Novel Algorithms for Smoothly Differentiable and Efficiently Vectorizable Contact Manifold Construction

Onur Beker, A. René Geist, Anselm Paulus, Georg Martius,
University of Tübingen

***Abstract*— Generating intelligent robot behavior in contact-rich settings is a research problem where zeroth-order methods currently prevail. Developing methods that make use of first/second order information about the dynamics holds great promise in terms of increasing the solution speed and computational efficiency. The main bottleneck in this research direction is the difficulty in obtaining useful gradients and Hessians, due to pathologies in all three steps of a common simulation pipeline: i) collision detection, ii) contact dynamics, iii) time integration. This abstract proposes a method that can address the collision detection part of the puzzle in a manner that is smoothly differentiable and massively vectorizable. This is achieved via two contributions: i) a highly expressive class of analytical SDF primitives that can efficiently represent complex 3D surfaces, ii) a novel contact manifold generation routine that makes use of this geometry representation.**

I. Introduction and Related Work

Our exposition proceeds as follows. We first briefly describe how existing collision detection pipelines based on convex primitive decomposition work, which are employed as the de-facto standard by all commonly used (non-differentiable) simulators in robotics [1, 2, 3]. We then outline existing approaches for making such convex primitive based routines smoothly differentiable. Finally, we list the reasons why such approaches still fall short of providing a satisfactory, generally applicable solution to differentiable collision detection.

The standard non-differentiable process employed by commonly used simulators to generate a contact manifold (i.e., a properly distributed set of contact points) between two surfaces can be abstracted into the following steps [4, 5]:

(i) Converting each non-convex surface into a mesh with well-distributed faces using a computational geometry algorithm [6, 7, 8], and then decomposing it into a set of convex meshes [9]. All of the remaining steps operate on pairs of these convex meshes.
(ii) Running standard collision detection routines like GJK+EPA [10, 11, 12] or SAT [13] 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.

The last step of going from two witness points to a manifold is needed to stabilize simulations in pathological settings such as completely parallel faces (e.g. two boxes stacked on top of each other). Alternative ways of implementing it are [5]:

- 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 [14] to the closest mesh-facets.

Approaches for obtaining smoothed gradients for the witness point positions between two convex primitives can be grouped in two: analytical smoothing and randomized smoothing. A prominent example of the former is the work of Tracy et al. [15], which solves for the growth distance [16] 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 [17]. A prominent example of the randomized smoothing approach is the work by Montaut et al. [18], which operates by: i) sampling M deviations around a nominal kinematic configuration, ii) running GJK and EPA algorithms [10] M times to get separation distances, iii) getting a zeroth-order estimate of the gradient via the score function estimator [19]. While both methods can robustly generate useful gradients for the witness point positions between two convex primitives, we argue that the underlying convex primitive based paradigm itself is pathological for differentiable contact simulation, as we elaborate on next.

The first pathology of the convex primitive based formulation is that witness points are conceptually not well defined (i.e. non-unique) for configurations that involve parallel contacts between planar faces, which are very common in contact simulations since they often constitute stable points for the dynamics (e.g. a box on a plane). For the analytical smoothing approach, this means in such pathological configurations (where the non-smoothed witness point optimization problem is not strictly convex), the uniqueness comes from the (strictly convex) log-barrier term, which needs to be sufficiently flat for the gradients to remain physically plausible, thus leading to slow convergence. For the randomized smoothing approach, this means the estimator has large variance at such configurations, since witness points can jump drastically in arbitrary directions based on the sampled deviation. The second pathology is that a single witness points pair is not sufficient for stable simulation, and only addressing

**Please see https://arxiv.org/abs/2604.17538 for an extended version with late-breaking results such as a new differentiable broad-phase routine.*

the differentiability of the convex-convex witness point routine is therefore insufficient. For the subsequent step of building a contact manifold from witness points, maintaining a buffer couples gradients across timesteps which is unsuitable for optimal control, and making polygon clipping routines differentiable in a computationally efficient manner is non-trivial (due to branching code and non-smooth operations such as top-K selection) [20] and remains an open problem. The most commonly used option is therefore inducing incremental rotations along tangential axes and re-running the convex-convex routine to build a manifold. The main problem is that there is currently no standard, mathematically principled way to pick the tangential axes uniquely (since they are only defined up to a rotation along the normal direction), in a way that is guaranteed to change smoothly across timesteps. A third and final pathology is that in the absence of a broad-phase routine for filtering primitive pairs without running the complete convex-convex routine, the convex primitive based formulation is inefficient. This is because for two surfaces decomposed into N and M primitives each, $N \times M$ convex-convex checks are required. Therefore only addressing the narrow-phase convex-convex routine is insufficient since the broad-phase routine can still create jumps in the gradient (due to primitive pairs activating and deactivating).

All of these considerations motivate us to design a collision detection routine from scratch with differentiability and vectorizability as the primary concern (which constitutes the main contribution of this paper), inspired by the analogous routines of barrier-based contact simulation approaches [21]. We opt for this approach rather than trying to make existing routines differentiable, because these standard routines were conceived within a computer graphics, video game design, or computational science and engineering context with speed and minimal memory footprint as the primary concern (rather than differentiability and vectorization).

II. Methods

A. Smooth Approximations to Non-Smooth Operators

The proposed framework extensively uses smooth approximations of operators such as comparisons and clipping, implemented by the SoftJAX library [22]. The sigmoid function $\sigma(x) = (1+\exp(-x))^{-1}$ can smoothly approximate a binary comparison operator $[\![x > a]\!] : \mathbb{R} \to \{0,1\}$ via $\sigma((x-a)/\tau)$. The softplus function $\mathrm{s}_+(x) = \tau \log(1+\exp(x/\tau))$ can approximate the relu function, two of which can be combined to approximate the clip function. Finally, the softmax and logsumexp functions can smoothly approximate the argmax and max functions, notated as $s_{\text{argmax}}(\boldsymbol{x})_i = \exp(x_i/\tau)/\sum_{j=1}^{D}\exp(x_j/\tau)$ and $\mathrm{LSE}(\boldsymbol{x}) = \tau \log[\sum_{i=1}^{D}\exp(x_i/\tau)]$.

B. XPSQ: A Novel Analytical SDF Primitive

This section introduces the proposed "extruded plane-superquadric intersection" (XPSQ) primitive, used for efficiently representing complex 3D surfaces in a smoothly differentiable manner. It extends the line of work by [23], in particular on the superquadric (SQ) primitive. At a high-level, the XPSQ primitive analytically solves for the SDF associated with the volume traced by sweeping a superquadric (potentially intersected with N half-spaces) along a quadratic

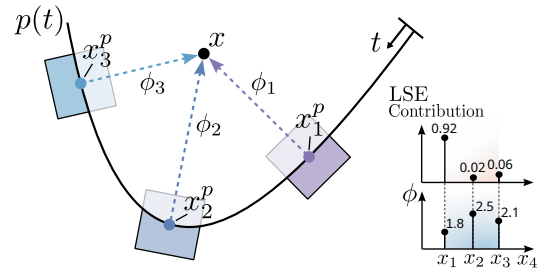

Fig. 1: The SDF of the XPSQ primitive is evaluated by: (i) analytically projecting a point $\boldsymbol{x}$ onto a spline $p(t)$, (ii) transforming the PSQ SDF expression such that the surface is moved to coincide with each of the three projection points, and (iii) obtaining the SDF for $\boldsymbol{x}$ via the smooth-minimum of the three transformed PSQ SDFs.

spline. This enables describing geometries such as the handle of a cup with a single primitive, which would not be possible via convex decomposition [9] or an SQ decomposition [24].

1) Superquadrics. Superquadrics (SQ) are a family of surfaces that subsume basic shape primitives such as cubes, cylinders, and ellipsoids. An SQ in its canonical form is defined by the unit level set the level set $f(\boldsymbol{x}) = 1$ of an *inside–outside* implicit function $f : \mathbb{R}^3 \to \mathbb{R}$ [24]:

$$f(\mathbf{x}) = \left(\left(\frac{x}{a_x}\right)^{\frac{2}{\varepsilon_2}} + \left(\frac{y}{a_y}\right)^{\frac{2}{\varepsilon_2}} \right)^{\frac{\varepsilon_2}{\varepsilon_1}} + \left(\frac{z}{a_z}\right)^{\frac{2}{\varepsilon_1}} \tag{1}$$

where ε_1 controls the shape's "roundness", ε_2 adjusts the shape's "pointedness", and a_x scales the x-dimension. In total there are 11 parameters: 6 for 3D pose, 3 for per axis scales, 2 for roundness and pointedness.

2) Differentiable Combinations of SDFs. Any two SDFs $\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}$ (e.g. representing a half space, SQ, or XPSQ) can be combined through *union* $\phi_3 = \min(\phi_1, \phi_2)$, *intersection* $\phi_3 = \max(\phi_1, \phi_2)$, or *subtraction* $\phi_3 = \max(\phi_1, -\phi_2)$ operations. Smoothly differentiable versions of these operations are obtained through the use of the logsumexp operator [25, 26]:

$$\phi_{\cup}(\boldsymbol{x}) = -\mathrm{LSE}([-\phi_1(\boldsymbol{x}), -\phi_2(\boldsymbol{x})]) \text{ (smooth union)}, \tag{2}$$
$$\phi_{\cap}(\boldsymbol{x}) = \mathrm{LSE}([\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x})]) \text{ (smooth intersection)}, \tag{3}$$
$$\phi_{\ominus}(\boldsymbol{x}) = \mathrm{LSE}(\phi_+(\boldsymbol{x}), -\phi_-(\boldsymbol{x})) \text{ (smooth subtraction)}. \tag{4}$$

3) PSQ: Combining Half Spaces with SQs. The SDF of a half-space is defined as $\phi_N = \boldsymbol{x} \cdot \boldsymbol{n} + h$, where $\boldsymbol{n}$ denotes the plane normal and h the offset from the origin. Through the smooth intersection (3) of an SQ and a single half space, most of the geometric primitives provided by common simulation frameworks can be defined, including boxes, ellipsoids, cylinders, elliptic cones, square pyramids, and tetrahedrons (although our framework and software implementation does not assume any restrictions on the number of half spaces intersecting an SQ). We refer to the primitive obtained by intersecting N half spaces $\mathbf{P} \in \mathbb{R}^{N \times 4}$ (3 parameters for $\boldsymbol{n}$ and 1 for h) and an SQ as a "plane-SQ intersection" (PSQ).

4) XPSQ: Tracing a Spline with a PSQ. The XPSQ primitive represents the volume obtained by tracing a PSQ along a 3D quadratic spline. A quadratic spline admits a 1D parameterization using $t \in [0,1]$ and three control points $\boldsymbol{p}_{1:3}$:

$$\boldsymbol{p}(t) = (1-t)^2\boldsymbol{p}_1 + 2t(1-t)\boldsymbol{p}_2 + t^2\boldsymbol{p}_3. \tag{5}$$

The data-structure that represents an XPSQ primitive therefore maintains the following information: i) a quadratic spline $\boldsymbol{p}(t)$, ii) a smooth function $\mathbf{R}(t) \in \mathrm{SO}(3)$ that prescribes the rotational pose of the PSQ as it traces the spline (e.g. the Frenet frame associated with the spline [27]), iii) smooth functions $\varepsilon_{\{1,2\}}(t)$ and $a_{\{x,y,z\}}(t)$ that prescribe the shape and scale parameters of the SQ part of the PSQ, iii) $\mathbf{P}(t) \in \mathbb{R}^{N\times 4}$ that prescribes the plane normal (expressed in the PSQ base frame) and shift for the N half spaces within the PSQ.

4) The SDF Associated with an XPSQ. To evaluate the SDF of the XPSQ for a given point $\boldsymbol{x}$, one first projects it onto the spline to obtain the projection point $\boldsymbol{p}(t^*)$, and then evaluates the SDF of the PSQ at that point (as prescribed by $\mathbf{R}(t^*), \varepsilon_{\{1,2\}}(t^*), a_{\{x,y,z\}}(t^*), \mathbf{P}(t^*)$). As was also noted by [28], this projection can be obtained by minimizing $\|\boldsymbol{x} - \boldsymbol{p}(t)\|^2$, which is a fourth order polynomial in t. Taking its gradient and setting it to zero in turn gives a cubic equation in t, which can be solved analytically via Cardano's formula [29]. The solution is not unique in general, as for some constellations there are multiple points on the spline with equally minimal distance to $\boldsymbol{x}$, e.g. when the spline is wrapped around $\boldsymbol{x}$. In Cardano's formula, these cases are distinguished by the sign of the discriminant $\Delta \in \mathbb{R}$, which is a function of the control points, and the computation of the roots branches depending on it. This branching leads to a discontinuity, which we soften by always treating both branches simultaneously and merging the results. In the negative branch we use the negative projected discriminant $\Delta^- = -s_+(-\Delta) \in \mathbb{R}^-$ to obtain the unique real root t^- of the cubic. In the positive branch, we use the positive projected discriminant $\Delta^+ = s_+(\Delta) \in \mathbb{R}^+$, yielding three real roots (t_1^+, t_2^+, t_3^+). Note that in both cases we apply a soft clipping operation to keep the roots in the range $(0,1)$, as Cardano's formula can return roots outside of this range. Next, we always return three roots as the convex combination $\boldsymbol{t}^* \in \mathbb{R}^3$ of the two root sets:

$$\boldsymbol{t}^* = [\![\Delta < 0]\!] * [t^-, t^-, t^-] + [\![\Delta > 0]\!] * [t_1^+, t_2^+, t_3^+] \tag{6}$$

where the brackets are smoothed as described in S.II-A. Then, we get the PSQs prescribed at all three roots using $\mathbf{R}(t_i^*)$, $\boldsymbol{x}_i^*$, $\varepsilon_{\{1,2\}}(t_i^*)$, $a_{\{x,y,z\}}(t_i^*)$, and $\mathbf{P}(t_i^*)$ with $i \in \{1,2,3\}$. The associated SDFs for the three PSQs are $\{\phi(\boldsymbol{x}_1^*), \phi(\boldsymbol{x}_2^*), \phi(\boldsymbol{x}_3^*)\}$, and we finally return their smooth-minimum $\phi_\cup(\boldsymbol{x}) = -\mathrm{LSE}([-\phi_1(\boldsymbol{x}), -\phi_2(\boldsymbol{x}), -\phi_3(\boldsymbol{x})])$ as shown in Fig.1.

C. A Novel Contact Manifold Generation Routine

Given that the XPSQ primitive provides an efficient way to represent SDFs of complex surfaces, the next step is using such SDFs to create contact manifolds between two colliding surfaces in a differentiable and efficiently vectorizable manner. In the proposed framework, one of the surfaces is represented by an SDF and the other by a mesh, and contact points are created on the surface represented by the mesh (if symmetry is desired, one can simply run the same routine again with the roles transposed). There are two main steps involved in this: i) finding the intersections between all edges of the mesh and the SDF, ii) using these per-edge intersection points to transcribe per face contact depths, normals, and Jacobians.

1) Finding Edge-SDF Intersection Points. Points $\boldsymbol{e}$ on an edge between corners $\boldsymbol{v}_I, \boldsymbol{v}_{II} \in \mathbb{R}^3$ can be parameterized as

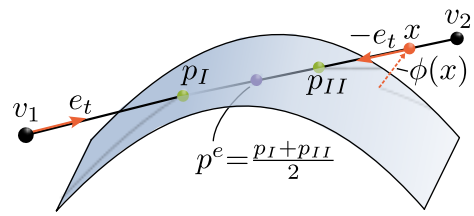

Fig. 2: Intersection points p_I and p_{II} between the SDF $\phi(x)$ of one surface and the edge $v_1 - v_2$ of the other surface are determined via sphere tracing. At every sphere tracing step, the current intersection point candidate x is moved towards the SDF by $\pm[\![\phi(x) > 0]\!]\phi(x)e_t$. The $[\![\phi(x) > 0]\!]$ term ensures that if v_1 or v_2 already lie inside the surface, they are not moved by sphere tracing.

$\boldsymbol{e}(\alpha) = \boldsymbol{v}_I + \alpha \boldsymbol{e}_t$ with $\boldsymbol{e}_t = \frac{\boldsymbol{v}_{II} - \boldsymbol{v}_I}{\|\boldsymbol{v}_{II} - \boldsymbol{v}_I\|}$ and $0 \le \alpha \le \|\boldsymbol{v}_{II} - \boldsymbol{v}_I\|$. This means the maximal penetration point between the edge e and any SDF ϕ can hypothetically be found via $\min_\alpha \phi(\boldsymbol{e}(\alpha))$ subject to $0 \le \alpha \le \|\boldsymbol{v}_{II} - \boldsymbol{v}_I\|$. One can then obtain smooth gradients through this constrained optimization problem by employing log barrier smoothing and applying the implicit function theorem to the resulting KKT conditions [15, 17]. That being said, there are a number of pathologies associated with employing such an edge-SDF collision routine:

- The optimization problem is non-linear, meaning the solution and its gradients obtained via implicit differentiation would still exhibit jumps when the problem switches from one local optimum to another as parameters are varied.
- Unrolling the solver steps (e.g. Newton's method in 1D) and applying automatic differentiation is also not an option since it is computationally expensive. This is because: i) computing Hessians is expensive, and ii) the number of Newton iterations needs to be large (empirically ≥ 10).
- As shown in [23], such procedures are inefficient for vectorization, because a vectorized iterative solver applied to a batch of problems will only finish as quickly as the slowest-converging problem in the batch.

2) Collision Detection with Sphere Tracing. Therefore, we devise a custom edge-SDF collision routine based on sphere-tracing [30]. Given an initial point $\boldsymbol{p}_0$ and a ray direction $\boldsymbol{n}$, sphere-tracing is an algorithm to find ray-SDF intersections (widely considered to be the most efficient way of doing so even for very complex surfaces [31]). It proceeds via updates of the form $\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + [\![\phi(x) > 0]\!]\phi(x)\boldsymbol{n}$. The proposed edge-SDF routine then proceeds as follows:

- Sphere trace the edge corners $\boldsymbol{v}_I$ and $\boldsymbol{v}_{II}$ along $\boldsymbol{e}_t$ and $-\boldsymbol{e}_t$ respectively to obtain $\boldsymbol{p}_I$ and $\boldsymbol{p}_{II}$. We have empirically found three iterations to be more than sufficient for convergence.
- Clip $\boldsymbol{p}_I$ and $\boldsymbol{p}_{II}$ within the edge boundaries via a soft-clip operation, and return the midpoint $\boldsymbol{p}^e = (\boldsymbol{p}_I + \boldsymbol{p}_{II})/2$.

This process is illustrated in Figure 2. Naturally, it is guaranteed to succeed only in cases when an edge penetrates the SDF only once or never. We argue that this is sufficient, because if there exists an edge that penetrates an SDF more than once, this indicates that the mesh tessellation is not at a density adequate for the complexity of the surfaces involved in the collision, and should hence be increased.

3) Computing Per-Face Contact Information. In a triangle mesh, every face has three edges (each with one edge-SDF intersection point obtained in the previous step), and three vertices. This means every face has 6 potential contact points

on it, which we denote as $p^f_{1:6}$. Technically, one can create a contact point for every edge and every vertex in a mesh, resulting in $V+E$ contact points in total (where V and E denote the number of vertices and edges respectively). Given that $E \approx 3V$ for most meshes [32], this creates approximately $4V$ contact points. If desired, this number can be reduced to F (i.e. the number of faces) as follows:

- Compute the penetration depths $d_i = \phi(p^f_i)$, activity indicators $\gamma_i = [\![d_i < 0]\!]$, contact normals $\boldsymbol{n}_i = \nabla\phi(p^f_i)$, and contact Jacobians $\mathbf{J}_i$ for all 6 points.
- Obtain softmax weights $z_i = s_{\text{argmax}}(-\boldsymbol{d})_i$ based on the penetration distances d_i, and fuse the normals and Jacobians via $\boldsymbol{n} = \sum_i \boldsymbol{z}_i \gamma_i \boldsymbol{n}_i$ and $\mathbf{J} = \sum_i \boldsymbol{z}_i \gamma_i \mathbf{J}_i$, as proposed by [33].

Note that the fusion of the contact normals and Jacobians means whenever any subset of the 6 potential contact points of a face have nearly equal penetration distances (e.g. a cube lying on a plane), the effect of the per face contact force (i.e. returned by the subsequent contact dynamics solver that consumes the contact manifold) is distributed nearly equally across them. This results in more stable contact interactions compared to first fusing $\boldsymbol{p} = \sum_i \boldsymbol{z}_i \gamma_i \boldsymbol{p}_i$ and then computing the contact normal and Jacobian at this single fused point.

III. Experiments

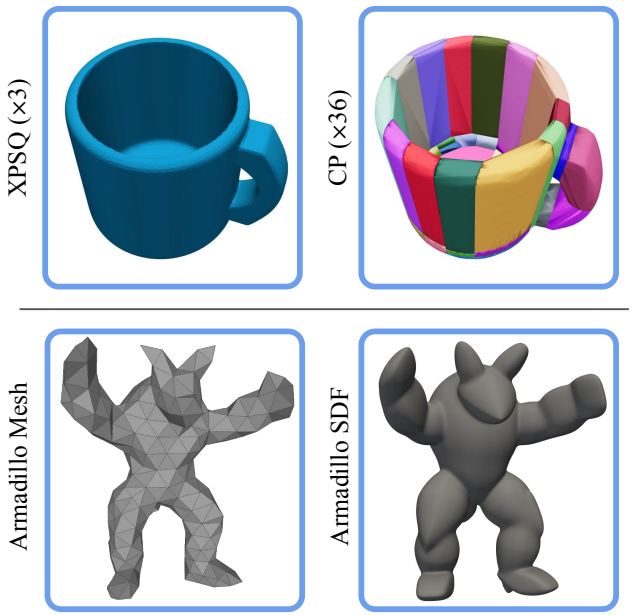

Fig. 3: Examples that demonstrate the expressivity of XPSQs.

1) Expressivity of the XPSQ Primitive. The top part of Fig.3 illustrates how a cup can be represented with only 3 XPSQ primitives (i.e. one cylinder subtracted from another, combined with a handle obtained by sweeping a box along a spline). In contrast, convex decomposition using CoACD [9] results in 36 primitives, which, in the absence of a differentiable broad-phase routine, is highly inefficient. The bottom part shows how an armadillo can be efficiently represented with 18 SQs, obtained using the method of [24].

2) Behavior of the Contact Manifold. Fig.4 illustrates the qualitative behavior of the contact manifold compared to the polygon clipping routine employed in Mujoco [1], on

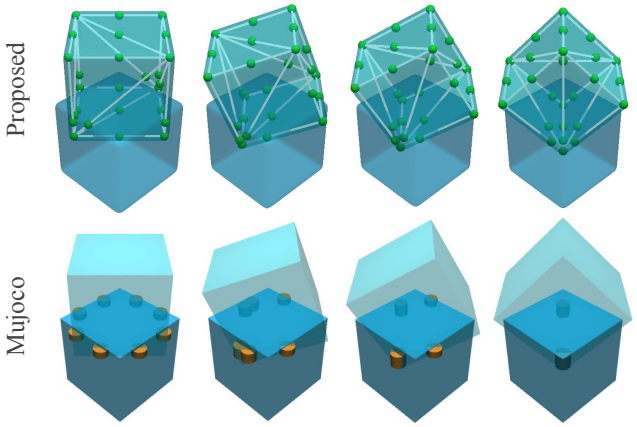

Fig. 4: An example illustrating the behavior of the contact manifold.

an example with two boxes stacked on top of each other. It can be seen that polygon clipping results in non-smooth behavior of the contact points, whereas the proposed edge-SDF contact points adapt smoothly to the surface, adequately sampling the volume of intersection between the colliding surfaces. From this perspective, the proposed routine can be interpreted as a middle ground between polygon clipping and the hydroelastic contact model [34], without needing to explicitly re-mesh the intersection volume like the latter (which is a non-differentiable and non-vectorizable routine).

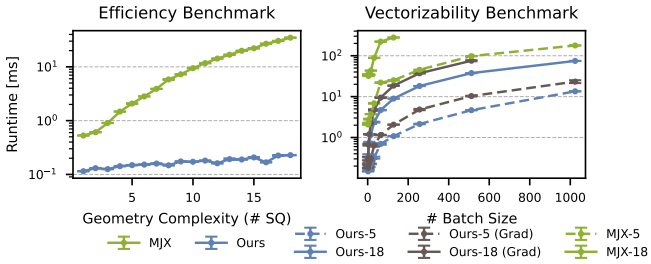

Fig. 5: A benchmark characterizing computational efficiency.

3) Computational Efficiency. Fig.5 contains 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 order of magnitude improvement compared to MJX. The experiment does not contain any broad-phase routine to filter out edges without needing to execute the full edge-SDF collision routine. Another order of magnitude improvement can potentially be gained by employing any existing differentiable broadphase routine for edge filtering, such as the soft topK selection approach of [23], or the approach of [35] that utilizes spatial hashing and distance barrier functions that are smoothly mollified to zero to deactivate non-penetrating contact points.

IV. Limitations and Conclusion

We have presented two contributions: i) XPSQ primitives that can efficiently represent SDFs associated with highly

complex surfaces, ii) a differentiable and vectorizable routine that generates a contact manifold between a mesh and an SDF. The main limitation of the proposed method is the lack of an established routine to automatically decompose a mesh into XPSQ primitives, which we leave to future work.

References

[1] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033. 1, 4

[2] R. Tedrake and the Drake Team, "Drake: Model-based design and verification for robotics," 2019. 1

[3] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016. 1

[4] K. Erleben, "Methodology for assessing mesh-based contact point methods," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 3, pp. 1–30, 2018. 1

[5] D. Gregorius, "Robust contact creation for physics simulations," in *GDC*, 2015. 1

[6] W. Jakob, M. Tarini, D. Panozzo, and O. Sorkine-Hornung, "Instant field-aligned meshes," in *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)*, vol. 34, no. 6, Nov. 2015. 1

[7] E. Corman and K. Crane, "Rectangular surface parameterization," *ACM Trans. Graph.*, vol. 44, 2025. 1

[8] S. Oh, X. Yuan, X. Wei, R. Shi, F. Xiang, M. Liu, and H. Su, "Pamo: Parallel mesh optimization for intersection-free low-poly modeling on the gpu," in *Computer Graphics Forum*. Wiley Online Library, 2025, p. e70267. 1

[9] X. Wei, M. Liu, Z. Ling, and H. Su, "Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–18, 2022. 1, 2, 4

[10] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, pp. 193–203, 1988. 1

[11] L. Montaut, Q. L. Lidec, V. Petrik, J. Sivic, and J. Carpentier, "Collision detection accelerated: An optimization perspective," *arXiv preprint arXiv:2205.09663*, 2022. 1

[12] G. Van Den Bergen, "Proximity queries and penetration depth computation on 3d game objects," in *GDC*, vol. 170, 2001, p. 209. 1

[13] D. Gregorius, "The separating axis test between convex polyhedra," in *GDC*, 2013. 1

[14] I. E. Sutherland and G. W. Hodgman, "Reentrant polygon clipping," *Communications of the ACM*, vol. 17, no. 1, pp. 32–42, 1974. 1

[15] K. Tracy, T. A. Howell, and Z. Manchester, "Differentiable collision detection for a set of convex primitives," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3663–3670. 1, 3

[16] C. J. Ong and E. G. Gilbert, "Growth distances: New measures for object separation and penetration," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 6, pp. 888–903, 1996. 1

[17] K. Tracy and Z. Manchester, "On the differentiability of the primal-dual interior-point method," 2024. 1, 3

[18] L. Montaut, Q. Le Lidec *et al.*, "Differentiable collision detection: a randomized smoothing approach," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3240–3246. 1

[19] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992. 1

[20] A. Paulus, A. R. Geist, P. Schumacher, V. Musil, and G. Martius, "Hard contacts with soft gradients: Refining differentiable simulators for learning and control," *arXiv preprint arXiv:2506.14186*, 2025. 2

[21] M. Li, Z. Ferguson, T. Schneider, T. Langlois, D. Zorin, D. Panozzo, C. Jiang, and D. M. Kaufman, "Incremental potential contact: Intersection- and inversion-free large deformation dynamics," *ACM Trans. Graph. (SIGGRAPH)*, vol. 39, no. 4, 2020. 2

[22] A. Paulus, A. R. Geist, V. Musil, S. Hoffmann, O. Beker, and G. Martius, "SoftJAX & SoftTorch: Empowering automatic differentiation libraries with informative gradients," *arXiv preprint*, 2026. 2

[23] O. Beker, A. R. Geist, A. Paulus, N. Gürtler, J. Shi, S. Calinon, and G. Martius, "Smoothly differentiable and efficiently vectorizable contact manifold generation," *arXiv preprint arXiv:2602.20304*, 2026. 2, 3, 4

[24] W. Liu, Y. Wu, S. Ruan, and G. S. Chirikjian, "Marching-primitives: Shape abstraction from signed distance function," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 8771–8780. 2, 4

[25] I. Quilez, "Signed distance functions," 2013. 2

[26] ——, "Smooth minimum," 2013. 2

[27] M. Spivak, *A Comprehensive Introduction to Differential Geometry*, 3rd ed. Publish or Perish, Inc., 1999, vol. 2, ch. Curves In the Plane and In Space (The Serret–Frenet formulas), pp. 34–36. 3

[28] Y. Li and S. Calinon, "From movement primitives to distance fields to dynamical systems," *IEEE Robotics and Automation Letters (RA-L)*, vol. 10, no. 9, pp. 9550–9556, 2025. 3

[29] M. Artin, *Algebra*, 2nd ed. Pearson, 2013, ch. 16 – Galois Theory, pp. 501–502. 3

[30] J. C. Hart, "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996. 3

[31] K. Crane, "Ray tracing quaternion julia sets on the gpu," 2005. 3

[32] ——, "Discrete differential geometry: An applied introduction." 4

[33] O. Beker, N. Gürtler, J. Shi, R. Geist, A. Razmjoo, G. Martius, and S. 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

[34] R. Elandt, E. Drumwright, M. Sherman, and A. Ruina,

"A pressure field model for fast, robust approximation of net contact force and moment between nominally rigid objects," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2019, p. 8238–8245. 4

[35] Z. Ferguson *et al.*, "IPC Toolkit," 2020. 4