GPU-Accelerated Subsystem-Based ADMM for Large-Scale Interactive Simulation

Harim Ji, Hyunsu Kim, Jeongmin Lee, Somang Lee,
Seoki An, Jinuk Heo, Youngseon Lee, Yongseok Lee, and Dongjun Lee

***Abstract*— In this paper, we implement the GPU-accelerated subsystem-based Alternating Direction Method of Multipliers (SubADMM) for interactive simulation. The challenging objective for interactive simulations is to deliver realistic results under tight performance, even for large-scale scenarios. We aim to achieve this by exploiting the parallelizable nature of SubADMM to the fullest extent. We introduce a new subsystem division strategy to make SubADMM 'GPU friendly' along with custom kernel designs and optimization regarding efficient memory access patterns. We successfully implement the GPU-accelerated SubADMM and show the accuracy and speed of the framework for large-scale scenarios, highlighted with an interactive 'Hand demo' scenario. We also show improved robustness and accuracy compared to other state-of-the-art interactive simulators with several challenging scenarios that introduce large-scale ill-conditioned dynamics problems.**

I. INTRODUCTION

Interactive physics simulation has been essential for the fields of haptics [21] and virtual reality (VR) [16], with its importance also recognized in diverse fields such as interactive computer animation [7], digital twins, and mechanical components design [1]. More recently, this interactive physics simulation is considered the key enabler for the framework of Learning from Demonstration (LfD) since it can drastically improve sample efficiency, particularly for contact-involving and long-horizon tasks [20], for which it is difficult even to finish the task autonomously.

Arguably, the three most essential requirements for this interactive physical simulation would be speed, accuracy, and scalability. First, the simulation should deliver at least 60 frames per second (fps) speed [2] with the ratio of simulation time step size to computation time near or larger than one. Second, its accuracy to the ground truth (or real physical experiment results [22]) should be small enough for believable realism or good sim-to-real performance. Third, it should allow for implementing a large-scale environment with not much compromise of speed and accuracy. All these become particularly challenging when contacts and constraints are involved among the objects.

For this, various off-the-shelf simulators have been proposed over the past decades. PhysX [6] is a state-of-the-art simulator which is widely used as an interactive simulator in both robotics [15] and haptics & VR [24], [8], [14], [17], [19]. MuJoCo physics engine [23] shows the ability to be used as an interactive simulator with a reasonable level of accuracy for successful manipulation with a simulated human hand model from a simulation platform, HAPTIX [10]. However, there are limitations to these simulators. These simulators relax the actual dynamics problems to avoid tackling the non-linear complementarity problem (NCP) originating from constrained dynamics problems with contact conditions. These simulators also have limited scalability since the underlying solvers for these simulators are not directly parallelizable. While XPBD can utilize parallelizable constraint projection, Jacobi solve, this introduces a slow convergence.

Fig. 1: Overview of the 'Hand demo' scenario. The scenario contains a 281 number of rigid bodies dynamically coupled with an average number of 1773 contacts and 33 joint constraints.

In this context, we present a novel framework that tackles the exact NCP while still being interactive and scalable for large-scale scenarios. This framework is based on a GPU implementation of the recently proposed subsystem-based Alternating Direction Method of Multipliers (SubADMM [12]) algorithm. Since SubADMM can solve the exact dynamics problem and naturally handles operations for each subsystem and constraint in parallel, we exploit this parallelizable nature to the fullest extent by accelerating the SubADMM with GPU. We explain how parallelized computation can be equally distributed among threads with technical details to achieve accurate and interactive behavior. We demonstrate the 'Hand demo' scenario, shown in Fig. 1, and other challenging scenarios, showing that our framework can manage diverse large-scale scenarios accurately at interactive rates that are not achievable with current simulators.

II. GPU-ACCELERATED SUBADMM

A. Subsystem-Based ADMM

In this paper, we utilize the SubADMM framework for its ability to tackle the NCP while handling operations for each subsystem and constraint in parallel. Here, we explain the details of the SubADMM framework.

SubADMM formulates the discrete constrained dynamics problem as an optimization problem based on augmented Lagrangian by introducing slack variables $x_k \in \mathbb{R}^{n_{c,k}}$,

*This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) (RS-2022-00144468, RS-2023-00208052) and the Ministry of Trade, Industry & Energy (MOTIE) of Korea (RS-2024-00419641).

The authors are with the Department of Mechanical & Aerospace Engineering, IAMD and IOER, Seoul National University, Seoul, Republic of Korea. {jiharim0911, hyunsu.kim, ljmlgh, hopelee, seoki97s, jinukheo, yslee1765, yongseoklee, djlee}@snu.ac.kr

$z_k \in \mathbb{R}^{n_{c,k}}$ and Lagrange multiplier $u_k \in \mathbb{R}^{n_{c,k}}$ for each kth constraint. This optimization problem is solved using the ADMM [3] by executing the following iterative steps:

Step 1 Velocity update:
The representative velocity of the ith subsystem is updated as follows:

$$\forall i,\ \left(A_i + \beta_i \sum_{k \in \mathcal{I}_i} J_{i,k}^T J_{i,k}\right) \hat{v}_i^{l+1} = b_i + \sum_{k \in \mathcal{I}_i} J_{i,k}^T \left(\beta_i z_{i,k}^l - u_{i,k}^l\right) \tag{1}$$

where the superscript l denotes the iteration step and $\beta_i \in \mathbb{R}$ is the penalty weight of the ith subsystem.
Step 2 x-update:
For ith subsystem with kth constraint, $x_{i,k}$ represents the constraint space velocity, which is updated as below:

$$\forall k,\ x_{i,k}^{l+1} = J_{i,k} \hat{v}_i^{l+1} \tag{2}$$

Step 3 z-update:
Suppose the kth constraint involves subsystem i and j, that is, $f_k = i,\ s_k = j$. This step updates $z_{i,k}^{l+1}$ and $z_{j,k}^{l+1}$ to satisfy the constraint condition described as:

$$\forall k,\ (z_{i,k}^{l+1} + z_{j,k}^{l+1}, \lambda_k^{l+1}) \in \mathcal{C}(e_k) \tag{3}$$

This can be done with the following update rules:

$$\begin{aligned} \beta_i z_{i,k}^{l+1} &= \underbrace{\beta_i x_{i,k}^{l+1} + u_{i,k}^l}_{y_{i,k}^{l+1}} + \lambda_k^{l+1} \\ \beta_j z_{j,k}^{l+1} &= \underbrace{\beta_j x_{j,k}^{l+1} + u_{j,k}^l}_{y_{j,k}^{l+1}} + \lambda_k^{l+1} \end{aligned} \tag{4}$$

where $\lambda_k^{l+1} \in \mathbb{R}^{n_{c,k}}$ can be obtained by simple scalar operations for each of three types of constraints :

– Soft constraint :

$$\lambda_k^{l+1} = -\frac{k e_k + c\left(\beta_i^{-1} y_i^{l+1} + \beta_j^{-1} y_j^{l+1}\right)}{1 + c\left(\beta_i^{-1} + \beta_j^{-1}\right)}$$

– Hard constraint :

$$\lambda_k^{l+1} = -\frac{e_k + \beta_i^{-1} y_i^{l+1} + \beta_j^{-1} y_j^{l+1}}{\beta_i^{-1} + \beta_j^{-1}}$$

– Contact constraint :

$$\lambda_k^{l+1} = \Pi_{\mathcal{C}}\left(-\frac{e_k + \beta_i^{-1} y_i^{l+1} + \beta_j^{-1} y_j^{l+1}}{\beta_i^{-1} + \beta_j^{-1}}\right)$$

where $\Pi_{\mathcal{C}}$ denotes a strict projection on the friction cone [11].

Step 4 u-update:
The Lagrangian multiplier for the ith subsystem with the kth constraint, $u_{i,k}$, is updated with a simple computation:

$$\forall k,\ u_{i,k}^{l+1} = u_{i,k}^l + \beta_i \left(x_{i,k}^{l+1} - z_{i,k}^{l+1}\right) \tag{5}$$

Notice that **Step 1** can be done subsystem-wise in parallel while **Step2~Step4** can be done constraint-wise in parallel.

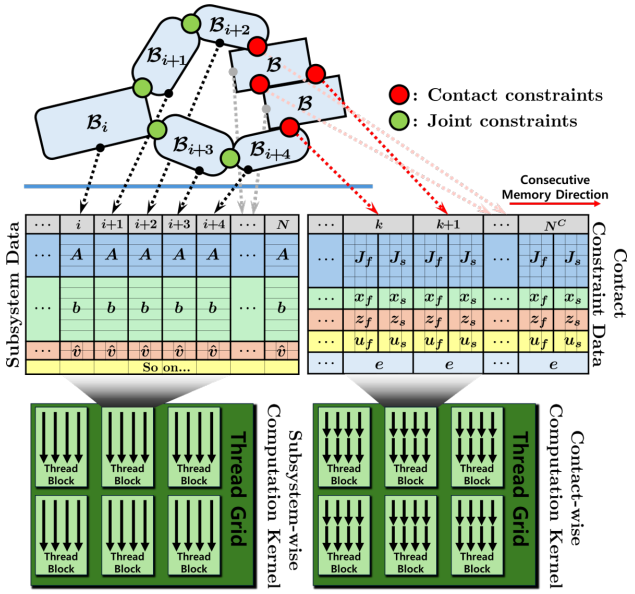

Fig. 2: GPU-accelerated SubADMM overview under a scenario with N number of subsystems, N^C number of contact constraints, and N^J number of joint constraints. Each subsystem and contact constraint is designated a memory region with the same layout pattern. GPU threads are assigned to each memory region for parallelized computation. This structure is the same for joint constraints, which is not shown in the figure. Notice that the contact-wise computation kernel is generally assigned more threads than the subsystem-wise computation kernel in large-scale scenarios.

In this work, we utilize this parallelizable nature of SubADMM. We adopt a new subsystem division strategy and implement GPU-accelerated SubADMM using the CUDA Toolkit [5]. By utilizing the massive computation power of modern GPUs, we achieve sub-linear scalability and handle large scenarios (a few thousand bodies) at an interactive rate.

B. Subsystem Division

The overall parallelization scheme of GPU-accelerated SubADMM is shown in Fig. 2. We set every subsystem as an individual rigid body with its origin in the center of mass. This lets every subsystem have an equal 6 degrees of freedom (DoF) and makes all matrices and vectors within the same category (e.g., contact Jacobian, slack variables, etc.) equally sized and computed with a unified rule. Thus, we can distribute every parallelizable computation, explained in Sec. II-A, equally along the threads with minimized divergence (each thread doing the same computations) between threads and straightforwardly design a global memory layout for coalesced memory access patterns (consecutive threads access consecutive memories). This enables SubADMM to handle large-scale scenarios with sub-linear scalability with modern GPUs. From the N number of subsystems, we introduce a mass matrix $A_i \in \mathbb{R}^{6\times6}$ and a dynamics vector $b_i \in \mathbb{R}^6$ for $i = 1 \ldots N$. Notice that the mass matrix, A_i, is constant, diagonal, and only 4 values can be stored: m, I_{xx}, I_{yy}, and I_{zz}. The dynamics vector b_i includes the effect of momentum, Coriolis force, and external wrench from user input such as virtual coupling [4].

C. GPU Kernel

The detailed algorithm is shown in Fig. 3. We design custom GPU kernels for every block labeled with a bold alphabet appearing in Fig. 3, where the number in the

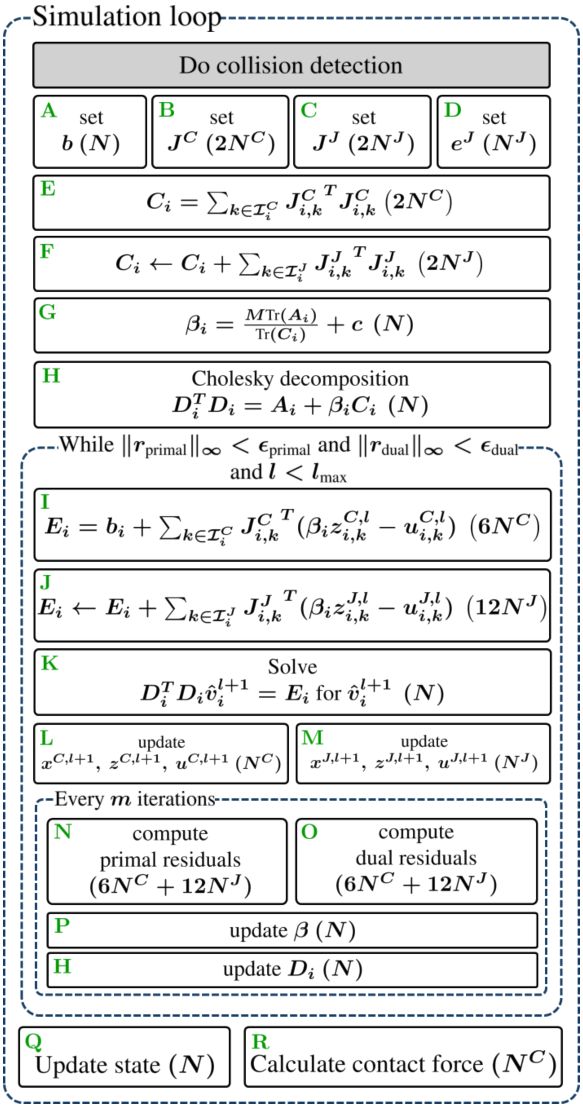

Fig. 3: Kernels and kernel scheduling for complete simulation loop including collision detection and GPU-accelerated SubADMM. Each block labeled with a bold alphabet letter represents a custom-built kernel.

parenthesis is the number of GPU threads assigned for the kernel call. When assembling the C_i matrices for each body, as shown in the kernels labeled as **E** and **F** in Fig. 3, we parallelize the execution at the constraint level rather than the body level. This can further exploit the computation power of the GPU since the number of constraints is generally much larger than the number of bodies, and it can also reduce thread divergences coming from different numbers of constraints between the bodies. However, naively using the add operation can cause race conditions (different threads writing on the same memory simultaneously) since different threads for different constraints may add to the same C_i matrix for the same body. We thus use the `atomicAdd` operation provided by the CUDA Toolkit [18] to avoid this issue. This style of kernel design is also used for assembling $E_i \in \mathbb{R}^6$ vectors for each ith subsystem as shown in the kernels labeled as **I** and **J** and constructing the dual residual, shown in the kernel labeled as **O**. Notice that some kernels that computes b_i, $J_{i,k}^C$, $J_{i,k}^J$, and e_k^J, each labeled as **A**, **B**, **C**, and **D**, are decoupled to each other. In this case,

Fig. 4: Experiment setup and scene captures for 'Hand demo' scenario.

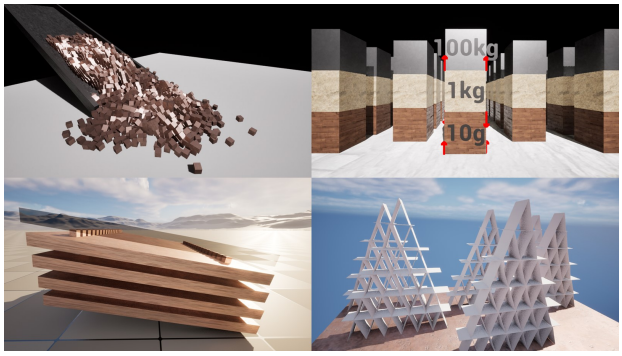

Fig. 5: Scene capture for stress test scenarios. From left to right, top to bottom: 'Sliding cubes,' 'Vertical stacks,' 'Oblique stacks,' and 'Card houses' after 912 seconds of simulation time. For 'Vertical stacks,' contact forces for the front boxes are rendered as red arrows.

we can schedule the kernels to be called asynchronously to keep as many threads busy as possible. This asynchronous scheduling is also used to schedule kernels that update the slack variables and the Lagrangian multipliers for each contact constraint and joint constraint, labeled as **N** and **O**, and kernels that update the state of each body and calculate the contact forces, labeled as **Q** and **R**.

III. Experiments

In this section, we conduct several test scenarios. Every scenario is run with an Intel Core i9-13900K CPU and an RTX 4080 GPU. Implementation is done with a custom-built Unreal Engine 5.3 plugin, rendering quality set to 'High.' The input of the hand motion used in the 'Hand demo' scenario, which will be described in Sec. III-A, is obtained using Visual Inertial Skeletal Tracking (VIST) [13]. The overall results and settings for each scenario are shown in Table. I. We also recommend that the readers see the supplemental video for more details.

A. Hand demo

To highlight our simulator's ability to handle large-scale scenarios accurately and interactively, we implement a 'Hand demo' scenario in which a user can interact with the simulation in real-time using a virtual hand. This scenario includes tasks such as dexterous manipulation with various objects (apple, small cube, solderer, thin spoon, wooden toy car, etc.), stacking objects (tricky cube), and grasping

TABLE I: Settings and performance of the solver for test scenarios described in Sec. III. Residuals and time cost (GPU-accelerated SubADMM) are calculated as the average for the scenario's number of frames.

Scenario	DoFs	Iterations	δt (ms)	Time Cost (ms)	$\|r_{\text{primal}}\|_\infty$	$\|r_{\text{dual}}\|_\infty$	Contact Average (Max)	Joints
Hand demo ($m = 20$)	1686	100	4	5.00	1.48E-5	3.99E-5	1773(1901)	33
Sliding 100 cubes ($m = 30$)	600	120	5	3.82	2.26E-7	2.05E-7	488(456)	-
Sliding 500 cubes ($m = 30$)	3000	120	5	4.15	3.75E-6	3.62E-6	3864(3968)	-
Sliding 1000 cubes ($m = 30$)	6000	120	5	4.85	5.19E-6	5.10E-6	8961(9140)	-
Sliding 1500 cubes ($m = 30$)	9000	120	5	5.26	9.37E-6	9.16E-6	14813(15972)	-
Sliding 2000 cubes ($m = 30$)	12000	120	5	5.51	1.05E-5	1.72E-5	21156(22660)	-
Vertical stacks ($m = 50$)	2160	200	5	5.81	1.09E-5	7.47E-05	1438(1440)	-
Oblique stacks ($m = 30$)	840	150	5	4.52	6.97E-7	8.62E-6	1087(1088)	-
Card houses ($m = 100$)	2376	300	1	9.12	3.06E-8	6.64E-7	2551(2680)	-

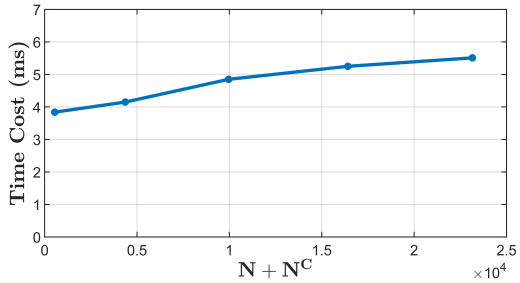

Fig. 6: Time cost of GPU-accelerated SubADMM for sliding cubes scenarios

heavy objects (golden cube, Stanford bunny). To deliver physically realistic results, the simulator should solve the NCP with hundreds of objects and thousands of constraints in real-time. It also should be robust for ill-conditioned problems arising from the interaction between odd mass ratios, such as between the fingertip (10g) and the golden cube (1kg) or the Stanford bunny (1.5kg). As shown in Table. I, our simulator can achieve such tight requirements. Our framework can provide interaction force data during the simulation, showing potential use for a rich data collection from expert demonstration. The detailed results are provided in the supplemental video.

B. Stress test

1) Scalability: We test the scalability of GPU-accelerated ADMM with a 'Sliding cubes' scenario. We test the scenario with 100, 500, 1000, 1500, and 2000 cubes. As shown in Fig. 6, GPU-accelerated SubADMM demonstrates sub-linear scalability regarding the number of bodies and contact constraints.

2) Robustness under odd mass ratio: As described in Sec. III-A, the interaction between odd mass ratios should be robustly handled for interactive simulations. We further test the robustness under odd mass ratios with two scenarios: 'Vertical stacks' and 'Oblique stacks.' In 'Vertical stacks, ' a 100kg box, slightly perturbed with a pitch angle, is dropped on top of a 1kg box sitting on a 10g box. The scenario contains 120 number of these stacks. In 'Oblique stacks,' 10kg plates are stacked upon 50g cubes. The friction coefficient of each plate and cube is set to 3, which is enough to prevent sliding. These scenarios are simulated successfully, showing that our framework can handle such large-scale and ill-conditioned problems in real-time. We test the same scenarios using Chaos and other state-of-the-art interactive simulators, PhysX and Mujoco, explained in Sec. I, with an extensive number of iterations. For 'Vertical stacks,' every simulator fails to stack the heavy block, as shown in Fig. 7. For 'Oblique stacks,' Chaos exhibits springy

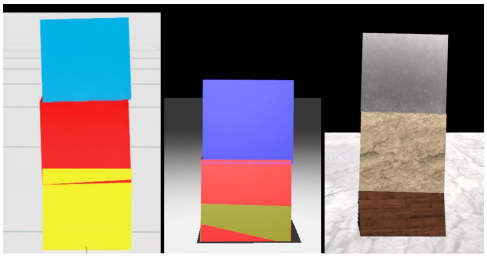

Fig. 7: Robustness test with odd mass ratios. From left to right: PhysX, Mujoco, and Chaos.

motion but successfully maintains the stack, while PhysX and Mujoco show sliding even if we increase the friction coefficient to 10. To enable stacking, PhysX needs techniques such as 'sleeping' (ignoring consecutive small velocities), and Mujoco needs an additional 'No slip iteration' but with a large trade-off (more than 10ms) in computation time. The detailed settings and results are provided in the supplemental video.

3) Stacking stability: We test the stacking stability of our simulator with a 'Card houses' scenario. The challenging aspects of this scenario are well explained in [9]. The six card houses are initially shocked due to small gaps between each card in the first frame, and quickly stabilize. We observe stable behavior for more than 15 minutes of simulation time even though no stabilization methods, such as sleeping, are used.

IV. CONCLUSIONS

In this paper, we implement the GPU-accelerated SubADMM, which exploits the parallelizable nature of SubADMM to the fullest extent. We introduce a new subsystem division strategy to make SubADMM 'GPU friendly' and apply optimization techniques with GPU programming. With an interactive 'Hand demo' scenario, we demonstrate the ability of this new framework to handle large-scale scenarios accurately at an interactive rate. We also show improved scalability compared to the original SubADMM and accuracy and robustness compared with several state-of-the-art simulators with challenging scenarios. This work can be used to tackle challenging problems in robotics. Utilizing our framework for an efficient collection of expert demonstrations of hand-object manipulation, including contact force measurements and fast roll-out of a control policy under complicated environments, seems to be a promising and interesting direction.

REFERENCES

[1] Inc Ansys. Ansys vrxperience hmi. https://www.ansys.com/content/dam/product/optical/vrxperience/ansys-vrxperience-hmi-datasheet.pdf, 2024. Accessed: 2024-09-15.

[2] Jan Bender, Kenny Erleben, and Jeff Trinkle. Interactive simulation of rigid body dynamics in computer graphics. In *Computer Graphics Forum*, volume 33, pages 246–270. Wiley Online Library, 2014.

[3] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[4] J Edward Colgate, Michael C Stanley, and J Michael Brown. Issues in the haptic display of tool use. In *Proceedings 1995 IEEE/RSJ international conference on intelligent robots and systems. Human robot interaction and cooperative robots*, volume 3, pages 140–145. IEEE, 1995.

[5] Nvidia Corporation. Cuda c++ programming guide. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`, 2024. Accessed: 2024-09-01.

[6] Nvidia Corporation. Nvidia physx. `https://developer.nvidia.com/physx-sdk`, 2024. Accessed: 2024-08-27.

[7] DeepMotion. Physics based multiple 3 point tracked vr avatar interaction. `https://www.youtube.com/watch?v=SaGezfGzFQs`, January 2018. YouTube video.

[8] Markus Höll, Markus Oberweger, Clemens Arth, and Vincent Lepetit. Efficient physics-based implementation for realistic hand-object interaction in virtual reality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 175–182, 2018.

[9] Danny M Kaufman, Shinjiro Sueda, Doug L James, and Dinesh K Pai. Staggered projections for frictional contact in multibody systems. In *ACM SIGGRAPH Asia 2008 papers*, pages 1–11. 2008.

[10] Vikash Kumar and Emanuel Todorov. Mujoco haptix: A virtual reality system for hand manipulation. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 657–663. IEEE, 2015.

[11] Jeongmin Lee, Minji Lee, and Dongjun Lee. Large-dimensional multibody dynamics simulation using contact nodalization and diagonalization. *IEEE Transactions on Robotics*, 39(2):1419–1438, 2022.

[12] Jeongmin Lee, Minji Lee, and Dongjun Lee. Modular and parallelizable multibody physics simulation via subsystem-based admm. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10132–10138. IEEE, 2023.

[13] Yongseok Lee, Wonkyung Do, Hanbyeol Yoon, Jinuk Heo, WonHa Lee, and Dongjun Lee. Visual-inertial hand motion tracking with robustness against occlusion, interference, and contact. *Science Robotics*, 6(58):eabe1315, 2021.

[14] Christos Lougiakis, Jorge Juan González, Giorgos Ganias, Akrivi Katifori, Ioannis-Panagiotis, and Maria Roussou. Comparing physics-based hand interaction in virtual reality: Custom soft body simulation vs. off-the-shelf integrated solution. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 743–753, 2024.

[15] Malte Mosbach, Kara Moraw, and Sven Behnke. Accelerating interactive human-like manipulation learning with gpu-based simulation and high-quality demonstrations. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 435–441. IEEE, 2022.

[16] Syed T Mubarrat, Antonio Fernandes, and Suman K Chowdhury. A physics-based virtual reality haptic system design and evaluation by simulating human-robot collaboration. *IEEE Transactions on Human-Machine Systems*, 2024.

[17] Kiran Nasim and Young J Kim. Physics-based assistive grasping for robust object manipulation in virtual reality. *Computer Animation and Virtual Worlds*, 29(3-4):e1820, 2018.

[18] NVIDIA. *CUDA C Programming Guide - Atomic Functions*, 2024. Accessed: 2024-09-14.

[19] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, John A Castro-Vargas, Sergio Orts-Escolano, and Jose Garcia-Rodriguez. A visually realistic grasping system for object manipulation and interaction in virtual reality environments. *Computers & Graphics*, 83:77–86, 2019.

[20] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[21] Kenneth Salisbury, Francois Conti, and Federico Barbagli. Haptic rendering: introductory concepts. *IEEE computer graphics and applications*, 24(2):24–32, 2004.

[22] Dongwon Son, Hyunsoo Yang, and Dongjun Lee. Sim-to-real transfer of bolting tasks with tight tolerance. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9056–9063. IEEE, 2020.

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[24] Mickeal Verschoor, Daniel Lobo, and Miguel A Otaduy. Soft hand simulation for smooth and robust natural interaction. In *2018 IEEE conference on virtual reality and 3D user interfaces (VR)*, pages 183–190. IEEE, 2018.