DiffCloud: Real-to-Sim from Point Clouds with Differentiable Simulation and Rendering of Deformable Objects

Priya Sundaresan[1†], Rika Antonova[1*], and Jeannette Bohg[1]

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 solution is to train an inverse model on data generated with a black-box (non-differentiable) simulator. The input to such models usually consists of trajectories of a low-dimensional state of the system, e.g. position and orientation (pose) 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 pose, since the object deforms during motion. Hence, we represent objects with point clouds obtained from depth cameras. Recent neural network architectures, such as PointNet++ [5] and MeteorNet [6], are well suited for processing 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. However, 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. For highly deformable objects, even small changes in these parameters can have a significant impact on the behavior of the object. We show that 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) view 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.

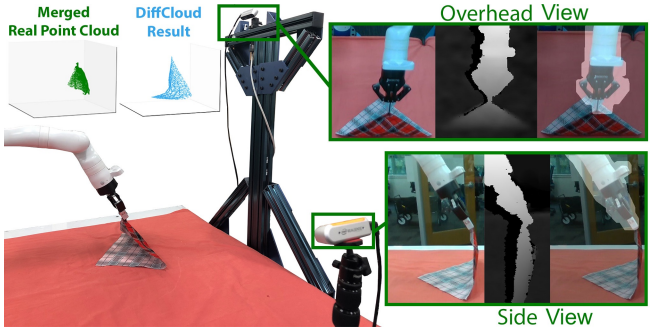

Fig. 1. Experimental setup. We execute deformable manipulation trajectories using a Kinova Gen3 arm. We post-process observations recorded from two stereo depth cameras (Intel D435) to generate merged point clouds with the robot arm masked from view. These observations are fed to our proposed method DiffCloud for *real-to-sim* parameter estimation.

II. 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 assume that the initial geometry of objects and location of the grasp are known. 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 and simulate their interactions. For end-to-end differentiability we implement a differentiable point cloud sampler, which allows to propagate loss gradients all the way to the simulation parameters. Figure 2 shows an overview.

Loss Definition : With low-cost depth sensors, we need a loss that avoids paying attention to noise artifacts in the real point cloud. Our insight is that such a loss can be obtained by using a unidirectional Chamfer distance. This yields a loss that relieves the pressure for the simulated point clouds to match the noisy parts of the real point clouds:

$$d_{\text{Chamf}}^{\text{sim}\rightarrow\text{real}}(\mathcal{P}^{\text{sim}},\mathcal{P}^{\text{real}}) = \sum_{\boldsymbol{x}^{\text{sim}}\in\mathcal{P}^{\text{sim}}} \min_{\boldsymbol{x}^{\text{real}}\in\mathcal{P}^{\text{real}}} ||\boldsymbol{x}^{\text{sim}} - \boldsymbol{x}^{\text{real}}||_2^2. \quad (1)$$

While the naive computation of the Chamfer distance can be expensive, PyTorch3D provides an efficient GPU-based

[1]Department of Computer Science, Stanford University, Stanford, CA 94305, USA {priyasun, rika.antonova, bohg}@stanford.edu

[†]P. Sundaresan was supported by the NSF Graduate Research Fellowship.
[*]Supported by the National Science Foundation grant No.2030859 to the Computing Research Association for the CIFellows Project.
This project was supported in part by a research award from Meta.

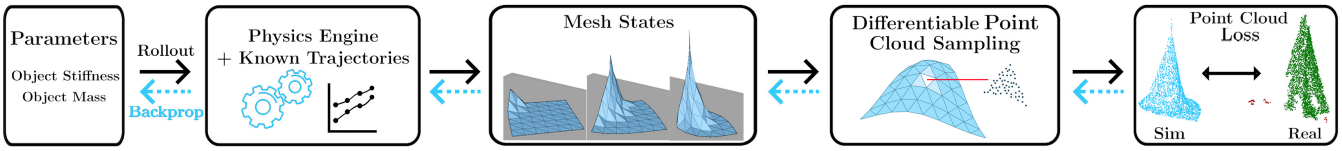

Fig. 2. **Overview of DIFFCLOUD**: the proposed method for real-to-sim parameter estimation from point clouds. DIFFCLOUD combines differentiable point cloud sampling with a differentiable mesh-based simulator to propagate losses computed between simulated (blue) and real (green, with red noise artifacts) point clouds to the underlying simulation properties. We visualize updating mass and stiffness of a simulated cloth lifted off of a table.

implementation (see Section 3.1 in [7]). With that, we can quickly propagate gradients from the point clouds through the mesh representation to optimize the low-level physical parameters of the simulator. We found that including point clouds from one or two depth cameras is sufficient to construct a partially occluded point cloud that is still informative enough for the overall optimization to be successful.

We build upon the differentiable simulator DiffSim [8], 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 $\mathbf{q} = [\mathbf{q_1}^T, \mathbf{q_2}^T, \ldots, \mathbf{q_n}^T]^T$ of all objects in the simulation with corresponding velocities $\dot{\mathbf{q}} = [\dot{\mathbf{q}}_\mathbf{1}^T, \dot{\mathbf{q}}_\mathbf{2}^T, \ldots, \dot{\mathbf{q}}_\mathbf{n}^T]^T$. The generalized coordinates of a rigid body have $\mathbf{q_i} \in \mathbb{R}^6$ for rigid object poses. Deformables consist of multiple nodes $\mathbf{q_i} \in \mathbb{R}^3$, denoting node positions. 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. A mesh vertex p has coordinate $\mathbf{p_0} = (p_x, p_y, p_z)^T$ in the body frame and world coordinate $\mathbf{p} = \mathbf{f}(\mathbf{q}) = [\mathbf{r}]\mathbf{p_0} + \mathbf{t}$, where $\mathbf{r} = (\phi, \theta, \psi)^T$ and $\mathbf{t} = (t_x, t_y, t_z)$ is the 6-DoF pose of the mesh. Propagating gradients from vertex $\mathbf{p}$ to the generalized coordinates $\mathbf{q}$ involves computing the Jacobian $\nabla\mathbf{f}$ and obtaining the partial derivatives $\partial\mathbf{f}(\mathbf{q})/\partial\mathbf{q}$. DiffSim can do sim-to-sim alignment by comparing current mesh states to target mesh states and propagating gradients from differences in node positions.

To propagate gradients from real point clouds, we implement differentiable point cloud sampling. A triangular mesh face can be represented by its enclosing vertices $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}$. 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 $u + v + w \leq 1$ [7]. We can obtain a random point $\boldsymbol{x}$ inside the triangle as: $\boldsymbol{x} = u\mathbf{p_1} + v\mathbf{p_2} + w\mathbf{p_3}$. To generate an N-point, uniform-density point cloud from a mesh, we first sample N triangular faces $\{(\mathbf{p_{i,1}}, \mathbf{p_{i,2}}, \mathbf{p_{i,3}})\}_{i=1,\ldots,N}$, weighted proportionally to the area of each face. For the i^{th} sampled face $(\mathbf{p_{i,1}}, \mathbf{p_{i,2}}, \mathbf{p_{i,3}})$, we generate random coefficients (u_i, v_i, w_i) and compute a point $\boldsymbol{x}_i$ that lies on the face. Concatenating the points obtained by applying this procedure to the N sampled faces and coefficients yields the point cloud $\mathcal{P} = \{\boldsymbol{x}_i\}_{i=1,\ldots,N}$. We connect the PyTorch3D [7] implementation of this sampling procedure to the output of a differentiable simulator. With that, gradients from loss on $\{\boldsymbol{x}_i\}_{i=1,\ldots,N}$ can be propagated to mesh vertices $(\mathbf{p_{i,1}}, \mathbf{p_{i,2}}, \mathbf{p_{i,3}})$ via chain rule using:

$$\partial\boldsymbol{x}_i/\partial\mathbf{p_{i,1}} = u_i \quad \partial\boldsymbol{x}_i/\partial\mathbf{p_{i,2}} = v_i \quad \partial\boldsymbol{x}_i/\partial\mathbf{p_{i,3}} = w_i \quad (2)$$

III. EXPERIMENTS

Baseline Inverse Models : As baselines, we use methods that treat simulators as black-box, i.e. don't differentiate w.r.t 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 simulation parameters and synthetic noise.

- METEORNET: An architecture for learning representations of 3D point cloud sequences from [6], which we modify to predict k simulation parameters. We use MeteorNet-cls (Appendix D.2 in [6]) for this regressor.
- 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.

As training data for the regressors we initialize N simulations with uniformly sampled parameters and record the resulting point cloud sequences and ground truth parameters. This yields a dataset $\mathcal{D} = \left\{\{\boldsymbol{x}_t\}_{t=1}^T, \boldsymbol{w} = [w_{\text{stiff}}, w_{\text{mass}}]\right\}_{i=1}^N$. We use 1500 training and 375 test point cloud sequences with

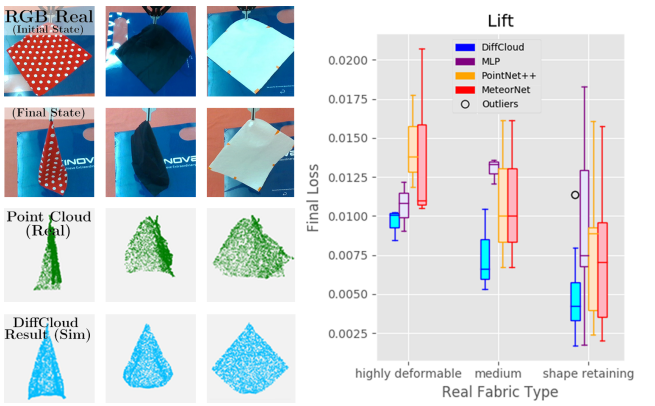

Fig. 3. Left: A Kinova robot executes trajectories to lift real cloths from a flat starting state. From these trajectories, DIFFCLOUD accurately infers stiffness and mass parameters capturing the observed degree of collapsibility in the real cloths. Right: Across all cloth types, DIFFCLOUD achieves lower loss on average than all competing baselines on the lift scenario.

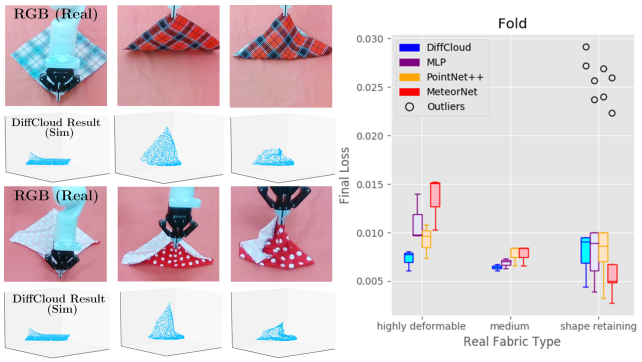

Fig. 4. Left: DIFFCLOUD correctly learns to approximate low mass/high stiffness for the shape retaining cloth (1st row) such that three corners lift off the table mid-fold (2nd row, 2nd column), and high mass/low stiffness result for the heavy, highly deformable polka dot fabric (3rd row) such that all ungrasped corners rest on the table mid-fold (4th row, 2nd column). Right: Compared to data-driven baselines, DIFFCLOUD achieves lower or comparable loss in 13/15 trajectories across categories. Due to difficulties perceiving very thin sheets in motion, all methods struggle with 2/15 paper towel (shape retaining) trajectories, which appear as outliers.

3,500 points per frame across methods. METEORNET regressor learns a mapping $f : \{\boldsymbol{x}\}_{i=1}^{T} \rightarrow \boldsymbol{w}$ from input point cloud sequence $\{\boldsymbol{x}_t\}_{t=1}^{T}$ to simulator parameters $\boldsymbol{w}$. POINTNET++ and MLP methods learn a mapping $g : \boldsymbol{x}_t \rightarrow \boldsymbol{w}$. We use the same $\boldsymbol{x}_t$ as the one used to compute the unidirectional Chamfer distance in DIFFCLOUD optimization. For each baseline, using the target point cloud trajectory as input, we first infer the predicted parameters. These serve as input to the simulator; we then run the simulator 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.

Real Robot Experiments : Figures 3,4 show `lift` and `fold` scenarios with 5 different fabrics grouped into three categories: highly deformable, medium, and shape retaining. For each fabric, we execute 3 trajectories (15 trajectories per scenario), ≈ 2.5 seconds each. During lift and fold the cloth either collapses or maintains its shape depending on the underlying cloth properties – stiffness and mass, which we aim to learn. DIFFCLOUD achieves a lower loss than the baselines across the 3 cloth categories in the lift scenario (Figure 3). The final parameters found by DIFFCLOUD align with the qualitative descriptions of the three categories of cloth types considered (Figure 5). Self-occlusion is more severe in the fold scenario, yet DIFFCLOUD achieves a loss on par with the baselines, which are (a) trained from thousands of examples, (b) use data augmentations to be invariant to varying degrees of self-occlusion, and (c) require significantly more compute time. While baseline regressors run inference in milliseconds, each baseline takes >2 hours for dataset generation and additional 40 minutes to multiple hours per scenario for training. DIFFCLOUD takes on average 10 minutes per trajectory (Figure 5). With significantly less computational footprint, DIFFCLOUD achieves parameter estimation results on real data that are comparable and in some cases better than baselines.

Further Simulation Experiments : Figure 6 shows evaluation on contact-rich simulated scenarios: hanging one

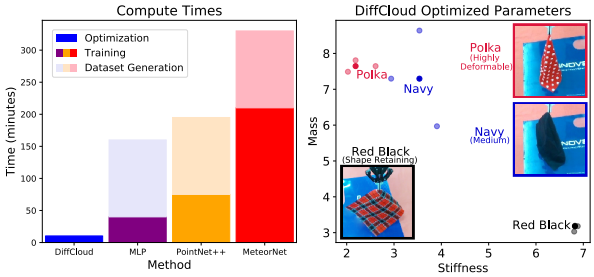

Fig. 5. Left: We visualize the average compute times across all methods for performing parameter estimation in the real lift and fold scenarios. Compared to the baseline inverse models, DIFFCLOUD achieves more than an order of magnitude speedup, since it eliminates the need to pre-generate a dataset and train on it. Right: The optimized DIFFCLOUD parameters found in the lift scenario correspond to the intuitive physical properties of real cloths, ranging from highly deformable to shape retaining. Each darkened circle represents the category median across three trajectories per cloth type.

shoulder of a vest onto a pole (`vest hang`); stretching an elastic band against a pole (`band stretch`). The mass and stiffness of the vest mesh determine the outline of the vest when hung on the pole. 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. For evaluation, we generate a held-out test set of 10 simulated episodes for each scenario with mass and stiffness parameters sampled uniformly from $[0.1, 10]$ range. Since synthetic point clouds are noiseless, it is possible to choose a termination criteria for DIFFCLOUD based on when the computed loss falls below a some threshold. We find that loss$<5e$-4 corresponds to a well-aligned match, hence, we run DIFFCLOUD until the loss falls below this threshold or until the number of optimization iterations exceeds 50 (same as for `lift`,`fold`). DIFFCLOUD converges to a set of parameters that yield a Chamfer distance below the threshold in all cases, while the baselines only reach sub-threshold alignment in 50-80% of runs (Figure 6). DIFFCLOUD 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.

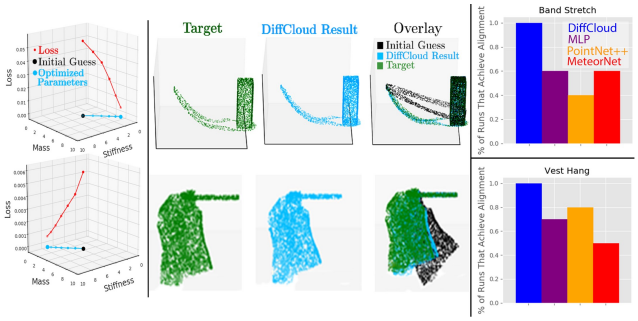

Fig. 6. Two contact-rich, simulated scenarios: `band stretch` and `vest hang`. Starting from an initial guess for the parameters (black), DIFFCLOUD takes gradient steps to update the parameters such that the loss taken between point cloud observations with optimized parameters (blue) and target point clouds (green) is minimized. The optimization terminates once the loss falls below a threshold of $5e$-4, denoting close visual alignment. DIFFCLOUD optimizes for the mass and stiffness parameters of simulated deformables to match a highly deformable target elastic band (top row) and a shape retaining target vest (bottom row). Across 10 runs per scenario, DIFFCLOUD achieves alignment between simulated and target point clouds in all runs, while baseline regressors yield mixed success (right bar plots).

REFERENCES

[1] A. Prakash, S. Debnath, J.-F. Lafleche, 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. 16 044–16 054.

[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. Liu, Z. Li, Y. Han, J. Lu, F. Richter, and M. C. Yip, "Real-to-sim registration of deformable soft tissue with position-based dynamics for surgical robot autonomy," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 328–12 334.

[5] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[6] X. Liu, M. Yan, and J. Bohg, "Meteornet: Deep learning on dynamic 3d point cloud sequences," in *ICCV*, 2019.

[7] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, "Accelerating 3d deep learning with pytorch3d," *arXiv preprint arXiv:2007.08501*, 2020.

[8] Y.-L. Qiao, J. Liang, V. Koltun, and M. Lin, "Scalable differentiable physics for learning and control," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7847–7856.