TacEx: GelSight Tactile Simulation in Isaac Sim – Combining Soft-Body and Visuotactile Simulators

Duc Huy Nguyen[1,3], **Tim Schneider**[1,2], **Guillaume Duret**[1,2],
Alap Kshirsagar[1], **Boris Belousov**[3], **Jan Peters**[1,3,4]

[1]TU Darmstadt [2]École centrale de Lyon [3]DFKI [4]Hessian.AI
duc_huy.nguyen@dfki.de

Abstract: Training robot policies in simulation is becoming increasingly popular; nevertheless, a precise, reliable, and easy-to-use tactile simulator for contact-rich manipulation tasks is still missing. To close this gap, we develop TacEx – a modular tactile simulation framework. We embed a state-of-the-art soft-body simulator for contacts named GIPC and vision-based tactile simulators Taxim and FOTS into Isaac Sim to achieve robust and plausible simulation of the visuotactile sensor GelSight Mini. We implement several Isaac Lab environments for Reinforcement Learning (RL) leveraging our TacEx simulation, including object pushing, lifting, and pole balancing. We validate that the simulation is stable and that the high-dimensional observations, such as the gel deformation and the RGB images from the GelSight camera, can be used for training. The code, videos, and additional results will be released online https://sites.google.com/view/tacex.

Keywords: Vision-based Tactile Sensor, Simulation, Reinforcement Learning

1 Introduction

Tactile sensing plays an important role for human perception of touch [1] and for advanced manipulation tasks in robotics [2, 3, 4]. Contact properties such as contact geometry, object stiffness, and surface texture can be estimated using tactile sensors [5]. Furthermore, slip detection [6], hardness estimation [7], and grasping of soft objects [8] are facilitated by the sense of touch. However, finger coordination based on tactile feedback is a complex control problem with high-dimensional observation space, therefore several Deep RL approaches have been explored [9, 10]. A crucial bottleneck for applying RL to tactile-rich manipulation tasks is the lack of stable and reliable contact simulation that includes soft-body interaction and tactile sensing. Although a number of simulators have appeared recently that aim to remedy this issue [10, 11, 12, 13, 14], each simulator uses a different physics engine, simulates a different tactile sensor, different robot, and runs in a different robotics simulator altogether – making comparison and interoperability challenging.

To address these issues, we develop TacEx – a novel tactile simulation framework embedded in NVIDIA's Isaac Sim [15] and Isaac Lab [16, 17] that is modular, extensible, and based on the latest advancements in tactile simulation. We additionally integrate GIPC [18] for GPU-accelerated and inversion-free simulation of soft-body contacts. By leveraging Isaac Sim, we gain access to powerful features, such as photorealistic rendering, ROS support, and GPU-accelerated physics simulation, and by integrating TacEx into Isaac Lab – an extensible RL framework built on top of Isaac Sim – we enable support for teleoperation, GPU-parallelized training, and various RL libraries.

Related Work A simulation of a GelSight tactile sensor generally requires three components: physics simulation (to capture contact properties), optical simulation (to generate perceived RGB images), and marker simulation (to generate marker motion field, which reflects gel deformation). In this paper, we chiefly focus on the physics simulation, leveraging existing GelSight simulators

8th Conference on Robot Learning (CoRL 2024), Munich, Germany.

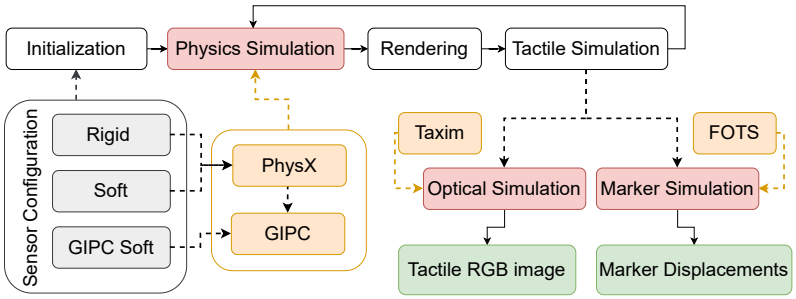


Figure 1: **Overview of the TacEx Tactile Simulation Pipeline**. First, the simulation is initialized according to a given Sensor Configuration. Then the physics are simulated using PhysX and GIPC, followed by the scene rendering. Finally, the tactile sensor is simulated using the optical simulation (Taxim) and marker simulation (FOTS), yielding a tactile RGB image and a marker displacements field. After this, the physics are simulated again and the process repeats.

for the other components: Taxim [11] for optical simulation and FOTS [19] for marker motion field simulation. For physics simulation, PyBullet's rigid body dynamics has been used in TACTO [20] and in [21], whereas [14] used a penalty-based contact model to approximate the soft gelpad deformation with rigid body dynamics. Though fast, these methods are less accurate compared to Finite Element Methods (FEM). More recent approaches rely on FEM: TacIPC [22] and [10] use the incremental potential contact (IPC) [23] model to simulate the gelpad deformation in an FEM-like manner. DiffTactile [13] also simulates the gelpad deformation with an FEM-based approach. Furthermore, the FEM simulation of Isaac Gym's Flex engine has also been used for accurate tactile simulation [24, 25], but it is slow and unsuited for RL.

Our approach is closest to the concurrent work TacSL [26] which also incorporates visuotactile simulation into Isaac Sim, however in contrast to TacSL, we leverage GIPC [18] for FEM-based soft-body simulation (instead of a simplified soft contact model). GIPC additionally allows for soft-to-soft contact simulation. Importantly, our method is modular, enabling the user to select which simulations should be enabled depending on the task requirements.

2 TacEx Simulation Framework

In this section, we present TacEx – our modular framework for tactile simulation (c.f., Fig. 4). We compare three different approaches for simulating the physical behavior of sensors and objects: i) PhysX to simulate the gelpad as a rigid body with compliant contact; ii) PhysX FEM-based soft body simulation for the gelpad; iii) GIPC [18] to simulate the gelpad as a soft body. Since PhysX is the built-in physics engine of Isaac Sim, baselines i) and ii) are straightforward to implement by directly setting asset properties; for iii), we modified the GIPC code and created Python bindings.

We integrate the GIPC simulation with Isaac Sim in the following manner. Isaac Sim is used for scene setup, robot simulation, and rendering. The gelpad is attached to the sensor case and moves kinematically in response to the robot's motion, which is handled by PhysX. The non-attached gelpad vertices are handled by GIPC: as the robot moves in Isaac Sim, the sensor case moves and the attachment points are recomputed, followed by a call to the GIPC solver that computes new positions for the remaining gelpad vertices and other GIPC-modeled objects. This enables the gelpad and objects to move, deform, and interact dynamically in Isaac Sim.

For optical simulation, we use Taxim [11]; specifically, a GPU-accelerated implementation [27]. First, we generate height maps with cameras in Isaac Sim and smooth them with pyramid Gaussian kernels. Then we use a polynomial lookup table to map the surface normals of the height maps to RGB values. As a final step, we attach shadows to the images. We further use the generated height maps to simulate the marker motion with FOTS [19]. For this, we compute the contact centers based on the height maps and extract the z rotation of the objects relative to the gelpads from Isaac Sim.

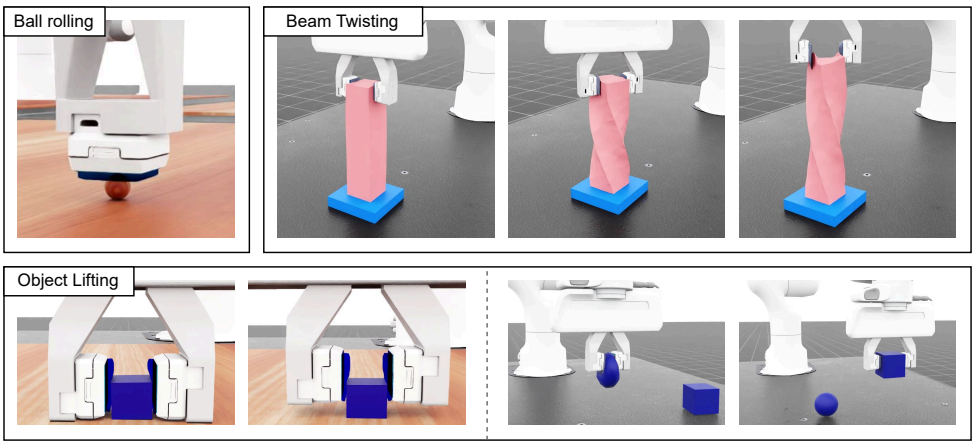


Figure 2: **Simulation Showcases**. We do a ball rolling experiment for testing the simulation performance, and we further evaluate the capabalities of our GIPC simulation by twisting and stretching a soft body beam and test how well the gelpads can be used for lifting objects (see website for videos).

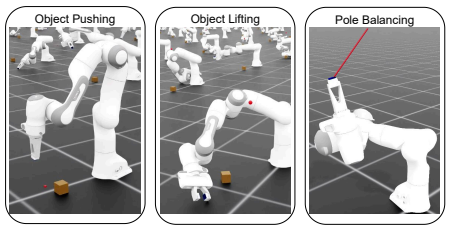
3 Demonstrations and Evaluation

We showcase the behavior, capabilities, and limitations of our framework in a series of experiments (visualized in Figure 2). First, a ball rolling experiment demonstrates a contact-rich manipulation task with a single GelSight sensor. Second, object lifting with two soft gelpads showcases robust grasping capabilities. Third, the limits of GIPC simulation are tested in a challenging beam twisting environment. Subsequently, we implement three RL tasks: object pushing, object lifting, and pole balancing – to demonstrate how TacEx can be used within Isaac Lab for RL training.

Ball Rolling. To showcase how the simulation behaves in a dynamic setting, we do a ball-rolling experiment, similar to [14]. A robot with a single GelSight Mini sensor uses the gelpad to roll the ball around. For this, we define goal positions for the end-effector and compute the required joint values with differential inverse kinematics. We can simulate about 18 robots at the same time with the rigid body configuration till we reach our VRAM limits due to camera simulations requiring much memory. When using soft-body GIPC-based simulation, we can only simulate a single robot properly due to our VRAM limit.

Object Lifting. In this example, we try to grasp and lift primitive objects using two GelSight sensors. The example reveals that the PhysX soft body setup cannot be used to grasp and lift objects. The objects always slip away even with several variations of the soft body parameters. Also, using a single soft body gelpad and a rigid body gelpad is unreliable for grasping and lifting. One reason for this failure is that the soft body simulation of PhysX currently does not support static friction.

Beam Twisting. We use this example to showcase the capabilities of the GIPC soft body simulation. A beam is simulated as a soft body and attached to a plate. The robot grasps the top of the beam with two soft body gelpads, twists and stretches the beam till it snaps back. This environment demonstrates that the simulation stays stable even under extreme deformations. Additionally, it showcases that friction is reasonably simulated.



RL environments. We implemented 3 environments in Isaac Lab and trained policies to validate that our framework can be used for Reinforcement Learning. In each environment, we used the marker displacements from our tactile simulation. For training the RL policies, we used PPO [28] as implemented in [29]. We have validated that the training pipeline works, and we are currently working towards obtaining successful policies that leverage tactile feedback.

Table 1: **Tactile Simulation Speed.** We measure the average simulation time per frame in ms for optical (GPU accelerated Taxim with shadows) and for marker simulation (FOTS running on CPU without parallelization) during contact in the ball rolling experiment with rigid body gelpads. We additionally measure the performance of the height map generation with the Isaac Sim USD cameras. We generate tactile RGB images with a resolution of 480×640 and simulate 10×10 markers per environment. We were only able to test up to approximately 18 environments, else we run into out-of-memory issues. We assume that the performance loss at 16 environments is related to our GPU being near its limit.

num_envs	height map gen	optical sim	marker sim
1	1.3718	5.9015	4.4863
2	0.8508	3.8886	2.8838
4	0.5988	3.0424	2.1184
8	0.4323	2.5773	1.7587
16	2.8827	5.7314	5.0450
18	3.5149	5.931	5.2343

Table 2: **PhysX Simulation Speed.** We measure the average simulation time per frame in ms for the physics simulation with PhysX during the ball rolling experiment (without tactile simulation). We run into out-of-memory issues with the soft body simulation at 256 environments. The soft body gelpad has a mesh resolution of 10 and uses 16 solver iterations.

num_envs	1	16	32	64	128	256	512	1024
rigid	3.6930	0.2426	0.1286	0.0673	0.0361	0.0212	0.0143	0.0093
soft	4.7069	0.4496	0.2718	0.1798	0.1267	-	-	-

Table 3: **GIPC Simulation Speed.**

num_vert	num_tetra	GIPC
1029	3717	24.95 ms
7900	40370	110.47 ms
12509	66563	221.61 ms

Speed Evaluations. We evaluate the simulation time of the optical simulation Taxim/FOTS in Table 1. Results for PhysX are presented in Table 2. The runtimes for soft-body GIPC simulation are shown in Table 3. We measure the average simulation time per frame in ms for the physics simulation with GIPC during the ball rolling experiment without tactile simulation. Compared to the ball rolling experiments with PhysX, the ball here is a soft body. We use different mesh resolutions for the ball to measure the performance w.r.t the amount of vertices and tetrahedra.

4 Conclusion and Future Work

We presented TacEx – a novel framework for simulating GelSight tactile sensors. The framework enables the usage of GelSight Mini sensors for Reinforcement Learning. It is built on top of Isaac Sim and Isaac Lab, which gives the user access to a wealth of features non-existent in current tactile simulators. We designed the framework to be modular, extendable, and easy to use. The framework integrates multiple different simulation approaches. The gelpad can either be simulated as a rigid body with compliant contact, or as a soft body. For the soft body simulation, one can use PhysX or our integration of GIPC. To simulate the sensor output, we create height maps with cameras in Isaac Sim and use the approach from Taxim [11] for the optical and the one from FOTS [19] for the marker simulation. We demonstrated framework features and simulation behavior with multiple examples. Additionally, we provide three environments for RL with tactile sensing.

A limitation of our current work is that it only contains qualitative experiments and demonstrations in simulation. Our tactile simulation, specifically the physics simulation approaches, lacks experiments that investigate whether they can be used for Sim2Real or not. Therefore, we aim to do more quantitative experiments for comparing different tactile simulation approaches, as well as Sim2Real experiments in future works. We also plan to extend our framework to include more RL environments and more tactile simulation approaches with the goal of creating a benchmarking platform for tactile simulators and algorithms for tactile sensing.

Acknowledgments

This research was supported by Research Clusters "The Adaptive Mind" and "Third Wave of AI", funded by the Excellence Program of the Hessian Ministry of Higher Education, Science, Research and the Arts. This work was supported by the French Research Agency, l'Agence Nationale de Recherche (ANR), and the German Federal Ministry of Education and Research (BMBF) through the project Aristotle (ANR-21-FAI1-0009-01). The authors acknowledge the grant "Einrichtung eines Labors des Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) an der Technischen Universität Darmstadt" of the Hessisches Ministerium für Wissenschaft und Kunst. The authors gratefully acknowledge the computing time provided to them on the high-performance computer Lichtenberg II at TU Darmstadt, funded by the BMBF and the State of Hesse.

References

[1] R. S. Dahiya, G. Metta, M. Valle, and G. Sandini. Tactile sensing—from humans to humanoids. *IEEE transactions on robotics*, 26(1):1–20, 2009.

[2] Z. Yu, W. Xu, S. Yao, J. Ren, T. Tang, Y. Li, G. Gu, and C. Lu. Precise Robotic Needle-Threading with Tactile Perception and Reinforcement Learning.

[3] A. Wilson, H. Jiang, W. Lian, and W. Yuan. Cable Routing and Assembly using Tactile-driven Motion Primitives.

[4] G. Wang, X. Liu, Z. Liu, P. Huang, and Y. Yang. Visual-Tactile Perception Based Control Strategy for Complex Robot Peg-in-Hole Process via Topological and Geometric Reasoning. 9(10):8410–8417. ISSN 2377-3766. doi:10.1109/LRA.2024.3436334.

[5] Q. Li, O. Kroemer, Z. Su, F. F. Veiga, M. Kaboli, and H. J. Ritter. A review of tactile information: Perception and action through touch. 36(6):1619–1634. ISSN 1941-0468. doi:10.1109/TRO.2020.3003230. URL https://ieeexplore.ieee.org/document/9136877/?arnumber=9136877. Conference Name: IEEE Transactions on Robotics.

[6] W. Yuan, R. Li, M. A. Srinivasan, and E. H. Adelson. Measurement of shear and slip with a GelSight tactile sensor. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 304–311. IEEE, . ISBN 978-1-4799-6923-4. doi:10.1109/ICRA.2015.7139016.

[7] W. Yuan, C. Zhu, A. Owens, M. A. Srinivasan, and E. H. Adelson. Shape-independent hardness estimation using deep learning and a GelSight tactile sensor. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 951–958, . doi:10.1109/ICRA.2017.7989116. URL http://arxiv.org/abs/1704.03955.

[8] Y. Han, K. Yu, R. Batra, N. Boyd, C. Mehta, T. Zhao, Y. She, S. Hutchinson, and Y. Zhao. Learning generalizable vision-tactile robotic grasping strategy for deformable objects via transformer. pages 1–13. ISSN 1941-014X. doi:10.1109/TMECH.2024.3400789. URL https://ieeexplore.ieee.org/document/10552075/?arnumber=10552075. Conference Name: IEEE/ASME Transactions on Mechatronics.

[9] A. Church, J. Lloyd, R. Hadsell, and N. F. Lepora. Deep Reinforcement Learning for Tactile Robotics: Learning to Type on a Braille Keyboard. 5(4):6145–6152. ISSN 2377-3766. doi:10.1109/LRA.2020.3010461.

[10] W. Chen, J. Xu, F. Xiang, X. Yuan, H. Su, and R. Chen. General-Purpose Sim2Real Protocol for Learning Contact-Rich Manipulation With Marker-Based Visuotactile Sensors. 40:1509–1526. ISSN 1552-3098, 1941-0468. doi:10.1109/TRO.2024.3352969.

[11] Z. Si and W. Yuan. Taxim: An Example-based Simulation Model for GelSight Tactile Sensors.

[12] Y. Lin, J. Lloyd, A. Church, and N. F. Lepora. Tactile Gym 2.0: Sim-to-real Deep Reinforcement Learning for Comparing Low-cost High-Resolution Robot Touch.

[13] Z. Si, G. Zhang, Q. Ben, B. Romero, Z. Xian, C. Liu, and C. Gan. DIFFTACTILE: A Physics-based Differentiable Tactile Simulator for Contact-rich Robotic Manipulation.

[14] J. Xu, S. Kim, T. Chen, A. R. Garcia, P. Agrawal, W. Matusik, and S. Sueda. Efficient tactile simulation with differentiability for robotic manipulation. In *6th Annual Conference on Robot Learning*.

[15] NVIDIA. Nvidia isaac sim, . URL https://developer.nvidia.com/physx-sdk.

[16] NVIDIA. Isaac lab, . URL https://isaac-sim.github.io/IsaacLab/.

[17] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi:10.1109/LRA.2023.3270034.

[18] K. Huang, F. M. Chitalu, H. Lin, and T. Komura. Gipc: Fast and stable gauss-newton optimization of ipc barrier energy. *ACM Trans. Graph.*, 43(2), 3 2024. ISSN 0730-0301. doi:10.1145/3643028. URL https://doi.org/10.1145/3643028.

[19] Y. Zhao, K. Qian, B. Duan, and S. Luo. FOTS: A Fast Optical Tactile Simulator for Sim2Real Learning of Tactile-motor Robot Manipulation Skills.

[20] S. Wang, M. Lambeta, P.-W. Chou, and R. Calandra. TACTO: A Fast, Flexible, and Open-source Simulator for High-Resolution Vision-based Tactile Sensors. 7(2):3930–3937. ISSN 2377-3766, 2377-3774. doi:10.1109/LRA.2022.3146945.

[21] A. Church and J. Lloyd. Tactile Sim-to-Real Policy Transfer via Real-to-Sim Image Translation.

[22] W. Du, W. Xu, J. Ren, Z. Yu, and C. Lu. TacIPC: Intersection- and Inversion-Free FEM-Based Elastomer Simulation for Optical Tactile Sensors. 9(3):2559–2566. ISSN 2377-3766, 2377-3774. doi:10.1109/LRA.2024.3357030.

[23] 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. 39(4). ISSN 0730-0301, 1557-7368. doi:10.1145/3386569.3392425.

[24] S. Cui, Y. Wang, S. Wang, Q. Li, R. Wang, and C. Zhang. Tactile Imprint Simulation of GelStereo Visuotactile Sensors. In *2023 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 650–656. IEEE. ISBN 9798350320848. doi:10.1109/ICMA57826.2023.10216245.

[25] G. Duret, F. Zara, J. Peters, and L. Chen. Toward synthetic data generation for robotic tactile manipulations. In *Workshop on "Robot Embodiment through Visuo-Tactile Perception" - 2024 IEEE International Conference on Robotics and Automation (ICRA) Conference Workshop*, Yokohama, Japan, May 2024. URL https://hal.science/hal-04566202.

[26] I. Akinola, J. Xu, J. Carius, D. Fox, and Y. Narang. Tacsl: A library for visuotactile sensor simulation and learning. *arXiv preprint arXiv:2408.06506*, 2024.

[27] T. Schneider. Taxim-gpu. URL https://git.ias.informatik.tu-darmstadt.de/tactile-sensing/taxim-gpu.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. URL https://arxiv.org/abs/1707.06347.

[29] L. Schneider, J. Frey, T. Miki, and M. Hutter. Learning risk-aware quadrupedal locomotion using distributional reinforcement learning, 2023.

[30] C. Higuera, B. Boots, and M. Mukadam. Learning to Read Braille: Bridging the Tactile Reality Gap with Diffusion Models.

[31] S. Zhong, A. Albini, O. P. Jones, P. Maiolino, and I. Posner. Touching a NeRF: Leveraging Neural Radiance Fields for Tactile Sensory Data Generation.

[32] W. D. Kim, S. Yang, W. Kim, J.-J. Kim, C.-H. Kim, and J. Kim. Marker-Embedded Tactile Image Generation via Generative Adversarial Networks. 8(8):4481–4488. ISSN 2377-3766, 2377-3774. doi:10.1109/LRA.2023.3284370.

[33] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull. Active Domain Randomization.

[34] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. doi:10.1109/IROS.2017.8202133.

[35] Y. Hu, T. Schneider, B. Wang, D. Zorin, and D. Panozzo. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi:10.1145/3386569.3392385. URL https://doi.org/10.1145/3386569.3392385.

[36] NVIDIA. Usd, fabric and usdrt. URL https://docs.omniverse.nvidia.com/kit/docs/usdrt/latest/docs/usd_fabric_usdrt.html.

A Related Work

A simulation of a GelSight tactile sensor generally requires three components: physics simulation (to capture contact properties), optical simulation (to generate perceived RGB images), and marker simulation (to generate marker motion field, which reflects gel deformation). This section provides a general overview of the approaches used by tactile simulators.

Physics Simulation Rigid body based approaches allow for fast simulations. TACTO [20] and Church and Lloyd [21] use PyBullet's rigid body simulation. Xu et al. [14] use a penalty-based contact model to approximate the soft gelpad's deformation with rigid body dynamics. While fast, they are inaccurate compared to Finite Element Methods (FEM). DiffTactile [13] simulates the gelpad deformation with a FEM-based approach. Chen et al. [10] and TacIPC [22] use incremental potential contact (IPC) [23] to simulate the gelpad deformation in a FEM-based manner. Additionally, the FEM simulation of Isaac Gym's Flex engine also has been used for tactile simulation [24, 25].

Optical Simulation Common for optical simulation approaches is the generation of a height map, which represents the surface of the gelpad and its deformation after contact. The gelpad deformation is approximated by smoothing the height map with, for example, pyramid Gaussian kernels, and then surface normals of the height map are mapped to RGB values (Taxim [11], DiffTactile [13], and FOTS [19]). Taxim uses a polynomial look-up table to map surface normals to RGB values. DiffTactile and FOTS use trained MLPs for the mapping. These methods need relatively few real tactile images to work, compared to Higuera et al. [30], for example. Here, tactile RGB images are generated via a diffusion model. Another example of a data-intensive approach is the work from Zhong et al. [31], which uses Neural Radiance Fields and a conditional Generative Adversarial Network.

Marker Simulation Chen et al. [10] and DiffTactile [13] use their accurate soft body simulation of the gelpad for the marker motion simulation. A mapping, which relates markers to faces of the tetrahedra mesh, is precomputed. If the gelpad deforms, the new marker world coordinates can be computed according to the vertex positions of the corresponding facet. A marker image is then created by projecting the marker world positions onto the image plane of a camera. Xu et al. [14] simulate the normal and shear tactile force fields. They compute the force fields with a penalty-based contact model. Simulated tactile force fields and the marker motion fields of real sensors are normalized before they are used as input for an RL policy. FOTS [19] simulates the marker motion field with exponential functions, which model the marker displacement distributions for normal, shear, and twist loads. Kim et al. [32] use a generative adversarial network that takes a sequence of depth images as input and outputs a tactile RGB image with markers.

B Choice of the Simulation Methods

For the physics simulation, we leverage PhysX. Not only because it is the built-in physics engine of Isaac Sim, but also due its fast GPU-accelerated simulation. We simulate the gelpad as a rigid body with compliant contacts for extremely fast tactile simulation. This can be extremely useful for prototyping new RL environments and algorithms, for example. Besides that, we simulate the gelpad as a soft body to have an approach with accurate gelpad simulation. Additionally, Isaac Sims's soft body simulation has not been used for tactile simulation yet. In this way, our work also provides a first study of the capabilities of Isaac Sims soft body simulation for simulating GelSight sensors. Unfortunately, some of our initial experiments revealed that the built-in soft body simulation is currently lacking in some aspects. We tried to grasp and pick up objects with soft body gelpads, but the objects were constantly slipping away. Even after extensive experimentation with different soft body parameters, this behavior did not change. One reason is the lack of static friction in their soft body simulation.

Therefore, we also wanted to integrate an external physics simulator into our framework. It would additionally serve as an example of the extensibility of our framework. But the question here is, which soft body simulation should we use? IPC [23] seemed to be a promising candidate. IPC is extremely robust. It guarantees intersection and inversion free simulation of soft bodies, regardless of material parameters and severity of deformation. This allows us to freely change parameters without worrying about unstable and inaccurate simulations. This is crucial for RL since one technique for closing the Sim2Real gap is domain randomization [33, 34]. Another benefit is that the need to fine-tune simulation parameters till reasonable behavior is achieved is omitted, or at least significantly reduced. IPC also simulates static and dynamic friction. Furthermore, it has already been shown that IPC can be used for accurate tactile simulation [10]. Instead of IPC, we use GIPC [18], a completely GPU-based variant of IPC with massive speedups.

For the optical simulation, we use the approach from Taxim [11] and for the marker simulation FOTS [19]. Both approaches are based on generating a height map, which approximates the gelpad deformation. Generating a height map is a common step for the optical simulation. By already having this step implemented, it is easier to integrate other optical simulation approaches. Additionally, both approaches do not rely on accurate simulation of the gelpad. Not only is this beneficial performance-wise but also for using different types of physics simulation. Compared to, for instance, the simulator from Chen et al. [10], we can simulate the marker motion field, even with a rigid body gelpad. Another benefit of Taxim and FOTS is that they can be easily adjusted to simulate other GelSight sensor models. Both use a relatively simple calibration process.

C Simulation Details

In this section, we describe our GIPC integration and our tactile simulation in more detail. Subsection C.1 describes the GIPC simulation pipeline in general and subsection C.2 the attachment creation. Subsection C.3 explains the steps of the optical and marker simulation.

C.1 General GIPC Simulation Pipeline

The general simulation pipeline with GIPC looks as follows. First, the GIPC simulation needs to be set up and initialized. This happens after initialization of the Isaac Sim simulation and generation of the scene. The scene generation involves spawning assets into Isaac Sim. For our GIPC simulation, we spawn assets without physical properties into Isaac Sim and then use them to create GIPC objects. To create a GIPC object out of an asset, we first extract the triangle mesh data of the corresponding USD mesh, i.e., world position and triangle indices of the mesh points. We then generate a tetrahedra mesh, which is required for the GIPC simulation. We use the Wildmeshing [35] python bindings for the tetrahedra generation. The topologies of the Isaac Sim USD meshes are updated according to the surface vertices and triangles of the tetrahedra meshes. This is necessary for the rendering of the objects.

After the initialization of the simulation, we compute the simulation state after a time step. For this, we first do a PhysX step, i.e., compute the new simulation state for objects simulated by PhysX. For instance, this involves the simulation of the robot's movement. The PhysX step is directly followed by a GIPC step. A step in our integrated GIPC simulation consists of first computing the new positions of attachment points. These values are set as target vertex values for the vertices that are attachment points. This allows us to move the GIPC objects kinematically. At the end of the GIPC step, we compute the new vertex positions for all GIPC objects with the GIPC solver. Additionally, we update the object position data by computing the mean of the new vertex positions. The object position data is helpful for RL environments as observations, for example.

To render the results of the GIPC simulations inside Isaac Sim, we update the position of the USD mesh vertices with the new computed vertex positions. For fast updates of the USD meshes, we use the USDRT [36] API. The rest is done by Isaac Sim's rendering engine. The general simulation pipeline with GIPC and PhysX is visualized in Figure 3.

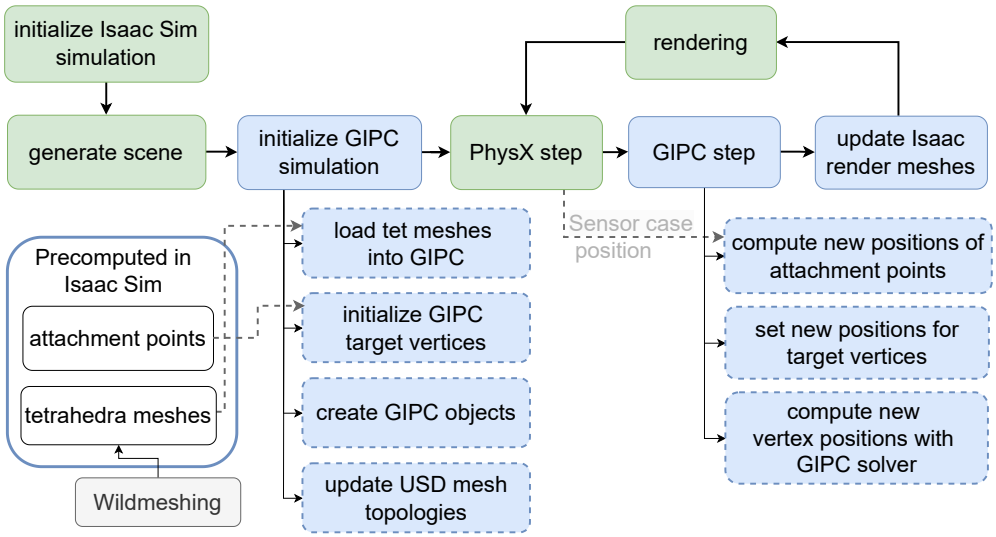


Figure 3: **Overview of our GIPC simulation pipeline.** First, the Isaac Sim simulation is initialized and the scene is created. Then the GIPC simulation is initialized. This includes, loading tetrahedra meshes into GIPC, creating GIPC objects, and updating the corresponding meshes in Isaac Sim. The tetrahedra (tet) meshes are computed with Wildmeshing. Attachment points for the gelpads are also precomputed. For the physics simulation the simulation state after a time step is computed. First, with PhysX, which leads to, for example, the robot moving, and then with GIPC. The GIPC simulation takes the newest position of the sensor case and uses it to compute the new positions of the attachment points. Then the GIPC solver computes the new vertex positions of every GIPC object. After that, the meshes in Isaac Sim are updated correspondingly and the scene is rendered.

For RL, we also need to reset the GIPC simulation occasionally. Resetting means bringing the scene back to the initial state. To achieve this, we save the initial positions of the GIPC vertices during the scene initialization. When the reset happens, we set the initial positions as the position of the vertices. The velocities are set to $(0, 0, 0)$.

C.2 GIPC Attachments

The two core questions regarding the attachment points are:

1. How do we find attachment points?
2. How do we compute the new vertex positions for the attachment points after each robot movement?

Finding attachment points means finding the IDs of vertices that should be attachment points. Attachment points should be the vertices inside the sensor case or at least close to it. To attach a GIPC object to the sensor case, we first query the world position and orientation of the sensor case. Secondly, we iterate through each point of the GIPC object's tetrahedra mesh (tet point) and do sphere ray casting in Isaac Sim with the PhysX scene query interface. A sphere with a specified radius is swept out from an origin point in a direction with the specified maximum. If the sphere hits a collider mesh, the impact point is returned. By using the world position of the tet points as the origin point, a very small sphere radius, and a very small maximum distance, we can check if a tet point is inside or close enough to a rigid body. This way, we find the attachment points.

For computing the new positions of the attachment points, we use that the relative positions of the attachment points to the sensor case position are constant. We precompute and save the offsets between attachment points and sensor case position. These offsets stay the same throughout the simulation. The attachment points positions are then updated by first querying the current pose, i.e.,

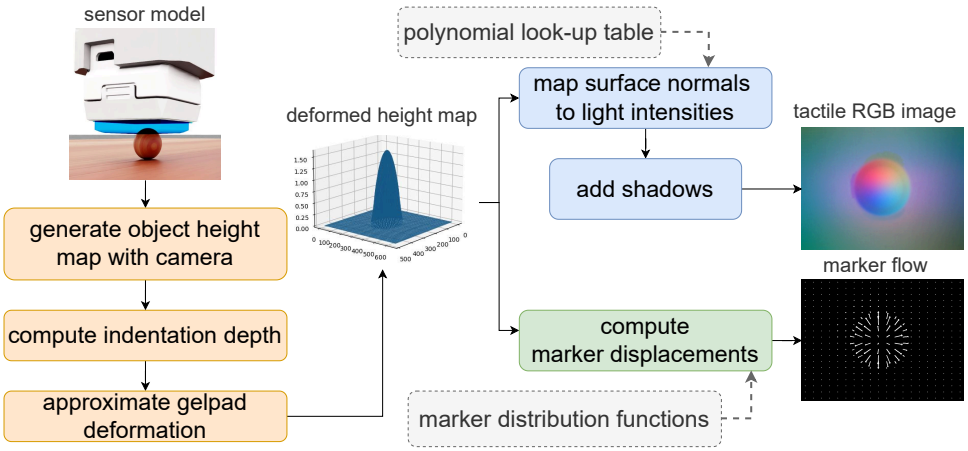


Figure 4: **Overview of our simulation pipeline for the sensor output.** We use approaches that rely on a height map which approximates the gelpad deformation. The height map generation is outlined in orange. For generating tactile RGB images, we use Taxim [11] (blue), which uses a polynomial look-up table. For the marker flow, we use FOTS [19] (green), which uses functions that model the marker displacements distributions.

position and orientation, of the rigid body. Then the attachment offsets are transformed based on the current pose. The transformed offset points are the new attachment point positions.

Computing which vertices are attachment points and the corresponding offsets happens before the simulation is run. For this, we wrote a script, which can be used in the Isaac Sim GUI. The attachment data is saved as USD properties and retrieved during the GIPC initialization.

C.3 Tactile Simulation

Our sensor model inside Isaac Sim contains a camera, which generates a height map of object surfaces inside the gelpad. The indentation depth is computed based on the height map and the gelpad thickness. Indentation depth and pyramid Gaussian kernels are used to approximate the gelpad deformation. For the tactile RGB image generation, the surface normals are mapped to light intensities with a polynomial look-up table. Then shadows are added to the RGB image to make the image more realistic. For the marker flow, we compute the marker displacement with exponential functions that model the marker displacements distribution under different loads. The main steps of the sensor output simulation are visualized in Figure 4.

D System Specification

We run the experiments from section 3 on an Ubuntu system with an AMD Ryzen 9 5950X 16-Core CPU, 32 GB RAM and an NVIDIA RTX 3080Ti GPU with 12 GB VRAM.