Learning to Assemble with Alternative Plans

ZIQI WANG, EPFL & HKUST, Hong Kong, China
WENJUN LIU, HKUST, Hong Kong, China
JINGWEN WANG, EPFL, Switzerland
GABRIEL VALLAT, EPFL, Switzerland
FAN SHI, NUS, Singapore
STEFANA PARASCHO, EPFL, Switzerland
MARYAM KAMGARPOUR, EPFL, Switzerland

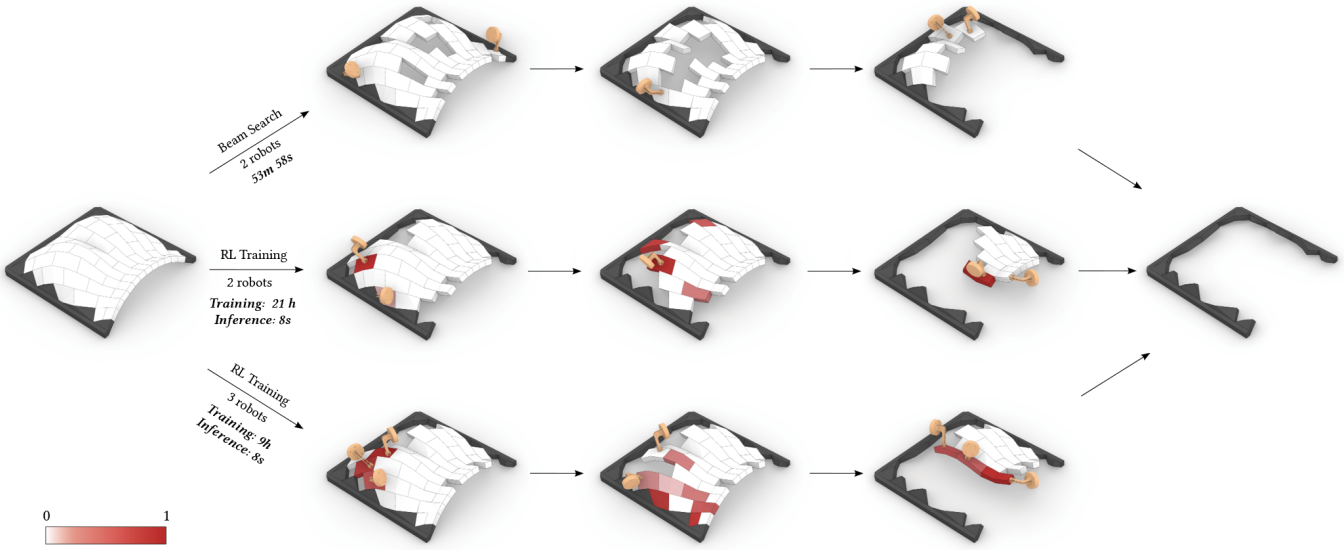


Fig. 1. We plan the assembly process of the Vault from [Deuss et al. 2014] using our reinforcement learning (RL) framework, which consists of 62 parts. By employing two or three robots, the structure remains stable throughout the assembly, eliminating the need for additional scaffolding. The first row shows results from a classic search-based method, while the second and third rows present results generated by our RL-based approach. Our framework adopts an assembly-by-disassembly strategy, in which every generated disassembly plan can be reversed to produce a feasible assembly plan. Only the grippers of the robots are visualized, while their bodies are omitted. The color highlights on the parts indicate the probabilities of the following disassembly actions as proposed by our RL agents. This work considers two types of disassembly actions: a robot holding a part and a robot removing a part. Assuming all robots are identical, these two actions cannot be performed simultaneously on the same part. Their action probabilities are visualized together. The second and fourth columns depict disassembly steps where all robots are occupied. The third column depicts steps where one of the robots is free to hold a new part. Training the RL policies takes 21 hours for 2 robots and 9 hours for 3 robots. Using three robots to disassemble the Vault is less challenging than using two, which leads to faster training. Once trained, the RL policies can generate alternative disassembly plans in 8 seconds. This is 400 times faster than re-running a search-based method, which takes 53 minutes.

Authors' addresses: Ziqi Wang, EPFL & HKUST, Hong Kong, China, ziqiw@ust.hk; Wenjun Liu, HKUST, Hong Kong, China, wenjunliu@ust.hk; Jingwen Wang, EPFL, Lausanne, Switzerland, jingwen.wang@epfl.ch; Gabriel Vallat, EPFL, Lausanne, Switzerland, gabriel.vallat@epfl.ch; Fan Shi, NUS, Singapore, fan.shi@nus.edu.sg; Stefana Parascho, EPFL, Lausanne, Switzerland, stefana.parascho@epfl.ch; Maryam Kamgarpour, EPFL, Lausanne, Switzerland, maryam.kamgarpour@epfl.ch.

We present a reinforcement learning framework for constructing assemblies composed of rigid parts, which are commonly seen in many historical masonry buildings and bridges. Traditional construction methods for such structures often depend on dense scaffolding to stabilize their intermediate assembly steps, making the process both labor-intensive and time-consuming. This work utilizes multiple robots to collaboratively assemble structures, offering temporary support by holding parts in place without additional scaffolding. Precomputing the robotic assembly process to ensure structural

ACM 0730-0301/2025/8-ART
https://doi.org/10.1145/3730824

stability involves a time-consuming offline process due to the combinatorial nature of its search space. However, the precomputed assembly plans may get disrupted during real-world execution due to unforeseen changes, such as setup modifications or delays in part delivery. Recomputing these plans using traditional offline methods results in significant project delays. Therefore, we propose a reinforcement learning-based approach in which a neural network is trained to efficiently generate alternative assembly plans for a given structure online, enabling adaptation to external changes. To enable effective and efficient training, we introduce three key innovations: a GPU-based stability simulator for parallelizing simulations, a novel curriculum-based training scheme to address sparse rewards during training, and a new graph neural network architecture for efficiently encoding assembly geometry. We validate our approach by training reinforcement learning agents on various assemblies and evaluating their performance on unseen assembly tasks. Furthermore, we demonstrate the effectiveness of our framework in planning multi-robot assembly processes, effectively handling disruptions in both simulation and physical environments.

CCS Concepts: • **Computing methodologies → Planning for deterministic actions**; **Massively parallel and high-performance simulations**.

Additional Key Words and Phrases: Assembly sequence planning, reinforcement learning, GPU-accelerated simulation.

ACM Reference Format:
Ziqi Wang, Wenjun Liu, Jingwen Wang, Gabriel Vallat, Fan Shi, Stefana Parascho, and Maryam Kamgarpour. 2025. Learning to Assemble with Alternative Plans. *ACM Trans. Graph.* 44, 4 (August 2025), 16 pages. https://doi.org/10.1145/3730824

1 INTRODUCTION

Assembly is a key element of modern manufacturing. It enables the production of complex and large-scale products by combining simpler and modular components. Recent advancements in automation, particularly the introduction of robotic assembly lines, have significantly improved the efficiency and precision of many assembly processes. However, the majority of assembly automation is only feasible in mass production due to its repetitive nature. The growing demand for low-volume yet highly customized products, such as personalized toys [Chen et al. 2022; Ge et al. 2024] and architectural structures [Deuss et al. 2014; Huang et al. 2021], poses significant challenges to existing robotic assembly processes. Addressing these challenges requires developing new computational methods to automate the planning of robotic assembly processes.

This work aims to advance customized fabrication [Baudisch and Mueller 2017] by automating the assembly of rigid parts, which has applications such as customized masonry vaults [Block 2009] and buildings [Whiting et al. 2009]. Assembling such structures has traditionally relied on dense scaffolding to stabilize intermediate assembly steps, in a time-consuming and labor-intensive process. Developing novel, cost-effective methods for constructing these structures remains an ongoing research challenge [Deuss et al. 2014]. Inspired by recent advancements in robotic construction [Johns et al. 2023; Wang et al. 2023b], we aim to utilize a multi-robotic assembly system to collaboratively construct structures composed of rigid parts without the use of additional scaffolding.

Most existing assembly planning approaches rely on precomputing a single assembly plan through computationally intensive offline algorithms [Tian et al. 2024; Wang et al. 2023b]. However, in practice, external changes—such as modifications to the robotic assembly setup, part delivery delays, or future repair and replacement operations—can disrupt the precomputed plan. Addressing these disruptions by re-computing new assembly plans using existing methods causes significant delays due to their intensive offline computations. This motivates us to develop a robust assembly planning method capable of efficiently handling unexpected changes.

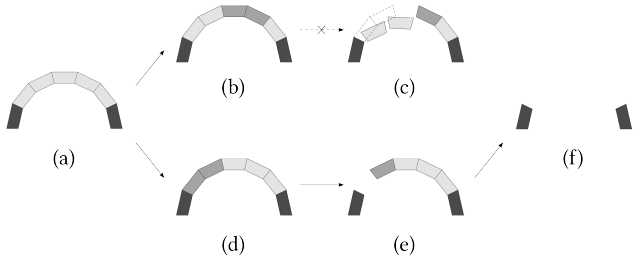


Fig. 2. Disassembling an arch from the initial disassembly state (a) to the target disassembly state (f) using two robots. Robots held parts are shown in gray, and boundary parts are colored in black. (b)(c) are disassembly states of an infeasible plan using a simple sorted-by-height policy. The disassembly state of (c) is structurally unstable. (d)(e) are disassembly states of a feasible plan, which disassembles parts from left to right using two robots collaboratively. This figure illustrates that a top-down strategy—removing parts from highest to lowest—can fail when disassembling an arch structure.

A straightforward approach is to enumerate all assembly plans for constructing the given structure in advance. However, the number of plans grows combinatorially with the increase in the number of parts and the number of robots involved, making the enumeration computationally infeasible. Figure 1 illustrates three examples of feasible plans for constructing the Vault structure [Deuss et al. 2014] using two or three robotic arms.

To address this problem, we first simplify it by considering an abstract planning problem where robots and parts are assumed to have no collision bodies, and all robots are treated as identical. We also adopt the assembly-by-disassembly strategy [Wilson 1992], which first computes the disassembly plan and reverses it to obtain a feasible assembly plan. This strategy simplifies the planning problem by gradually reducing its size through the removal of parts. In the following, we primarily focus on disassembly planning as our core methodology; however, our main objective remains to solve assembly planning problems.

We propose addressing this abstract disassembly planning problem for a given structure using reinforcement learning (RL). The core of our approach is to train a neural network, referred to as the disassembly policy, which maps a disassembly state to a probability distribution over potential next disassembly actions, prioritizing those with the highest likelihood of success. A disassembly state encodes the status of each part, such as whether it is installed and whether it is temporarily held. A disassembly action involves either removing a part or holding it in place. The advantage of training such a disassembly policy is that, when an unexpected event occurs, a new disassembly action can be efficiently generated by sampling from the policy based on the updated disassembly state. Note that manually designing an effective disassembly policy is challenging.

Simple policies, such as sorting by height, cannot disassemble an arch using two robots, as illustrated in Figure 2.

Training an effective disassembly policy faces two significant challenges. First, training requires evaluating the structural stability of numerous incomplete disassembly states using simulation, which becomes a significant bottleneck in training. Second, each training episode involves a large number of disassembly steps, with a reward only provided at the end of the episode. This results in sparse rewards during training, reducing the signal available to train on, and limiting its ability to generate diverse disassembly plans. To address these challenges, we first develop a new GPU-accelerated physics simulator that solves structural stability problems in parallel. Then, we introduce a novel learning scheme inspired by the curriculum learning approach [Bengio et al. 2009], in which our disassembly policy is trained on tasks with progressively increasing complexity. Finally, a physics-inspired graph neural network (GNN) is proposed as the architecture for the disassembly policy. This novel architecture improves data efficiency during training and facilitates potential policy transfer across similar structures.

Our disassembly policy can be extended to handle robotic disassembly processes that require strict collision avoidance. First, part disassemblability is incorporated into the training of our disassembly policy, ensuring that each part can be removed along a linear trajectory without colliding with the remaining structure. Next, we introduce a hybrid method for robotic disassembly planning. Our approach leverages the pretrained disassembly policy as a heuristic to guide the selection of high-level disassembly tasks, while a robotic motion planner [Sundaralingam et al. 2023] is employed to compute the low-level robotic disassembly motions. Specifically, we make the following contributions:

- We develop a GPU-based rigid body equilibrium simulator that solves batches of stability problems in parallel on GPUs using a modified alternating direction method of multipliers.
- We introduce a novel curriculum-based learning scheme to train disassembly policies via reinforcement learning.
- We propose a novel force-torque graph attention network architecture to serve as our disassembly policy.
- We propose a hybrid approach that combines our pre-trained policy with a robotic motion planner, showcasing its effectiveness in assembling structures using multiple robot arms in both simulation and real-world environments.

The code and dataset are available at https://github.com/KIKI007/LearningToAssemble.

2 RELATED WORK

Assembly sequence planning (ASP) is a foundational topic in computer-aided design and manufacturing. The goal of assembly sequence planning is to determine a physically feasible sequence for constructing a given assembly. Structural stability and disassemblability are two key feasibility criteria commonly considered in assembly sequence planning, as discussed in Section 2.1. The combinatorial nature of the planning space makes assembly sequence planning a challenging NP-hard problem. Traditional methods formulate assembly sequence planning as a tree search and utilize problem-specific search heuristics to accelerate the planning process, as detailed in Section 2.2. Recently, learning-based approaches have been introduced to address the challenges in assembly sequence planning, as described in Section 2.3.

2.1 Feasibility Assessment for Assembly Plans

Structural stability. To ensure safety, all intermediate assembly states must be structurally stable under gravity. Structural stability is often verified by a simulator, which evaluates each step of the planned sequence by predicting the movement of parts.

One area of research relevant to this work focuses on evaluating the structural stability of assemblies composed of rigid parts, which have applications in mechanical systems [Tian et al. 2024], masonry walls [Johns et al. 2020], and masonry shells [Deuss et al. 2014; Wang et al. 2023b]. Physically-based simulators, such as MuJoCo [Todorov et al. 2012], Isaac Sim [Liang et al. 2018], and incremental potential contact (IPC) [Ferguson et al. 2021; Li et al. 2020a], are capable of simulating the dynamics of complex assemblies. When assessing stability during assembly, static analysis is generally sufficient and computationally more efficient. Two widely used approaches for modeling contact problems in the evaluation of structural stability are the rigid body equilibrium method [Whiting et al. 2009] and the linear complementarity problem [Kao et al. 2022; Kaufman et al. 2008; Yao et al. 2017]. A comprehensive summary of simulating contact problems can be found in a recent SIGGRAPH course [Andrews et al. 2022a]. GPUs have been employed to parallelize stability simulations [Lan et al. 2022; Liang et al. 2018]. However, previous works have primarily focused on parallelizing simulations across different structures. In contrast, this work emphasizes parallelizing the simulation of different assembly states for the same structure. This problem presents a unique mathematical structure that can be further optimized for efficient computation on GPUs.

Recent research also evaluates the structural stability of assemblies made of deformable elements [Huang et al. 2021; Wang et al. 2023a]. For such assemblies, structural stability is measured as a continuous metric rather than a binary state, considering factors like maximum structural deformation or stress. In fact, most additive manufacturing processes, such as robotic spatial printing [Huang et al. 2024, 2016], can be viewed as a continuous assembly of deformable parts. These additive manufacturing studies face similar challenges, including the need to minimize structural deformation during the printing process. Our method is not directly applicable to planning assembly sequences involving deformable parts, as this lies outside the scope of the current work. However, the proposed approach—particularly the curriculum-based training scheme—has the potential to be extended to handle assemblies with deformable elements in future research.

Disassemblability. During assembly, whether parts are installed manually or by robots, avoiding collisions is essential to prevent damage and ensure safety. For manual assembly, a collision-free part insertion trajectory must be computed before installing the part. The directional blocking graph [Wilson 1992] is an effective method for computing linear installation trajectories for parts. If parts are assembled using complex non-linear trajectories, such as screwing, [Tian et al. 2022] introduces a novel algorithm based on physics-based simulation. This approach calculates the sequence of external

forces and torques required to remove a part, which can then be reversed to determine the corresponding parts' installation trajectories. This work focuses on installing parts using linear trajectories, with complex non-linear trajectories left for future work.

When robots are used for assembly, it is necessary to plan both the part insertion trajectories and the corresponding robot joint angle trajectories. Collisions must be avoided between the robots and the installed components, as well as between the robots themselves. Sampling-based methods, such as the rapidly exploring random tree (RRT) [LaValle 1998] and the probability road map (PRM) [Kavraki et al. 1996], along with their variants, are commonly employed to determine collision-free robot trajectories. Optimization-based approaches are employed to minimize assembly time in collaborative scenarios involving multiple robots [Hartmann et al. 2022]. This work leverages the GPU-based robotic motion planner, CuRobo [Sundaralingam et al. 2023], which facilitates the parallelization of planning robot motions for a batch of assembly tasks. However, the CuRobo framework is limited to low-level motion planning functions, and directly applying it can result in inconsistent robotic motions during transitions between assembly actions. To address these motion incompatibilities, we develop a novel graph-based motion planning framework to provide high-level motion planning for both assembly and disassembly actions.

2.2 Heuristics-based Assembly Sequence Planning

Tree search with backtracking is one of the most commonly used methods in assembly sequence planning [Tian et al. 2022]. To avoid collisions, the assembly-by-disassembly strategy [Halperin et al. 1998] is often employed, in which a disassembly sequence is first determined using tree search and then reversed to derive the corresponding assembly sequence.

However, for assemblies with many parts, the time required to search for a feasible assembly plan becomes impractical due to the enormous search space. To address this, the layer decomposition approach [Deuss et al. 2014; Huang et al. 2016; Wang et al. 2023a] has been proposed as a search heuristic to accelerate the tree search process. The method partitions the entire assembly process into several sub-assembly planning tasks, with each task having predefined start and end assembly states. These critical assembly states, known as layers, are determined using mixed-integer optimization. Tree searches are then used to solve each sub-assembly planning task, which begins and ends at two adjacent layers. By limiting the number of parts to plan within each sub-assembly planning task, this approach significantly reduces the overall planning time. Still, both approaches are expensive offline precomputation algorithms that generate only a single assembly plan. This work focuses on developing an efficient assembly sequence planner for the same structure, designed to effectively mitigate real-world disruptions.

2.3 Learning-based Assembly Planning

Learning-based approaches have recently gained significant attention in addressing challenges in assembly. First, reinforcement learning-based approaches are employed to address the simulation-to-reality gaps in controlling robots for assembling parts in real-world scenarios. Applications include assembling timber beams [Apolinarska et al. 2021], LEGO structures [Chen et al. 2023], magnetically connected parts [Kataoka et al. 2023], and irregular stones [Menezes et al. 2021]. A recent work [Ha et al. 2020] utilizes expert data generated by classical motion planning algorithms to accelerate the planning of multiple robot trajectories.

Learning-based approaches have been applied to generate feasible assembly sequences. Funk et al. [2021] employ a combination of deep RL and Monte Carlo Tree Search (MCTS) to learn to stack cubes. Ghasemipour et al. [2022] utilize Proximal Policy Optimization (PPO) to assemble parts connected by magnets. Kulshrestha et al. [2023] leverage simulation data to train a neural network for the rearrangement of block-stacking assemblies. Li et al. [2020b] introduce a curriculum-based approach to address block-stacking problems using robots. Graph neural networks (GNN) have been utilized to encode assembly geometry for training assembly sequence planning policies [Funk et al. 2021; Ghasemipour et al. 2022; Tian et al. 2024]. A hybrid approach combining search algorithms with learning-based heuristics is employed to address challenging assembly planning problems, such as maze-like 3D puzzles [Zhang et al. 2020] and mechanical assemblies [Tian et al. 2024]. Additionally, recent works leverage reinforcement learning to explore the design of stable assemblies that can be constructed by robots, including two-dimensional structures [Vallat et al. 2023], stacking blocks [Li et al. 2022], and LEGO designs [Chung et al. 2021].

Our work builds upon the development of learning-based assembly sequence planning. However, most existing works focus on training neural networks to solve general assembly planning problems, which become increasingly difficult to train as the number of parts grows. Besides, creating a training dataset of customized masonry vaults and buildings is highly challenging due to the vast design space. Therefore, our work focuses on addressing assembly planning problems for the same structure, enabling the efficient generation of alternative assembly plans with varying initial and target states. Our approach is designed to effectively mitigate real-world disruptions in assembly sequence planning.

3 PROBLEM FORMULATION

This work aims to address a collaborative assembly planning problem in which n robots are used to assemble a given structure made of m rigid parts. The $n (n < m)$ robots can temporarily hold parts to provide additional support during assembly. Our approach follows the assembly-by-disassembly strategy, in which a disassembly plan is generated first and then reversed to obtain the assembly plan. Illustrated in Figure 3, a disassembly state $\boldsymbol{s}$ represents a partial disassembly step, indicating whether each part remains in the assembly and whether a robot temporarily holds it. Our main goal is to train a disassembly policy $\pi(\cdot \mid \boldsymbol{s})$ for a given structure. The policy maps the current disassembly state $\boldsymbol{s}$ to a probability distribution of the following disassembly actions. When a disruption changes the current disassembly state $\boldsymbol{s}'$, an alternative disassembly action $\boldsymbol{a}'$ can be efficiently generated by sampling from the disassembly policy $\boldsymbol{a}' \sim \pi(\cdot \mid \boldsymbol{s}')$. This facilitates real-time disassembly planning processes for the given structure.

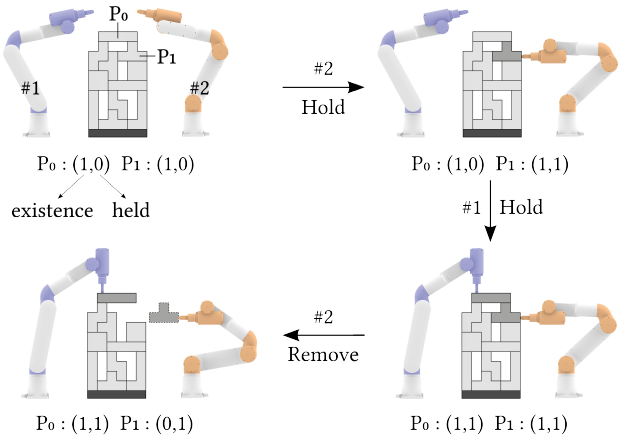


Fig. 3. We demonstrate the first three steps of a disassembly process for the Bottle model using dual robotic arms. Each part's state is represented by two binary values: the first indicates existence, while the second indicates whether it is held by robots. Parts held by robots are shown in grey, and remaining installed parts are displayed in white. Boundary parts are emphasized with a dark color. To remove part P_1, robot #2 needs first to hold the part and proceed with its removal, while robot #1 is required to hold P_0 to provide temporary support.

To enhance readability without introducing excessive physical constraints, we present our framework by first disregarding collisions between robots or parts and treating all robots as interchangeable. This allows us to concentrate on abstract disassembly states and disassembly actions without specifying which robot holds a particular part. Later, we extend our method to include robot-related physical constraints in Section 6.2.

The component of a disassembly state s^i consists of two binary values indicating whether the i-th part still exists and whether it is held by any of the robots. The component of the disassembly action a^i consists of two binary values indicating whether the i-th part is to be removed or to be held. Figure 3 illustrates the state and action in a two-robot disassembly process. Again, the reason each a^i has only two binary values is that we do not distinguish which robot executes the action on the part. The disassembly rules are:

(1) Each part can only be held once.
(2) At most n parts can be held simultaneously.
(3) A part must be held before it can be removed.
(4) One action (holding or removing) is executed per step.

In this work, releasing a held part without removing it (e.g., regrasp) is not permitted to prevent an infinite planning horizon. Additionally, simultaneously removing multiple parts per step is not allowed. Both are considered future work discussed in Section 8.

A disassembly process starts with a given disassembly state, $\boldsymbol{s}_0$. The first disassembly action $\boldsymbol{a}_0$ is sampled using $\boldsymbol{a}_0 \sim \pi(\cdot \mid \boldsymbol{s}_0)$. The next disassembly state $\boldsymbol{s}_1$ is obtained deterministically by applying the disassembly action $\boldsymbol{a}_0$ to the current disassembly state $\boldsymbol{s}_0$. A physically-based simulator is employed to evaluate the stability of $\boldsymbol{s}_1$. If $\boldsymbol{s}_1$ is found to be unstable, a reward of -1 is assigned. Otherwise, the process continues by sampling new disassembly actions, $\boldsymbol{a}_1 \sim \pi(\cdot \mid \boldsymbol{s}_1)$, $\boldsymbol{a}_2 \sim \pi(\cdot \mid \boldsymbol{s}_2)$, and so on. This process terminates with a reward of +1 when all installed parts in $\boldsymbol{s}_0$ have been successfully removed. The intermediate disassembly states form a disassembly sequence, $\{\boldsymbol{s}_0, \boldsymbol{s}_1, \cdots, \boldsymbol{s}_N\}$. By reversing this disassembly sequence, a feasible assembly sequence is obtained $\{\boldsymbol{s}_N, \cdots, \boldsymbol{s}_0\}$.

4 GPU-BASED STABILITY SIMULATOR

This section presents a fully parallelizable, physically-based simulator optimized for analyzing the structural stability of multiple incomplete assemblies on GPUs. Section 4.1 explains the general formulation of our structural stability simulations as a specific type of quadratic programming (QP) problem and outlines the approach for solving these QP problems on GPUs. Section 4.2 presents the rigid body equilibrium method and discusses the adjustments necessary to optimize it for GPU computation.

4.1 ADMM-QP Solver

Evaluating the stability of a structure in a given disassembly state $\boldsymbol{s}$ is formulated as a quadratic programming (QP) [Whiting et al. 2009]:

$$\begin{aligned} \min_{\boldsymbol{x}} \ & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}(s)\boldsymbol{x} + \boldsymbol{q}(s)^T\boldsymbol{x} \\ \text{s.t.} \ & \boldsymbol{b}_l(\boldsymbol{s}) \le \boldsymbol{A}(s)\boldsymbol{x} \le \boldsymbol{b}_u(\boldsymbol{s}) \end{aligned} \tag{1}$$

Our key observation is that changes to the disassembly state $\boldsymbol{s}$ do not affect the quadratic term ($\boldsymbol{Q} = \boldsymbol{Q}(s)$) or the linear constraints ($\boldsymbol{A} = \boldsymbol{A}(s)$). This unique property enables the development of an efficient GPU-based solver, inspired by the Alternating Direction Method of Multipliers (ADMM) [Parikh et al. 2014] and its QP-specific version, Operator Splitting Quadratic Programming (OSQP) [Stellato et al. 2020]. We refer to our new GPU-based QP solver as the ADMM-QP solver.

$$\boldsymbol{L} = \boldsymbol{Q} + \sigma\boldsymbol{I} + \boldsymbol{A}^T\boldsymbol{\rho}\boldsymbol{A} \tag{2}$$

$$\hat{\boldsymbol{x}}_{k+1} = \boldsymbol{L}^{-1}\left[\sigma\boldsymbol{x}_k - \boldsymbol{q} + \boldsymbol{A}^T\left(\boldsymbol{\rho}\boldsymbol{z}_k - \boldsymbol{y}_k\right)\right] \tag{3}$$

$$\boldsymbol{x}_{k+1} = \alpha\hat{\boldsymbol{x}}_{k+1} + (1-\alpha)\boldsymbol{x}_k \tag{4}$$

$$\boldsymbol{z}_{k+1} = \Pi_{[\boldsymbol{b}_l,\boldsymbol{b}_u]}(\alpha\boldsymbol{A}\hat{\boldsymbol{x}}_{k+1} + (1-\alpha)\boldsymbol{z}_k + \boldsymbol{\rho}^{-1}\boldsymbol{y}_k) \tag{5}$$

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \boldsymbol{\rho}(\alpha\boldsymbol{A}\hat{\boldsymbol{x}}_{k+1} + (1-\alpha)\boldsymbol{z}_k - \boldsymbol{z}^{k+1}) \tag{6}$$

where $\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{z}_k$ are primal, dual, and slack variables, and α, σ, and $\boldsymbol{\rho}$ are parameters introduced in the OSQP paper [Stellato et al. 2020]. We choose $\alpha = 1.6$ and $\sigma = 10^{-6}$.

The most computationally expensive step is solving the sparse linear system defined in Equation 3. Fortunately, by selecting a constant $\boldsymbol{\rho} = \rho\boldsymbol{I}$, the matrix $\boldsymbol{L}$ becomes constant and can be pre-computed. We choose $\rho = 0.1$. In contrast, second-order methods, such as the interior point method [Boyd 2004], have a non-constant Karush–Kuhn–Tucker (KKT) system that depends on the disassembly state $\boldsymbol{s}$, making them challenging to solve efficiently on GPUs.

In our experiment, the most effective approach to solving the linear system is to pre-compute the inverse matrix $\boldsymbol{L}^{-1}$ directly, as demonstrated in Equation 3. GPU-based pre-factorization methods, such as Cholesky LLT [Nicolet et al. 2021], while providing higher accuracy, are significantly slower than the direct matrix inversion

Table 1. List of symbols used for stability simulations.

Variable	Description
$\boldsymbol{\lambda}_n$	The magnitude of the normal force impulse.
$\boldsymbol{\lambda}_t$	The magnitude of friction impulse.
$\boldsymbol{J}_n$	The normal parts of the contact Jacobian.
$\boldsymbol{J}_t$	The tangential parts of the contact Jacobian.
$\boldsymbol{f}$	The support force provided by robots
$\boldsymbol{g}$	The generalized gravity force.
μ	The friction coefficient.
$\boldsymbol{M}$	The generalized mass matrix.
f^u	The maximum absolute value of support force.
$\boldsymbol{\lambda}^u$	The maximum magnitude of contact force impulse.
$\boldsymbol{P}$	The part projection matrix.
$\boldsymbol{C}$	The contact projection matrix.

method on GPUs. Since our ADMM-QP solver relies solely on matrix multiplication, a batch of stability instances can be efficiently simulated in parallel on GPUs. In our implementation, the variables x_k, y_k, and z_k are matrices where each column represents a sample in the batch. These variables are initialized to zero. To achieve sufficient precision, our solver is executed on an NVIDIA H100 GPU with support for double-precision floating-point arithmetic (FP64).

4.2 GPU-based Stability Simulator

In this section, we present our GPU-based stability simulator, adapted from the rigid body equilibrium method [Whiting et al. 2009], which simulates the stability of rigid bodies under frictional contacts. Our contribution is to leverage the fact that L^{-1} is constant for different disassembly states $\boldsymbol{s}$, allowing for an efficient parallelization. Our notation, summarized in Table 1, follows that of the SIGGRAPH course [Andrews et al. 2022b].

The $\boldsymbol{\lambda}_n$ and $\boldsymbol{\lambda}_t = \{\boldsymbol{\lambda}_{t_k}\}$ are normal and tangential force impulses defined on contact points. Each $\boldsymbol{\lambda}_{t_k}$ represents a frictional force along a tangential direction of the contact plane. This work uses eight tangential directions to approximate the friction cone. The $\boldsymbol{J}_n^T\boldsymbol{\lambda}_n$ and $\boldsymbol{J}_t^T\boldsymbol{\lambda}_t$ ($= \sum_k \boldsymbol{J}_{t_k}^T\boldsymbol{\lambda}_{t_k}$) are the resultant of the normal and friction force impulse acting on parts. The rigid body equilibrium method seeks to identify a feasible $\boldsymbol{\lambda}_n, \boldsymbol{\lambda}_t$ that satisfies the linear inequality constraints:

$$\boldsymbol{P}\left(\boldsymbol{J}_n^T\boldsymbol{\lambda}_n + \boldsymbol{J}_t^T\boldsymbol{\lambda}_t + \boldsymbol{g}\right) = 0 \tag{7}$$

$$0 \le \boldsymbol{\lambda}_n \le \boldsymbol{C}\boldsymbol{\lambda}^u \tag{8}$$

$$0 \le \boldsymbol{\lambda}_{t_k} \le \boldsymbol{C}\boldsymbol{\lambda}^u \tag{9}$$

$$0 \le \mu\boldsymbol{\lambda}_n - \textstyle\sum\boldsymbol{\lambda}_{t_k} \tag{10}$$

Equation 7 represents the force-torque equilibrium conditions, where the part projection matrix $\boldsymbol{P}$ excludes constraints for held parts as well as removed parts. Equation 8, 9 imposes constraints on the range of the force impulses, where the contact projection matrix $\boldsymbol{C}$ ensures that normal or frictional impulses are zero on parts that have been removed. Equation 10 represents the Coulomb friction condition, with $\mu = 0.55$.

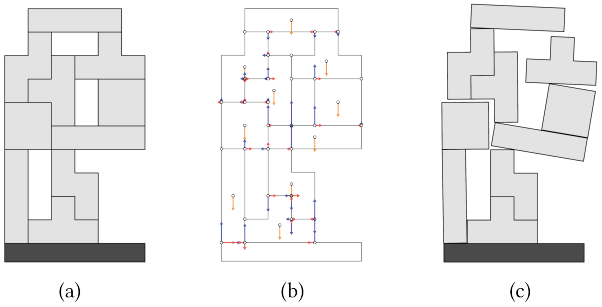


Fig. 4. Structural stability simulation of the Bottle using the rigid body equilibrium method. (a) A disassembly state of the Bottle with one missing part. (b) The imbalanced internal forces computed using Equation 13, including normal (blue), frictional (red), and gravitational forces (orange), and (c) the imbalanced motions of parts computed using Equation 14.

This feasibility problem can be addressed using a QP solver by introducing a zero-valued objective. However, our ADMM-QP solver cannot be used to solve this problem because the matrix $\boldsymbol{P}$ in Equation 7 depends on the disassembly state $\boldsymbol{s}$. To tackle this challenge, we introduce an additional variable, the support force $\boldsymbol{f}$, and replace Equation 7 with the following:

$$\boldsymbol{J}_n^T\boldsymbol{\lambda}_n + \boldsymbol{J}_t^T\boldsymbol{\lambda}_t + \boldsymbol{g} + \boldsymbol{f} = \boldsymbol{0} \tag{11}$$

$$-[\boldsymbol{1} - \boldsymbol{P}]f^u \le \boldsymbol{f} \le [\boldsymbol{1} - \boldsymbol{P}]f^u \tag{12}$$

Equation 12 ensures that support forces are only applied to uninstalled and temporarily held parts. The new GPU-based stability simulator is then written as:

$$\begin{aligned} \min_{\boldsymbol{\lambda}_n,\boldsymbol{\lambda}_t,\boldsymbol{f}} \quad & \frac{1}{2}\|\boldsymbol{J}_n^T\boldsymbol{\lambda}_n + \boldsymbol{J}_t^T\boldsymbol{\lambda}_t + \boldsymbol{g} + \boldsymbol{f}\|^2_{\boldsymbol{M}^{-1}} \\ \text{s.t.} \quad & 0 \le \boldsymbol{\lambda}_n \le \boldsymbol{C}\boldsymbol{\lambda}^u \\ \text{s.t.} \quad & 0 \le \boldsymbol{\lambda}_{t_k} \le \boldsymbol{C}\boldsymbol{\lambda}^u \\ & -[\boldsymbol{1} - \boldsymbol{P}]f^u \le \boldsymbol{f} \le [\boldsymbol{1} - \boldsymbol{P}]f^u \\ & 0 \le \mu\boldsymbol{\lambda}_n - \textstyle\sum\boldsymbol{\lambda}_{t_k} \end{aligned} \tag{13}$$

where $\|\boldsymbol{x}\|^2_{\boldsymbol{M}^{-1}} = \boldsymbol{x}^T\boldsymbol{M}^{-1}\boldsymbol{x}$. The generalized mass matrix M is a block-diagonal matrix, where each block represents the generalized mass matrix of a part. Each component of $\boldsymbol{\lambda}^u$ and f^u is set to 10^5. The objective function minimizes the work done by the residual forces. Its objective and linear constraint coefficients are independent of the disassembly states $\boldsymbol{s}$, making it compatible with our ADMM-QP solver.

Figure 4 (b) shows the optimal force outputs generated by our simulator for an unstable disassembly state. Our simulator uses residual velocity to verify the stability of a disassembly state $\boldsymbol{s}$, which is computed as follows:

$$\boldsymbol{v} = \boldsymbol{M}^{-1}(\boldsymbol{J}_n^T\boldsymbol{\lambda}_n + \boldsymbol{J}_t^T\boldsymbol{\lambda}_t + \boldsymbol{g} + \boldsymbol{f}) \tag{14}$$

The disassembly state $\boldsymbol{s}$ is considered stable if $\|\boldsymbol{v}\|_\infty \le 10^{-3}$, and otherwise unstable. The residual velocity provides a more intuitive way to understand the instability by illustrating the movement of parts in an unstable disassembly state; see Figure 4(c).

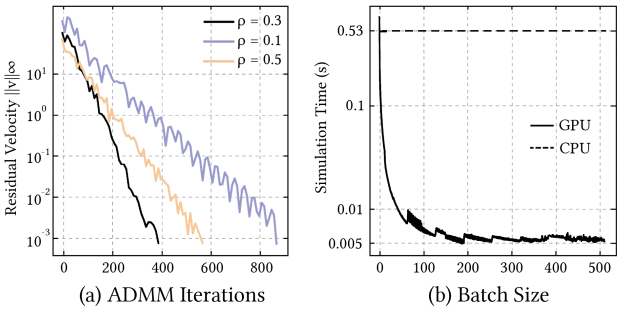


Fig. 5. Simulate the stability of the Vault model from Figure 1. (a) The infinity norm of the velocity residual is plotted against the number of ADMM iterations for various penalty coefficients ρ, as defined in Equation 3. The y-axis is plotted on a log scale. (b) A comparison of the performance between a CPU-based QP solver (Gurobi) and our ADMM-QP solver when simulating the stability of the Vault multiple times in batch by setting $\rho = 0.3$. The graph illustrates the average simulation time as a function of batch size.

Using our ADMM-QP solver, our stability simulator efficiently evaluates a batch of disassembly states in parallel on GPUs. The simulator evaluates the disassembly states' residual velocity every 200 ADMM iterations and excludes the disassembly states that already satisfy the stability condition. Additional ADMM iterations are performed until the stability of all partial states is determined or the iteration limit (e.g., 3000) is reached. The remaining undecided disassembly states are then classified as unstable.

The simulation time is directly proportional to the total number of ADMM iterations performed. The ADMM method typically exhibits a linear convergence rate, as visualized in Figure 5 (b). For this test case, compared to commercial CPU-based QP solvers like Gurobi, our ADMM-QP solver can achieve up to a 100× speedup. Figure 5(c) illustrates the performance of our GPU-based stability simulator across various batch sizes when simulating the Vault model multiple times. A detailed performance evaluation is provided in Section 7.1. It is worth noting that less efficient performance is observed for small batch sizes (≤ 16), primarily due to the significant overhead of GPU computation.

5 REINFORCEMENT LEARNING

This section introduces our reinforcement learning approach for planning the disassembly process by training a disassembly policy. Our disassembly policy is trained to disassemble an m-part structure using n robots. Collisions are ignored, and all robots are considered indistinguishable. Section 5.1 discusses the challenges of using an off-the-shelf reinforcement learning method, Proximal Policy Optimization (PPO) [Parikh et al. 2014], to train our disassembly policy. Section 5.2 presents our curriculum-based training scheme to efficiently address the challenge of sparse rewards. Section 5.3 extends our policy to plan disassembly processes between two arbitrary disassembly states. Lastly, Section 5.4 introduces a graph-based neural network architecture for the disassembly policy.

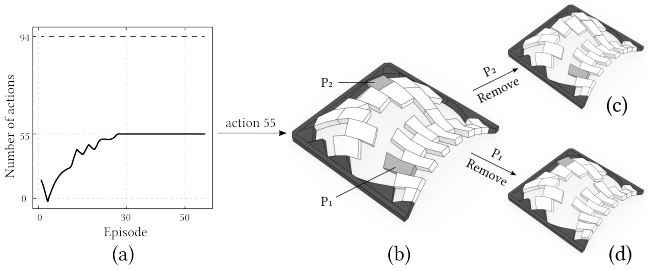


Fig. 6. We train an off-the-shelf PPO agent to plan the disassembly process of the Vault, starting from its complete state using two robots. (a) The PPO agent converges to a local minimum. Disassembling the Vault requires at least 94 actions (highlighted by the dashed line), but all training episode terminates after taking 55 actions. (b) This is one of the most common disassembly states encountered before termination. The agent cannot proceed further, as removing either $P1$ or $P2$ leads to unstable states, as illustrated in (c) and (d).

5.1 Disassembly Policy Training

A disassembly policy $\pi(\cdot|\boldsymbol{s})$ takes the current disassembly state $\boldsymbol{s}$ as input, outputting a probability distribution over the next possible disassembly actions. A training rollout, or episode, involves using the policy to plan the disassembly process, starting from an initial disassembly state and continuing until reaching the empty state. A reward is given at the end of each episode. The reward is +1 if a feasible disassembly plan is found; otherwise, it is −1. Offline reinforcement learning methods, such as Deep Q-Networks (DQN) [Mnih 2013] and Soft Actor-Critic (SAC) [Haarnoja et al. 2018], can be employed for training. This work trains the disassembly policy using the Proximal Policy Optimization (PPO) algorithm [Schulman et al. 2017], selected for its widespread adoption and robust performance across various applications [Berner et al. 2019; Hoeller et al. 2024; Yang et al. 2023].

Training a policy using PPO to disassemble a given structure is challenging due to the sparsity of the reward signal, which is provided only at the termination of each episode. Our PPO agent struggles to robustly plan the disassembly process for structures with more than 10 parts, even when multiple parallel disassembly rollouts (e.g., 64) are employed to improve its exploration. Figure 6 illustrates the results of training an agent with an off-the-shelf PPO algorithm to disassemble the Vault model using two robots. We employ a simple 3-layer multi-layer perceptron (MLP) as the policy network. Due to the lack of positive rewards, the disassembly policy converges to a local minimum, repeatedly getting stuck in the same disassembly state, as shown in Figure 6 (b).

5.2 Curriculum Learning

In our experiment, we can train PPO policies to disassemble structures with a small number of parts (e.g., ≤ 10), but fail to robustly disassemble structures with more than 10 parts. The training difficulty lies in the fact that the ratio of feasible plans to the total number of possible plans decreases significantly as the number of parts increases. The off-the-shelf PPO algorithm cannot sample a single feasible disassembly plan, not even by chance.

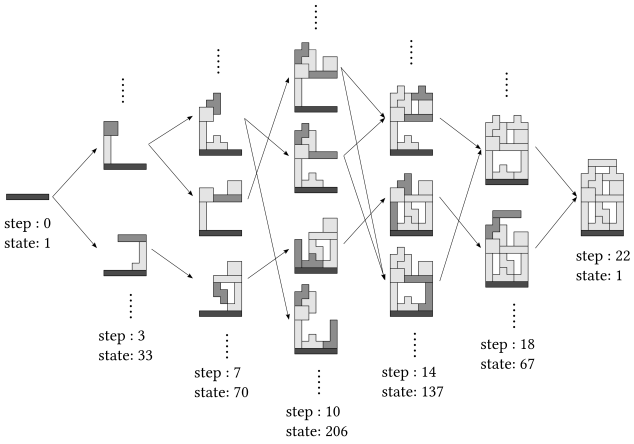


Fig. 7. An assembly state graph of the Bottle is generated using a beam search to plan its assembly process with two robots. Each node represents a stable assembly state, while duplicated and unstable states are excluded from the graph. Each edge corresponds to a feasible assembly action. The nodes form an effective dataset for training our disassembly policy.

The key research question is how to enable our PPO agent to obtain positive rewards during training without changing the reward function. A key insight is that instead of initializing the training to disassemble the entire structure, we start with a partial disassembly state. If the initial disassembly state contains only a small number of parts, the PPO method can find a feasible disassembly plan and receive a positive reward. We gradually increase the number of parts in the initial disassembly state if the policy has already learned to handle disassembly tasks with fewer parts. Training on disassembly tasks with an increasing number of parts aligns with the concept of curriculum learning [Bengio et al. 2009], a strategy widely adopted in the character animation community to address challenges such as sparse rewards and complex motion dynamics [Xie et al. 2020]. Additionally, introducing random state initialization has been shown to be effective in exploring the state-action space, as demonstrated in DeepMimic [Peng et al. 2018].

A key technical challenge is generating a training dataset of valid initial disassembly states. An initial disassembly state is considered invalid if no plan exists to disassemble the remaining parts starting from it. Initializing an RL agent to disassemble starting from an invalid initial disassembly state is bound to fail, preventing the agent from learning effectively.

We found that the assembly state graph, generated as a byproduct of most search-based assembly planning algorithms, serves as an effective dataset for training our disassembly policy, as illustrated in Figure 7. Nodes in this assembly state graph represent partial assembly states, with the leftmost node, also known as the root node, representing the empty state. Nodes are connected by directed edges, each representing a feasible assembly action. An important feature of this assembly state graph is that tracing a path from the root node to any node represents a feasible assembly plan, while backtracking along this path yields a valid disassembly plan. Thus, all nodes in this state graph form our training dataset, as each represents a valid initial disassembly state with a guaranteed disassembly plan.

This assembly state graph can be efficiently generated using a beam search, accelerated by our GPU-based stability simulator. The search begins with the root node of the empty state. In each iteration, the search selects W nodes to apply assembly actions and adds the newly discovered assembly states that are structurally stable to the state graph. The search process terminates upon reaching the complete structure. Please refer to the detailed algorithm provided in the supplementary material.

5.3 Disassembly Policy Between Two States

We can extend our disassembly policy to plan the disassembly process from an initial disassembly state $\boldsymbol{s}$ to a non-empty target disassembly state $\boldsymbol{s}_{\text{target}}$, denoted as $\pi(\cdot \mid \boldsymbol{s}, \boldsymbol{s}_{\text{target}})$. Our extended disassembly policy can be applied to restore structures that have experienced partial structural collapse due to earthquakes.

A straightforward approach is to reuse the disassembly policy $\pi(\cdot \mid \boldsymbol{s})$ while ensuring no parts present in the target disassembly state $\boldsymbol{s}_{\text{target}}$ can be removed. However, as demonstrated in the results (Section 7.2), this masked policy exhibits a significantly lower success rate, as changing the target state can significantly alter the disassembly process. To improve the success rate, we enhance our training dataset by including new disassembly tasks that terminate at non-empty target disassembly states. The new training data is generated using the same search process described in Section 5.2. However, instead of using an assembly state graph generated by a search-based assembly planning method, we utilize a disassembly state graph generated by a search-based disassembly planning method.

We can further enhance the data efficiency by leveraging Hindsight Experience Replay (HER) [Andrychowicz et al. 2017]. A failed disassembly plan $\{\boldsymbol{s}_0, \cdots, \boldsymbol{s}_N\}$ inherently contains a successful disassembly sub-plan $\{\boldsymbol{s}_0, \cdots, \boldsymbol{s}_{N-1}\}$, which can be reused by setting a new target disassembly state $\boldsymbol{s}_{\text{target}} = \boldsymbol{s}_{N-1}$. Leveraging this successful sub-plan enables the RL agent to receive positive rewards, thereby improving data efficiency.

5.4 Force-Torque Graph Attention Network

This section discusses the neural network architecture for our disassembly policy. The policy takes a disassembly state as input and outputs the probabilities of the next possible disassembly actions. The neural network for our disassembly policy must satisfy two critical requirements: (1) it must produce consistent outputs regardless of the indexing of parts, and (2) it must effectively utilize contact geometry information, such as contact points and normals. The commonly used multilayer perceptron (MLP) is unsuitable for this purpose because it cannot handle permutation invariance with respect to part indexing.

To address this, we employ a graph attention network as our disassembly policy [Veličković et al. 2017], which is permutation invariant by design. The key research question is how to effectively represent the contact geometry and the disassembly state $\boldsymbol{s}$ using a graph-based representation. A commonly used approach for encoding parts' geometry is the part graph [Tian et al. 2024], a

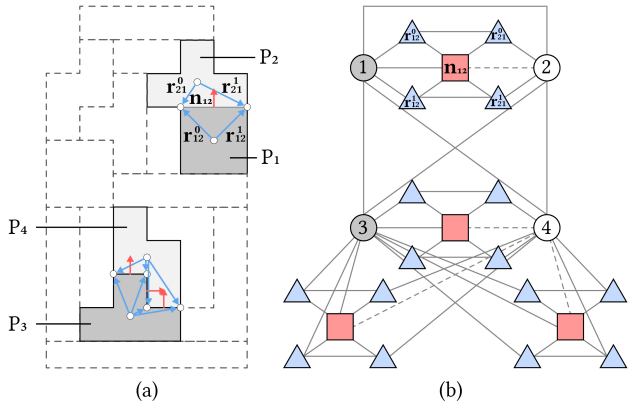


Fig. 8. We present our force-torque graph of a four-part disassembly state. Circles represent part nodes, red squares represent force nodes, and blue triangles represent torque nodes. A gray circle indicates a temporarily held part. P_1 and P_2 have two contact points which share the same force node ($\boldsymbol{n}_{12}$) and four torque nodes ($\boldsymbol{r}_{12}^0$, $\boldsymbol{r}_{21}^0$, $\boldsymbol{r}_{12}^1$, and $\boldsymbol{r}_{21}^1$). To represent the directionality of the normal vector $\mathbf{n}_{12}$—which points from P_1 to P_2—we use two distinct edge types (a solid line and a dashed line) to connect the force node to the corresponding part nodes. P_3 and P_4 share 3 force nodes and 12 torque nodes. We use 12 torque nodes instead of 8 because a separate torque node is assigned to the same contact point for each distinct contact normal. Despite not being in direct contact, skip connections are included between P_1-P_3 and P_2-P_3. Three force nodes are used between P_3 and P_4 to represent their three distinct contact normals.

homogeneous graph in which each node represents a part. However, this graph inevitably leads to duplications when encoding contact information, such as contact normals and points, which are critical components in the stability simulation. Specifically, the same contact normal and contact point are redundantly encoded into both of their associated part nodes.

To address this challenge, this work introduces a novel heterogeneous Force-Torque Graph (FT-Graph), inspired by the physically-based stability simulator (Equation 13). Our FT-Graph consists of three types of nodes: part nodes (circles), force nodes (squares), and torque nodes (triangles); see Figure 8. Each part node includes a binary attribute that represents whether the part is held by a robot, as specified in the disassembly state $\boldsymbol{s}$, and a real-valued attribute that represents the mass of the part. For the extended policy $\pi(\cdot \mid \boldsymbol{s}, \boldsymbol{s}_{\text{target}})$, the target disassembly state $\boldsymbol{s}_{\text{target}}$ is also encoded into the part nodes. Parts not included in the current disassembly state $\boldsymbol{s}$ are excluded from the graph, as no further disassembly actions can be performed on them. Contacts between parts are encoded using both force nodes and torque nodes. Each contact is composed of a contact normal $\boldsymbol{n}$ and a contact point $\boldsymbol{r}$. To achieve translational invariance, we use relative contact points $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, where each represents the vector between the contact point and the centroid of one of the associated parts; see Figure 8(a). Each force node contains a contact normal $\boldsymbol{n}$, while two torque nodes store the relative contact points $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$, respectively. Force nodes that share the same contact normal are merged into a single force node.

Each force node is connected to its two associated part nodes using two distinct edge types to represent the pointing direction of the normal vector. Each torque node is connected to exactly one force node, one part node, and one other torque node. Indeed, torque nodes that share the same contact point are connected. All part nodes are interconnected through skip connections, even if the parts are not in contact; see Figure 8(b). When a part is removed, its corresponding part node is removed along with all connected force and torque nodes.

Our disassembly policy begins with converting the disassembly state $\boldsymbol{s}$ and the contact geometry into an FT-Graph. The policy then processes this graph using the 8 Graph Attention Network (GAT) layers, each with 16 hidden features [Veličković et al. 2017], implemented with the PyG framework [Fey and Lenssen 2019]. The output of the GAT is an FT-Graph, where each part node is represented by a feature vector containing 16 hidden channels. These feature vectors are subsequently passed through an additional linear layer to compute the disassembly action probabilities. Additionally, the state value function V, required by the PPO method, shares the same architecture as the disassembly policy but uses a separate linear layer for its output.

6 ROBOTIC DISASSEMBLY PLANNING

At this stage, our reinforcement learning framework allows us to compute abstract disassembly sequences. However, additional physical constraints need to be verified when real robots are used to execute these sequences. In this section, we demonstrate how our framework can be extended to plan robotic disassembly processes.

Before introducing robotics, Section 6.1 discusses how to train a disassembly policy to ensure that parts can be removed physically. Section 6.2 presents our hybrid approach for planning robotic disassembly processes by combining our trained disassembly policy with a robot motion planner.

6.1 Part Disassemblability

In this work, a part can be physically disassembled if it can be removed along a linear trajectory without colliding with the remaining structure. Complex non-linear trajectories, such as those explored in [Tian et al. 2022], are considered a direction for future work. For a given part, we sample d linear trajectories (e.g., $d = 1000$) and evaluate each trajectory to determine which parts, if any, collide with the given part. A binary lookup table T is precomputed before training, containing $d \times m \times m$ entries, where m is the number of parts. $T_{ijk} = 0$ indicates that using the i-th trajectory to remove the j-th part causes collisions with the k-th part. A part can be removed if at least one trajectory in the lookup table allows its removal without colliding with any remaining parts. The part disassemblability is enforced using an action mask that sets the probabilities of invalid removal actions to zero.

6.2 Hybrid Robotic Disassembly Planner

We propose a hybrid method to jointly plan the disassembly sequence and the robot motions, as outlined in Algorithm 1. The algorithm's input is an initial disassembly state $\boldsymbol{s}$ and the robots'

Algorithm 1: Hybrid Robotic Disassembly Planner

Data: A disassembly state $\boldsymbol{s}$ and robots' joint angles $\boldsymbol{\theta}$
Result: A feasible robot disassembly plan.
$T \leftarrow \text{Tree}(\boldsymbol{s}, \boldsymbol{\theta})$;
while $|T| \leq N$ **and** $\boldsymbol{s} \neq \mathbf{0}$ **do**
  $\boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s})$;
  $\tilde{\boldsymbol{s}} \leftarrow \text{Next State}(\boldsymbol{s}, \boldsymbol{a})$;
  if Stable $(\tilde{\boldsymbol{s}})$ **then**
    for $\tilde{\boldsymbol{\theta}} \in \text{Keyframe}(\boldsymbol{a}, \tilde{\boldsymbol{s}})$ **do**
      $\tau \leftarrow \text{Motion Planning}(\boldsymbol{s}, \boldsymbol{\theta}, \tilde{\boldsymbol{\theta}})$;
      if Collision-free $(\boldsymbol{s}, \tau)$ **then**
        $(\boldsymbol{s}, \boldsymbol{\theta}) \leftarrow \text{Add Node } (T, \tilde{\boldsymbol{s}}, \tilde{\boldsymbol{\theta}})$;
        break;
      end
    end
  end
  if $\boldsymbol{s} \neq \tilde{\boldsymbol{s}}$ **then**
    $\pi(\boldsymbol{a}|\boldsymbol{s}) \leftarrow 0$;
    if $\sum \pi(\cdot|\boldsymbol{s}) = 0$ **then**
      $(\boldsymbol{s}, \boldsymbol{\theta}) \leftarrow \text{Backtrack } (T, \boldsymbol{s}, \boldsymbol{\theta})$;
    end
  end
end
if $\boldsymbol{s} = \mathbf{0}$ **then**
  return Disassembly Plan (T);
end

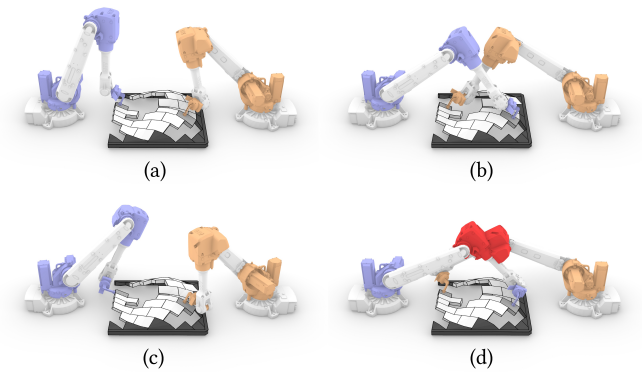


Fig. 9. We illustrate the keyframe generation process, showcasing various robot joint configurations that can achieve the same disassembly state. (a) The purple robot holds the left part, and the orange robot holds the right part. (b) Robots switch the parts they hold while still corresponding to the same disassembly state. (c) Robots change their joint configurations while maintaining the same grasping configuration as shown in (a). (d) A robot joint configuration discarded due to self-collisions (highlighted in red).

initial joint angles $\boldsymbol{\theta}$. The output of the algorithm is a robotic disassembly plan, which consists of a list of disassembly states and robot joint angles $\{(\boldsymbol{s}_0, \boldsymbol{\theta}_0), \cdots, (\boldsymbol{s}_N, \boldsymbol{\theta}_N)\}$. Our hybrid algorithm is a tree search with backtracking. A disassembly action $\boldsymbol{a}$ is sampled at each iteration using a pre-trained disassembly policy π. The feasibility of the sampled action $\boldsymbol{a}$ is evaluated by simulating its structural stability and planning the corresponding robotic motion. If the structure of the next disassembly state is stable and the planned robot motion is collision-free, the sampled action is executed, allowing the search process to continue. Otherwise, the sampled action is discarded by setting its probability in the disassembly policy to zero. If all actions suggested by the disassembly policy have been attempted and failed, the algorithm backtracks to an earlier disassembly state.

Our robot motion planning function utilizes the Curobo framework [Sundaralingam et al. 2023], which offers low-level motion planning utility functions. It can efficiently plan the robots' motion to transition from one joint angle configuration to another while avoiding collisions. However, the Curobo framework does not offer high-level utility functions for planning complex robotic disassembly tasks. Specifically, the framework currently lacks a function to find feasible robot disassembly motions to realize the disassembly action $\boldsymbol{a}$ generated by the pre-trained disassembly policy.

A robot configuration $\boldsymbol{\theta}$ that corresponds to an abstract disassembly state $\boldsymbol{s}$ is referred to as a keyframe. Multiple keyframes exist for the same abstract disassembly state $\boldsymbol{s}$. Since the abstract disassembly state does not differentiate between different robots, multiple grasping configurations can correspond to the same disassembly state, as illustrated in Fig. 9(a) and (b). Additionally, different robots' joint angles can achieve the same grasping configuration, as shown in Fig. 9(a) and (c). Keyframes that result in self-collisions or collisions with the parts must be discarded, as illustrated in Fig. 9(d).

The keyframes form a graph referred to as the keyframe graph, as illustrated in Fig. 10. Each row of the keyframe graph represents feasible keyframes corresponding to the same abstract disassembly state associated with that row. Only compatible pairs of keyframes are connected by edges, indicating that a collision-free trajectory exists for the robots to transition between the two keyframes. This work assumes that only one robot can move at a time. Adjacent keyframes in which more than one robot changes its joint angles are incompatible and are not connected by edges. Additionally, the collision body of the part is attached to the robot's end effector during the execution of the removal action. If there is no outgoing edge from the current disassembly state in the keyframe graph, the search backtracks. A feasible disassembly plan is a path in the keyframe graph that connects a keyframe of the complete state to a keyframe of the empty state. Reversing this robotic disassembly plan results in a feasible robotic assembly plan.

7 RESULTS

We implement our method in Python, using the PyTorch and PyTorch Geometric frameworks for GPU-based simulation and reinforcement learning. The training of the disassembly policy is conducted on a cluster equipped with an NVIDIA H100 GPU, 64 virtual CPU cores, and 128 GB of memory. The robotic assembly planning is performed separately on a desktop with an NVIDIA 4090 GPU, 32 CPU cores, and 32 GB of memory. We train our PPO agent using a batch size of 128, a learning rate of 2×10^{-3}, and a discount factor of 0.95. The disassembly policy network is composed of 8 Graph Attention (GAT) layers, each with 16 hidden features. Section 7.1

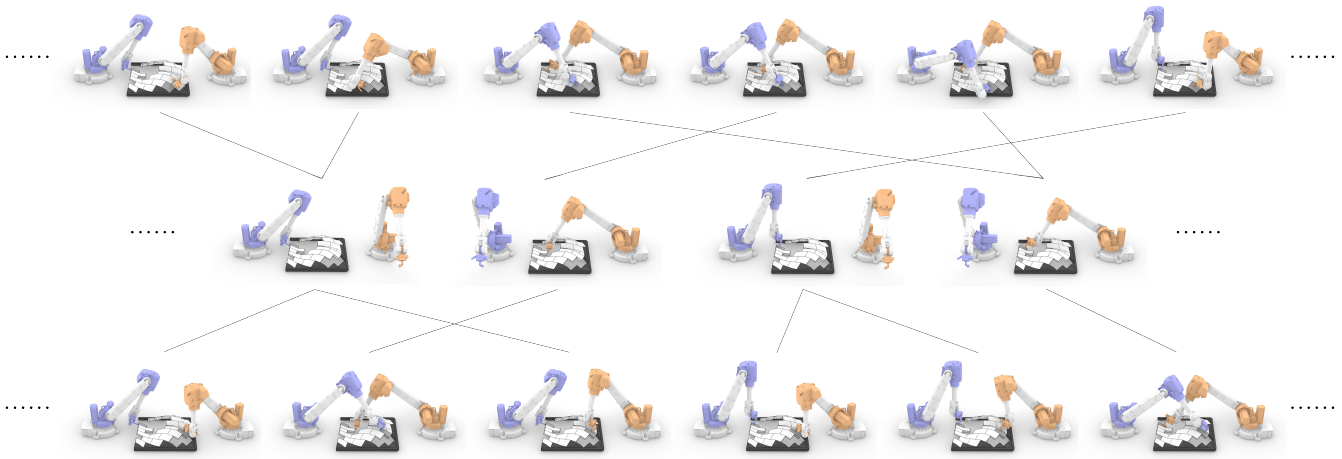


Fig. 10. The illustration depicts a keyframe graph, where each node represents a keyframe. Each row in the graph corresponds to a disassembly state and contains all robot configurations (keyframes) associated with that disassembly state. Edges between keyframes indicate the existence of collision-free robot trajectories for transitioning from one keyframe to another.

Table 2. A performance comparison is conducted to evaluate the effectiveness of our ADMM-QP solver against the Gurobi solver in solving the stability simulation of assemblies with rigid parts. The testing data is generated using a beam search with a width of 64 to plan assembly sequences involving two robots. The results produced by the Gurobi solver serve as the ground truth for this evaluation, and the computation time of the ADMM-QP was done using batches of 256 samples.

Model Name	Test Data Stable / Unstable	ADMM-QP		Gurobi
		Time (ms)	Acc. (%)	Time (ms)
Bottle-12	2'180 / 739	4.21	99.86	6.44
Dog-35	45'658 / 1'687	5.52	99.99	8.40
Dome-37	46'226 / 3'179	4.56	99.82	15.79
Vault-62	78'185 / 7'927	16.84	99.99	90.32
Dome-72	147'694 / 8'159	15.13	99.96	101.16

evaluates the performance of our GPU-based stability simulator. Section 7.2 evaluates the performance of our disassembly policy. Section 7.3 demonstrates results on robotic disassembly planning.

7.1 Statistics on GPU-based Stability Simulation

We compare our ADMM-QP solver with a commercial CPU-based QP solver, Gurobi, for solving the same rigid equilibrium problem introduced in Section 4.2. Specifically, we compare our solver with the CPU-based interior point method implemented in Gurobi. The stability evaluation results generated by the Gurobi solver are considered the ground truth. Rendered versions of all test 3D models are included in the supplementary material and can be accessed through our project repository.

To generate a test dataset, we employ a beam search using the Gurobi solver as the stability checker to plan the assembly process of a given structure. All assembly states, whether stable or not, are collected as a labeled dataset for evaluating our GPU-based stability simulator. The beam search is performed with a width of 64, utilizing two robots for the assembly process. Table 2 presents statistics of our solver's performance on various sizes and shapes of structures. In the worst case, our solver achieves an accuracy of 99.82% and accelerates the simulation time by a factor of 1.5 compared to the CPU-based solver. Note that in addition to these already good results, our method scales better with the number of parts, achieving an average speedup of 6 times acceleration on the largest structure.

Our GPU-based stability simulator does not misclassify an unstable state as stable, as the solver outputs the internal forces. Verifying the equilibrium conditions of these forces ensures that their structural stability results are accurately assessed. However, our GPU-based stability simulator may misclassify a stable state as unstable if the solver fails to converge within the limited number of iterations.

7.2 Statistics on Disassembly Policy

Disassembly from an initial state. We evaluate the performance of our disassembly policy $\pi(\cdot \mid \boldsymbol{s})$, which plans the disassembly sequence for a given structure starting from an initial state $\boldsymbol{s}$ and disassembles all parts that are not on the boundary. The disassembly policy is trained on a dataset generated using a beam search with a beam width of $W = 64$. For testing, we evaluate the policy on a dataset generated using a beam search with double the beam width, $W = 128$. All training data are excluded from the test dataset. Table 3 provides statistics on the performance of our disassembly policies across various sizes and shapes of structures. Note that our algorithm performs less effectively on the test set of the Vault using two robots. Our hypothesis for this lower accuracy is that the disassembly tasks in the test set differ significantly from those in the training set, likely due to its highly challenging geometry. However, we also observe an increase in test accuracy for Vault as the number of robots increases, since the difficulty of the disassembly task decreases with more robots.

Table 3. Evaluation of our disassembly policy for planning the disassembly process with different initial states. The model name is appended with its number of parts. All models are assembled using two robots, except for the Vault, which is also assembled with 3 to 6 robots.

Model Name	Training ($W = 64$)			Testing ($W = 128$)	
	Data	Time (h)	Acc.(%)	Data	Acc.(%)
Bottle-12	1077	0.28	95.70	1159	86.53
Tetris-17	1757	0.25	98.04	2571	92.64
Tetris-14	1337	0.21	94.29	1734	86.55
Tetris-13	1228	0.15	97.33	1447	92.32
Cube-13	1820	0.29	93.74	2680	90.85
Dog-35	4216	1.14	99.15	8130	98.98
Dome-37	4540	1.6	97.36	8433	90.96
Vault-62 (2)	4781	21.3	92.32	10229	38.90
Vault-62 (3)	5876	9.5	97.86	11494	53.44
Vault-62 (4)	5082	7.62	96.18	11054	69.04
Vault-62 (5)	5618	9.85	94.59	10699	70.77
Vault-62 (6)	5619	9.84	91.24	11004	78.39
Dome-72	9018	16.05	97.64	17801	96.40

Table 4. Accuracy statistics for planning disassembly task ends at non-empty target disassembly states using disassembly policies $\pi(\cdot \mid \boldsymbol{s})$ and $\pi(\cdot \mid \boldsymbol{s}, \boldsymbol{s}_{\text{target}})$.

Method	Bottle 12	Tetris 17	Tetris 14	Tetris 13	Cube 13	Dog 35	Dome 37
$\pi(\cdot \mid \boldsymbol{s})$	6.46	3.46	3.51	1.01	2.00	0.04	3.29
$\pi(\cdot \mid \boldsymbol{s}, \boldsymbol{s}_{\text{target}})$	66.03	81.03	77.44	79.65	90.89	90.49	85.23

Additionally, our graph neural network architecture supports policy transfer between different structures. For instance, directly applying the policy trained on the Bottle to plan the disassembly process of the Dog achieves an accuracy of 66% on the training set of the Dog. This promising result highlights the potential of our approach to serve as a foundation model for assembly planning in future work.

Disassembly between two states. We further evaluate the accuracy of our extended disassembly policies $\pi(\cdot \mid \boldsymbol{s}, \boldsymbol{s}_{\text{target}})$, for planning disassembly between two disassembly states. Note that the extended policy, $\pi(\cdot \mid \boldsymbol{s}, \boldsymbol{s}_{\text{target}})$, is trained on extra disassembly tasks with non-empty target disassembly states $\boldsymbol{s}_{\text{target}}$ generated by a beam search with a beam width of 64. For comparison, we modify the baseline policy, $\pi(\cdot \mid \boldsymbol{s})$, by setting the probability of disassembly actions that remove parts in $\boldsymbol{s}_{\text{target}}$ to zero. The test dataset is generated using a beam search to plan the disassembly process of the given structure with a beam width of 128, ensuring that the training set is excluded. Table 4 presents the accuracy of the two policies on the test dataset.

Policy network architecture comparisons. We compare our GNN-based policy with a baseline policy implemented using a multi-layer perceptron (MLP). The MLP policy comprises three hidden layers with 64 neurons each and employs the tanh activation function.

Table 5. Comparisons of neural network architectures. The Bottle model uses two robot arms for disassembly, whereas the Vault model employs four robot arms. The GNN-based policy demonstrates better generalization on the test set compared to the MLP-based policy.

Name	$\pi(\cdot \mid \boldsymbol{s})$	Steps	Time (h)	Train acc. (%)	Test acc. (%)
Bottle-12 (2)	MLP	8'675	0.084	99.82	88.97
	GNN	4'119	0.38	99.45	95.86
Vault-62 (4)	MLP	558'351	9.51	90.35	17.85
	GNN	67'131	7.62	96.18	69.04

Training a PPO agent with this MLP-based policy requires a separate MLP-based value network with the same architecture. The comparisons are shown in Table 5. First, the GNN-based policy exhibits a significantly higher average training time per step, primarily due to its less efficient computational architecture. Second, although both policies can achieve comparable training accuracy, the MLP policy tends to generalize poorly when solving disassembly tasks from novel initial disassembly states. Lastly, policy transfer across different structures is not feasible with the MLP-based approach.

Comparisons with depth-first search. We implement a depth-first search (DFS) method to plan the disassembly processes for Bottle-12 and Dome-37. While DFS successfully identifies a disassembly sequence for Bottle-12 in 82 steps using 0.54s, it fails to find a valid plan for Dome-37 within a 100,000-step limit. These results are consistent with the sparse reward issue encountered during our disassembly policy training, where the proportion of feasible disassembly plans is extremely low compared to the total search space, making it difficult to sample a feasible plan using depth-first search.

7.3 Results on Robotic Disassembly Planning

Part disassemblability. We train disassembly policies with part assemblability to solve disassembly problems presented in [Tian et al. 2024]; see Figure 11. Both models are disassembled using two robots. Our disassembly policy is successfully trained to disassemble the Bird House (top) but fails to disassemble the Bookshelf (bottom). Our disassembly policy of the Bookshelf terminates at a deadlock sub-assembly. This sub-assembly cannot be further disassembled if only one part is allowed to be removed at a time. Figure 11 further clarifies that this failure is caused by the deadlock design of the Bookshelf.

Hybrid robotic assembly planner. We test our hybrid robotic assembly planner for assembling the Dome-37 model using two ABB IRB 2600 robot arms with fixed bases. Figure 12 presents two disassembly plans generated by the same pretrained policy, each starting from a different assembly state, with an average planning time of 5 minutes. The disassembly policy for Dome-37 is trained without considering part assemblability, as each part can be removed along the spherical normal direction without colliding with neighboring parts. The training curriculum is constructed using a dataset generated through beam search with a beam width of 64.

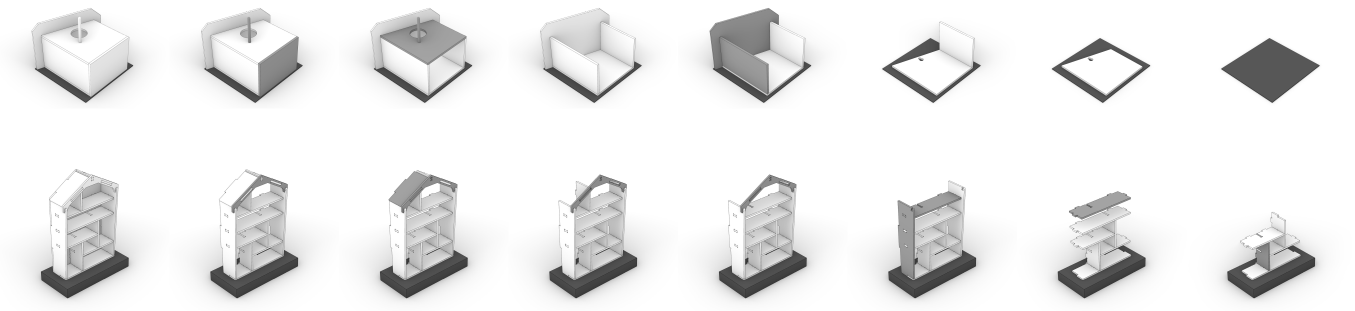


Fig. 11. (Top) A successful sequence for disassembling the Bird House, generated using our disassembly policy. (Bottom) A failed sequence for disassembling the Bookshelf generated using our disassembly policy. The rightmost figure is a deadlock partial assembly state, where each of the four top plates has two incompatible joints. Disassembling this structure requires the simultaneous removal of two parts, an operation not supported by either our method or [Tian et al. 2024].

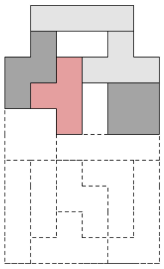
We test our hybrid planner on assembling the Vault model using three ABB IRB 2600 robots. Figure 13 illustrates two disassembly plans with different robot layouts, achieving an average planning time of 96 minutes. Since our policy is trained without differentiating between robots, the robot layout can be adjusted while using the same pre-trained policy. The disassembly policy for the Vault considers part assemblability due to its non-planar contact faces. The policy is trained using a dataset generated through a beam search with a beam width of 256. The total training time is 33.9 hours.

Physical demonstration. We assemble the Bottle model using two ABB GoFa robotic arms. All parts are wooden toy pieces purchased from stores, and the robot grasps them using vacuum suction cups. Ideally, the structure would be constructed following a precomputed assembly sequence, as shown in Figure 14 (top). However, delays in the delivery of the bottom three parts disrupt this planned sequence. In response, our pre-trained disassembly policy efficiently generates an alternative assembly plan for the robots to execute, as shown in Figure 14 (bottom). The average deviation between the real and simulated positions is 1.67 mm, with a maximum deviation of 5 mm. Further details on the physical experiments, including sim-to-real gap analysis, are provided in the supplementary material.

8 CONCLUSION & FUTURE WORK

In this work, we propose a reinforcement learning framework for planning robotic assembly and disassembly sequences to construct structures composed of rigid parts while ensuring the stability of intermediate assembly steps and avoiding collisions between the robots and the structures. We train our disassembly policy using proximal policy optimization. To accelerate training, we develop a GPU-based stability simulator that facilitates parallelized training by running batches of simulations simultaneously on GPUs. To address the challenges posed by sparse rewards in the disassembly process, we introduce a novel curriculum-based training scheme that leverages data generated by assembly planning methods to train the disassembly policy. To improve the data efficiency of training, we propose a graph-based neural network architecture to serve as our disassembly policy. Our framework is further extended to accommodate additional physical constraints, such as part disassemblability and collision-free robot motions. To validate the effectiveness of our framework, we physically construct the Bottle using two robotic arms and generate alternative plans to accommodate the disruption caused by the absence of three bottom parts.

Limitation and future works. First, our current approach operates under the constraint that only one robot is used at a time, with each action removing a single part through linear motion. These assumptions reduce the action space, thereby making the training process more efficient. However, this simplification limits the generality of the method. For instance, as shown in Figure 11, some tasks require the simultaneous removal of multiple parts—something our method cannot currently accommodate. Moreover, disassembly tasks involving rotational motions, such as unscrewing, are also beyond our method's capability. More importantly, deploying multiple robots simultaneously enables parallel task execution, significantly reducing the overall completion time. To overcome these limitations, we plan to extend our framework by enabling dynamic task allocation across multiple robots and incorporating a wider range of disassembly motions in future work.



Second, as pointed out by previous studies [Kao et al. 2022; Yao et al. 2017], a limitation of the rigid body equilibrium method is its inability to detect structural instability caused by sliding failures. For example, the highlighted part in the inset, which is expected to fall under gravitational forces, is erroneously identified as stable by the rigid body equilibrium method. Recent works [Kaufman et al. 2008; Yao et al. 2017] propose a staggered projection approach to address this sliding failure issue by alternately solving two QP problems. The supplementary material demonstrates that the two QP problems in the staggered projection approach are compatible with our ADMM-QP solver. However, the staggered projection approach, even when

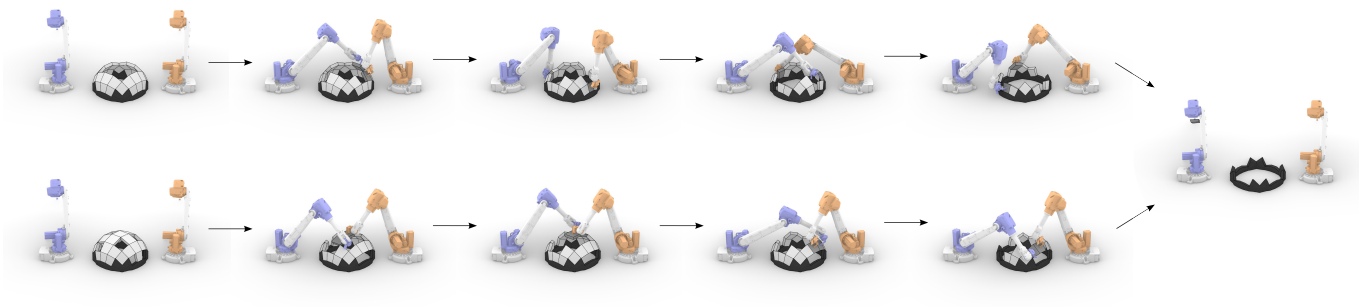


Fig. 12. Two disassembly sequences of the Dome-37 generated using our framework with two robots. Our disassembly policy can efficiently generate alternative disassembly plans from different assembly states.

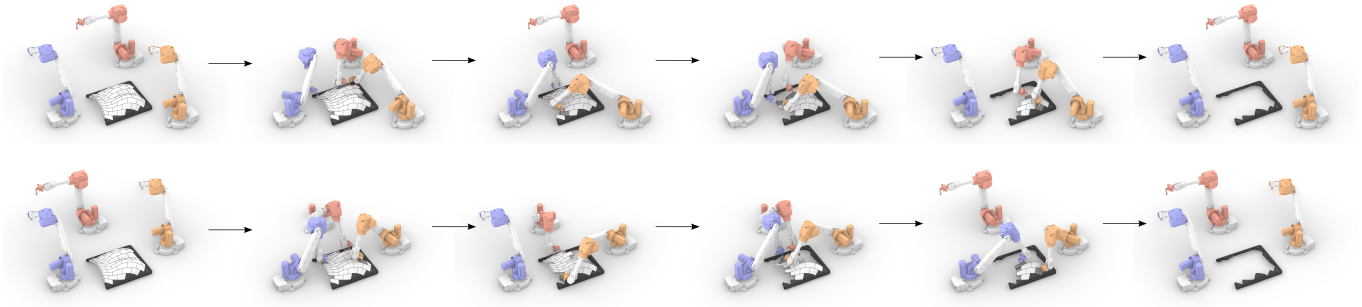


Fig. 13. Two disassembly sequences of the Vault generated using our framework with three robots. We can reuse the same disassembly policy to replan the process, effectively accommodating changes in robot layouts.

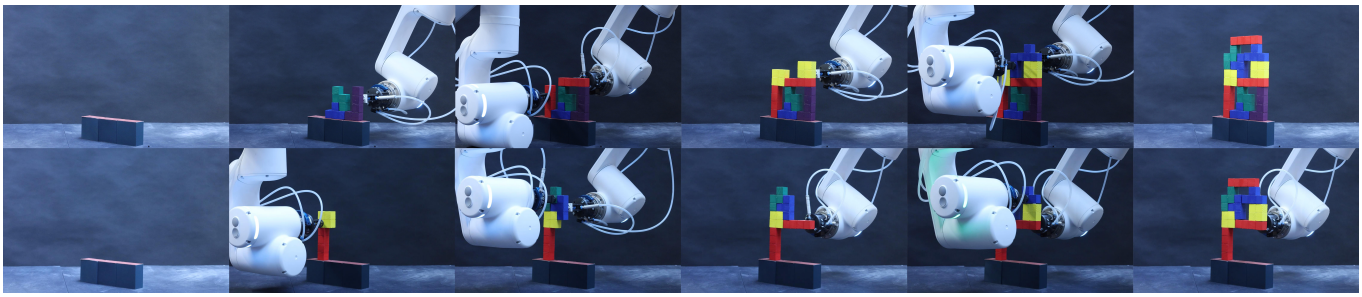


Fig. 14. Physically assembling the Bottle using two ABB GoFa robots. (Top) The robots follow a precomputed assembly plan when all parts are available. (Bottom) Our disassembly policy generates an alternative assembly plan to adapt to the absence of the bottom three parts.

accelerated by GPUs, remains too slow for practical use in RL training. Further accelerating the staggered projection approach on GPUs helps to narrow the sim-to-real gaps.

Third, our approach does not support re-grasping, which would allow robots to release a part they are holding without removing it. The challenge of considering re-grasping actions is the resulting infinite planning horizon, where robots can repeatedly hold and release parts without achieving meaningful progress. This infinite planning horizon problem is difficult to solve using search-based methods, but can be addressed with reinforcement learning by employing a discount factor of less than 1 ($\gamma < 1$). We plan to extend our framework to include re-grasping actions by redesigning the reward function to penalize disassembly plans that require more steps.

Fourth, the maximum number of parts in our test case is 72. Although our GPU-based simulator can verify the stability of assemblies with up to 150 parts, training a policy to disassemble models with such a large number of parts remains extremely time-consuming. To mitigate this, we plan to further optimize our framework to support policy training on large-scale clusters using multiple GPUs. Additionally, our current policy must be retrained for each new model, as its transferability is limited and requires further investigation. To enhance generalizability and accelerate training, we are also interested in exploring alternative reinforcement learning

methods, such as DQN [Mnih 2013], SAC [Haarnoja et al. 2018], and MCTS-based approaches like AlphaGo [Silver et al. 2017].

Fifth, our FT-Graph is effective for encoding assemblies with planar contacts. However, it becomes less efficient when encoding assemblies with curved contacts due to the large number of contact points and normals involved. To address this, we plan to cluster these contact points and normals and encode them as hidden features, reducing the total number of nodes in our FT-Graph, thereby accelerating both training and inference.

Lastly, despite our disassembly policy being capable of achieving instant inference, the robotic motion planner remains a bottleneck in our hybrid disassembly planning framework, as it still requires a considerable amount of planning time. Moreover, although our disassembly policy can accommodate many robots, the hybrid robotic disassembly planning method does not scale well due to the combinatorial complexity of the task. In this work, our method is limited to generating robotic assembly plans involving up to three robot arms. We plan to train a separate robotic motion planning policy by leveraging methods such as [Ha et al. 2020]. This future work will pave the way for the development of a truly real-time robotic assembly planner.

ACKNOWLEDGMENTS

We thank the reviewers for their valuable comments. This research was supported by the EPFL Center for Intelligent Systems, the HKUST start-up grant and NUS Presidential Young Professorship grant (A-0009982-00-00).

REFERENCES

Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. 2022a. Contact and Friction Simulation for Computer Graphics. In *ACM SIGGRAPH 2022 Courses* (Hybrid Event, Vancouver, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 2, 124 pages.

Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. 2022b. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH 2022 Courses*. Association for Computing Machinery, New York, NY, USA, 1–172.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. *Advances in neural information processing systems* 30 (2017).

Aleksandra Anna Apolinarska, Matteo Pacher, Hui Li, Nicholas Cote, Rafael Pastrana, Fabio Gramazio, and Matthias Kohler. 2021. Robotic assembly of timber joints using reinforcement learning. *Automation in Construction* 125 (May 2021), 103569. https://doi.org/10.1016/j.autcon.2021.103569

Patrick Baudisch and Stefanie Mueller. 2017. Personal Fabrication. *Foundations and Trends® in Human–Computer Interaction* 10, 3–4 (May 2017), 165–293. https://doi.org/10.1561/1100000055

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. Association for Computing Machinery, New York, NY, USA, 41–48.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

Philippe Philippe Camille Vincent Block. 2009. *Thrust network analysis: exploring three-dimensional equilibrium*. Ph. D. Dissertation. Massachusetts Institute of Technology.

Stephen Boyd. 2004. *Convex optimization*. Cambridge university press, Cambridge, UK.

Rulin Chen, Ziqi Wang, Peng Song, and Bernd Bickel. 2022. Computational design of high-level interlocking puzzles. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–15.

Yuanpei Chen, Chen Wang, Li Fei-Fei, and C Karen Liu. 2023. Sequential Dexterity: Chaining Dexterous Policies for Long-Horizon Manipulation. In *7th Conference on Robot Learning*.

Hyunsoo Chung, Jungtaek Kim, Boris Knyazev, Jinhwi Lee, Graham W Taylor, Jaesik Park, and Minsu Cho. 2021. Brick-by-brick: Combinatorial construction with deep reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 5745–5757.

Mario Deuss, Daniele Panozzo, Emily Whiting, Yang Liu, Philippe Block, Olga Sorkine-Hornung, and Mark Pauly. 2014. Assembling Self-Supporting Structures. *SIGGRAPH Asia* 33, 6 (2014), 214:1–214:10.

Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M Kaufman, and Daniele Panozzo. 2021. Intersection-free rigid body dynamics. *ACM Transactions on Graphics* 40, 4 (2021).

Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

Niklas Funk, Georgia Chalvatzaki, Boris Belousov, and Jan Peters. 2021. Learn2Assemble with Structured Representations and Search for Robotic Architectural Construction. In *5th Annual Conference on Robot Learning*. London., UK.

Jiahao Ge, Mingjun Zhou, Wenrui Bao, Hao Xu, and Chi-Wing Fu. 2024. Creating LEGO Figurines from Single Images. *ACM Transactions on Graphics (TOG)* 43, 4 (2024), 1–16.

Seyed Kamyar Seyed Ghasemipour, Satoshi Kataoka, Byron David, Daniel Freeman, Shixiang Shane Gu, and Igor Mordatch. 2022. Blocks assemble! learning to assemble with large-scale structured reinforcement learning. In *International Conference on Machine Learning*. PMLR, 7435–7469.

Huy Ha, Jingxi Xu, and Shuran Song. 2020. Learning a Decentralized Multi-arm Motion Planner. In *Conference on Robotic Learning (CoRL)*. arXiv.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

Dan Halperin, Jean-Claude Latombe, and Randall H Wilson. 1998. A general framework for assembly planning: The motion space approach. In *Proceedings of the fourteenth annual symposium on Computational geometry*. 9–18.

Valentin N Hartmann, Andreas Orthey, Danny Driess, Ozgur S Oguz, and Marc Toussaint. 2022. Long-horizon multi-robot rearrangement planning for construction assembly. *IEEE Transactions on Robotics* 39, 1 (2022), 239–252.

David Hoeller, Nikita Rudin, Dhionis Sako, and Marco Hutter. 2024. Anymal parkour: Learning agile navigation for quadrupedal robots. *Science Robotics* 9, 88 (2024), eadi7566.

Yijiang Huang, Caelan Reed Garrett, Ian Ting, Stefana Parascho, and Caitlin Tobin Mueller. 2021. Robotic additive construction of bar structures: Unified sequence and motion planning. *Construction Robotics* (2021), 115–130.

Yuming Huang, Yuhu Guo, Renbo Su, Xingjian Han, Junhao Ding, Tianyu Zhang, Tao Liu, Weiming Wang, Guoxin Fang, Xu Song, et al. 2024. Learning Based Toolpath Planner on Diverse Graphs for 3D Printing. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–16.

Yijiang Huang, Juyong Zhang, Xin Hu, Guoxian Song, Zhongyuan Liu, Lei Yu, and Ligang Liu. 2016. Framefab: Robotic fabrication of frame shapes. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–11.

Ryan Luke Johns, Martin Wermelinger, Ruben Mascaro, Dominic Jud, Fabio Gramazio, Matthias Kohler, Margarita Chli, and Marco Hutter. 2020. Autonomous dry stone: On-site planning and assembly of stone walls with a robotic excavator. *Construction Robotics* 4, 3-4 (Dec. 2020), 127–140. https://doi.org/10.1007/s41693-020-00037-6

Ryan Luke Johns, Martin Wermelinger, Ruben Mascaro, Dominic Jud, Ilmar Hurkxkens, Lauren Vasey, Margarita Chli, Fabio Gramazio, Matthias Kohler, and Marco Hutter. 2023. A framework for robotic excavation and dry stone construction using on-site materials. *Science Robotics* 8, 84 (2023), eabp9758.

Gene Ting-Chun Kao, Antonino Iannuzzo, Bernhard Thomaszewski, Stelian Coros, Tom Van Mele, and Philippe Block. 2022. Coupled rigid-block analysis: Stability-aware design of complex discrete-element assemblies. *Computer-Aided Design* 146 (2022), 103216.

Satoshi Kataoka, Youngseog Chung, Seyed Kamyar Seyed Ghasemipour, Pannag Sanketi, Shixiang Shane Gu, and Igor Mordatch. 2023. Bi-Manual Block Assembly via Sim-to-Real Reinforcement Learning. *arXiv preprint arXiv:2303.14870* (2023).

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

Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12, 4 (1996), 566–580.

Manav Kulshrestha and Ahmed H Qureshi. 2023. Structural Concept Learning via Graph Attention for Multi-Level Rearrangement Planning. In *7th Annual Conference on Robot Learning*.

Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022. Penetration-free projective dynamics on the GPU. *ACM Transactions on Graphics* 41, 4 (July 2022), 1–16. https://doi.org/10.1145/3528223.3530069

Steven LaValle. 1998. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811* (1998).

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. 2020a. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM*

Trans. Graph. 39, 4 (2020), 49.

Richard Li, Allan Jabri, Trevor Darrell, and Pulkit Agrawal. 2020b. Towards Practical Multi-Object Manipulation using Relational Reinforcement Learning. In *ICRA*. IEEE, 4051–4058. https://doi.org/10.1109/ICRA40945.2020.9197468

Yunfei Li, Tao Kong, Lei Li, and Yi Wu. 2022. Learning Design and Construction with Varying-Sized Materials via Prioritized Memory Resets. In *ICRA*. IEEE Press, Philadelphia, PA, USA, 7469–7476. https://doi.org/10.1109/ICRA46639.2022.9811624

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. 2018. Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning*. PMLR, 270–282.

Andre Menezes, Pedro Vicente, Alexandre Bernardino, and Rodrigo Ventura. 2021. From Rocks to Walls: a Model-free Reinforcement Learning Approach to Dry Stacking with Irregular Rocks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, Nashville, TN, USA, 2057–2065. https://doi.org/10.1109/CVPRW53098.2021.00234

Volodymyr Mnih. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. 2021. Large steps in inverse rendering of geometry. , 13 pages. https://github.com/rgl-epfl/cholespy

Neal Parikh, Stephen Boyd, et al. 2014. Proximal algorithms. *Foundations and trends® in Optimization* 1, 3 (2014), 127–239.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)* 37, 4 (2018), 1–14.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv:1712.01815 [cs.AI] https://arxiv.org/abs/1712.01815

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. 2020. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation* 12, 4 (2020), 637–672.

Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. 2023. CuRobo: Parallelized Collision-Free Robot Motion Generation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 8112–8119. https://doi.org/10.1109/ICRA48891.2023.10160765

Yunsheng Tian, Karl DD Willis, Bassel Al Omari, Jieliang Luo, Pingchuan Ma, Yichen Li, Farhad Javid, Edward Gu, Joshua Jacob, Shinjiro Sueda, et al. 2024. ASAP: automated sequence planning for complex robotic assembly with physical feasibility. In *ICRA*. IEEE, 4380–4386.

Yunsheng Tian, Jie Xu, Yichen Li, Jieliang Luo, Shinjiro Sueda, Hui Li, Karl D.D. Willis, and Wojciech Matusik. 2022. Assemble Them All: Physics-Based Planning for Generalizable Assembly by Disassembly. *ACM Trans. Graph.* 41, 6, Article 278 (2022), 15 pages.

Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

Gabriel Vallat, Jingwen Wang, Anna Maddux, Maryam Kamgarpour, and Stefana Parascho. 2023. Reinforcement learning for scaffold-free construction of spanning structures. In *SCF '23: Proceedings of the 7th Annual ACM Symposium on Computational Fabrication*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3623263.3623359

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

Jingwen Wang, Wenjun Liu, Gene Ting-Chun Kao, Ioanna Mitropoulou, Francesco Ranaudo, Philippe Block, and Benjamin Dillenburger. 2023b. Multi-robotic assembly of discrete shell structures. In *Advances in Architectural Geometry 2023*. De Gruyter, 261–274.

Ziqi Wang, Florian Kennel-Maushart, Yijiang Huang, Bernhard Thomaszewski, and Stelian Coros. 2023a. A Temporal Coherent Topology Optimization Approach for Assembly Planning of Bespoke Frame Structures. *ACM Transactions on Graphics (SIGGRAPH 2023)* 42, 4 (2023).

Emily Whiting, John Ochsendorf, and Frédo Durand. 2009. Procedural modeling of structurally-sound masonry buildings. In *ACM SIGGRAPH Asia 2009 papers*. 1–9.

Randall H. Wilson. 1992. *On Geometric Assembly Planning*. Ph. D. Dissertation. Stanford University.

Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. 2020. Allsteps: curriculum-driven learning of stepping stone skills. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 213–224.

Zeshi Yang, Zherong Pan, Manyi Li, Kui Wu, and Xifeng Gao. 2023. Learning based 2D irregular shape packing. *ACM Transactions on Graphics (TOG)* 42, 6 (2023), 1–16.

Jiaxian Yao, Danny M Kaufman, Yotam Gingold, and Maneesh Agrawala. 2017. Interactive design and stability analysis of decorative joinery for furniture. *ACM Transactions on Graphics (TOG)* 36, 2 (2017), 1–16.

Xinya Zhang, Robert Belfer, Paul G Kry, and Etienne Vouga. 2020. C-space tunnel discovery for puzzle path planning. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 104–1.