Abstract—Research in manipulation of deformable objects is typically conducted on a limited range of scenarios, because handling each scenario on hardware takes significant effort. Realistic simulators with support for various types of deformations and interactions have the potential to speed up experimentation with novel tasks and algorithms. However, for highly deformable objects it is challenging to align the output of a simulator with the behavior of real objects. Manual tuning is not intuitive, hence automated methods are needed. We view this alignment problem as a joint perception-inference challenge and demonstrate how to use recent neural network architectures to successfully perform simulation parameter inference from real point clouds. We analyze the performance of various architectures, comparing their data and training requirements. Furthermore, we propose to leverage differentiable point cloud sampling and differentiable simulation to significantly reduce the time to achieve the alignment. We employ an efficient way to propagate gradients from point clouds to simulated meshes and further through to the physical simulation parameters, such as mass and stiffness. Experiments with highly deformable objects show that our method can achieve comparable or better alignment with real object behavior, while reducing the time needed to achieve this by more than an order of magnitude. Videos and supplementary material are available at https://tinyurl.com/diffcloud.

I. INTRODUCTION

We consider the real-to-sim problem of inferring parameters of general-purpose simulators from real observations such that the gap between reality and simulation is reduced [1]–[4]. A common approach to solving this problem is to train an inverse model on data generated with a black-box (non-differentiable) simulator using a wide range of parameter settings. The input to such models usually consists of trajectories of a low-dimensional state of the system, e.g. position and orientation of rigid objects in the scene. The output are parameters, such as mass, friction, and other physical object properties.

The state of a highly deformable object cannot be captured by only its position and orientation, since the object deforms during motion. Hence, we represent objects with point clouds as observed through depth cameras. Recent neural network architectures, such as PointNet++ [5] and MeteorNet [6], are well suited for learning to process point clouds. As we will show, they can yield inverse models that offer a viable solution to the challenging task of real-to-sim for deformables from point clouds. Nonetheless, data collection and training for these can be computationally demanding.

In this work, we propose an alternative approach that employs a differentiable simulator to allow adjusting simulation parameters directly via gradient descent without the need for dataset collection and pre-training. Our approach combines differentiable point cloud rendering and differentiable simulation to bring the behavior of simulated, highly deformable objects closer to that of real objects. We instantiate a scene with a simulated object and create an end-to-end differentiable pipeline that lets us seamlessly propagate the gradients from real point clouds to the low-level physical simulation parameters. Even small changes in these parameters can have a significant impact on the behavior of deformable objects.

We show that establishing end-to-end differentiability yields a faster alignment between simulation and reality, compared to training inverse models with a black-box (i.e. gradient-free) treatment of the simulator.

We conduct a set of experiments where a robot manipulates highly deformable real objects, such as cloth and paper towels. We show that our approach successfully infers simulation parameters, such as mass and stiffness, making the behavior of simulated deformables match the real ones. In simulation experiments, we explore interactions of the deformables with rigid objects: stretching a band on a pole, and hanging a vest onto a rigid pole. Overall, our experiments show that we can obtain similar or better alignment between the simulated and the target object, compared to the inverse model baselines. The major benefit of our approach is that it obtains such an alignment after on average 10 minutes of direct gradient descent, replacing 2.5 – 5.5 hours of data collection and training for the inverse models.
II. BACKGROUND

A. Real-to-Sim for Deformable Objects

Many approaches in deformable object manipulation are limited to specific scenarios due to the complexity of real hardware setups [7]–[11]. Furthermore, there is a lack of easily tunable yet realistic simulators that support deformables and could aid experimentation with novel tasks. Hence, automated ways to find simulation parameters are needed to make the behavior of simulated deformables resemble that of their real-world counterpart. In robotics, this real-to-sim problem has been extensively studied for rigid objects [12]–[15]. However, these methods assume access to a low-dimensional state, such as object poses. Deformable objects on the other hand comprise a high-dimensional state space that is only partially observable in raw images and point clouds or in contact-rich scenarios. Recent surveys have reviewed learning-based approaches for manipulation and perception of deformables [16], [17], including methods for tracking and registration. However, such methods either employ markers [18], assume a known and simplified deformation model [19], or show applicability only to specific cases, e.g. a rope lying flat on a surface [20]. As of now, there is no generic tracking or registration approach for deformables that is robust in a wide range of scenarios.

B. Inverse Models for Real-to-Sim

One approach towards real-to-sim is to learn an inverse model that maps from a sequence of sensor observations to simulation parameters [21]. In robotics, several earlier works explored using inverse models for highly deformable objects. For example, in [22] the authors propose a piecewise linear elastic material model for cloth, then they fit the material model to real cloths by applying controlled forces and measuring the deformation response. This results in a paired dataset of cloth types and estimated material parameters, but limits generalization to unseen cloths without additional physical measurements. In [23], the authors propose using the dataset of [22] as a source of supervision for training a network to regress material parameters from RGB videos of cloth. This supervised training approach relies heavily on domain randomization to achieve generalization to cloths with unseen colors or patterns. Additionally, the proposed regression framework discretizes the space of outputs to deal with the high dimensional model of cloth deformation, limiting the range of cloth types that can ultimately be accurately modelled. In this work, we use a sequence of depth images (converted to point clouds) as input to our models. Using point clouds instead or RGB images allows us to circumvent the need for extensive domain randomization and for contrasting textures (for foreground/background segmentation). To compare our gradient-based approach against inverse model baselines, we adopt established methods for processing point clouds, such as PointNet/PointNet++ [5], [24], MeteorNet [6] and train them to regress to continuous simulation parameters.

C. Differentiable Rendering and Differentiable Simulation

Differentiable rendering of images allows using 2D observations to inform 3D understanding, by propagating gradients from images to illumination models, 3D objects in the scene, and camera pose. Various works have explored making the rasterization process (coloring image pixels based on the assignment of triangular mesh faces) fully differentiable, including DIB-R [25] and SoftRAS [26]. Other works explore making the processes of ray tracing [27] and volumetric rendering [28], [29] differentiable. Toolkits, such as Kaolin [30] and PyTorch3D [31] provide batched implementations of differentiable rasterization, allowing a significant speedup of these computationally expensive processes. In this work, we aim to do correspondence-free alignment of simulated object states, such as meshes, with real visual observations. While images are one such observation space to bridge the gap, achieving close alignment between images that are differentiably rendered from simulated meshes and real images still remains an open challenge due to the range of colors, textures, and fine details present in real images. Thus, we use point clouds to represent objects and build on PyTorch3D [31], which offers a way to differentiably sample a point cloud from a given mesh such that gradients from losses in the point clouds propagate to the mesh vertex gradients. Furthermore, PyTorch3D library exposes efficient data structures and CUDA-enabled batched operations for 3D meshes and point clouds, allowing seamless integration into simulator backends that support PyTorch.

Differentiable simulation is a promising paradigm that allows to adapt simulators by propagating gradients of the simulator outputs w.r.t. the lower-level physical simulation parameters. Recent differentiable simulators with support for deformables include [32]–[41]. Most of these only support a limited set of interactions and types of deformation. For example, out of the above, only [32] (a differentiable version of ARCSim [42]) supports arbitrary meshes and modeling interactions of rigid and deformable objects. From the simulators listed above, only gradSim [33] provides a combination of differentiable RGB rendering and simulation. This approach is conceptually related to our work, but is still substantially different, since it does not handle point clouds, does not support interactions between rigid and deformable objects, and presents only sim-to-sim results with simple simulated images and plain textures. The differentiable simulation framework proposed in [43] shows results on inferring deformation properties of a real object from point clouds. We do not build on this approach directly, because it has several key limitations. First, there are limitations in the underlying simulator: the model is aimed for objects with low-to-medium levels of deformability, such as tight plush toys and pillows. In contrast, we are interested in highly deformable objects, such as cloth, garments, and stretchable bands. Second, there are limitations in the loss formulation: the point cloud loss proposed in [43] relies on a simple diffusion of the observed deformation into a deformation field. This is not suitable for the case of highly deformable objects.
that rapidly and drastically change shape under gravity and upon interacting with other objects. Finally, this prior work is geared towards analyzing cases with passive dynamics of a single object falling and colliding with the table or bending under gravity; the simulator also lacks support for modeling frictional contacts. We are instead interested in actuating a robot to perform manipulation with highly deformable objects that can also interact in non-trivial ways with rigid objects in the scene. We found that [32] is the only available simulator that readily supports such functionality, hence we incorporate it into our approach.

III. OUR APPROACH: DIFFCLOUD

Our objective is to discover the physical simulation parameters (e.g. stiffness, mass, friction) that would cause the behavior of the simulated deformable objects to match the behavior of observed real objects. We start by recording a sequence of point cloud observations of the scene, where a robot manipulates a deformable object. Then, we construct a simulated environment in a differentiable simulator that can load meshes of deformable & rigid objects in the scene and simulate their interactions. To obtain simulated point clouds we implement a differentiable point cloud sampler: we sample points on the surface of the simulated objects at each time step and performs collision resolution in localized outlier filtering and post-processing techniques (see the red patches highlighted in the real point cloud in Figure 2). Thus, we opt for the unidirectional Chamfer distance between simulated and real point clouds:

\[
\text{Equation 1 is not directly applicable to our setting where real point clouds captured by low-cost depth sensors contain noise and partial occlusion not present in clean, simulated point clouds. We consider the following two comparisons with different tradeoffs: the unidirectional Chamfer distance between (a) real and simulated points and (b) simulated and real points. (a) can be artificially inflated due to real sensor noise for otherwise aligned point clouds, and (b) can wrongly penalize parts that are occluded in the real point cloud but are visible in the fully observable simulated point cloud. Empirically, we find both (a) and (b) to be functionally comparable depending on the degree of noise and occlusion present. In our case, the effects of partial observability are easier to mitigate by using merged point clouds from two depth cameras, but sensor noise can still persist even after outlier filtering and post-processing techniques (see the red patches highlighted in the real point cloud in Figure 2). Thus, we opt for the unidirectional Chamfer distance between simulated and real point clouds:}

\[
d_{\text{Chamf}}(\mathcal{P}_{\text{sim}}, \mathcal{P}_{\text{real}}) = \sum_{x^\text{sim}, \epsilon = \mathcal{P}_{\text{sim}}} \min_{x^\text{real} \in \mathcal{P}_{\text{real}}} ||x^\text{sim} - x^\text{real}||^2. \tag{2}
\]

Using an efficient GPU-based implementation in PyTorch3D (see Section 3.1 in [31]), we can quickly propagate gradients from the point clouds through the mesh representation to optimize the low-level physical simulator parameters.

B. Gradient Propagation

We build upon the differentiable simulator DiffSim [32], which supports mesh-based simulation and contact-handling of rigid objects and thin-shell deformables (e.g. cloth). In DiffSim, the simulation state is represented by generalized coordinates \( \mathbf{q} = [q_1^T, q_2^T, \ldots, q_n^T]^T \) of all objects in the simulation with corresponding velocities \( \dot{\mathbf{q}} = [\dot{q}_1^T, \dot{q}_2^T, \ldots, \dot{q}_n^T]^T \). The generalized coordinates of a rigid body have \( q_i \in \mathbb{R}^6 \) for rigid objects, denoting position and orientation, and \( q_i \in \mathbb{R}^3 \) for deformable nodes with 3 DoF for position alone. Here, \( n \) is the cumulative total of the number of deformable nodes and rigid bodies in the scene. DiffSim uses the implicit Euler method to compute \( \mathbf{q}, \dot{\mathbf{q}} \) at each time step and performs collision resolution in localized impact zones. A given mesh has a body frame with the origin set to its center of mass at the start of simulation.

\[\text{Ablations: https://tinyurl.com/diffcloud-ablations}\]
A mesh vertex \( p \) has coordinate \( p_0 = (p_x, p_y, p_z)^T \) in the body frame and world coordinate \( p = f(q) = [r]p_0 + t \), where \( r = (\phi, \theta, \psi)^T \) and \( t = (t_x, t_y, t_z) \) is the 6-DoF pose of the mesh. Propagating gradients from vertex \( p \) to the generalized coordinates \( q \) involves computing the Jacobian \( \nabla F \) and obtaining the partial derivatives \( \partial f(q)/\partial q \). In this way, DiffSim is able to solve sim-to-sim optimizations by comparing current mesh states to target mesh states and propagating gradients from observed positional differences.

Since ground truth mesh states are not readily available for real deformables, we use point clouds as an observation space. To allow losses in the point cloud space to be propagated to simulated cloth vertices, we implement a differentiable point cloud sampling step drawing from widely-used techniques in graphics [45]. A triangular mesh face can be represented by its enclosing vertices \( p_1, p_2, p_3 \). In barycentric coordinates, a random point on the surface of the face can be generated by sampling 3 random numbers \((u, v, w)\) such that \( u + v + w \leq 1 \) [31], [45]. Hence, we can obtain a random point \( x \) inside the triangle as:

\[
x = up_1 + vp_2 + wp_3.
\]  

(3)

To generate an \( N \)-point, uniform-density point cloud from a mesh, we first sample \( N \) triangular faces \( \{(p_1, p_2, p_3)\}_{i=1,...,N} \), weighted proportionally to the area of each face. This step ensures that the resulting point cloud is evenly distributed over the surface of the mesh, instead of being oversaturated in regions with closely-packed mesh faces. For the \( i \)-th sampled face \( (p_1, p_2, p_3) \), we generate random coefficients \((u_i, v_i, w_i)\) and use Equation 3 to compute a point \( x_i \) that lies on the face. Concatenating the points obtained by applying this procedure to the \( N \) sampled faces and coefficients yields the point cloud \( P = \{x_i\}_{i=1,...,N} \). We connect the PyTorch3D [31] implementation of this sampling procedure to the output of a differentiable simulator. With that, gradients from loss functions operating on \( \{x_i\}_{i=1,...,N} \) can be propagated to mesh vertices \( \{p_1, p_2, p_3\} \) via chain rule, observing that:

\[
\partial x_i / \partial p_{1,1} = u_i \quad \partial x_i / \partial p_{1,2} = v_i \quad \partial x_i / \partial p_{1,3} = w_i
\]  

(4)

C. Optimization

We consider scenarios where a robot executes a trajectory to manipulate a deformable object, which potentially also interacts with other objects in the scene. For each scenario, we get a real point cloud sequence of length \( T \) and a resolution of \( N \) points per frame: \( \{P_{t}^{\text{real}} = \{x_{i}^{\text{real}}\}_{i=1,...,N}\}_{t=1}^{T} \). We instantiate each real scenario in DiffSim using representative meshes to model the deformable and rigid objects in the scene initially. We begin optimizing the material properties of the simulated cloth such that the discovered parameters best explain the observed target point cloud data. Each simulated scenario involves manipulating a simulated cloth with the same trajectory as in real. Even with identical trajectory execution, we expect a discrepancy between the real and simulated point cloud sequences due to mismatch in the motion of the deformables, whose physical properties we aim to optimize. For each iteration of optimization, we run the simulation for the number of steps in the scenario horizon using the current set of estimated parameters. At each timestep \( t \), we sample the simulated object surfaces in the scene according to Equation 3 to obtain a point cloud. The concatenated point clouds across frames give the simulated point cloud sequence: \( \{P_{t}^{\text{sim}} = \{x_{i}^{\text{sim}}\}_{i=1,...,N}\}_{t=1}^{T} \). In practice, we compute the unidirectional Chamfer distance (Equation 2) on \( \{P_{t}^{\text{sim}}, P_{t}^{\text{real}}\} \) for the corresponding point clouds observed at time \( t \). For most experiments, we found that using only the last frame works well, i.e. \( t = T \), as in [32], [33]. Other frames can be used, and their selection could be considered as a hyperparameter. This loss is propagated across all frames of the simulation and used to update the underlying parameters of the deformable object. This procedure is repeated for a fixed number of iterations or until the Chamfer distance falls below a threshold.

IV. EXPERIMENTS

In this section, we compare DiffCloud with non-differentiable methods for parameter estimation: we evaluate the degree of point cloud alignment between simulated and real trajectories and the compute efficiency across methods.

A. Baseline Inverse Models

As experimental baselines, we use methods that view simulators as black-box, i.e. not treating observations as end-to-end differentiable w.r.t the parameters. These inverse models are implemented as regression networks that take point cloud sequences as inputs and predict \( k \) simulation parameters. They are trained on simulated point cloud sequences generated by simulations with various parameters.

- **METEORNET**: A state-of-the-art architecture for learning representations of 3D point cloud sequences from [6], which we modify to predict \( k \) simulation parameters. Specifically, we use MeteorNet-clss (Appendix D.2 in [6]).
- **POINTNET++**: A regressor similar to the above, but using the multi-scale group architecture from [5] (Appendix B.1) to extract features from a single point cloud; uses three set abstraction layers followed by three fully connected layers with output sizes \((512, 256, k)\).
MLP: A regressor with a fully connected network that also operates on a single frame; uses five 2D convolutional layers with output sizes (64, 64, 64, 128, 1024), a symmetric max pooling layer, two fully connected layers with output sizes (512, 256), a dropout layer, and a final fully connected layer with k outputs.

Real point cloud sequences suffer from partial observability, self-occlusion, and noise. To minimize the gap between simulation and reality, we aim to generate a training dataset for each of the regressor methods that is as realistic as possible. As training data for the regressors (METEORNET, POINTNET++, MLP), we initialize N simulations with uniformly sampled parameters and record the resulting point cloud sequences and ground truth parameters. This yields a dataset \( D = \{\{x_t\}_{t=1}^{T}, w = [w_{\text{stiff}}, w_{\text{mass}}]\}_{i=1}^{N} \). For each simulation run, we generate a point cloud sequence from mesh states according to Equation 3. From bias the sampling of points to be on a subset instead of all faces of the deformable to mimic a partial point cloud with occlusion. Then, we apply random Gaussian noise to the point cloud to mimic the effect of real sensor noise. We generate a dataset of 1500 training and 375 test point cloud sequences with 3,500 points per frame across methods. For comparison, METEORNET was trained on a dataset of 576 examples and the PointNet++ [5] literature uses thousands of examples.

We train METEORNET to learn a function \( f : \{x_t\}_{t=1}^{T} \rightarrow w \), mapping an input point cloud sequence \( \{x_t\}_{t=1}^{T} \) to simulator parameters \( w \) that explain the observed behavior. POINTNET++ and MLP methods learn a function \( g : x_t \rightarrow w \) mapping only a single frame \( t \) (selected from a sequence) to the material parameters. In practice, we choose the frame \( t \) to be the same frame for which we compute the unidirectional Chamfer distance in DIFFCLOUD optimization. We train each network using \( L_1 \) loss but bias the predicted and ground truth parameters, using the Adam optimizer with a learning rate of 0.001 for 100 epochs. We run dataset generation on an Intel i5-9400F CPU. We use an NVIDIA GeForce GTX 1070 GPU for training and optimization.

B. DIFFCLOUD Implementation Details

DIFFCLOUD is our proposed approach described in Section III. For all simulations we use DiffSim [32] – a differentiable version of the garment simulator ARCSim [42]. The original parameter space for cloth in ARCSim consists of high-dimensional tensors representing the material model. However, [23] finds that initializing cloth with a basis material and scaling the base parameters can adequately model a range of deformation behaviors while reducing the dimensionality of the parameter space, leading to our optimization on only scalar stiffness and mass multipliers, \((w_{\text{stiff}}, w_{\text{mass}})\). These multipliers represent the simulation parameters we aim to learn (Figure 3). We initialize them to the midpoint of each parameter range. We empirically determine this range as \([0.1, 10]\) simulation units for all parameters, so that forward simulation is numerically stable.

To encourage small changes in the learnable parameters to make a visually compelling difference in deformation, we optimize intermediate variables \((s_{\text{stiff}}, s_{\text{mass}})\) instead of directly optimizing \((w_{\text{stiff}}, w_{\text{mass}})\). Here, \((w_{\text{stiff}}, w_{\text{mass}}) = \sigma(s_{\text{stiff}}, s_{\text{mass}}) \times (10 - 0.1) + 0.1\), and \(\sigma(x) = \frac{1}{1 + e^{-x}}\) is the standard sigmoid function. Intuitively, this maps \((s_{\text{stiff}}, s_{\text{mass}})\) first to the range \([0, 1]\) via the sigmoid function, and then interpolates this value within the parameter range of \([0.1, 10]\). We choose the final parameters to be those that incurred the lowest loss during optimization.

C. Evaluation Metrics

Alignment: For quantitative evaluation we use unidirectional Chamfer distance (Equation 2) that characterizes how well the motions of the simulated and real point clouds align. For each of the baselines, we first infer the predicted parameters using the target point cloud trajectory as input. These parameters serve as input to the simulation engine. We then run the simulator using the inferred parameters and generate point clouds from the simulated meshes. We compute the Chamfer distance between the point clouds generated by the baselines and the real point cloud, then compare this against the loss from running DIFFCLOUD on the real point cloud.

Efficiency: Aside from alignment, we also evaluate the computational resources of all methods. Across experiments, we report (1) the time it takes to run DIFFCLOUD optimization on trajectories and (2) the combined dataset generation, training, and inference times for the supervised baselines.

D. Real Experimental Setup

1) Hardware Setup: Our hardware setup consists of a Kinova Gen3 robot arm with a Robotiq 2F-85 gripper, a 50 × 43 cm table, and two Intel RealSense D435 cameras, mounted on the side and overhead, respectively (Figure 1). For each scenario outlined below, we execute trajectories using velocity control in the Cartesian (end-effector) space and record robot joint states and point cloud observations at each step. We merge point clouds from both camera views and use a standard camera-to-robot transformation obtained from checkerboard calibration to obtain point clouds in the robot frame of reference, which has a corresponding frame of reference in simulation. Using recorded joint states and known robot geometry, we mask out the end-effector from the point clouds as depicted in Figure 1. This yields a set of real, post-processed point cloud observations \(\{x_{i,\text{real}}\}_{i=1}^{T}\).

2) Real Scenarios: We consider two real manipulation scenarios using cloth as the deformable object of interest. The lift scenario involves grasping and lifting a cloth from a flat start state on a table surface. At the end of the trajectory, the cloth is lifted off the table and suspended mid-air. The fold scenario involves folding a cloth in half, starting from a diamond shape and ending in a triangular configuration. We perform both scenarios on 5 different fabrics grouped into three categories: highly deformable, medium, and shape retaining. The highly deformable class consists of cloths that are especially low stiffness and collapsible, such as silk-like materials. The medium class consists of cloths with modest deformation that can still crumple subject to enough
force. The \textit{shape retaining} class contains cloths that resist deformation more than the other categories, such as paper towels and stiff washcloths. For each fabric, we execute 3 trajectories for a total of 15 trajectories per scenario. Each trajectory is ≈2.5 seconds long with robot commands sent at 10 Hz., resulting in a horizon length of $T = 25$. During the lift and fold trajectories, the cloth either collapses or maintains its shape depending on the underlying cloth properties (which we attempt to learn).

### E. Real Robot Experiments

We aim to infer cloth stiffness and mass from the \textit{lift} and \textit{fold} scenarios discussed in Section IV-D.

1) \textbf{DIFFCLOUD Specifications:} For \textbf{DIFFCLOUD} loss and evaluation metrics, we need to consider the frames for which deformation is most visible. The lift trajectory exhibits least occlusion at the end of the trajectory when the cloth is off the table. For folding, the cloth is most visible in the middle of the trajectory. The level of occlusion at the end of real lift trajectories is minimal, so simulated point clouds are generated by sampling from all cloth faces. In the fold scenario, the half of the real cloth that remains flat on the table does not appear in the real point cloud. Hence, we sample simulated point clouds only from cloth faces on the upper half of the cloth in simulation, as a heuristic for generating occlusion-sensitive point clouds. Occlusion handling is out of scope with the sampling procedure discussed in Section III-B, but we hope to relax this in future work. The simulated counterparts for the lift and fold scenarios use a 2D cloth mesh with 49 vertices, where we keyframe the position of the top corner vertex to produce the same motion that is executed in real for $T = 25$ steps. For both scenarios, we run \textbf{DIFFCLOUD} for 50 iterations using the Adam optimizer with a learning rate of 0.2 to infer $(w_{\text{stiff}}, w_{\text{mass}})$.

2) Results: We find that \textbf{DIFFCLOUD} is able to estimate mass and stiffness cloth parameters that visually explain the observed real point cloud behavior in the lift (Figure 4) and fold (Figure 5) scenarios. We note that \textbf{DIFFCLOUD} achieves a lower loss than \textbf{METEORNET}, \textbf{POINTNET++}, and \textbf{MLP} across the 3 cloth categories in the lift scenario (Figure 4), according to the evaluation metric from Section IV-C. We also visualize the final parameters found by \textbf{DIFFCLOUD} and observe that the parameters align with the qualitative descriptions of the 3 categories of cloth types considered (Figure 3). For instance, \textbf{DIFFCLOUD} discovers high mass, low stiffness parameters for the polka dot cloth and low mass, high stiffness parameters for the red black cloth, which belong to the highly deformable and shape retaining categories, respectively. Similarly, for the fold scenario, we find that \textbf{DIFFCLOUD} is able to find parameters that account for variations in deformation during execution, such as collapsing inwards mid-fold or maintaining shape (Figure 5). The severity of self-occlusion is more prominent in the fold scenario, complicating robust parameter estimation. Still, \textbf{DIFFCLOUD} achieves a loss on par with the baselines, which are (1) trained from thousands of examples, (2) use data augmentations to be invariant to varying degrees of self-occlusion, and (3) require significantly more compute time. \textbf{DIFFCLOUD} takes on average 10 minutes per trajectory in the lift and fold scenarios to optimize the simulation parameters (Figure 3). While the baseline regressors all have inference times on the order of milliseconds, each method incurs an up-front cost of more than two hours of dataset generation. Furthermore, the training procedure requires 40 minutes to multiple hours per scenario. With significantly less computational footprint, \textbf{DIFFCLOUD} achieves parameter estimation results on real data that are comparable and in some cases better than baselines.

### F. Further Simulation Experiments

1) Simulation Scenarios: We further evaluate the performance of \textbf{DIFFCLOUD} on two additional simulated scenarios: hanging one shoulder of a vest onto a pole (\textit{vest hang}), and stretching an elastic band against a pole (\textit{band stretch}), shown in Figure 6. The mass and stiffness of the vest mesh determine the outline of the vest when hung on
Fig. 6. We evaluate all competing methods using the metric provided in Section IV-C on parameter estimation in simulation: band stretch and vest hang. Across 10 runs per scenario, DIFFCLOUD achieves alignment between simulated and target point clouds in all runs, while baseline regressors achieve mixed success.

The mass and stiffness of the elastic band dictate the extent to which the band travels up or down along the pole when pulled taut. We aim to analyze how the contact-rich aspects of these scenarios influence the performance of various methods. To separate these effects from point cloud quality considerations, we use fully observable noise-free point cloud observations. Aside from skipping augmentations (random jittering, dropout of cloth faces) during dataset generation, we use an identical training procedure for MLP, POINTNET++, and METEORNET as in Section IV-E.

2) DIFFCLOUD Specifications: DIFFCLOUD performs optimization using the Adam optimizer (with learning rate 0.3 for vest hang, 0.4 for band stretch). We take the unidirectional Chamfer loss on the last frame for band stretch. For vest hang, most of the deformation happens in the second half of the trajectory, so we pick an intermediate frame in this part of the trajectory (instead of the final frame). We compute the loss between the simulated cloth point cloud only (no poles) and the entire target scene (including poles). This focuses optimization on the deformables. Since synthetic point clouds are noiseless, it is possible to choose a termination criterion for DIFFCLOUD based on when the computed loss falls below a pre-defined threshold. In practice, we find that a loss threshold of 0.0005 corresponds to a well-aligned match. Thus, for each target trajectory, we run DIFFCLOUD until the loss falls below this threshold (Figure 7) or until the number of optimization iterations exceeds 50, as in the lift and fold scenarios.

3) Results: For evaluation purposes, we generate a held-out test set of 10 simulated episodes for each scenario with mass and stiffness parameters sampled uniformly at random in the range $[0.1, 1.0]$. For each episode, we render the corresponding point cloud sequences according to the procedure from Section III-B without additional augmentations. We evaluate the performance of all methods on estimating the parameters of this set of target point cloud sequences. Across all target point cloud sequences in both scenarios, we find that DIFFCLOUD is able to converge to a set of parameters that yield a Chamfer distance below the threshold. The baselines are only able to achieve sub-threshold alignment in 50-80% of runs (Figure 6). Due to the threshold-based stopping criteria for DIFFCLOUD in the simulated experiments, running optimization for a given target point cloud sequence takes 5 minutes averaged across the vest hang and band stretch scenarios, compared to a combined dataset generation, training, and inference time on the order of hours for the baselines.

V. CONCLUSION AND FUTURE WORK

In this work, we proposed DIFFCLOUD: an approach to combine differentiable point cloud sampling with differentiable simulation for tackling real-to-sim problems with highly deformable objects. Our experiments showed that DIFFCLOUD reduced the time needed for obtaining real-to-sim alignment by more than an order of magnitude compared to non-differentiable, data-driven inverse models. This result opens the way to more agile experimentation with real-to-sim for highly deformable objects from real, correspondence-free visual input. Possible avenues for future work include relaxing the need for robot masking by including the robot model into simulation; improving point cloud rendering with realistic occlusion and camera noise models; and jointly optimizing robot actions along with physical simulation parameters. Furthermore, it would be interesting to apply DIFFCLOUD to contact-rich tasks that involve multiple rigid and deformable objects. Another interesting direction is to use DIFFCLOUD to infer the initial poses and morphology of a real deformable object before manipulation. This would open the way for instantiating real scenarios in simulation on the fly.

REFERENCES

[1] A. Prakash, S. Debnath, J.-F. Lalonde, E. Cameracci, S. Birchfield, M. T. Law et al., “Self-supervised real-to-sim scene generation,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 16044–16054.

[2] P. Chang and T. Padif, “Sim2real2sim: Bridging the gap between simulation and real-world in flexible object manipulation,” in 2020 Fourth IEEE International Conference on Robotic Computing (IRC), IEEE, 2020, pp. 56–62.
[3] J. Zhang, L. Tai, P. Yun, Y. Xiong, M. Liu, J. Boedecker, and W. Burgard, “Vr-goggles for robots: Real-to-sim domain adaptation for visual control,” IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 1148–1155, 2019.

[4] F. Muratore, T. G. Gruner, F. Wiese, B. Belousov, M. Gienger, and C. R. Qi, “Parameter estimation from dynamic motion,” in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 4383–4393.

[5] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 652–660.

[6] W. Chen, H. Ling, J. Gao, E. Smith, J. Lehtinen, A. Jacobson, and S. Fidler, “Learning to predict 3d objects with an interpolation-based differentiable renderer,” Advances in Neural Information Processing Systems, vol. 32, 2019.

[7] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, “DiffPD: Differentiable projective dynamics with contact,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4686–4695.

[8] M. A. Z. Mora, M. P. Peychev, S. Ha, M. Vechev, and S. Coros, “Pods: Policy optimization via differentiable simulation,” in International Conference on Machine Learning. PMLR, 2021, pp. 7805–7817.

[9] R. Marain, A. Samii, and J. F. O’Brien, “Data-driven anisotropic remeshing for cloth simulation,” ACM transactions on graphics (TOG), vol. 31, no. 6, pp. 1–10, 2012. [Online]. Available: graphics.berkeley.edu/resources/ARCSim

[10] S. Weiss, R. Maier, D. Cremers, R. Westermann, and N. Thueray, “Correspondence-free material reconstruction using sparse surface constraints,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4686–4695.

[11] M. W. Jones, J. A. Baerentzen, and M. Sramek, “3d distance fields for view synthesis,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2019, pp. 7708–7717.

[12] R. Marain, A. Samii, and J. F. O’Brien, “Adaptive anisotropic remeshing for cloth simulation,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1–10, 2012. [Online]. Available: graphics.berkeley.edu/resources/ARCSim

[13] S. Weiss, R. Maier, D. Cremers, R. Westermann, and N. Thueray, “Correspondence-free material reconstruction using sparse surface constraints,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4686–4695.

[14] M. W. Jones, J. A. Baerentzen, and M. Sramek, “3d distance fields for view synthesis,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2019, pp. 7708–7717.

[15] R. Marain, A. Samii, and J. F. O’Brien, “Adaptive anisotropic remeshing for cloth simulation,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1–10, 2012. [Online]. Available: graphics.berkeley.edu/resources/ARCSim

[16] S. Weiss, R. Maier, D. Cremers, R. Westermann, and N. Thueray, “Correspondence-free material reconstruction using sparse surface constraints,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4686–4695.

[17] M. W. Jones, J. A. Baerentzen, and M. Sramek, “3d distance fields for view synthesis,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2019, pp. 7708–7717.

[18] R. Marain, A. Samii, and J. F. O’Brien, “Adaptive anisotropic remeshing for cloth simulation,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1–10, 2012. [Online]. Available: graphics.berkeley.edu/resources/ARCSim

[19] S. Weiss, R. Maier, D. Cremers, R. Westermann, and N. Thueray, “Correspondence-free material reconstruction using sparse surface constraints,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4686–4695.

[20] M. W. Jones, J. A. Baerentzen, and M. Sramek, “3d distance fields for view synthesis,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2019, pp. 7708–7717.