`foam`: A Tool for Spherical Approximation of Robot Geometry

Sai Coumar, Gilbert Chang, Nihar Kodkani, and Zachary Kingston

Abstract—**Many applications in robotics require primitive spherical geometry, especially in cases where efficient distance queries are necessary. Manual creation of spherical models is time-consuming and prone to errors. This paper presents `foam`, a tool to generate spherical approximations of robot geometry from an input Universal Robot Description Format (URDF) file. `foam` provides a robust preprocessing pipeline to handle mesh defects and a number of configuration parameters to control the level and approximation of the spherization, and generates an output URDF with collision geometry specified only by spheres. We demonstrate `foam` on a number of standard robot models on common tasks, and demonstrate improved collision checking and distance query performance with only a minor loss in fidelity compared to the true collision geometry. We release our tool as an open source Python library and containerized command-line application to facilitate adoption across the robotics community.**

I. INTRODUCTION

Efficient collision detection (both as binary classification and measuring distance to collision) is a fundamental challenge throughout robotics; algorithms and control strategies from real-time model-predictive control [1, 2], sampling-based motion planning [3, 4], simulation [5, 6], and more all rely on effective collision detection of the robot's geometry with that of the environments. Although effective collision detection strategies exist for complex geometries (e.g., space decompositions, parallel approaches, GPU-accelerated algorithms), these approaches scale on the complexity of the involved geometry, and performance improves when the representative geometry of the robot and environment is simple.

Moreover, many robot descriptions (usually in URDF) are exported from CAD tools [7], which may contain extraneous details for coarse collision checking. These complex meshes also often contain non-manifold geometry (that is, non-watertight) and may contain geometry that violates assumptions made by geometric processing pipelines, e.g., flat planes or single faces. A common approach is to provide simplified geometry to represent the collision geometry of the robot, either with a set of convex meshes [8] or with a set of primitives [9]. This is advantageous as collision checking with primitive approximations is more efficient for collision checking and signed distance queries (e.g., as in Sundaralingam et al. [3] and Mukadam et al. [10]), which are prevalent in modern robotics applications. However, although these primitive approximations are useful and widely used, existing tools lack robust automation for converting existing robot geometry into these primitive decompositions.

SC, GC, NK, and ZK are with the CoMMA Lab, Department of Computer Science, Purdue University, West Lafayette, IN, USA
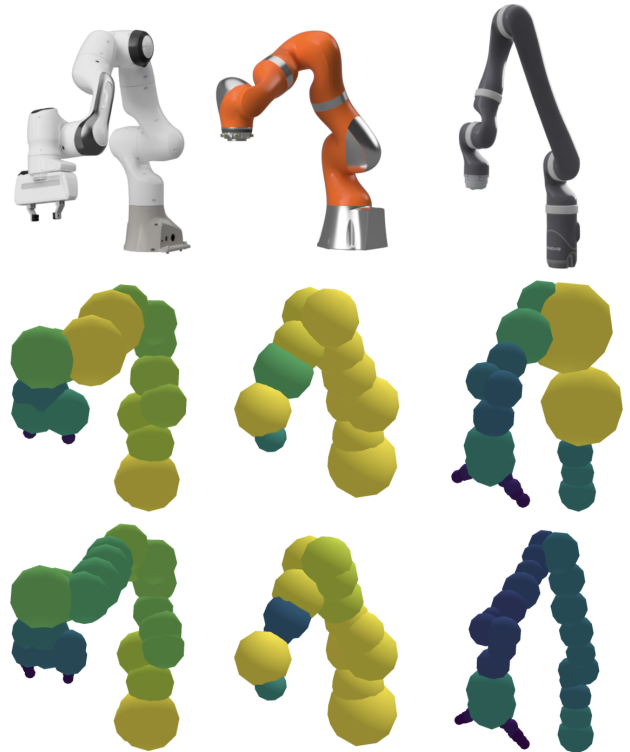{scoumar, chang940, nkodkani, zkingston}@purdue.edu

Fig. 1. Results of spherizing commonly used robots with `foam`. From left to right: Franka Emika Panda, KUKA IIWA, and Kinova Jaco robots (top row), alongside spherized approximations (middle and bottom row). Displayed are spherizations of the Panda with 22 and 28 spheres, the IIWA with 13 and 19 spheres, and the Kinova with 22 and 39 spheres. Our `foam` library automatically processes robots specified with a URDF and creates a URDF with spherical approximations of the robot's collision geometry.

This paper presents `foam`, an automated framework for generating spherical approximations of robot geometries directly from Universal Robot Description Format (URDF) specifications. The `foam` spherization pipeline balances geometric accuracy with computational efficiency through a number of configurable approximation parameters, building upon existing work on the medial sphere-tree approximation for meshes [11]. Our preprocessing pipeline handles and fixes common mesh defects gracefully, while maintaining critical geometric features needed for collision detection. The output of `foam` is an equivalent URDF of the input robot, but with all collision geometry replaced by spheres. We evaluated the output URDFs of `foam` across multiple standard robots and applications, and we demonstrated that `foam`-generated models achieve collision checking performance improvements while maintaining geometric

fidelity suitable for practical robotic tasks. To the authors' knowledge, while spherization tools for just meshes exist open source, nothing has been packaged as a convenient tool for roboticists; we release our tool as open source at https://github.com/CoMMALab/foam/ as a Python library and containerized command-line tool.

II. RELATED WORK

`foam` focuses on generating the spherical decomposition of a given robot geometry. Sphere-based modeling of robots has become increasingly prominent, as spherical approximations facilitate both efficient collision checking and efficient computation of Euclidean distance transforms (EDTs) [12] and signed distance fields (SDFs) [13–15] with respect to a robot's current configuration, which are both essential for many gradient-based and cost-map methods in robotics. For example, in motion planning alone, trajectory optimization approaches such as CHOMP [9], STOMP [16], motion planning as inference methods such as GPMP2 [10, 17], and cuRobo [3] all rely on spherical models for SDF computation. The recently proposed SIMD-accelerated sampling-based planning methods in Thomason et al. [4] also rely on spherical decompositions of the robot's geometry for efficient collision detection.

Our tool is built upon existing methods for automatic approximation of a mesh by a set of spheres. Early approaches used simple voxelizations (which are still used in, e.g., Sundaralingam et al. [3]) or octree decompositions [18]. Other tools fit spheres along the medial axis of the geometry [19], that is, along the set of points inside the mesh having more than one closest point on the mesh's boundary. For robots with simple collision geometry, e.g., cylinders, this is simple and can be done analytically, as done by Völz and Graichen [20]. `foam` is built on top of an improvement to these medial axis approaches, the Adaptive Medial Axis Approximation (AMAA) method of [11]. Although subsequent approaches have refined the AMAA approach (e.g., [21]), open source code was not available to build on. Sphere decompositions have also been used in discrete element models for particle simulations [22].

Given the impact of mesh complexity on the performance of collision checking, rendering, and other geometric algorithms [23, 24], there has been much research on a variety of decomposition, simplification, and approximation methods for the processing of geometry. While alternative primitive representations have been explored in the literature, such as cuboid decompositions [25, 26] and quadric-based models [27], these have seen less adoption due to their complexity and lack of benefit for computing SDF and EDT functions. Convex decomposition methods, e.g., V-HACD [28] and CoACD [29], decompose a non-convex mesh into a set of convex meshes that approximate the original. Highlighting the importance of geometry preprocessing, these decomposition techniques are critical not only for efficient geometric representation, but also for modern dataset curation. This is evidenced by recent works that incorporate CoACD into large-scale robotics datasets [30, 31]. There are also other mesh simplification and low-poly approximation pipelines [32–35] that reduce the number of vertices in a mesh, but do not change the mesh into convex or primitive geometry.

III. THE FOAM LIBRARY

The `foam` Library provides an automated framework for creating spherical approximations of robot geometries directly from Universal Robot Description Format (URDF) specifications. By converting complex polygonal meshes into simplified sphere-based representations, `foam` significantly improves computational efficiency for collision detection while maintaining sufficient geometric fidelity for practical robotics applications. The library offers a configurable pipeline that handles common mesh defects gracefully, preserves critical geometric features, and outputs equivalent URDFs with collision geometry replaced by spheres.

A. URDF Collision Geometry

In a robot's URDF, the collision geometry is specified by some number of `<collision/>` tags in each of the robot's `<link/>` tags. These can either be primitives (e.g., spheres, cuboids, cylinders), or by most of the time referencing external mesh files that are composed of complex polygons. Some robot manufacturers provide convex hulls, simplified mesh geometry, or primitive approximations for collision checking, but many are simply raw CAD file output [7]; in some cases the details are potentially so fine that screw holes are captured in the geometry.

`foam` creates simple, accurate spherized approximations of the geometries within a robot's URDF, either specified as meshes or as primitives. Our tool can either output spheres in a JSON database or in standard URDF representation (that is, a copy of the original robot's URDF with all the collision geometry replaced by a set of sphere primitives). The `foam` library provides some utility functions to load URDFs, retrieve all mesh files used in a URDF, modify a URDF to include spheres, and save the output file, respectively **load_urdf()**, **get_urdf_meshes()**, **set_urdf_spheres()** and **save_urdf()**.

We note that our simplified spherized models can be easily utilized in simulators such as MuJoCo [6], and spherization removes the dependency on external mesh assets. Generated spheres can be used in other applications that may benefit from simplified spherical approximations. For example, Pinocchio, which uses FCL collision checking, benefits greatly from mesh simplification due to using a collision library that is optimized for simplified geometry [36, 37].

B. Mesh Preprocessing

Many polygonal mesh processing algorithms (including our desired spherization approach) assume that the input mesh is *watertight* or *manifold* (specifically a 2-manifold). As implied by the name, a watertight mesh is such that it can "hold water", that is, it does not have holes or other broken geometry, e.g., edges with more than two incident faces, mesh faces intersecting each other, faces that are not connected

(1) Original Mesh   **(2)** Watertight Mesh   **(3)** Simplified Mesh   **(4)** Smoothed Mesh   **(5)** 9 Sphere Mesh   **(6)** cuRobo Meshes
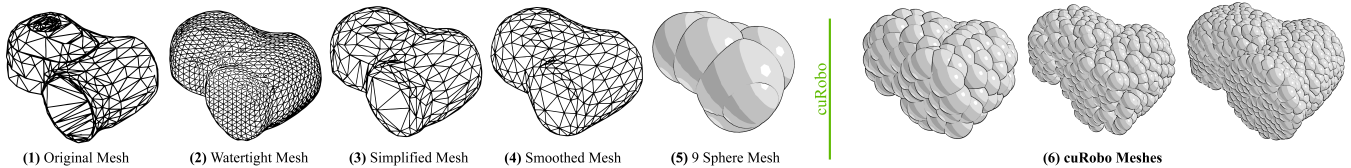
Fig. 2. A representative example of a mesh being processed by `foam`. Here, the 6th link of the Franka Emika Panda [38] is processed into a set of spheres. From left to right: **(1)** the original mesh is made into a **(2)** guaranteed *watertight* or *manifold* mesh through Huang et al. [39]. The watertight mesh is then simplified **(3)** using a quadric-based edge collapse mesh simplification method [33] and smoothed **(4)** with Laplacian smoothing and Humphrey filtering [40]. Finally, spheres are fit to the processed mesh **(5)** with adaptive medial axis approximation method [11]. We also show the **(6)** spherization approach provided in cuRobo [3], which voxelizes the mesh into spheres with a given sphere radius. From left to right, we show decompositions with 200 spheres and 2cm radius, 500 spheres and 1cm radius, 1000 spheres and 1cm radius

together, etc. Watertight meshes also have a clear definition of the interior and exterior of the mesh. Unfortunately, many URDFs contain robot geometry that has been exported from automated tools [7], and therefore may contain multiple defects. These defects are also common among many meshes used in everyday tools, e.g., it is known that many of the meshes in ShapeNet [41] are non-manifold. Moreover, for our downstream tasks of collision checking and spherization, these meshes may have extraneous detail that adds additional complexity to the mesh which is unessential.

In order to handle non-watertight meshes, `foam` uses a preprocessing pipeline to handle these defects while preserving essential geometric features required for collision detection. This preprocessing pipeline also reduces the complexity of the mesh, improving the performance of the final spherization algorithm. A representative output from each stage of the pipeline is shown in Fig. 2. Each of these steps has a number of configurable hyperparameters to control the amount of simplification, smoothing, and processing the input mesh goes through.

The preprocessing workflow begins after the collision meshes are loaded with Trimesh [42] (Fig. 2-1). The mesh is then transformed into a guaranteed watertight mesh using the method of Huang et al. [39] (Fig. 2-2). This step is crucial and ensures topological consistency by eliminating non-manifold edges, dangling faces, and small holes that could otherwise compromise the accuracy of subsequent spherization. Following this step, we then simplify the manifold mesh with a quadric-based edge collapse simplification method [33] to reduce complexity, as downstream algorithms scale with the number of triangles in the mesh and are more robust with less complex input (e.g., see Fig. 3). This produces a simplified mesh (Fig. 2-3) with significantly fewer vertices and faces, yet preserves the essential shape characteristics of the original geometry. The process continues with a smoothing phase (Fig. 2-4) that applies both Laplacian smoothing and Humphrey filtering [40] (`smooth_manifold()` in the `foam` library) to remove small-scale irregularities and noise from the mesh surface, creating a more uniform geometric representation suitable for sphere fitting.

C. Spherization of Meshes

After the mesh has been processed into a simplified, watertight mesh, it is then spherized with an implementation of the
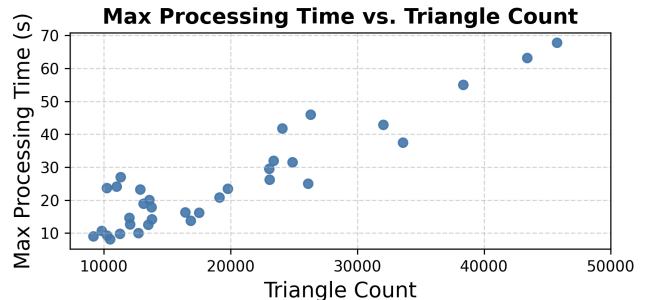

Fig. 3. Time to spherize an input mesh against the number of triangles in the input mesh.

Adaptive Medial Axis Approximation (AMAA) algorithm of Bradshaw and O'Sullivan [11] (`spherize_mesh` using the `medial` option in the `foam` library). An output from the AMAA processing can be seen in Fig. 2-5, which shows how the mesh can be effectively covered with only 9 spheres. We contrast the output of AMAA after our preprocessing to the provided spherization tool in cuRobo [3], where a variety of outputs are shown in Fig. 2-6. Their approach performs a simplified voxelization of the mesh into spheres with a given sphere radius.

We also expose other methods for spherization (e.g., a voxel grid-based decomposition with `grid`), but these notably provide less high-fidelity spherizations There are also a number of keyword arguments to control the spherization process—a number of them are listed in the example code in Fig. 5 (e.g., the maximum number of desired spheres can be specified with the `branch` parameter). This control of the amount of desired spherization is illustrated in Fig. 4, which shows the Franka Emika Panda robot spherized with increasing amounts of spheres. This allows users to control the fidelity and complexity of the output spherized model, either remaining coarse or reaching a close-fitting approximation. We contrast the output of `foam` (Fig. 4-1–5) with a custom-made spherization from Fishman et al. [43] (Fig. 4-6). `foam`'s spherization more closely matches the true robot collision geometry (Fig. 4-7) and is a conservative overapproximation, rather than an underapproximation.

D. The Foam Tool

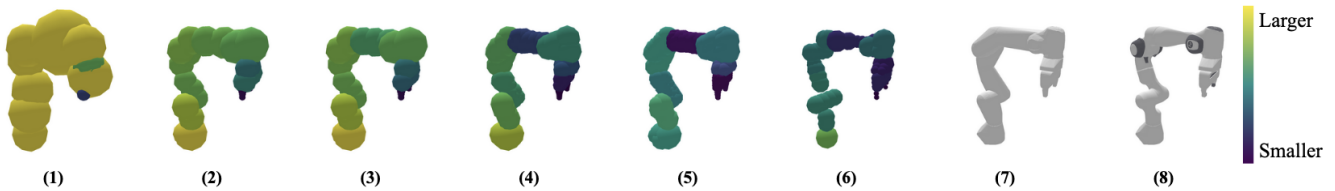We provide the mesh preprocessing pipeline and spher-

Fig. 4. Franka Emika Panda Robot [38] processed with `foam` to generate spherized models with increasing level of fidelity, with (left to right) **(1)** 11, **(2)** 22, **(3)** 27, **(4)** 92, and **(5)** 316 spheres. The spheres on the robot are colored by size, indicated by the colorbar on the far right. We compare the output of `foam` to **(6)** a custom-made spherization of the Panda from [43] with 59 spheres, which is a clear under-approximation of the robot's **(7)** true collision geometry. We also display **(8)**, the robot's visual geometry, as reference.

```python
def main(mesh: str, output: str):
    mesh_filepath = Path(mesh)
    # kwargs for spherization
    sphere_kwargs = {
        'depth': 1,
        'branch': 8,
        'method': medial,
        'testerLevels': 2,
        'numCover': 500,
        'minCover': 5,
        'initSpheres': 1000,
        'minSpheres': 200,
        'erFact': 2,
        'expand': True,
        'merge': True,
        'burst': False,
        'optimise': True,
        'maxOptLevel': 1,
        'balExcess': 0.05,
        'verify': True,
        'num_samples': 500,
        'min_samples': 1
    }
    # kwargs for mesh processing
    process_kwargs = {
        'manifold_leaves': 1000,
        'ratio': 0.2,
    }
    # Call spherize_mesh with kwargs
    spheres = spherize_mesh(
        mesh_filepath,
        spherization_kwargs=sphere_kwargs,
        process_kwargs=process_kwargs
    )
    # Write the result to a JSON file
    output = mesh_filepath.stem + "-spheres"
    with open(output + ".json", 'w') as f:
        f.write(dumps(spheres, indent=4, cls=
SphereEncoder))
```

Fig. 5. Example script to process a mesh using `foam`.

ization tools as the `foam` Python library and command-line tool, along with a Docker container to provide easy use to end users. The code is available in https://github.com/CoMMALab/foam/. A simple example script that demonstrates how `foam` can be used as a library to convert a mesh into a set of spheres is given in Fig. 5. After defining the `spherization_kwargs` and `process_kwargs`, the **spherize_mesh()** function is called and the spheres are written to a JSON file.

Beyond the `foam` library, we provide standalone scripts that can be used to spherize an input mesh or URDF:

```
$ python generate_spheres.py <mesh>
$ python generate_sphere_urdf.py <urdf>
```

`foam` has already proven useful for other publications. For example, `foam` was used to generate the spherized geometry for the UR5, Fetch, Baxter robots as well as some environments used in Thomason et al. [4], Ramsey et al. [44], and Quintero-Peña et al. [45].

IV. EMPIRICAL RESULTS

We observe `foam`'s effectiveness across a variety of fundamental robotics tasks in various simulators and collision checking backends. We selected MuJoCo [6], PyBullet [5], and Pinocchio [36] as these are widely used throughout the field—MuJoCo and PyBullet are popular off-the-shelf robotics-focused physics simulators due to their wide functionality, high fidelity, and accessibility. Pinocchio is a library for robot kinematics and dynamics modeling and uses the FCL library for collision checking [37]. We present results in Table I that show the difference in collision checking and signed distance query speed over these three libraries for five robots: the Hello Robot Stretch, the Franka Emika Panda, the KUKA IIWA, the Kinova Jaco, and the Rethink Robotics Baxter. Examples of the spherized models for the Panda, KUKA, and Kinova are shown in Fig. 1. While collision checking speed remains relatively consistent in PyBullet and MuJoCo between the spherized models and base models, signed distance query time is significantly faster in all engines. FCL in particular is highly compatible with the spherized primitive representation, giving large performance gains over all robots except for the Stretch.

Additionally, we also investigate how model simplification can affect physics iteration time in a parallelized simulator, Genesis [46], shown in Table II. The demand for parallelized simulators has seen a recent uptick as parallelized training for large models has become more prevalent, requiring a large-scale high-speed simulation for training and validation—simplified models may be a way to increase performance in certain tasks. Benchmarking the frames per second (FPS) in Genesis with 300 parallel simulations of the Franka Panda shows a 10% speedup when using a simplified model with 30 spheres, compared to the original model, shown in Table II. Spherization also improved FPS across all models. These results demonstrate that spherized models outperform the original, offering better computational efficiency in applications that demand numerous parallelized environments.
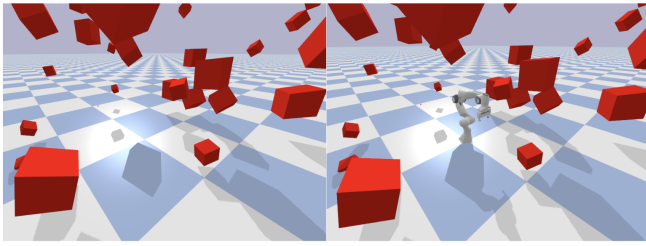
Fig. 6. Left: Random environment used to evaluate collision checking, visualized in PyBullet. Right: the same environment with the Franka Emika Panda loaded inside.

All experiments were performed on a PC with a 13th Gen Intel(R) Core(TM) i7-13700K CPU, an NVIDIA GeForce RTX 4070Ti, and 32GB of system memory.

V. DISCUSSION

We have presented `foam`, a library and tool to generate spherical approximations of robot geometry from an input URDF, addressing a critical limitation in existing tools to bridge general robots to methods that require spherical approximations. `foam` handles common mesh defects gracefully while preserving essential geometric features; our pipeline enables roboticists to easily convert complex CAD-derived meshes into computationally efficient primitive representations. `foam` also provides a number configurable parameters allow users to balance geometric fidelity with computational efficiency according to downstream requirements.

Our empirical results also demonstrate that spherical approximations of robot geometries provide significant computational advantages while maintaining acceptable geometric fidelity for practical robotics tasks. Across multiple standard robots (Franka Emika Panda, Baxter, KUKA IIWA, Kinova, and Stretch), we observed substantial performance improvements in collision checking times, particularly for distance queries, with speedups of up to two orders of magnitude in some cases (e.g., Kinova through FCL-based collision checking in the Pinocchio framework). These findings suggest that simplified robot geometry should be preferred for robotics tasks unless absolute geometric precision is essential, especially for applications involving signed distance fields or real-time control.

`foam` also opens the door for general robots to be applied to tools that require spherical models, such as cuRobo [3], GPMP2 [10], VAMP [4], and more. In the future, we plan to provide not only spherical approximations in `foam`, but also convex decompositions, cuboid decompositions, and other mesh processing tools. We are especially interested in other spherization algorithms that could be run more tightly "in-the-loop" to address situations that require faster spherization, such as dynamic manipulation requiring spherization of objects on the fly.

ACKNOWLEDGMENTS

REFERENCES

[1] J.-R. Chiu, J.-P. Sleiman, M. Mittal, F. Farshidian, and M. Hutter. "A collision-free mpc for whole-body dynamic locomotion and manipulation". In: *2022 international conference on robotics and automation (ICRA)*. IEEE. 2022, pp. 4686–4693.

[2] M. Gaertner, M. Bjelonic, F. Farshidian, and M. Hutter. "Collision-free MPC for legged robots in static and dynamic scenes". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 8266–8272.

[3] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. Van Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, et al. "CuRobo: Parallelized collision-free minimum-jerk robot motion generation". In: *arXiv preprint arXiv:2310.17274* (2023).

[4] W. Thomason, Z. Kingston, and L. E. Kavraki. "Motions in microseconds via vectorized sampling-based planning". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 8749–8756.

[5] E. Coumans and Y. Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2021.

[6] 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.

[7] D. Tola and P. Corke. "Understanding URDF: A dataset and analysis". In: *IEEE Robotics and Automation Letters* (2024).

[8] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. "Motion planning with sequential convex optimization and convex collision checking". In: *The International Journal of Robotics Research* 33.9 (2014), pp. 1251–1270.

[9] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. "Chomp: Covariant hamiltonian optimization for motion planning". In: *The International journal of robotics research* 32.9-10 (2013), pp. 1164–1193.

[10] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots. "Continuous-time Gaussian process motion planning via probabilistic inference". In: *The International Journal of Robotics Research* 37.11 (2018), pp. 1319–1340.

[11] G. Bradshaw and C. O'Sullivan. "Adaptive medial-axis approximation for sphere-tree construction". In: *ACM Transactions on Graphics (TOG)* 23.1 (2004), pp. 1–26.

[12] P. F. Felzenszwalb and D. P. Huttenlocher. "Distance transforms of sampled functions". In: *Theory of computing* 8.1 (2012), pp. 415–428.

[13] Q.-Z. Ye. "The signed Euclidean distance transform and its applications". In: *9th International conference on pattern recognition*. IEEE Computer Society. 1988, pp. 495–496.

[14] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart. "Signed distance fields: A natural representation for both mapping and planning". In: *RSS 2016 workshop: geometry and beyond-representations, physics, and scene understanding for robotics*. University of Michigan. 2016.

[15] J. Ortiz, A. Clegg, J. Dong, E. Sucar, D. Novotny, M. Zollhoefer, and M. Mukadam. "isdf: Real-time neural signed distance fields for robot perception". In: *arXiv preprint arXiv:2204.02296* (2022).

[16] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. "STOMP: Stochastic trajectory optimization for motion planning". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4569–4574.

[17] J. Dong, M. Mukadam, F. Dellaert, and B. Boots. "Motion planning as probabilistic inference using Gaussian processes and factor graphs." In: *Robotics: Science and Systems*. Vol. 12. 4. 2016.

[18] P. M. Hubbard. "Interactive collision detection". In: *Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium*. IEEE. 1993, pp. 24–31.

[19] P. M. Hubbard. "Approximating polyhedra with spheres for time-critical collision detection". In: *ACM Transactions on Graphics (TOG)* 15.3 (1996), pp. 179–210.

[20] A. Völz and K. Graichen. "Computation of collision distance and gradient using an automatic sphere approximation of the robot model with bounded error". In: *ISR 2018; 50th International Symposium on Robotics*. VDE. 2018, pp. 1–8.

[21] R. Wang, K. Zhou, J. Snyder, X. Liu, H. Bao, Q. Peng, and B. Guo. "Variational sphere set approximation for solid objects". In: *The Visual Computer* 22 (2006), pp. 612–621.

Robot	PyBullet			MuJoCo			Pinocchio (FCL)		
	Δ Col. (ms)	Δ Dis. (ms)	% Diff.	Δ Col. (ms)	Δ Dis. (ms)	% Diff.	Δ Col. (ms)	Δ Dis. (ms)	% Diff.
Stretch	-0.002	-4.286	-0.00001%	2.329	-96.189	0.00251%	0.649	-2.479	0.00386%
Panda	-0.002	-1.274	0.00000%	0.011	-14.785	0.00006%	-1.901	-2.558	0.00008%
KUKA	-0.000	-1.586	0.00000%	0.005	-1.985	0.00003%	-0.636	-27.824	0.00002%
Kinova	0.000	-4.309	0.00000%	-0.005	-0.233	0.00002%	-11.561	-44.469	0.00006%
Baxter	-0.001	-6.879	-0.00005%	7.363	-14.260	0.00008%	-1.431	-1.045	0.00014%

Table I. Comparison of collision checking and signed distance computation in three simulation and modeling frameworks: PyBullet [5], MuJoCo [6], and Pinocchio [36], which uses FCL [37] for collision checking. For each robot in each simulator, we evaluate the original robot URDF against the spherized URDF generated by `foam` over 100,000 queries in randomly generated environments, visualized in Fig. 6. We report the change in average query time for collision checking (Δ Col.) and the change in average distance query time (Δ Dis.) in milliseconds between the original and spherized model (negative meaning the spherized model is faster), and the percentage difference in collision check validity between the models (positive meaning more states are in collision with the spherized model). PyBullet implicitly convexifies all collision geometry and thus leads to the negative difference in collision, compared to the conservative but accurate spherized model generated by `foam`. We also note that sometimes the spherized model is less efficient for collision checking due to efficient spatial data structures used in collision testing, but distance queries are always faster.

Robot Model	Millions of FPS
Panda	5.20
11 Spheres	6.36
22 Spheres	5.36
27 Spheres	5.83
30 Spheres	5.73

Table II. Simulation framerate in millions of frames per second for a Panda grasping task in the Genesis [46] simulator. Using the spherized collision model provides around 10% speedup over base model using the 30 spheres.

[22] V. Angelidakis, S. Nadimi, M. Otsubo, and S. Utili. "CLUMP: a code library to generate universal multi-sphere particles". In: *SoftwareX* 15 (2021), p. 100735.

[23] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann Publishers, 2005.

[24] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and R. Rowe. "Collision detection: A survey". In: *2007 IEEE International Conference on Systems, Man and Cybernetics*. IEEE. 2007, pp. 4046–4051.

[25] C.-Y. Sun, Q.-F. Zou, X. Tong, and Y. Liu. "Learning adaptive hierarchical cuboid abstractions of 3d shape collections". In: *ACM Transactions on Graphics (TOG)* 38.6 (2019), pp. 1–13.

[26] K. Yang and X. Chen. "Unsupervised learning for cuboid shape abstraction via joint segmentation from point clouds". In: *ACM Transactions On Graphics (TOG)* 40.4 (2021), pp. 1–11.

[27] J.-M. Thiery, É. Guy, and T. Boubekeur. "Sphere-meshes: Shape approximation using spherical quadric error metrics". In: *ACM Transactions on Graphics (TOG)* 32.6 (2013), pp. 1–12.

[28] K. Mamou and F. Ghorbel. "A simple and efficient approach for 3D mesh approximate convex decomposition". In: *2009 16th IEEE international conference on image processing (ICIP)*. IEEE. 2009, pp. 3501–3504.

[29] X. Wei, M. Liu, Z. Ling, and H. Su. "Approximate convex decomposition for 3d meshes with collision-aware concavity and tree search". In: *ACM Transactions on Graphics (TOG)* 41.4 (2022), pp. 1–18.

[30] J. Gu, F. Xiang, X. Li, Z. Ling, X. Liu, T. Mu, Y. Tang, S. Tao, X. Wei, Y. Yao, et al. "Maniskill2: A unified benchmark for generalizable manipulation skills". In: *arXiv preprint arXiv:2302.04659* (2023).

[31] R. Wang, J. Zhang, J. Chen, Y. Xu, P. Li, T. Liu, and H. Wang. "Dexgraspnet: A large-scale robotic dexterous grasp dataset for general objects based on simulation". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 11359–11366.

[32] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. "Simplification envelopes". In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 119–128.

[33] M. Garland and P. S. Heckbert. "Surface simplification using quadric error metrics". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 1997, pp. 209–216.

[34] S. Calderon and T. Boubekeur. "Bounding proxies for shape approximation." In: *ACM Trans. Graph.* 36.4 (2017), pp. 57–1.

[35] Z. Chen, Z. Pan, K. Wu, E. Vouga, and X. Gao. "Robust low-poly meshing for general 3d models". In: *ACM Transactions on Graphics (TOG)* 42.4 (2023), pp. 1–20.

[36] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard. "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". In: *IEEE International Symposium on System Integrations (SII)*. 2019.

[37] J. Pan, S. Chitta, and D. Manocha. "FCL: A general purpose library for collision and proximity queries". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3859–3866.

[38] S. Haddadin. "The Franka Emika Robot: A Standard Platform in Robotics Research". In: *IEEE Robotics & Automation Magazine* (2024).

[39] J. Huang, H. Su, and L. Guibas. "Robust Watertight Manifold Surface Generation Method for ShapeNet Models". In: *arXiv preprint arXiv:1802.01698* (2018).

[40] J. Vollmer, R. Mencl, and H. Mueller. "Improved laplacian smoothing of noisy surface meshes". In: *Computer graphics forum*. Vol. 18. 3. Wiley Online Library. 1999, pp. 131–138.

[41] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. "Shapenet: An information-rich 3d model repository". In: *arXiv preprint arXiv:1512.03012* (2015).

[42] Dawson-Haggerty et al. *trimesh*. Version 3.2.0.

[43] A. Fishman, A. Murali, C. Eppner, B. Peele, B. Boots, and D. Fox. "Motion policy networks". In: *conference on Robot Learning*. PMLR. 2023, pp. 967–977.

[44] C. W. Ramsey, Z. Kingston, W. Thomason, and L. E. Kavraki. "Collision-Affording Point Trees: SIMD-Amenable Nearest Neighbors for Fast Collision Checking". In: *Robotics: Science and Systems*. 2024.

[45] C. Quintero-Peña, W. Thomason, Z. Kingston, A. Kyrillidis, and L. E. Kavraki. "Stochastic Implicit Neural Signed Distance Functions for Safe Motion Planning under Sensing Uncertainty". In: *IEEE International Conference on Robotics and Automation*. 2024, pp. 2360–2367.

[46] G. Authors. *Genesis: A Universal and Generative Physics Engine for Robotics and Beyond*. Dec. 2024.