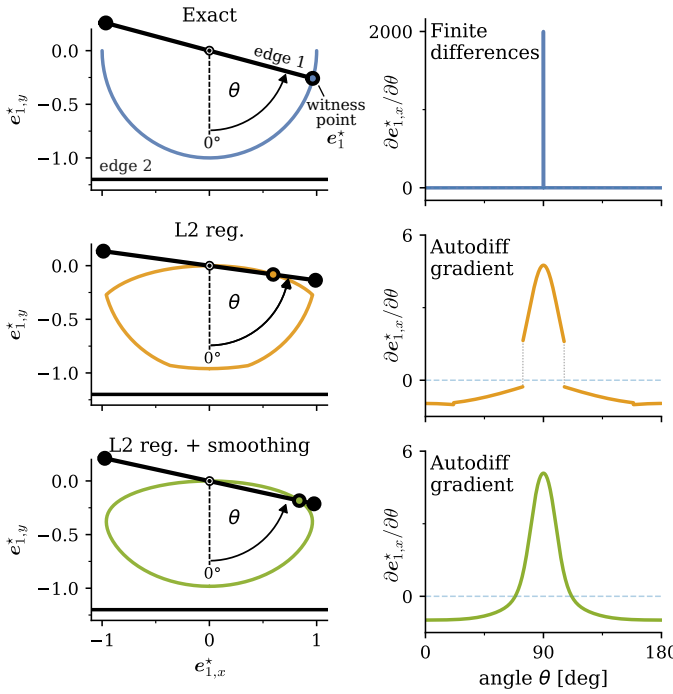


Fig. 3: **(Left)** A didactic experiment illustrating how E-E witness points behave under the proposed smoothing scheme. There are two edges lying on the xy plane, edge-2 being fixed and extending in the x direction with $y = -1.2$, and edge-1 rotating around the z axis of the origin. The colored lines plot the positions traced by the witness point $\mathbf{e}_1^*(\theta) = [\mathbf{e}_{1,x}^*, \mathbf{e}_{1,y}^*, 0]$ on edge-1 as it completes a half rotation $\theta : 0 \rightarrow \pi$. Without smoothing (top left), $\mathbf{e}_1^*(\theta)$ exhibits a jump as the system passes through the pathological parallel configuration $\theta = \pi/2$. The center and bottom plots visualize the smoothing contributions of L_2 regularization and employing the approximations from Section III. **(Right)** The gradients of $\mathbf{e}_1^*(\theta)$ for the three cases above. It can be seen that in the absence of any smoothing, the finite difference gradients are uninformative (e.g., very large near the pathological configuration), which is fixed by the proposed smoothing scheme.

$\mathbf{e}_1(\alpha_1^*)$ and $\mathbf{e}_2^* = \mathbf{e}_2(\alpha_2^*)$, yielding $\Delta \mathbf{e} = \mathbf{e}_1^* - \mathbf{e}_2^*$. Then, the unsigned distance and unsigned normal for the E-E pair are simply $\bar{d} = \|\Delta \mathbf{e}\|$ and $\bar{\mathbf{n}} = \Delta \mathbf{e} / \sqrt{\|\Delta \mathbf{e}\|^2 + \varepsilon}$.³

Determining the sign: The next step is to assign signs to \bar{d} and $\bar{\mathbf{n}}$ such that contact forces always oppose penetration. As shown in Fig. 5, the main difficulty of assigning the sign is in handling cases where there are penetrations, but the E-E witness points lie exactly on the intersection of the two surfaces. The standard way to address this problem is to rely on the separating plane between the surfaces to determine the sign

³The normal $\bar{\mathbf{n}}$ can be constructed from either $\mathbf{t}_1 \times \mathbf{t}_2$ or $\Delta \mathbf{e}$. The former cannot handle penetrations that happen when two edges are completely parallel (as $\mathbf{t}_1 \times \mathbf{t}_2 = \mathbf{0}$), and the latter cannot handle the case when two edges are exactly intersecting, but there is no penetration yet as $\Delta \mathbf{e} = \mathbf{0}$. Because most commonly used simulators rely on penalty formulations that only create forces after a penetration is present, the latter is the more suitable choice. Note that as long as an E-E contact is colliding, non-parallel, and non-intersecting, $\mathbf{t}_1 \times \mathbf{t}_2$ and $\Delta \mathbf{e}$ lie along the same line and are hence equivalent.

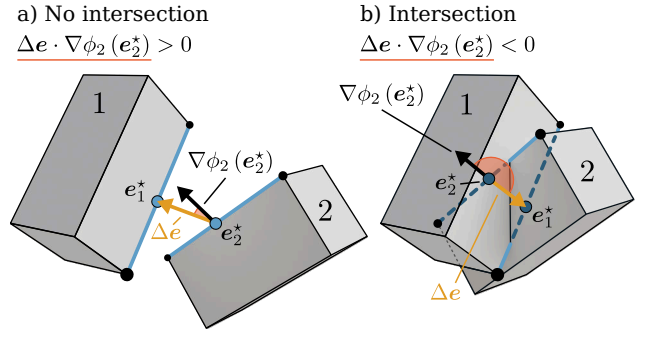


Fig. 4: Creating E-E Contacts: The witness points \mathbf{e}_i^* , as well as the corresponding unsigned distances and normals, are obtained from the (smoothed) analytical active set method described in S.IV-D2. The subsequent sign assignment for distances and normals in turn utilizes the SDFs by evaluating the dot product of the unsigned normal direction and the SDF gradient for the penetrated surface evaluated at each witness point.

[20]. When one has access to SDFs, the analogous approach is to realize that the contact normal for witness point \mathbf{e}_1^* should always point in the same direction as the normal $\mathbf{n}_2 := \nabla \phi_2(\mathbf{e}_2^*)$. Hence, if we construct a separating plane using \mathbf{e}_2^* and \mathbf{n}_2 , the correct sign is obtained as $\gamma_1 = \text{sign}(\mathbf{n}_2^\top \bar{\mathbf{n}})$ (and vice-versa for $\gamma_2 = \text{sign}(\mathbf{n}_1^\top \bar{\mathbf{n}})$), smoothed via a soft-sign. Therefore the signed distances and contact normals for both witness points \mathbf{e}_1^* and \mathbf{e}_2^* can be obtained as:

$$d_b = \gamma_b \bar{d} \text{ and } \mathbf{n}_b = \gamma_b \bar{\mathbf{n}} \text{ for } b \in \{0, 1\}. \quad (11)$$

Determining whether there is a penetration: To determine if a witness point is actually penetrating, one can compute

$$\text{pen} \gamma_1 = \sigma(-\phi_2(\mathbf{e}_1^*)/\tau), \quad (12)$$

to approximate $[\phi_2(\mathbf{e}_1^*) \leq 0]$ via a soft comparison operator.

Determining neighboring edges: E-E contacts should only be active if they are neighbors. Therefore, we need a mechanism that determines if edges are close to each other. After witness point computation, there are $|E_1| \times |E_2|$ unsigned distances $\bar{d}_{k,l}$ computed between all (ordered) E-E pairs, with $k \in \{1, \dots, |E_1|\}$ and $l \in \{1, \dots, |E_2|\}$. Let us arrange all $\bar{d}_{k,l}$ into a matrix $\mathbf{D} \in \mathbb{R}^{|E_1| \times |E_2|}$. To check if an (ordered) E-E pair is a nearest neighbor, we can compute two soft indicators (since computing nearest neighbors is an asymmetric operation)

$$\text{nn} \Gamma_{1,k,l} = s_{\text{argmax}} \left(- \left[D_{k,1} \cdots D_{k,|E_2|} \right] \right)_l \quad (13)$$

$$\text{nn} \Gamma_{2,k,l} = s_{\text{argmax}} \left(- \left[D_{1,l} \cdots D_{|E_1|,l} \right] \right)_k \quad (14)$$

by simply applying the soft argmin operation row- and column-wise on \mathbf{D} , to obtain $\text{nn} \Gamma_1, \text{nn} \Gamma_2 \in \mathbb{R}^{|E_1| \times |E_2|}$.

Determining clashing edges: E-E contacts should only be active if they are “clashing” (i.e. if they have opposing normals). This can be checked using the normals $\mathbf{n}_1 = \nabla \phi_1(\mathbf{e}_1^*)$ and $\mathbf{n}_2 = \nabla \phi_2(\mathbf{e}_2^*)$ evaluated at the witness points $\mathbf{e}_1^*, \mathbf{e}_2^*$ as:

$$\text{cl} \gamma_1 = \sigma \left(-\mathbf{n}_1^\top \mathbf{n}_2 / \tau \right), \quad (15)$$

approximating $[\mathbf{n}_1^\top \mathbf{n}_2 \leq 0]$ again via a soft comparison operator.

Determining which E-E collisions are active: A witness point in an E-E pair should be an active contact point if:

- The E-E pair is colliding.
- The witness point is penetrating into the opposing surface.
- The E-E pair constitutes nearest-neighbors.
- The E-E pair is clashing.

Much like the soft nearest-neighbor indicators $^{nn}\Gamma_{1,k,l}$ and $^{nn}\Gamma_{2,k,l}$; the soft indicators $^{con}\gamma$ for collision, $^{pen}\gamma$ for penetration, and $^{cl}\gamma$ for clashing are also defined for each of the two witness points in all $|E_1| \times |E_2|$ (ordered) E-E pairs. This means they can also be reshaped into a matrix format to obtain $^{con}\Gamma_1, ^{con}\Gamma_2, ^{pen}\Gamma_1, ^{pen}\Gamma_2, ^{cl}\Gamma_1, ^{cl}\Gamma_2 \in \mathbb{R}^{|E_1| \times |E_2|}$. In the proposed framework, all of these soft indicators are then combined to yield an “activity indicator”

$$\Gamma_{b,k,l} = ^{con}\Gamma_{b,k,l} * ^{pen}\Gamma_{b,k,l} * ^{nn}\Gamma_{b,k,l} * ^{cl}\Gamma_{b,k,l} \text{ for } b \in \{0, 1\} \quad (16)$$

for all E-E contacts. The subsequent contact dynamics routine can then use this information to turn contacts on or off.⁴

V. EXPERIMENTS

The aim of this section is to validate that the proposed routine can provide a contact manifold that adequately represents the volume of intersection between colliding surfaces in a manner that is differentiable, fast, and massively vectorizable.

1) *Validity of the contact manifold:* Two boxes resting on each other is one of the simplest cases in collision detection where witness point routines like GJK+EPA alone are not sufficient, and it is necessary to build a proper contact manifold (of up to 8 contact points [49]) with both V-F and E-E types for robust simulation. It is also a sufficiently representative case since non-convex meshes can be thought of as unions of convex polyhedra, thus making it the quintessential example covered by almost all pedagogical treatments of contact manifold generation [20]. Figure 5 also uses this example to visualize what contact manifolds generated by the proposed routine look like. As can be seen, the contact normals point in the correct directions and the contact activity indicators properly filter any contributions from non-active E-E and V-S pairs.

2) *Speed and Vectorizability of V-F, V-S, and E-E Collisions:* Figure 6 shows how the runtimes of the proposed V-F and E-E witness point routines change as one increases the number of vectorized queries from 1 to 10^7 . Queries are obtained by sampling V-F and E-E pairs from the uniform distribution over the unit cube, and the reported numbers are the mean and standard deviations from 100 trials. The baselines are:

- **Ours-NS** (no smoothing): Solving the V-F and E-E QPs via our proposed analytical active set routine, only without replacing non-smooth operations (e.g., comparisons and argmin operations) with their smooth approximations.
- **IP**: Solving the V-F and E-E QPs via the primal-dual interior point method (as implemented by the `qpax` library [50]), and smoothing them by changing the log-barrier coefficient.

⁴For example in a spring-damper model one can scale the stiffness with Γ , or analogously in the primal cone-complementarity formulation [48, 11] used by Mujoco [49] and Drake [47], the contact impedance can again be scaled.

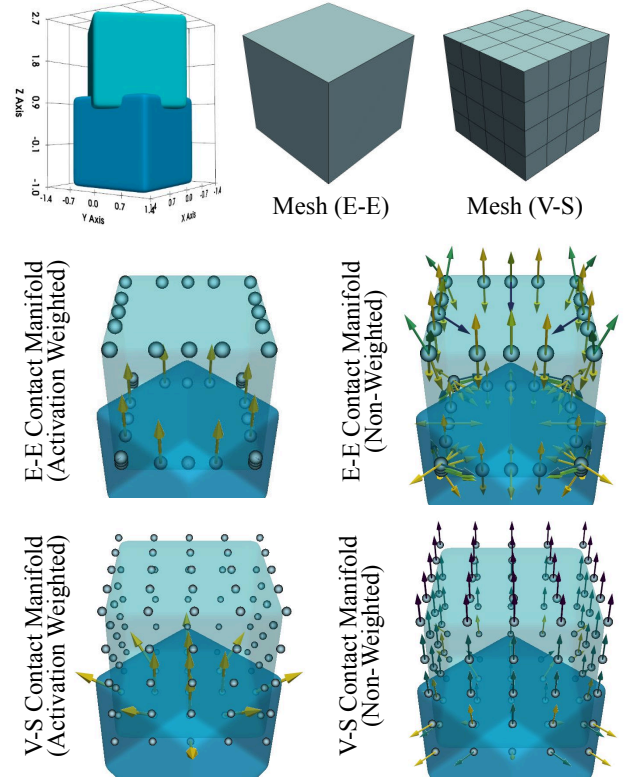


Fig. 5: Contact manifold visualizations for box-box collisions. SDFs are obtained from SQ primitives ($\epsilon_1 = \epsilon_2 = 0.1$). For illustration clarity, E-E contacts use the simplest cube mesh with 8 vertices, 12 edges, and 6 faces, whereas V-S contacts use a more densely meshed version of the same cube with 98 vertices. In both cases, only half of the contact points (belonging to the top box) are visualized, resulting in two manifolds: one with $12 \times 12 = 144$ E-E contacts and one with 98 V-S contacts. For both, two versions are shown: a version where normals are scaled with the activity indicator (to visualize which contacts are actually selected by the routine), and a version where they are unscaled. It can be seen that normals point in the right directions, and the correct points are selected.

- **IP-L**: Solving the exact same optimization objective (i.e. with the same log-barrier penalty) as the IP baseline, but using L-BFGS [9] for efficiency (as implemented by the `jaxopt` library [7]) rather than the the primal-dual interior point method. For V-F contacts, this requires parameterizing points on mesh faces via bilinear interpolation to ensure that there are only box constraints (i.e. to be able to apply LBFGS).

It can be seen that the proposed analytical smoothing approach is orders of magnitude faster and more suitable for vectorization compared to the standard log-barrier approach for smoothing inequality constrained optimization problems.

3) *Speed and Vectorizability of the Entire Routine:* This section compares the speed of the proposed contact manifold generation routine and that of the MJX simulator [18], in a setting that involves two nonconvex meshes of an armadillo. For the proposed routine, the smooth SDF representation for the

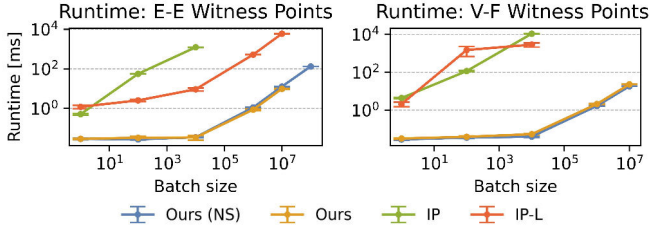


Fig. 6: Plots showing the speed and vectorizability of the proposed active-set approach for witness point computation (Ours, and the non-smoothed version Ours-NS), in comparison with log-barrier smoothing approaches. The latter are implemented via two interior point solvers, primal-dual Newton iterations on the KKT system (IP), and L-BFGS iterations with a log barrier penalty (IP-L). It can be seen that the proposed routines are fast and efficiently vectorizable.

original armadillo mesh is constructed from 18 SQ primitives whose parameters are obtained using the decomposition method of Liu et al. [30] (Fig. 7). The associated mesh representation (used by the proposed method) consists of 305 vertices, 605 faces, and 908 unique edges and is obtained by applying marching cubes to the smooth SDF representation and then applying the retopology method of [26]. There is no soft top-K selection on vertices, creating $305 \times 2 = 610$ potential V-S contacts. For E-E contacts, soft top-K selection with $K = 18$ is used (i.e., one edge per SQ), resulting in $18 \times 18 \times 2 = 648$ potential E-E contacts. To obtain collision meshes for the MJX routine, the 18 SQ primitives are converted to coarse convex meshes (i.e., since MJX requires all mesh geometries to be convex with fewer than 32 vertices)⁵, resulting in 18×18 convex-convex collision pairs and hence $18 \times 18 \times 4 = 1296$ potential contact points.

The first plot of Fig. 7 shows how the runtimes of the two routines change depending on geometry complexity in a non-vectorized setting as the armadillo geometry is gradually reconstructed from $(1 \times 1) \rightarrow (18 \times 18)$ SQ primitives. The reported numbers are again means and standard deviations over 100 trials, where the relative 6D poses of the two armadillos are randomized. It can be seen that the proposed framework is faster. The second plot of Fig. 7 runs the same experiment in a setting where the number of vectorized queries changes from $1 \rightarrow 1024$, only using the two cases with 5×5 and 18×18 SQ primitives in terms of geometry complexity for brevity. The experiment suggests that the proposed framework is significantly more suitable for massive vectorization. For this same experimental setup, the speed and vectorizability of gradient computations in the proposed framework (obtained via automatic differentiation) are also characterized in the same plot. Specifically, these plots correspond to the computation times for the 1×12 gradient of the mean distance of all contact points with respect to the 6D SE(3) representations for the poses

⁵We have also tried applying CoACD [53] to the same mesh representation (with 305 vertices) used for the proposed routine to directly obtain a convex decomposition that can then be passed to MJX. This was significantly slower than using 18 coarse SQ meshes.

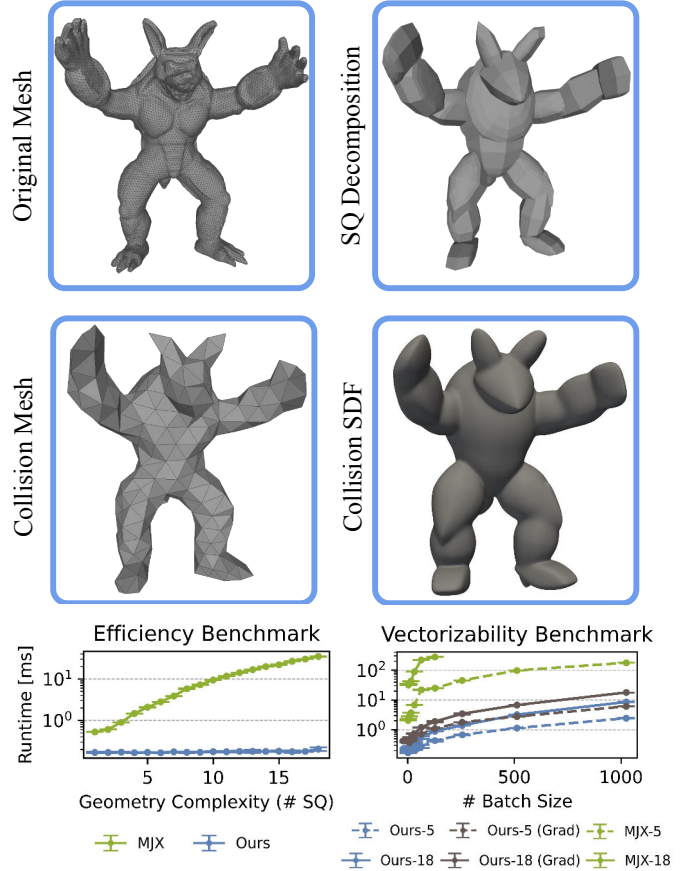


Fig. 7: Plots showing the speed and vectorizability of the proposed contact manifold generation routine, compared to the analogous routine of the well-established MJX simulator. The setup involves collisions between two non-convex armadillo geometries across 100 random configurations per-trial. Surfaces are represented with up to 18 SQs each, which is a controlled variable characterizing the effects of geometry complexity on the runtime. The second controlled variable is the vectorization batch size. Notice the logarithmic scale and the orders of magnitude improvement compared to MJX.

of the armadillos. It can be seen that the gradient computation is also fast (i.e., approximately half the speed of the original routine) and suitable for massive vectorization.

Please refer to the supplementary material for additional experiments that further characterize the proposed method.

VI. DISCUSSION AND LIMITATIONS

In general, the proposed framework can be flexibly configured by altering any subset of its interpretable parameters (e.g. number and detail of the SDF primitives and mesh resolutions, which contact types to activate, top K numbers for vertex and edge selection, smoothing coefficients for all approximations) to better suit the needs of a down-stream control algorithm. For example, all smoothing parameters can be set very small for obtaining forward rollouts with high physical realism, and gradient computations can then use larger smoothing coefficients to prevent pathologies due to

discontinuities. Similarly, the number of primitives can be reduced to trade-off geometry complexity with vectorizability, to facilitate applications such as coarse sweeps over hyper parameter grids or massively parallelized line-searches.

An important limitation of the proposed framework is that it expects a significant degree of expertise on the user side when building a simulation. For example: i) picking the correct SDF primitives for efficient shape abstraction, ii) making sure that there is no major discrepancy between the SDF and mesh representations of the same surface to avoid artifacts, iii) setting the numbers (N_1, N_2, M_1, M_2) for top-K selection appropriately. For (i), developing methods (similar to [30]) to automate the decomposition of arbitrary geometries into a mixture of SQ, CP, and OPC primitives in the most efficient way constitutes an interesting research direction. For (ii) and (iii), the supplementary material contains extended discussions on heuristic routines to determine (N_1, N_2, M_1, M_2) , as well as best practices and safeguard mechanisms to prevent any potential artifacts due to mesh-SDF discrepancies.

VII. CONCLUSION

We proposed a contact manifold generation routine that is smoothly differentiable, fast, and efficiently vectorizable. This was achieved via two main technical contributions. The first is a representative set of analytical primitives to build SDFs, which are then used for two purposes: i) as an acceleration structure to differentially select V-S and E-E pairs close to contact via soft top-K operations, and ii) to assign signed distances and normals to V-S and E-E contacts in a way that preserves curvature information (unlike traditional V-F and E-E contact routines between planar facets). The second is a routine to make witness point computations between V-F and E-E pairs smoothly differentiable via an analytical active-set method to solve the associated QPs, as well as differentiable schemes to assign signed distances, normals, and activity scores to them. The presented characterization experiments benchmark the proposed routine and compare it to the well-established MJX simulator. We observe a speedup by orders of magnitude. For future work, the most immediate direction is to use the proposed framework to develop efficient algorithms for optimal control and reinforcement learning that make use of first and second order information (gradients and Hessians) and exploit massive vectorizability.

ACKNOWLEDGMENTS

This work was supported by the ERC - 101045454 REAL-RL grant and the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039B). Georg Martius is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Onur Beker and Anselm Paulus.

REFERENCES

- [1] Superquadrics and angle-preserving transformations. *IEEE Computer graphics and Applications*, 1(01):11–23, 1981. 4
- [2] Arthur Allshire, Mayank Mittal, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11802–11809. IEEE, 2022. 1
- [3] Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH 2022 Courses*, pages 1–172. 2022. 1
- [4] Mihai Anitescu and Gary D Hart. A constraint-stabilized time-stepping approach for rigid multibody dynamics with joints, contact and friction. *International Journal for Numerical Methods in Engineering*, 60(14):2335–2371, 2004. 1
- [5] Mihai Anitescu and Florian A Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14(3):231–247, 1997. 1
- [6] Onur Beker, Nico Gürtler, Ji Shi, René Geist, Amirreza Razmjoo, Georg Martius, and Sylvain Calinon. A smooth analytical formulation of collision detection and rigid body dynamics with contact. In *2025 IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2025. 4
- [7] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021. 7
- [8] James Bradbury, Roy Frostig, et al. JAX: composable transformations of Python+NumPy programs, 2018. URL <http://github.com/jax-ml/jax>. 3
- [9] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyu Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995. 7
- [10] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 international conference on advanced robotics (ICAR)*. IEEE, 2015. 12
- [11] Alejandro M. Castro, Frank N. Permenter, and Xuchen Han. An unconstrained convex formulation of compliant contact. *IEEE Transactions on Robotics*, 39(2):1301–1320, 2023. doi: 10.1109/TRO.2022.3209077. 1, 7
- [12] Etienne Corman and Keenan Crane. Rectangular surface parameterization. *ACM Trans. Graph.*, 44(4), 2025. 2
- [13] MSC Software Corporation. Adams (automated dynamic analysis of mechanical systems). URL <https://hexagon.com/products/product-groups/>

- [14] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016. 1, 2
- [15] Tali Dekel, Shaul Oron, Michael Rubinstein, Shai Avidan, and William T Freeman. Best-buddies similarity for robust template matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015. 13
- [16] Kenny Erleben. Methodology for assessing mesh-based contact point methods. *ACM Transactions on Graphics (TOG)*, 37(3):1–30, 2018. 2
- [17] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4:193–203, 1988. 1, 2
- [18] Google Deepmind. Mujoco XLA (MJX), 2025. URL <https://mujoco.readthedocs.io/en/stable/mjx.html>. 3, 4, 7
- [19] Dirk Gregorius. The separating axis test between convex polyhedra. In *Game Developers Conference*, 2013. 2
- [20] Dirk Gregorius. Robust contact creation for physics simulations. In *Game Developers Conference*, 2015. 1, 2, 3, 6, 7, 12
- [21] Jason Gregory. *Game engine architecture*. AK Peters/CRC Press, 2018. 1
- [22] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*, 2019. 3
- [23] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996. 14
- [24] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo. dec 2022. doi: 10.48550/arXiv.2212.00541. URL <https://arxiv.org/abs/2212.00541>. 1
- [25] Taylor A Howell, Simon Le Cleac’h, J Zico Kolter, Mac Schwager, and Zachary Manchester. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 9(2):4, 2022. 3
- [26] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM transactions on graphics (TOG)*, 34(6):1–15, 2015. 2, 8
- [27] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 313–324, 1996. 12
- [28] Vince Kurtz. Hydrax: Sampling-based model predictive control on gpu with jax and mujoco mjx, 2024. <https://github.com/vincekurtz/hydrax>. 1
- [29] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Trans. Graph. (SIGGRAPH)*, 39(4), 2020. 1, 2, 4
- [30] Weixiao Liu, Yuwei Wu, Sipu Ruan, and Gregory S Chirikjian. Marching-primitives: Shape abstraction from signed distance function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8771–8780, 2023. 4, 8, 9
- [31] Louis Montaut, Quentin Le Lidec, Vladimir Petrik, Josef Sivic, and Justin Carpentier. Collision detection accelerated: An optimization perspective. *arXiv preprint arXiv:2205.09663*, 2022. 2
- [32] Louis Montaut, Quentin Le Lidec, et al. Differentiable collision detection: a randomized smoothing approach. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3240–3246. IEEE, 2023. 2
- [33] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999. 1, 5
- [34] NVIDIA. Isaac Sim, 2025. URL <https://github.com/isaac-sim/IsaacSim>. 1
- [35] Seonghun Oh, Xiaodi Yuan, Xinyue Wei, Ruoxi Shi, Fanbo Xiang, Minghua Liu, and Hao Su. Pamo: Parallel mesh optimization for intersection-free low-poly modeling on the gpu. In *Computer Graphics Forum*, page e70267. Wiley Online Library, 2025. 2
- [36] Chong Jin Ong and Elmer G Gilbert. Growth distances: New measures for object separation and penetration. *IEEE Transactions on Robotics and Automation*, 12(6):888–903, 1996. 2
- [37] Anselm Paulus, A René Geist, Pierre Schumacher, Vít Musil, and Georg Martius. Hard contacts with soft gradients: Refining differentiable simulators for learning and control. *arXiv preprint arXiv:2506.14186*, 2025. 3
- [38] Anselm Paulus, René Geist, Vít Musil, Sebastian Hoffmann, Onur Beker, and Georg Martius. Softjax. <https://github.com/a-paulus/softjax>, 2025. 3
- [39] Sebastian Prillo and Julian Eisenschlos. Softsort: A continuous relaxation for the argsort operator. In *International Conference on Machine Learning*, pages 7793–7802. PMLR, 2020. 3
- [40] Inigo Quilez. Gallery of shadertoy examples. URL <https://www.shadertoy.com/user/iq>. 4
- [41] Inigo Quilez. Signed distance functions, 2013. URL <https://iquilezles.org/articles/distfunctions/>. 4
- [42] Inigo Quilez. Smooth minimum, 2013. URL <https://iquilezles.org/articles/smin/>. 4
- [43] Inigo Quilez. Unlocking creativity with signed distance fields, 2025. URL <https://san-francisco.siggraph.org/2025/04/05/inigo-quilez-unlocking-creativity-with-signed-distance-fields/>. 4
- [44] Ivan E Sutherland and Gary W Hodgman. Reentrant polygon clipping. *Communications of the ACM*, 17(1): 32–42, 1974. 1, 2
- [45] Dojo Development Team. Dojo documentation, . URL https://dojo-sim.github.io/Dojo.jl/dev/background_contact/collisions.html. 3

- [46] Nimble Development Team. Documentation, . URL <https://nimblephysics.org/docs/intro.html#does-nimble-do-general-purpose-robotics-simulation>. 3
- [47] Russ Tedrake and the Drake Team. Drake: Model-based design and verification for robotics, 2019. URL <https://drake.mit.edu>. 1, 2, 7
- [48] Emanuel Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061. IEEE, 2014. 1, 7
- [49] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. 1, 2, 7
- [50] Kevin Tracy and Zachary Manchester. On the differentiability of the primal-dual interior-point method, 2024. 2, 5, 7
- [51] Kevin Tracy, Taylor A Howell, and Zachary Manchester. Differentiable collision detection for a set of convex primitives. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3663–3670. IEEE, 2023. 2
- [52] Gino Van Den Bergen. Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference*, volume 170, page 209, 2001. 1, 2
- [53] Xinyue Wei, Minghua Liu, Zhan Ling, and Hao Su. Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search. *ACM Transactions on Graphics (TOG)*, 41(4):1–18, 2022. 2, 8
- [54] Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. *arXiv preprint arXiv:2103.16021*, 2021. 3
- [55] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992. 2
- [56] Andrew Witkin and David Baraff. Physically based modeling: Principles and practice. In *ACM SIGGRAPH 1997 Courses*. 1997. 1
- [57] Yujia Xie, Hanjun Dai, Minshuo Chen, et al. Differentiable top-k with optimal transport. In *Advances in Neural Information Processing Systems*, volume 33, pages 20520–20531. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/ec24a54d62ce57ba93a531b460fa8d18-Paper.pdf. 3
- [58] Wen Yang and Wanxin Jin. Contactsdf: Signed distance functions as multi-contact models for dexterous manipulation. *arXiv preprint arXiv:2408.09612*, 2024. 4

SUPPLEMENTARY MATERIAL

I. ADDITIONAL EXPERIMENTS AND VISUALIZATIONS

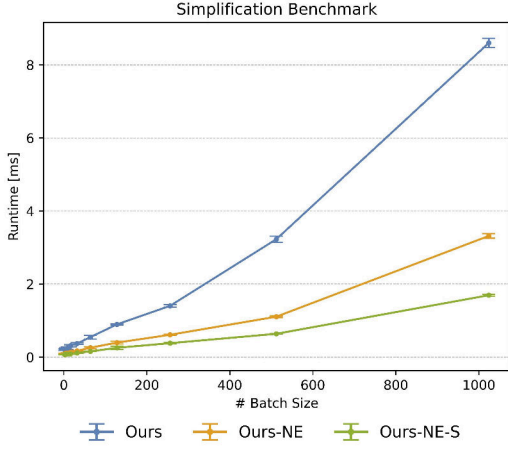


Fig. 8: A plot demonstrating how possible simplifications within the proposed contact manifold generation framework effect runtimes across different vectorization batch sizes. *Ours* denotes the proposed pipeline without any simplifications, *Ours-NE* denotes a setting where E-E contacts are turned off and only V-S contacts are considered, and *Ours-NE-S* denotes a setting where only one sided V-S contacts are created.

A. Possible Simplifications to Achieve Further Speed-up

As discussed in the main paper, a major goal of formulating physics for control is identifying the right simplifications. Within a collision detection setting, one example simplification is to turn off E-E contacts and only consider V-S contacts. The curve labeled *Ours-NE* in Fig. 8 shows how the runtime of such an approach scales with respect to vectorization batchsize, in the exact same experiment configuration as the vectorizability benchmark from Fig. 7 of the main paper. One can see that this simplification brings a speed-up of factor two. A further simplification can then be made by only creating one sided V-S contacts where vertices of one surface would collide with the SDF of the other, but not vice versa. In such an asymmetric setup, the surface that can be more efficiently tessellated in finer detail (e.g. with smaller surface area) would be assigned a mesh representation and the other surface would be represented with an SDF. The curve titled *Ours-NE-S* in Fig. 8 again shows that this brings an additional speed-up of factor two. The supplementary videos include simulations where E-E contacts are turned off, to demonstrate that these simplifications still maintain physical plausibility when a surface is meshed isotropically with dense enough tessellation. However one still needs to exercise caution, as when such conditions do not hold (e.g. the Mesh E-E case of the box-on-box collision from Fig.5 of the main paper, where tessellation density is low and long edges are present) these simplifications can result in artifacts.

B. Simulation Visualizations for Different Geometries

Fig. 9 shows primitive (SQ) decompositions, as well as the mesh and SDF representations for different geometries:

- Toy plane from the YCB dataset [10]. The mesh representation used for contact manifold generation contains 584 vertices and 1746 edges. The SDF is built from 32 SQ primitives.
- Drill from the YCB dataset. The mesh representation used for contact manifold generation contains 276 vertices and 822 edges. The SDF is built from 17 SQ primitives.
- The same armadillo mesh used in experiments from the main paper, borrowed from the Stanford 3D scanning repository [27]. The mesh representation used for contact manifold generation contains 305 vertices and 605 edges. The SDF representation is built from 18 SQ primitives.
- Screwdriver from the YCB dataset. The mesh representation used for contact manifold generation contains 165 vertices and 489 edges. The SDF is built from 5 SQ primitives.

Supplementary videos in turn contain example simulations with these objects, where the contact manifold was generated using the proposed method, and the subsequent contact dynamics step was implemented using the primal cone-complementarity formulation of Mujoco. Same as the benchmarking experiments from the main paper, simulations involving these objects use no soft top-K selection for V-S contacts, and E-E contacts use a soft top-K value equal to the number of SQ primitives constituting the object (i.e. one edge per SQ).

C. Larger Versions of All Plots

Please refer to Fig. 10 and Fig. 11 to find larger versions of plots from Fig.6 and Fig.7 of the main paper.

II. EXTENDED DISCUSSIONS ON LIMITATIONS

A. Picking the Top-K Numbers for Vertex and Edge Selection

It is important to note that by appropriately setting the numbers $|V_b|$ and $|E_b|$ for soft top-K selection of edges and vertices, the proposed contact manifold generation scheme can emulate both barrier methods and convex-primitive methods used in common robotics simulators. For the former barrier case, simply setting $|V_b| = |V|$ and $|E_b| = |E|$ for both surfaces is sufficient. For the latter convex primitive case, one can compute a convex decomposition of both surfaces to obtain K_b polytopes each, and set $|V_b|$ and $|E_b|$ to $\sim 4K_b$ (since it is common practice to generate up to four V-F contacts or eight E-E contacts between two convex polytopes [20]). It is important to realize that both of these estimates for $|V_b|$ and $|E_b|$ are very conservative upperbounds, and in most scenarios only a much smaller subset of all potential contact points are actually active (i.e. in or close to penetration). This observation is the main reason why the proposed framework uses analytical SDF primitives and soft top-K selection to implement a scheme analogous to a “smoothly differentiable acceleration structure”

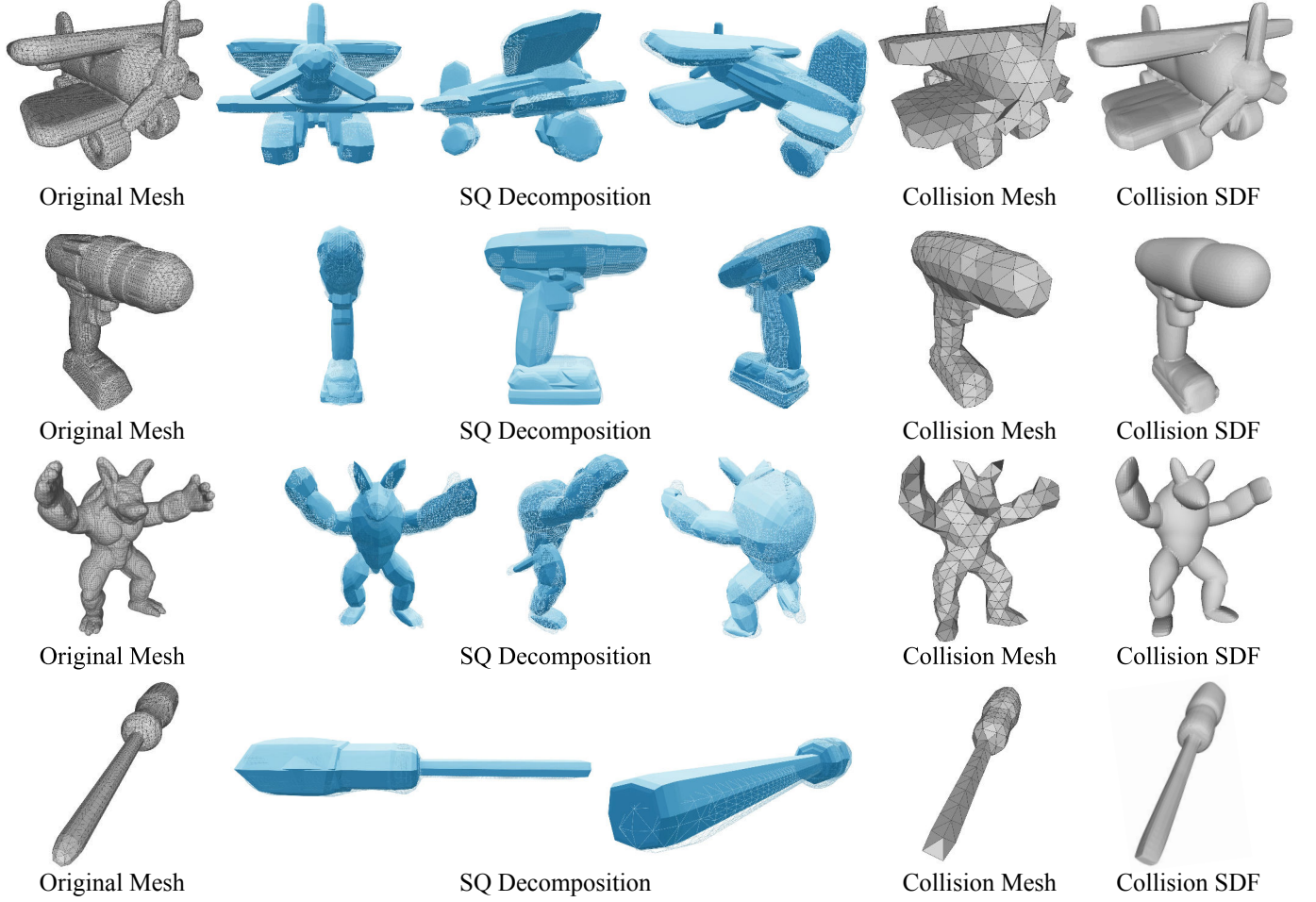


Fig. 9: Original high-resolution meshes, SQ primitive decompositions, as well as mesh and SDF representations (used by the proposed method) corresponding to the 4 objects used in simulation videos. The SQ decomposition visualizations also involve an overlay of the original high-resolution mesh, to demonstrate the representation power of SQ primitives.

(i.e. like a BVH, kd-tree, or a spatial hash), to only select those vertices and edges that are active. While this admittedly introduces a compromise between robustness and efficiency, we argue that this is reasonable (and expected) for a formulation of physics for control. A useful heuristic process to select the numbers $|V_b|$ and $|E_b|$ is as follows:

- Sample arbitrary SE3 configurations that create collisions between the two surfaces.
- Run any standard (non-differentiable) contact manifold generation process described in related work to obtain: i) the number N of best-buddy pairs [15] among all E-E pairs (to determine $|E_b|$), and ii) the number M of vertices that have a further penetration than all neighbors (to determine $|V_b|$).
- Observe the resulting histograms for N and M at the end of all Monte-Carlo trials to decide on $|V_b|$ and $|E_b|$ via a quantile that trades off efficiency and robustness (i.e. physics for control and physics for prediction).

B. Safeguarding Soft Top-K Selection

As was discussed in the background section of the main paper, soft-topK selection returns “soft-indices” (i.e. probabilities) $\Gamma_K(\mathbf{x}) \in [0, 1]^{K \times D}$ that correspond to the top-K

entries, and for any other arbitrary vector $\mathbf{y} \in \mathbb{R}^D$, the entries corresponding to these soft indices can be selected via a matrix product $\Gamma_K(\mathbf{x})\mathbf{y} \in \mathbb{R}^K$. Applying this scheme for selecting edges close to penetration involves averaging all edge endpoints (lying in rows of $\mathbf{E}_1, \mathbf{E}_2 \in \mathbb{R}^{|E| \times 3}$) using the soft-indices (i.e. $\Gamma_K(\mathbf{x})\mathbf{E}_1, \Gamma_K(\mathbf{x})\mathbf{E}_2 \in \mathbb{R}^{K \times 3}$). This means depending on the configuration, the averaged edge endpoints can in principle leave the interiors of their respective surfaces (since the proposed method operates on non-convex surfaces). As soft-topK probabilities are often very sharp (and they get even sharper as an edge penetrates further into an opposing surface), we have not observed this to cause any artifacts. Still, it is easy and computationally cheap to include a safeguard against such a hypothetical case by scaling the activity indicators for E-E contact points using the soft-indicators $[\phi_1(\mathbf{e}_1^*) \leq 0]$ and $[\phi_2(\mathbf{e}_2^*) \leq 0]$ (which capture whether a contact point \mathbf{e}_b^* lies within the surface described by ϕ_b).

C. Safeguarding Mesh-SDF Representation Discrepancies

To prevent any potential artifacts in the proposed framework due to a mismatch between the SDF and mesh representations, the following practices can be adopted:

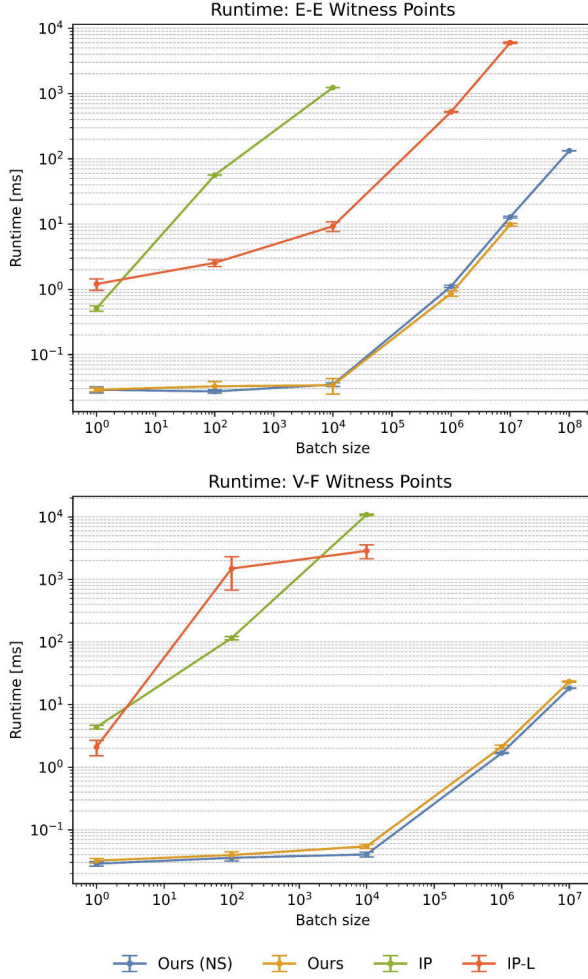


Fig. 10: Larger versions of plots from Fig.6 of the main paper.

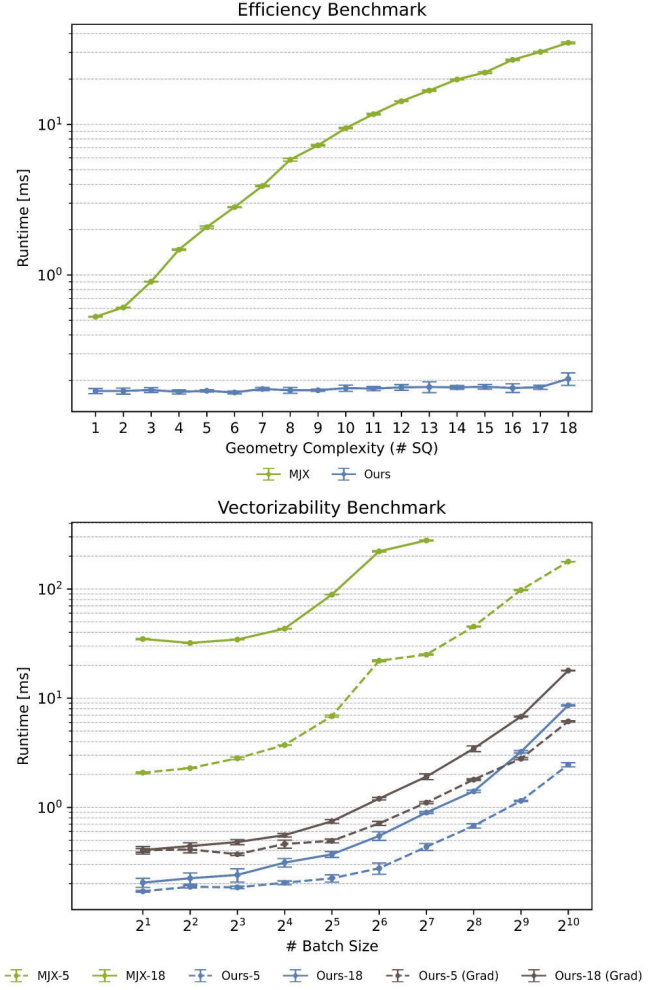


Fig. 11: Larger versions of plots from Fig.7 of the main paper.

- The mesh representation of the surface should be constructed such that: i) all vertices lie on the zero level set of the SDF representation, and ii) all edges should be contained inside the SDF representation (to the degree possible).
- After the witness points $\mathbf{e}_b^* = \mathbf{e}_b(\alpha_b^*)$ for $b \in \{0, 1\}$ are computed as described in subsection D-3 of the methods section, they can be projected onto the SDF surface via a fixed number of sphere tracing iterations [23] along the direction $\nabla \phi_b(\mathbf{e}_b^*)$. Since all of the SDF primitives introduced in subsection B of the methods section are metrically precise near the surface boundary (where witness points always lie if the mesh representation is constructed properly), the number of sphere tracing iterations needed is small enough (i.e. only 1 for a metrically correct SDF, often $\sim \leq 5$ for provided primitives) that the computation can be efficiently unrolled for automatic differentiation. All the subsequent computations for signed distances and normals in subsection D-3 should then use these adjusted witness points.

III. IMPLEMENTATIONS FOR THE PROPOSED V-F AND E-E WITNESS POINT ROUTINES

A. Smooth Approximations of Non-Smooth Operators

A subset of the smooth operators from the background section can be implemented in JAX as follows:

```

1 def sign_s(x, eps):
2     """
3     x: float
4     """
5     return jax.nn.tanh(x/eps)
6
7 def relu_s(x, eps=0.1):
8     """
9     x: float
10    """
11    return eps * jax.nn.softplus(x / eps)
12
13 def clip_s(x, a, b, eps=0.1):
14    """
15    x: float
16    """
17    tmp1 = relu_s(x - a, eps=eps)
18    tmp2 = relu_s(x - b, eps=eps)
19    return a + tmp1 - tmp2
20
21 def greater_s(x, l, eps=0.1):
22    """

```



```

23 x: float
24 """
25 return jax.nn.sigmoid((x - 1) / eps)
26
27 def smaller_s(x, u, eps=0.1):
28     """
29     x: float
30     """
31     return jax.nn.sigmoid((u - x) / eps)
32
33 def within_s(x, l, u, eps=0.1):
34     """
35     x: float
36     """
37     return greater_s(x, l, eps) * smaller_s(x, u, eps)
38
39 def argmin_s(x, eps=0.1):
40     """
41     x: (N,)
42     """
43     return jax.nn.softmax(-x/eps, axis=0)

```

B. E-E Witness Point Routine

The proposed smoothly differentiable active set routine to solve the E-E witness point QP (together with the utilities to then create the witness points) can be implemented as follows:

```

1 def solve_2by2_bqp(Q, c, eps_clip=0.1, eps_min=0.1,
2   eps_comp=0.1):
3     """
4     Smooth active set routine to solve a
5     2-by-2 box-constrained QP.
6
7     Q: (2,2)
8     c: (2,)
9     """
10
11     # Pre-compute necessary quantities
12     Q1 = Q[0, 0]
13     Q2 = Q[0, 1]
14     Q3 = Q[1, 1]
15     c1 = c[0]
16     c2 = c[1]
17
18     Q2_over_Q1 = Q2 / Q1
19     Q2_over_Q3 = Q2 / Q3
20     c1_over_Q1 = c1 / Q1
21     c2_over_Q3 = c2 / Q3
22
23     # Compute unconstrained minimum
24     x1_u = (Q2 * c2_over_Q3 - c1) / (Q1 - Q2 *
25     Q2_over_Q3)
26     x2_u = (Q2 * c1_over_Q1 - c2) / (Q3 - Q2 *
27     Q2_over_Q1)
28     x_u = jp.array([x1_u, x2_u])
29
30     # Compute all minima along the 4
31     # edges of the box constraints
32     x1_l_x2_u = clip_s(-(Q2_over_Q3 + c2_over_Q3),
33     0.0, 1.0, eps_clip)
34     x1_0_x2_u = clip_s(-c2_over_Q3, 0.0, 1.0,
35     eps_clip)
36     x1_u_x2_l = clip_s(-(Q2_over_Q1 + c1_over_Q1),
37     0.0, 1.0, eps_clip)
38     x1_u_x2_0 = clip_s(-c1_over_Q1, 0.0, 1.0,
39     eps_clip)
40
41     # Compute costs for the 4 minima
42     # along the box constraint edges
43     x1_l_x2_u_cost = (
44         0.5 * (Q1 + 2 * Q2 * x1_l_x2_u + Q3 *
45         x1_l_x2_u * x1_l_x2_u)

```

```

38         + c1
39         + c2 * x1_l_x2_u
40     )
41     x1_0_x2_u_cost = 0.5 * Q3 * x1_0_x2_u *
42     x1_0_x2_u + c2 * x1_0_x2_u
43     x1_u_x2_l_cost = (
44         0.5 * (Q1 * x1_u_x2_l * x1_u_x2_l + 2 * Q2 *
45         x1_u_x2_l + Q3)
46         + c1 * x1_u_x2_l
47         + c2
48     )
49     x1_u_x2_0_cost = 0.5 * Q1 * x1_u_x2_0 *
50     x1_u_x2_0 + c1 * x1_u_x2_0
51
52     # Soft-argminimum to select the smallest
53     # among the constrained minima
54     x1_l_x2_u_array = jp.array([1.0, x1_l_x2_u])
55     x1_0_x2_u_array = jp.array([0.0, x1_0_x2_u])
56     x1_u_x2_l_array = jp.array([x1_u_x2_l, 1.0])
57     x1_u_x2_0_array = jp.array([x1_u_x2_0, 0.0])
58
59     costs = jp.array([x1_l_x2_u_cost, x1_0_x2_u_cost,
60     x1_u_x2_l_cost, x1_u_x2_0_cost])
61     arrays = jp.array(
62         [x1_l_x2_u_array, x1_0_x2_u_array,
63         x1_u_x2_l_array, x1_u_x2_0_array]
64     )
65     min_probs = argmin_s(costs, eps_min)
66     x_c = (arrays * min_probs[:, None]).sum(axis=0)
67
68     # Compute the soft indicator for whether
69     # constraints are active
70     constraint_inactive = within_s(x1_u, 0, 1,
71     eps_comp) * within_s(
72         x2_u, 0, 1, eps_comp
73     )
74     constraint_active = 1 - constraint_inactive
75
76     # Pick whether the unconstrained or constrained
77     # minimum will be returned
78     x_opt = x_u * constraint_inactive + x_c *
79     constraint_active
80
81     return x_opt, constraint_active
82
83 def interpolate_edge(u, p):
84     """
85     u: float in [0, 1]
86     p: (2, 3)
87     """
88
89     return p[0] + (p[1] - p[0]) * u
90
91 def wp_E_E_bqp(
92     e1_p,
93     e2_p,
94     w_reg=0.01,
95     eps_clip=0.1,
96     eps_min=0.1,
97     eps_comp=0.1,
98 ):
99     """
100     Computes the two witness points of an E-E pair.
101
102     e1_p: (2,3), edge corner-points
103     e2_p: (2,3), edge corner-points
104     """
105
106     A = jp.stack([e1_p[1] - e1_p[0], e2_p[0] - e2_p
107     [1]], axis=1)
108     b = e1_p[0] - e2_p[0]
109
110     Q = A.T @ A + w_reg * jp.eye(2)
111     c = b.T @ A - w_reg * jp.array([0.5, 0.5])

```

```

104     u_u, gamma_con = solve_2by2_bqp_s(
105         Q,
106         c,
107         eps_clip=eps_clip,
108         eps_min=eps_min,
109         eps_comp=eps_comp,
110     )
111
112     witness_p1 = interpolate_edge(u_u[0], e1_p)
113     witness_p2 = interpolate_edge(u_u[1], e2_p)
114
115     return witness_p1, witness_p2, u_u, gamma_con
116
47         * within_s(v, 0, 1, eps_comp)
48         * within_s(w, 0, 1, eps_comp)
49     )
50     constraint_active = 1 - constraint_inactive
51
52     # Pick whether the unconstrained or constrained
53     # minimum will be returned
54     vp = vp_u * constraint_inactive + vp_c *
55         constraint_active
56     return vp

```

C. V-F Witness Point Routine

Analogous routines to the previous section can be implemented for V-F contacts as follows:

```

1 def V_E_proj(v, e_p):
2     e_t = normalize(e_p[1] - e_p[0])
3     return clip_s((v - e_p[0]).T @ e_t, 0, 1) * e_t
4     + e_p[0]
5
6 def V_T_proj_s(v, T_p, eps_clip=0.1, eps_min=0.1,
7     eps_comp=0.1):
8
9     # Pre-compute necessary quantities
10    delta_10 = T_p[1] - T_p[0]
11    delta_21 = T_p[2] - T_p[1]
12    delta_20 = T_p[2] - T_p[0]
13    delta_v0 = v - T_p[0]
14    delta_v1 = v - T_p[1]
15    e_t1 = normalize(delta_10)
16    e_t2 = normalize(delta_21)
17    e_t3 = normalize(delta_20)
18
19    # Compute all 3 minima along the constraints
20    # by projecting the vertex onto triangle edges
21    vp_e1 = clip_s((delta_v0 * e_t1).sum(), 0, 1,
22        eps_clip) * e_t1 + T_p[0]
23    vp_e2 = clip_s((delta_v1 * e_t2).sum(), 0, 1,
24        eps_clip) * e_t2 + T_p[1]
25    vp_e3 = clip_s((delta_v0 * e_t3).sum(), 0, 1,
26        eps_clip) * e_t3 + T_p[0]
27
28    # Soft-argminimum to select the smallest
29    # among the constrained minima
30    costs = jp.array(
31        [norm(v - vp_e1), norm(v - vp_e2), norm(v -
32        vp_e3)]
33    )
34    arrays = jp.array([vp_e1, vp_e2, vp_e3])
35    min_probs = argmin_s(costs, eps_min)
36    vp_c = (arrays * min_probs[:, None]).sum(axis=0)
37
38    # Compute the unconstrained minimum by
39    # projecting the vertex onto the triangle plane
40    n_bar = jp.cross(delta_10, delta_20)
41    n, n_bar_norm = normalize_with_norm(n_bar)
42    delta_vp_0 = delta_v0 - (delta_v0 * n).sum() * n
43    vp_u = delta_vp_0 + T_p[0]
44
45    # Check whether the unconstrained minimum is
46    # within
47    # the constraints, using barycentric coordinates
48    v = (jp.cross(delta_10, delta_vp_0) * n).sum() /
49        n_bar_norm
50    u = (jp.cross(delta_vp_0, delta_20) * n).sum() /
51        n_bar_norm
52    w = 1 - u - v
53
54    constraint_inactive = (
55        within_s(u, 0, 1, eps_comp)

```

ID	File	Description
Folder: armadillo_armadillo_rollout/		
A1	angle_1.mp4	Armadillo–Armadillo collision rollout (view angle 1). Shows contact manifold behavior on a non-convex mesh pair.
A2	angle_2.mp4	Same rollout as A1 from view angle 2.
A3	angle_3.mp4	Same rollout as A1 from view angle 3.
A4	angle_4.mp4	Same rollout as A1 from view angle 4.
A5	angle_1_E_E_contacts_off.mp4	Same setup as A1, but with E–E contacts disabled (V–S only). Demonstrates the “Ours-NE” simplification discussed in the supplementary material Fig. 8.
A6	E_E_on_off_comparison.mp4	Side-by-side comparison with E–E contacts enabled vs. disabled for Armadillo–Armadillo.
A7	interactive_visualizer_demo.mp4	Demo of the interactive visualizer used to inspect contact manifolds, normals, and activity indicators during simulation.
Folder: plane_drill_rollout/		
P1	angle_1.mp4	Plane (YCB) – Drill (YCB) collision rollout (view angle 1).
P2	angle_2.mp4	Same rollout as P1 from view angle 2.
P3	angle_3.mp4	Same rollout as P1 from view angle 3.
P4	angle_4.mp4	Same rollout as P1 from view angle 4.
P5	angle_1_E_E_contacts_off.mp4	Same setup as P1, but with E–E contacts disabled (V–S only).
P6	E_E_on_off_comparison.mp4	Side-by-side comparison with E–E contacts enabled vs. disabled for Plane–Drill.
P7	interactive_visualizer_demo.mp4	Interactive visualizer demo for Plane–Drill contact inspection.
Folder: drill_screwdriver_rollout/		
D1	angle_1.mp4	Drill (YCB) – Screwdriver (YCB) collision rollout (view angle 1).
D2	angle_2.mp4	Same rollout as D1 from view angle 2.
D3	angle_3.mp4	Same rollout as D1 from view angle 3.
D4	angle_4.mp4	Same rollout as D1 from view angle 4.
D5	angle_1_E_E_contacts_off.mp4	Same setup as D1, but with E–E contacts disabled (V–S only).
D6	E_E_on_off_comparison.mp4	Side-by-side comparison with E–E contacts enabled vs. disabled for Drill–Screwdriver.
D7	interactive_visualizer_demo.mp4	Interactive visualizer demo for Drill–Screwdriver contact inspection.

TABLE I: Supplementary videos. Each scenario is shown from four camera angles, plus an ablation with E–E contacts disabled (V–S only), an E–E on/off comparison, and an interactive visualizer demonstration.