Active Learning of Neural Collision Handler for Complex 3D Mesh Deformations

Qingyang Tan [1] **Zherong Pan** [2] **Breannan Smith** [3] **Takaaki Shiratori** [3] **Dinesh Manocha** [1]

Abstract

We present a robust learning algorithm to detect and handle collisions in 3D deforming meshes. We first train a neural network to detect collisions and then use a numerical optimization algorithm to resolve penetrations guided by the network. Our learned collision handler can resolve collisions for unseen, high-dimensional meshes with thousands of vertices. To obtain stable network performance in such large and unseen spaces, we apply active learning by progressively inserting new collision data based on the network inferences. We automatically label these new data using an analytical collision detector and progressively fine-tune our detection networks. We evaluate our method for collision handling of complex, 3D meshes coming from several datasets with different shapes and topologies, including datasets corresponding to dressed and undressed human poses, cloth simulations, and human hand poses acquired using multi-view capture systems.

1. Introduction

Learning to model or simulate deformable meshes is becoming an important topic in computer vision and computer graphics, with rich applications in real-time physics simulation (Holden et al., 2019), animation synthesis (Qiao et al., 2020), and cross-domain model transformation (Cudeiro et al., 2019). Central to these methods are generative models that map from high-dimensional deformed 3D meshes with rich details to low-dimensional latent spaces. These generative models can be trained from high-quality groundtruth datasets, and they infer visually or physically plausible meshes in real time. These 3D datasets can also be generated using physics simulations (Narain et al., 2012; Tang et al., 2012) or reconstructed from the physical world using multi-view capture systems (Smith et al., 2020). In general, 3D deformable meshes are more costly to acquire, so 3D mesh datasets typically come in smaller sizes than image or text datasets. Inference models trained using such small datasets can suffer from over-fitting and generate meshes with various visual artifacts. For example, human pose embedding networks (Tan et al., 2018b; Gao et al., 2018) can have excessive deformations, and interaction networks (Battaglia et al., 2016) can result in non-physically-based object motions.

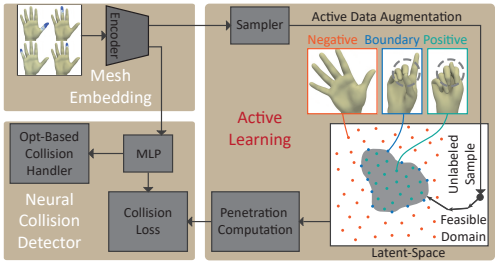

Figure 1. Our method consists of a (learned or analytic) latent space, a neural collision detector, and an optimization-based collision handler. We progressively insert data by randomly sampling in the latent space (sampler). We then use a Newton-type method to pull unlabeled samples towards the learned decision boundary (black arrows in the white, feasible domain). The groundtruth collision labels are generated using an analytic collision detector (penetration computation). Finally, we use three different loss functions for samples on the positive (orange), negative side (green), and near (blue) the decision boundaries.

The goal of our research is to resolve a major source of visual artifacts: self-collisions. Instead of acquiring more data, we argue that domain-specific knowledge could also be utilized to significantly improve the accuracy of inference models. There have been several prior research works along this line. For example, (Yang et al., 2020b) exploited the fact that near articulated meshes can be divided into multiple components, and they train a recursive autoencoder to stitch the components together. (Zheng et al., 2021) utilized the locality of secondary physics motions to learn re-targetable and scalable real-time dynamics animation. Recently, (Tan et al., 2021) studied learning-based collision avoidance for 3D meshes corresponding to human poses. They proposed a deep architecture to detect collisions and used numerical optimizations to resolve detected collisions. However, (Tan et al., 2021) used a large mesh dataset to obtain stable per-

[1]Department of Computer Science, University of Maryland at College Park [2]Lightspeed & Quantum Studio, Tencent America [3]Facebook Reality Labs Research. Correspondence to: Dinesh Manocha <dmanocha@umd.edu>.

formance of neural collision detection. Indeed, a deformed 3D mesh typically involves more than 10^4 elements (voxels, points, triangles) where any pair of two elements can have collisions. Therefore, a huge amount of data is required to present the inference model with enough examples of collisions between all possible element pairs.

Main Results: We present a robust method to train neural collision handler for complex, 3D deformable meshes using active learning. Our key observation is that the distribution of penetrating meshes can have a long tail and active learning is an effective method for modeling the tail (Geifman & El-Yaniv, 2017). Unfortunately, most 3D mesh datasets do not focus on generating samples in the tail and cannot be used to train stable collision detectors. In order to overcome these issues, our approach combines three main ideas:

- We use active learning to progressively insert new samples into the dataset. Collision labels for the new samples are automatically generated;
- We use a risk-seeking approach to prioritize samples near the decision boundary, so that the inserted samples can best help improve our accuracy;
- We use different loss functions for samples far from and close to the decision boundary.

Our overall approach is shown in Fig. 1. We show that our training method is versatile and can learn to detect collisions for meshes encoded in various low-dimensional spaces, such as learned latent-space (Yang et al., 2020a) and domain-specific human body deformation representation (Loper et al., 2015). For either encoding, we further show that our active learning technique outperforms supervised learning in terms of data efficacy and accuracy. We evaluate our method on three types of complex datasets:

- Dressed and undressed human poses, including SCAPE (Anguelov et al., 2005), MIT Swing (Vlasic et al., 2008), MIT Jump (Vlasic et al., 2008), and AMASS-MPIMosh (Mahmood et al., 2019) containing different genders and body shapes with 54387 meshes;
- Cloth simulations from (Yang et al., 2020a) with complex deformation and self-collisions;
- Human hands captured by multi-view camera system.

Compared to prior supervised learning approaches, our method exhibits much higher data efficacy and accuracy (up to 98.1%) and uses fewer training samples (up to 48.12% less). Given a training dataset of the same size, our method reduces the false negative rate by 14.27% on average, and we are able to resolve more self-colliding meshes. For a test size with 1×10^4 meshes, our method only cost $1.32s$ on GPU. Overall, ours is the first practical method for neural collision handling of general, 3D complex meshes.

2. Related Work

We review related works in mesh embedding, collision handling, and active learning.

Generative Model of Dense 3D Shapes: Categorized by shape representations, generative models can be based on point clouds (Qi et al., 2017), volumetric grids (Wu et al., 2015), multi-charts (Groueix et al., 2018), surface meshes (Tan et al., 2018a), or semantic data structures (Liu et al., 2019). We use mesh-based representations with fixed topologies, because most collision detection libraries are designed for meshes. Some generative models can learn to represent general meshes of changing topology, e.g., for modeling meshes of hierarchical structure (Yu et al., 2019) or modeling scenes with many objects (Ritchie et al., 2019). However, these applications typically involve only static meshes with no need for collision detection. There is a separate research direction on domain-specific mesh deformation representation, e.g., SMPL/STAR human models (Loper et al., 2015; Osman et al., 2020), wrinkle-enhanced cloth meshes (Lahner et al., 2018), and skeletal skinning meshes (Xu et al., 2020). Our method is versatile and can be combined with both learned embedding (Tan et al., 2018a) and SMPL representation (Loper et al., 2015; Osman et al., 2020).

Collision Prediction & Handling: Although collision handling is a well-studied area, they can still be non-trivial computational burden. Prior methods (Pan et al., 2012; Kim et al., 2018; Govindaraju et al., 2005) use spatial hashing, bounding volume hierarchies, and GPU to accelerate the computation by pruning non-colliding primitives, but they are incompatible with mesh represented in latent space. Handling collisions is even more challenging, and prior methods either use penalty forces coupled with discrete collision detectors (Tang et al., 2012) or hard constraints coupled with continuous collision detectors (Narain et al., 2012). All these methods rely on physics-based constraints to handle collisions. Recently, many learning methods such as (Gundogdu et al., 2019; Patel et al., 2020) have been designed to predict the cloth movement or deformation in 3D, but they do not perform collision handling explicitly.

Active Learning: An active learner alternates between drawing new samples and exploiting existing samples. These samples can be drawn guided by an acquisition function in Bayesian optimization (Niculescu et al., 2006) or from an expert algorithm (De Raedt et al., 2018). Active learning has been applied to approximate the boundary of the configuration space (Pan et al., 2013; Tian et al., 2016; Das et al., 2017), where the feasible domain of collision constraints is parameterized using kernel SVM. However, these methods are limited to rigid or articulated deformations and are not applicable to general 3D deformations. More broadly, active learning has been adopted in various prior

Variable	Definition
$\mathcal{G} = \langle V, E \rangle$	graph with vertices and edges
E, D	encoder, decoder
$\theta_{E,D,C}$	learnable parameters
$Z_{\text{all}} = \begin{pmatrix} Z_0 \, Z_1 \cdots Z_{\lvert Z_0 \rvert} \end{pmatrix}$	autoencoder latent code
$\mathcal{Z}$	latent region of sampling
PD	penetration depth
CSE	global collision state encoder
CP	local collision state predictor
MLP_c	collision classifier
$S_0, S_1, \cdots, S_{\lvert Z_0 \rvert}$	neural collision indicator

Variable	Definition
ACAP	feature transform function
θ_C	neural collision network parameters
I_c	collision state label
E	collision handler objective function
$\mathcal{D}$	dataset for learning $\theta_{E,D}$
$\mathcal{D}_{d,p,n,c}$	dataset for learning θ_C
ϵ	threshold for boundary samples
CE	cross-entropy loss
$\mathcal{L}_\bullet$	neural network losses
$w_\bullet$	weights for each loss

works to accelerate data labeling in image classification (Gal et al., 2017) and object detection (Aghdam et al., 2019) tasks. These methods progressively identify unlabeled images to be forwarded to experts for labelling. An alternative method for selecting the samples is identifying a coreset (Paul et al., 2014), and authors of (Sener & Savarese, 2018) propose a practical algorithm for coreset identification via k-center clustering. These methods consider a discrete dataset, while we assume a continuous latent space of samples for training generative models and use a risk-seeking method to identify critical new samples.

3. Neural Collision Handler

We introduce our neural collision handling architecture, based on which we build our active learning method. All notations are summarized in the symbol table.

A mesh is represented by the graph $\mathcal{G} = \langle V, E \rangle$, where V is a set of vertices, and E is a set of edges. We assume that all the meshes have the same topology, that is, all the meshes differ in V while the connectivity E stays the same. We further limit ourselves to manifold triangle meshes, i.e., each edge is incident to at most two triangles, and two triangles are adjacent if and only if they share an edge. No other assumptions are made on the mesh deformation. We denote a mesh as self-collision-free if and only if any pair of two non-adjacent triangles are not intersecting each other. Our goal is to design a mesh-based generative neural architecture where we take an input as a coordinate in the latent space and output a 3D mesh without self-collisions. The latent space is defined as a low-dimensional space that can be mapped to high-dimensional meshes injectively using a learned or analytic decoder function. Furthermore, the latent-to-mesh mapping is differentiable and supports multiple downstream applications explained in Sec. 3.3. We have experimented with two latent-spaces, learned bilevel autoencoder (Yang et al., 2020a) and SMPL human body representation (Loper et al., 2015), and we brief review these methods below.

3.1. Bilevel Autoencoder

The bilevel autoencoder architecture maps a deformed mesh to two levels of latent codes. We only want to encode intrinsic mesh information such as curvatures instead of extrinsic rigid transformations because mesh shapes are invariant to extrinsic transformation. Therefore, we first use as-consistent-as-possible (ACAP) feature transformation (Gao et al., 2019) to factor out rigid transformations. The ACAP feature vector is first brought through the level-1 autoencoder and mapped to a latent code Z_0. We further hypothesize that the error is sparsely distributed throughout the mesh vertices. Therefore, we then use an attention mechanism trained with a sparsity prior to decompose the mesh into near-rigid sub-domains. The sparsity prior is designed such that each domain can be mapped to a single axis of the latent space, i.e., a single entry of Z_0. Afterwards, a set of $|Z_0|$ level-2 autoencoders is introduced to further reduce the error, with each autoencoder dedicated to one entry of Z_0. Their latent codes are denoted as $Z_1, \cdots, Z_{|Z_0|}$. The ultimate mesh is reconstructed from Z_{all} by combining level-1 and level-2 latent codes:

$$
\begin{aligned}
Z_{\text{all}} &= \begin{pmatrix} Z_0 \, Z_1 \cdots Z_{|Z_0|} \end{pmatrix} \\
D(Z_{\text{all}}, \theta_D) &= \sum_{i=0}^{|Z_0|} D_i(Z_i, \theta_{D_i}) \\
V &= \text{ACAP}^{-1}(D(Z_{\text{all}}, \theta_D)),
\end{aligned}
$$

where D_i is the ith decoder, with θ_{D_i} being the learnable parameters. The mesh vertices V are reconstructed by inverting the ACAP transformation. Correspondingly, we have the encoder defined as $E(\text{ACAP}(V), \theta_E) = Z_{\text{all}}$, which maps the vertices of a mesh to the latent space, with θ_E being the learnable parameters.

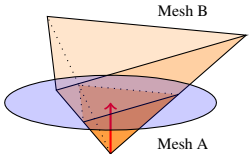

Figure 2. PD (red arrow) is the locally minimal translation for mesh B (orange) to be collision-free from mesh A.

Our neural collision detector predicts whether the mesh V is subject to self-collisions using latent information Z_{all}. The extent to which two meshes collide can be measured by the notion of Penetration Depth (PD) (Zhang et al., 2014), defined by the norm of the smallest configuration change needed for a mesh to be self-collision-free, as illustrated in Fig. 2. It is well-known that PD is a non-smooth function of V (esp. at the boundaries), thereby making it difficult to resolve collisions by minimizing PD. By choosing appropriate activation functions ($\tanh$ and CELU in our case), we design the neural collision detector to be a differentiable approximation of PD. As a result, gradient information can be propagated to a collision handler to minimize PD.

Since collisions can happen between any pairs of geometric mesh primitives, a collision detector should consider possible contacts between any pair of near-rigid sub-domains, leading to a quadratic complexity $\mathcal{O}(|Z_0|^2)$. We use a global-local detection architecture that effectively reduces the number of learnable parameters. Specifically, we introduce a global collision state encoder $S_0 = \text{CSE}(Z_{\text{all}}, \theta_C)$ and a set of local collision predictors $S_i = \text{CP}(S_0, Z_i, \theta_C)$, with $i = 1, \cdots, |Z_0|$, which predicts whether the ith sub-domain is in collision with the rest of the mesh. Finally,

the collision information for all local collision predictors is summarized using a classifier network $\text{MLP}_c(S_1, \cdots, S_{|Z_0|})$ to derive a single overall collision classifier:

$$\begin{aligned} S_0 &\triangleq \text{CSE}(Z_{\text{all}}, \theta_C) \\ S_i &\triangleq \text{CP}(S_0, Z_i, \theta_C) \\ I_c(Z_{\text{all}}) &\triangleq \mathbb{I}(\text{MLP}_c(S_1, \cdots, S_{|Z_0|}, \theta_C) \geq 0.5), \end{aligned}$$

where θ_C is the learnable parameters. The feasible space boundary of collision-free constraints corresponds to the 0.5-levelset of MLP_c. We refer readers to (Tan et al., 2021) for more details.

3.2. SMPL Human Pose Representation

The SMPL model represents a human body shape using two kinds of parameters: body pose and body shape. In order to determine mesh vertices V, SMPL uses a template mesh of a reference body shape and pose V^0. The variation in body shape is addressed by adding a linear perturbation: $V_s^0 = V^0 + \sum_i Z_{\beta_i} B_i$, where B_i is a set of shape variation bases and Z_{β_i} are the corresponding shape coefficients. From V_s^0, SMPL derives the final V using standard linear blended skinning: $V = D(Z_\theta, V_s^0)$, where we unify the notations for both mesh embedding methods and reuse the symbol D for the linear blended skinning function. Here Z_θ is the pose parameters. The overall latent information for the SMPL model is $Z_{\text{all}} = (Z_\beta \; Z_\theta)$. We refer readers to (Loper et al., 2015) for more details. Given that D is differentiable, we use standard MLP as our collision detector taking Z_{all} as input:

$$I_c(Z_{\text{all}}) \triangleq \mathbb{I}(\text{MLP}_c(Z_{\text{all}}, \theta_C) \geq 0.5).$$

We use the collision detector offered by (Muller et al., 2021) on the bodies generated by the SMPL model to neglect natural intersections around nearby tissues. Nevertheless, we still use PD to represent the energy measuring the collision extent.

3.3. Optimization-Based Collision Response

Following (Tan et al., 2021), we design our collision detector MLP_c to be differentiable. Suppose we take as input a randomly sampled latent code $Z_{\text{all}}^{\text{user}}$, which might not satisfy the collision-free constraints, we project that latent code back to the feasible domain by solving the following optimization problem under neural collision-free constraints using the Augmented Lagrangian Method (ALM):

$$\underset{Z_{\text{all}}}{\text{argmin}} \; E(Z_{\text{all}}) \quad \text{s.t. } \text{MLP}_c(S_1, \cdots, S_{|Z_0|}, \theta_C) \leq 0.5, \quad (1)$$

where $E(\bullet)$ is some objective function, which can take multiple forms, as specified by downstream applications. In the simplest case, we take as input a desired $Z_{\text{all}}^{\text{user}}$, and we can define $E(Z_{\text{all}}) = \|Z_{\text{all}} - Z_{\text{all}}^{\text{user}}\|^2/2$, which is only related to latent space variables. As a more intuitive interface, the user might want to change meshes in the Cartesian space instead of the of latent space. For example, if the user wants a human hand to be at a certain position V^{user}, we could define $E(Z_{\text{all}}) = \|D(Z_{\text{all}}) - V^{\text{user}}\|^2/2$. A desirable feature of Eq. 1 is an invariant problem size. However many vertices a mesh has, there is only one constraint, which guarantees high test-time performance. Moreover, it has been shown in Theorem 10.4.3 of (Sun & Yuan, 2006) that ALM either finds a feasible solution or returns an infeasible solution that is closest to the boundary of the feasible domain. In other words, ALM always makes a best effort to resolve collisions, even if feasible solutions are not available.

4. Active Learning Algorithm

The goal of active learning is to iteratively improve the accuracy of the neural collision detector. We assume the availability of an existing dataset $\mathcal{D}$ of "high-quality" meshes with deformed vertices $\mathcal{D} = \{V^{1,2,\cdots,|\mathcal{D}|}\}$, which is used to train the mesh embedding component. (Note that SMPL also requires a dataset $\mathcal{D}$ to learn the linear blended skinning weights). We assume that meshes in $\mathcal{D}$ are collision-free but the meshes reconstructed using function D can still suffer from collisions due to embedding error after training, and users might explore the latent space in regions that are not well covered by the training dataset. As a result, our neural collision detector cannot be trained with $\mathcal{D}$ alone. This is because $\mathcal{D}$ only contains negative (collision-free) samples, while the neural collision detector must learn the decision boundary between positive and negative samples. In other words, the neural detector must be presented with enough samples to cover all possible latent codes with both self-penetrating and collision-free meshes. We denote the training dataset of neural collision detectors as another set: $\mathcal{D}_c = \{\langle Z_{\text{all}}^i, I_c^*(Z_{\text{all}}^i)\rangle | i = 1, 2, \cdots, |\mathcal{D}_c|\}$, where $I_c^*(\bullet)$ is the groundtruth $0-1$ collision state label.

The groundtruth collision state label can be generated automatically using a robust algorithm such as (Pan et al., 2012), to compute PD, where a positive PD indicates self-collisions, so we can define $I_c^*(Z_{\text{all}}) \triangleq \mathbb{I}(\text{PD}(Z_{\text{all}}) > 0)$, where $\text{PD}(Z_{\text{all}}) > 0$ means first recovering the mesh V from Z_{all} and then compute PD via (Pan et al., 2012). However, the cost to compute penetration depth, $\text{PD}(\bullet)$, is superlinear in the number of mesh vertices, and computing PD for an entire dataset can still be a computational bottleneck. Moreover, we are considering a continuous space of possible training data that cannot be enumerated. To alleviate the computational burden, we design a three-stage method, as illustrated in Fig. 1. During the first stage of bootstrap, we sample an initial boundary set, by which we train MLP_c to approximate the true decision boundary. At the second stage of data augmentation, new training data is selected and progressively injected into a dataset. Finally, for the

third stage, our neural collision detector is updated to fit the augmented dataset. The criterion for selecting the subset is critical to the performance of active learning. We observe that our neural collision predictor is used as constraints for nonlinear optimization methods so that samples far from the boundary are not used by the optimizer and only the boundary of the feasible domain (gray area in Fig. 1 right) is useful. Therefore, we propose using a Newton-type risk-seeking method to push the samples towards the decision boundary. We provide more details for each step below.

4.1. Bootstrap

Active learning would progressively populate $\mathcal{D}_c$, so prior work (Aghdam et al., 2019) simply initializes the dataset to an empty set. However, we find that a good initial guess can significantly improve the convergence of training. This is because we select new data by moving (randomly sampled) latent codes towards the decision boundary of the PD function using a risk-seeking method. However, the true boundary of the collision-free constraints corresponds to the boundary of C-Obstacles, which is high-dimensional and unknown to us (PD is a non-smooth function, so we cannot even use gradient information to project a mesh to the zero level-set of PD). Instead, we propose using the learned neural decision boundary, i.e., the 0.5-levelset of MLP_c, as an approximation. If we initialize $\mathcal{D}_c = \varnothing$, the surrogate decision boundary is undefined, and the training might diverge or suffer from slow convergence. For our bootstrap training, we uniformly sample a small set of N_{init} latent codes $\mathcal{Z}_{\text{all}}$ at random positions from the latent space and compute PD for each of them. We define a valid space of sampling by mapping all the data $Z^i_{\text{all}} \in \mathcal{D}$ to their latent codes and compute a bounded box in the latent space:

$$\mathcal{Z} = \prod_{j=0}^{|Z_0|} \left[\min_{i=1,\cdots,|\mathcal{D}|} \left[e_j^T Z^i_{\text{all}} \right], \max_{i=1,\cdots,|\mathcal{D}|} \left[e_j^T Z^i_{\text{all}} \right] \right].$$

We hypothesize that all the meshes can be embedded using our autoencoder or the SMPL model with small error corresponding to latent codes in $\mathcal{Z}$, so we can initialize $\mathcal{D}_c = \{Z^{1,\cdots,N_{\text{init}}}_{\text{all}} | Z^i_{\text{all}} \sim U(\mathcal{Z})\}$. We then divide the data points into three subsets ($\mathcal{D}_c = \mathcal{D}_p \bigcup \mathcal{D}_n \bigcup \mathcal{D}_b$, illustrated in Fig. 1 right):

$$\begin{aligned} \mathcal{D}_p &\triangleq \{ \left(Z^i_{\text{all}}, I_c(Z^i_{\text{all}}) \right) | \text{PD}(Z_{\text{all}}) > \epsilon \} \\ \mathcal{D}_n &\triangleq \{ \left(Z^i_{\text{all}}, I_c(Z^i_{\text{all}}) \right) | \text{PD}(Z_{\text{all}}) < 0 \} \\ \mathcal{D}_b &\triangleq \{ \left(Z^i_{\text{all}}, I_c(Z^i_{\text{all}}) \right) | \text{PD}(Z_{\text{all}}) \in [0, \epsilon] \}. \end{aligned} \tag{2}$$

Here, $\mathcal{D}_p$ is the positive set consisting of samples with penetrations deeper than a threshold ϵ, $\mathcal{D}_n$ is the negative set consisting of collision-free samples, and $\mathcal{D}_b$ is a boundary set where samples are nearly collision-free and lie on the decision boundary. We propose using the l_1-loss function for $\mathcal{D}_b$ to approximate the decision boundary:

$$\mathcal{L}_b = E_{\{Z_{\text{all}} \in \mathcal{D}_b\}} \left[\| \text{MLP}_c(Z_{\text{all}}, \theta_C) - 0.5 \| \right]. \tag{3}$$

4.2. Data Aggregation

The accuracy of our neural collision detector can be measured by the discrepancy between the surrogate decision boundary deemed by MLP_c and the true decision boundary of PD, formulated as:

$$E_{\{Z_{\text{all}} \sim \mathcal{Z} | \text{PD}(Z_{\text{all}})=0\}} \left[\text{CE}(I_c(Z_{\text{all}}), I_c^*(Z_{\text{all}})) \right],$$

which is an expectation over the true decision boundary. Here CE is the cross-entropy loss. However, it is very difficult to derive a sampled approximation of the above metric because PD is a non-smooth function whose level-set is measure-zero, which corresponds to the boundaries of C-obstacles. Instead, we propose to take expectation over the surrogate decision boundary:

$$E_{\{Z_{\text{all}} \sim \mathcal{Z} | \text{MLP}_c(Z_{\text{all}}, \theta_C)=0.5\}} \left[\text{CE}(I_c(Z_{\text{all}}), I_c^*(Z_{\text{all}})) \right].$$

Generally speaking, the 0.5-level-set of MLP_c can also be measure-zero, but we have designed our neural networks $D, \text{CSE}, \text{CP}, \text{MLP}_c$ to be differentiable functions. As a result, we could always project samples onto the 0.5-level-set by solving the following risk-seeking unconstrained optimization:

$$\underset{Z_{\text{all}}}{\text{argmin}} \frac{1}{2} \| \text{MLP}_c(Z_{\text{all}}, \theta_C) - 0.5 \|^2.$$

We adopt the quasi-Newton method and update Z_{all} using the following recursion:

$$Z_{\text{all}} - \bar{H}^{-1} \nabla \text{MLP}_c (\text{MLP}_c - 0.5), \tag{4}$$

where $\bar{H}$ is some first-order approximation of the Hessian matrix, which is much faster to compute than the exact Hessian, which requires the second-order term $\nabla^2 \text{MLP}_c$. In summary, we would sample a new set of size $N_{\text{aug}}/2$ from previous $\mathcal{D}_c$ during each iteration of data augmentation. For each sampled Z_{all}, we project Z_{all} to the surrogate decision boundary using recursive Eq. 4 until the relative change within Z_{all} is smaller than ϵ_z between consecutive iterations. For datasets based on the SMPL model, we randomly choose to fix Z_β or not. We also sample $N_{\text{aug}}/2$ directly from $U(\mathcal{Z})$, using random samples to discover uncovered regions, which achieves a balance between exploitation and exploration. Finally, we classify Z_{all} into either one of $\mathcal{D}_{p,n,b}$, according to Eq. 2 using the penetration depth.

4.3. Model Update

After $\mathcal{D}_c$ has been updated, we fine-tune $\text{CSE}, \text{CP}, \text{MLP}_c$ by updating θ_C using the following loss functions:

$$\mathcal{L} = w_{\text{PD}} \mathcal{L}_{\text{PD}} + w_r \mathcal{L}_r + w_{\text{ce}} \mathcal{L}_{\text{ce}} + w_b \mathcal{L}_b,$$

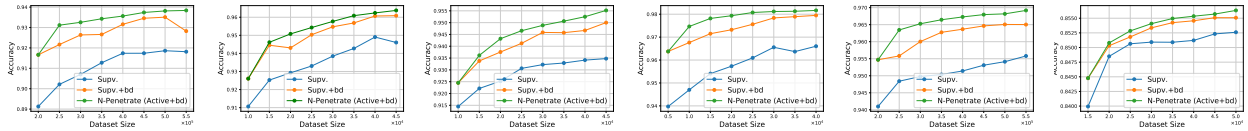

Figure 3. We plot the accuracy of the neural collision detector against the dataset size. The baselines are trained using the same amount of data. On average, ours achieves 1.62% higher accuracy than ***Supv***. From left to right: SCAPE, Swing, Jump, Skirt, Hand, and AMASS.

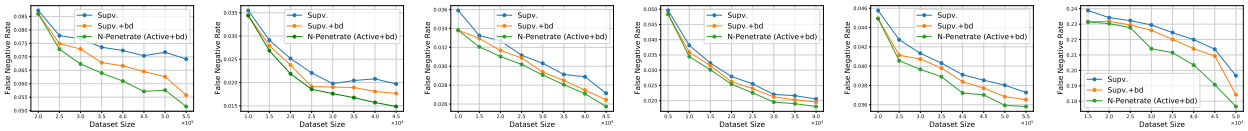

Figure 4. We plot the false negative rate against the dataset size. The baselines are trained using the same amount of data. On average, ours achieves a 13.59% lower false negative rate than ***Supv***. From left to right: SCAPE, Swing, Jump, Skirt, Hand, and AMASS.

where $w_\bullet$ are weights corresponding to each type of loss. Our first term $\mathcal{L}_{\text{PD}}$ is a regularization that enforces consistency between S_i and true PD, defined as:

$$\mathcal{L}_{\text{PD}} = E_{\{Z_{\text{all}} \in \mathcal{D}_C\}} \left[\sum_{i=1}^{|Z_0|} \| S_i - \text{PD}_i \|^2 + w_{\text{PDsum}} \| \sum_{i=1}^{|Z_0|} S_i - \text{PD} \|^2 \right],$$

where we penalize both domain-decomposed penetration depth PD_i defined in (Tan et al., 2021) and total penetration depth with weight w_{PDsum}. Note that the domain-decomposed term is used for bilevel autoencoders and omitted in SMPL representation. Our second term $\mathcal{L}_r$ is a marginal ranking loss that enforces the correct ordering of penetration depth to avoid over-fitting, which is defined as:

$$\mathcal{L}_r = E_{\{Z_{\text{all}}^{a,b} \in \mathcal{D}_C | \text{PD}^a < \text{PD}^b\}} \left[\max(0, \alpha - (\sum_{i=1}^{|Z_0|} S_i^a - \sum_{i=1}^{|Z_0|} S_i^b) \right],$$

where α is the maximal allowable order violation. We use superscripts to distinguish two samples drawn from $\mathcal{D}_C$. Our third term measures the discrepancy between MLP_c and PD over the entire latent space:

$$\mathcal{L}_{\text{ce}} = E_{\{Z_{\text{all}} \in \mathcal{D}_p \cup \mathcal{D}_n\}} \left[\text{CE}(I_c(Z_{\text{all}}), I_c^*(Z_{\text{all}})) \right].$$

We update our neural collision detector with objective function $\mathcal{L}$ by running a fixed number of training epochs, denoted as N_{epoch}, with θ_C warm-started from the last iteration of the model update.

5. Evaluation

Datasets: We evaluate our method on six datasets. The first three (SCAPE (Anguelov et al., 2005) with $N = 71$ meshes each having 2161 vertices, MIT Swing (Vlasic et al., 2008) with $N = 150$ meshes each having 9971 vertices, and MIT Jump (Vlasic et al., 2008) with $N = 150$ meshes each having 10002 vertices) contain human bodies with different sets of actions and poses. The forth one is a skirt dataset introduced by (Yang et al., 2020a) that contains $N = 201$ simulated skirt meshes synthesized by NVIDIA clothing tools, each of which has 2830 vertices. The skirt is deformable everywhere, and the dataset is rather small. Obtaining stable performance in this case is challenging and we observe reasonably good results using active learning. Our fifth one is a custom dataset of human hand poses. We captured various hand poses and transitions between the poses in a multi-view capture system. We ran 3D reconstruction (Galliani et al., 2015) and 3D keypoint detection (Simon et al., 2017) for the captured images and registered a linear blended skinning model consisting of 2825 vertices for each frame of the data (Gall et al., 2009), resulting in $N = 7314$ meshes. The first five datasets use the bilevel autoencoder for embedding. Our sixth dataset comes from the captured motion model repository AMASS (Mahmood et al., 2019), which uses SMPL to embed human body shapes. We choose one subset MPIMosh covering 19 subjects with different genders and body shapes. Suggested by (Muller et al., 2021), we sample at half of its original frame rate and get $N = 54387$ meshes each having 6890 vertices.

	SCAPE	Swing	Jump	Skirt	Hand	AMASS
N_{init}	200000	10000	10000	5000	200000	15000
N_{aug}	50000	5000	5000	5000	50000	5000

Table 1. N_{init} and N_{aug} used by each dataset.

Implementation: We implement our method using PyTorch and perform experiments on a desktop machine with an NVIDIA RTX 2080Ti GPU. If bilevel autoencoder is used, we begin by training $\langle \theta_E, \theta_D \rangle$ using Adam with a learning rate of 0.01 and a batch size of 128 over 3000 epochs. For neural collision detector training, we choose the following hyper-parameters: $\epsilon = 1{\times}10^{-4}, w_{\text{PD}} = 5, w_{\text{PDsum}} = 0.2, w_r = 2, w_{ce} = 2, w_b = 0.5$. We perform bootstrap by supervised learning θ_C on N_{init} data points. We progressively inject data points into N_{init} until the "elbow point" of the accuracy vs. the sample size is reached, which is detected using (Satopaa et al., 2011). Please check the appendix for detailed figures. For each experiment, we train θ_C using Adam with a learning rate of 0.001 and a batch size of 512 over $N_{\text{epoch}} = 100$ epochs. We choose suitable N_{aug} according to N_{init}. The N_{init} and N_{aug} used for each dataset are summarized in Table 1. During data aggregation, we terminate Newton's method when the relative changes of

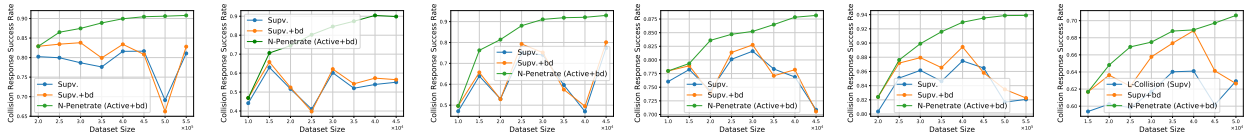

Figure 5. We plot the success rate of the neural collision handler against the dataset size. Our method resolves 22.73% more collisions than ***Supv+bd***. From left to right: SCAPE, Swing, Jump, Skirt, Hand, and AMASS.

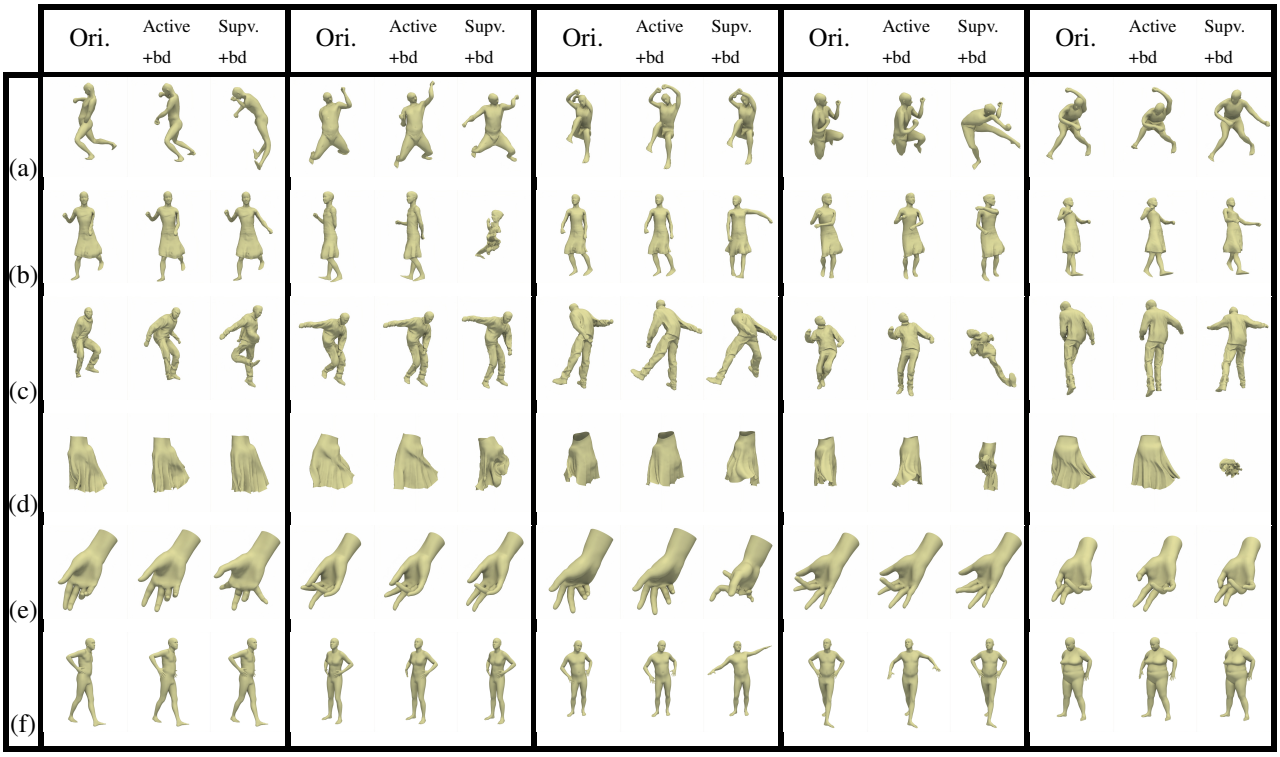

Figure 6. Representative examples of collision handling on the five datasets: (a) SCAPE; (b) MIT Swing; (c) MIT Jump; (d) Skirt; (e) Hand; (f) AMASS. For each example, we show the given self-penetrating mesh (left), our result (middle, ***Active+bd***), and ***Supv+bd*** (right).

Z_{all} are less than $\epsilon_z = 10^{-7}$. For each subsequent iteration of active data augmentation, we fine-tune θ_C using Adam and adjust learning rate, batch size, and N_{epoch} using the performance on the validation set. For collision handling, we run ALM until Eq. 1 is satisfied.

Collision Detection: We compare our method with two baseline algorithms. The first one (Tan et al., 2021) (denoted as ***Supv***) trains θ_C using supervised learning, where $\mathcal{D}_c$ is constructed by randomly sampled poses from $U(\mathcal{Z})$ and boundary set D_b where associated loss Eq. 3 mentioned in Sec. 4.1 is not used, i.e., $w_b = 0$. Our second baseline (denoted as ***Supv + bd***) also uses supervised learning, but the boundary set and loss in Eq. 3 are used. Our proposed method is denoted as ***Active + bd***. After k iterations of active data augmentations, we have a dataset with $N_{init} + kN_{aug}$ points for training θ_C. For fairness, we train our two baselines using $N_{init} + kN_{aug}$ points randomly sampled from $U(\mathcal{Z})$ for each iteration. For all the methods, we use 80% of the data for training and the rest is used as a validation set for hyperparameter tuning (learning rate, batch size, and N_{epoch}). For each dataset, we create a test set with 7.5×10^5 samples from $U(\mathcal{Z})$, which is unseen in the training stage, to evaluate the performances. The performances of neural collision detectors are evaluated based on two metrics: the fraction of successful predicates (accuracy) and the fraction of times a self-penetrating mesh is erroneously predicted as collision-free (false negative rate). False negatives are more detrimental to our applications than false positives as we want to detect and eliminate collided samples using our collision handler, while we can tolerate a few false positive samples. As illustrated in Fig. 3 and Fig. 4, our method effectively improves both metrics. The performance after active learning is summarized in Table 2. We reach an accuracy of 85.6 – 98.1% compared with the groundtruth generated by the exact method (Pan et al., 2012; Muller et al., 2021). On average, our method achieves 1.62% higher accuracy and a 13.59% lower false negative rate than ***Supv***. In the last row of Table 2, we measure an equivalent dataset size, which

metric	SCAPE	Swing	Jump	Skirt	Hand	AMASS
final dataset size	5.5×10^5	4.5×10^4	4.5×10^4	4×10^4	5.5×10^5	5×10^4
accuracy (Ours (***Active+bd***))	0.9383	0.9638	0.9552	0.9817	0.9692	0.8563
accuracy (***Supv+bd***)	0.9282	0.9609	0.9500	0.9795	0.9650	0.8551
accuracy (***Supv***)	0.9181	0.9460	0.9347	0.9660	0.9558	0.8526
false neg. (Ours (***Active+bd***))	0.05151	0.01485	0.02573	0.01808	0.03582	0.17656
false neg. (***Supv+bd***)	0.05576	0.01766	0.02644	0.01956	0.03652	0.18422
false neg. (***Supv***)	0.06914	0.01969	0.02713	0.02056	0.03727	0.19654
equi. dataset size (***Supv+bd***)	6.63×10^5	7.64×10^4	7.02×10^4	7.71×10^4	7.30×10^5	7.40×10^4

Table 2. We summarize the accuracy and false negative rate of three methods under comparison. We also include the equivalent dataset size for the baseline to reach the same performance as our method.

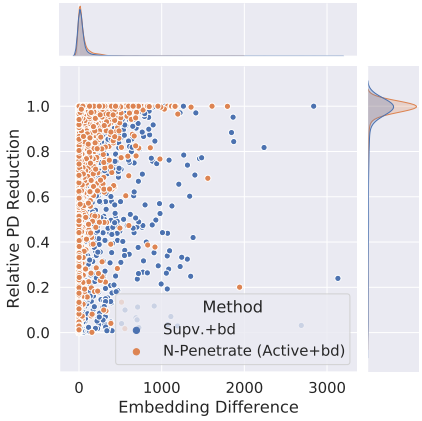

Figure 7. We plot joint distribution of relative PD reduction (Y-axis) and embedding difference (X-axis) over successfully collision-handled test meshes in the SCAPE dataset for our method and ***Supv+bd***. Our method resolves more collisions (average PD reduction 94.81% vs. 88.76%) while remaining closer to the input (average embedding difference 56.17 vs. 66.11) as compared with ***Supv+bd***.

is defined as the size of the dataset needed by the ***Supv+bd*** to achieve the same accuracy as our method. We derive this number by interpolating on experimental results of the ***Supv+bd***. The results show that our method achieves a similar accuracy using a 34.6% smaller dataset than ***Supv+bd*** on average.

Running Time: We have compared our neural collision detector with BVH-based exact method provided as part of FCL library (Pan et al., 2012) on a dataset with 1×10^4 meshes, which is unseen by our learning method. We compared the CPU-based implementations on a Intel Xeon Silver 4208 CPU (32 cores). Our neural method is implemented using PyTorch and we evaluate both CPU and GPU versions. We compare a parallel and tuned version of BVH-based algorithm in FCL using 15 threads. We can achieve $29 - 124\times$ speedup over the BVH-based collision checking (Table 3).

Collision Handling: We plug the trained neural collision detectors into ALM and compare our method, ***Supv***, against ***Supv+bd*** in terms of resolving self-penetrating meshes. To this end, we randomly sample 10000 self-penetrating, unseen Z_{all}^{user} from $U(\mathcal{Z})$, and use Eq. 1 to derive Z_{all}. For

	SCAPE	Swing	Jump	Skirt	Hand
FCL (parallel BVH-based collision checker)	37.84s	134.1s	135.7s	43.82s	42.7s
Ours (CPU)	1.05s	1.20s	1.16s	1.13s	1.46s
Ours (GPU)	0.922s	1.14s	1.09s	0.956s	1.32s
Speedup (CPU)	36.03	111.75	116.98	38.78	29.25
Speedup (GPU)	41.04	117.62	124.50	45.84	32.35

Table 3. Comparing our neural method with exact collision checking in FCL. Our method achieves up to $124\times$ speedup.

the SMPL model, we will fix Z_β in the optimization to maintain the body shape for each sample. We compare the performance based on relative PD reduction defined as:

$$\frac{\mathrm{PD}(\mathrm{ACAP}^{-1}(Z_{\mathrm{all}}^{user}, \theta_D)) - \mathrm{PD}(\mathrm{ACAP}^{-1}(Z_{\mathrm{all}}, \theta_D))}{\mathrm{PD}(\mathrm{ACAP}^{-1}(Z_{\mathrm{all}}^{user}, \theta_D))}.$$

Collision resolution is completely successful if this value equals one, which may not always happen because ALM uses soft penalties to relax hard constraints. Thus, we consider a solution successful if the value is greater than 0. We plot the success rate against the dataset size in Fig. 5, which shows that our method resolves 22.73% more collisions than ***Supv+bd***. Thanks to our risk-seeking data aggregation method, our method monotonically improves the collision handling success rate when more data points are injected, while ***Supv+bd*** exhibits unstable performance. Since ***Supv*** uses the same randomly sampled dataset as ***Supv+bd***, the performance exhibits similar instability. Meanwhile, our novel boundary loss improves the results for ***Supv+bd***, since it can better approximate the decision boundary. Another criterion for good collision handling is the embedding difference – the objective function in Eq. 1. We want the output to be as close to the input as possible. We plot the relative PD reduction vs. embedding difference over successfully collision-handled test meshes in the SCAPE dataset for our method and ***Supv+bd*** in Fig. 7. The mean relative PD reduction for our method is 94.81% and the mean embedding difference is 56.17, compared to 88.76% and 66.11, respectively, for ***Supv+bd***. The results show that our method resolves more collisions while the outputs stay closer to the input latent codes. Some exemplary results are shown in Fig. 6.

6. Conclusion & Limitations

We present an active learning method for training a neural collision detector in which training data are progressively sampled from the learned latent space using a risk-seeking approach. Our approach is designed for general 3D deformable meshes, and we highlight its benefits on many complex datasets. In practice, our method outperforms supervised learning in terms of accuracy, false negative rate, and stability. As a major limitation, our collision handler does not consider physics models. Physics models can be incorporated in the future via a learning-based physics sim-

ulation approach such as (Zheng et al., 2021). We are also considering extensions to meshes with changing topologies, e.g., using a level-set-based mesh representation.

References

Aghdam, H. H., Gonzalez-Garcia, A., Weijer, J. v. d., and López, A. M. Active learning for deep detection neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3672–3680, 2019.

Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., and Davis, J. Scape: shape completion and animation of people. In *ACM SIGGRAPH*, pp. 408–416. ACM, 2005.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D., and Kavukcuoglu, K. Interaction networks for learning about objects, relations and physics. *arXiv preprint arXiv:1612.00222*, 2016.

Cudeiro, D., Bolkart, T., Laidlaw, C., Ranjan, A., and Black, M. Capture, learning, and synthesis of 3D speaking styles. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 10101–10111, 2019. URL http://voca.is.tue.mpg.de/.

Das, N., Gupta, N., and Yip, M. Fastron: An online learning-based model and active learning strategy for proxy collision detection. In *Conference on Robot Learning*, pp. 496–504. PMLR, 2017.

De Raedt, L., Passerini, A., and Teso, S. Learning constraints from examples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Gal, Y., Islam, R., and Ghahramani, Z. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, pp. 1183–1192. PMLR, 2017.

Gall, J., Stoll, C., De Aguiar, E., Theobalt, C., Rosenhahn, B., and Seidel, H.-P. Motion capture using joint skeleton tracking and surface estimation. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 1746–1753. IEEE, 2009.

Galliani, S., Lasinger, K., and Schindler, K. Massively parallel multiview stereopsis by surface normal diffusion. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 873–881, 2015. doi: 10.1109/ICCV.2015.106.

Gao, L., Yang, J., Qiao, Y.-L., Lai, Y.-K., Rosin, P. L., Xu, W., and Xia, S. Automatic unpaired shape deformation transfer. *ACM Transactions on Graphics (TOG)*, 37(6): 1–15, 2018.

Gao, L., Lai, Y.-K., Yang, J., Ling-Xiao, Z., Xia, S., and Kobbelt, L. Sparse data driven mesh deformation. *IEEE transactions on visualization and computer graphics*, 2019.

Geifman, Y. and El-Yaniv, R. Deep active learning over the long tail. *arXiv preprint arXiv:1711.00941*, 2017.

Govindaraju, N. K., Lin, M. C., and Manocha, D. Quick-cullide: Fast inter-and intra-object collision culling using graphics hardware. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.*, pp. 59–66. IEEE, 2005.

Groueix, T., Fisher, M., Kim, V. G., Russell, B. C., and Aubry, M. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 216–224, 2018.

Gundogdu, E., Constantin, V., Seifoddini, A., Dang, M., Salzmann, M., and Fua, P. Garnet: A two-stream network for fast and accurate 3d cloth draping. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Holden, D., Duong, B. C., Datta, S., and Nowrouzezahrai, D. Subspace neural physics: Fast data-driven interactive simulation. In *Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 1–12, 2019.

Kim, Y., Lin, M., and Manocha, D. Collision and proximity queries. *Handbook of Discrete and Computational Geometry*, 2018.

Lahner, Z., Cremers, D., and Tung, T. Deepwrinkles: Accurate and realistic clothing modeling. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 667–684, 2018.

Liu, L., Zheng, Y., Tang, D., Yuan, Y., Fan, C., and Zhou, K. Neuroskinning: Automatic skin binding for production characters with deep graph networks. *ACM Trans. Graph.*, 38(4), July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322969. URL https://doi.org/10.1145/3306346.3322969.

Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., and Black, M. J. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6): 248:1–248:16, October 2015.

Mahmood, N., Ghorbani, N., Troje, N. F., Pons-Moll, G., and Black, M. J. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision*, pp. 5442–5451, October 2019.

Muller, L., Osman, A. A. A., Tang, S., Huang, C.-H. P., and Black, M. J. On self-contact and human pose. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9990–9999, June 2021.

Narain, R., Samii, A., and O'Brien, J. F. Adaptive anisotropic remeshing for cloth simulation. *ACM Transactions on Graphics*, 31(6):147:1–10, November 2012. URL http://graphics.berkeley.edu/papers/Narain-AAR-2012-11/. Proceedings of ACM SIGGRAPH Asia 2012, Singapore.

Niculescu, R. S., Mitchell, T. M., Rao, R. B., Bennett, K. P., and Parrado-Hernández, E. Bayesian network learning with parameter constraints. *Journal of machine learning research*, 7(7), 2006.

Osman, A. A. A., Bolkart, T., and Black, M. J. STAR: A sparse trained articulated human body regressor. In *European Conference on Computer Vision (ECCV)*, pp. 598–613, 2020. URL https://star.is.tue.mpg.de.

Pan, J., Chitta, S., and Manocha, D. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pp. 3859–3866. IEEE, 2012.

Pan, J., Zhang, X., and Manocha, D. Efficient penetration depth approximation using active learning. *ACM Transactions on Graphics (TOG)*, 32(6):1–12, 2013.

Patel, C., Liao, Z., and Pons-Moll, G. Tailornet: Predicting clothing in 3d as a function of human pose, shape and garment style. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2020.

Paul, R., Feldman, D., Rus, D., and Newman, P. Visual precis generation using coresets. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1304–1311. IEEE, 2014.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NIPS*, 2017.

Qiao, Y.-L., Lai, Y.-K., Fu, H., and Gao, L. Synthesizing mesh deformation sequences with bidirectional lstm. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2020. doi: 10.1109/TVCG.2020.3028961.

Ritchie, D., Wang, K., and Lin, Y.-a. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6182–6190, 2019.

Satopaa, V., Albrecht, J., Irwin, D., and Raghavan, B. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pp. 166–171. IEEE, 2011.

Sener, O. and Savarese, S. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1aIuk-RW.

Simon, T., Joo, H., Matthews, I., and Sheikh, Y. Hand keypoint detection in single images using multiview bootstrapping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Smith, B., Wu, C., Wen, H., Peluse, P., Sheikh, Y., Hodgins, J. K., and Shiratori, T. Constraining dense hand surface tracking with elasticity. *ACM Trans. Graph.*, 39(6), November 2020. ISSN 0730-0301. doi: 10.1145/3414685.3417768. URL https://doi.org/10.1145/3414685.3417768.

Sun, W. and Yuan, Y.-X. *Optimization theory and methods: nonlinear programming*, volume 1. Springer Science & Business Media, 2006.

Tan, Q., Gao, L., Lai, Y.-K., and Xia, S. Variational autoencoders for deforming 3d mesh models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5841–5850, 2018a.

Tan, Q., Gao, L., Lai, Y.-K., Yang, J., and Xia, S. Mesh-based autoencoders for localized deformation component analysis. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b.

Tan, Q., Pan, Z., and Manocha, D. Lcollision: Fast generation of collision-free human poses using learned non-penetration constraints. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, 2021.

Tang, M., Manocha, D., Otaduy, M. A., and Tong, R. Continuous penalty forces. *ACM Transactions on Graphics (TOG)*, 31(4):1–9, 2012.

Tian, H., Zhang, X., Wang, C., Pan, J., and Manocha, D. Efficient global penetration depth computation for articulated models. *Computer-Aided Design*, 70:116–125, 2016.

Vlasic, D., Baran, I., Matusik, W., and Popović, J. Articulated mesh animation from multi-view silhouettes. In *ACM SIGGRAPH 2008 papers*, pp. 1–9. ACM, 2008.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

Xu, Z., Zhou, Y., Kalogerakis, E., Landreth, C., and Singh, K. Rignet: Neural rigging for articulated characters. *ACM Trans. on Graphics*, 39, 2020.

Yang, J., Gao, L., Tan, Q., Huang, Y., Xia, S., and Lai, Y.-K. Multiscale mesh deformation component analysis with attention-based autoencoders, 2020a.

Yang, J., Mo, K., Lai, Y., Guibas, L. J., and Gao, L. Dsm-net: Disentangled structured mesh net for controllable generation of fine geometry. *CoRR*, abs/2008.05440, 2020b. URL https://arxiv.org/abs/2008.05440.

Yu, F., Liu, K., Zhang, Y., Zhu, C., and Xu, K. Partnet: A recursive part decomposition network for fine-grained and hierarchical shape segmentation. In *CVPR*, pp. to appear, 2019.

Zhang, X., Kim, Y. J., and Manocha, D. Continuous penetration depth. *Computer-Aided Design*, 46:3–13, 2014.

Zheng, M., Zhou, Y., Ceylan, D., and Barbic, J. A deep emulator for secondary motion of 3d characters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5932–5940, June 2021.

A. Long Tail Distribution of Data

We have included a figure (Fig. 8) showing histogram of the penetration depth (PD) over the test samples of Swing dataset. This forms a long-tail distribution.

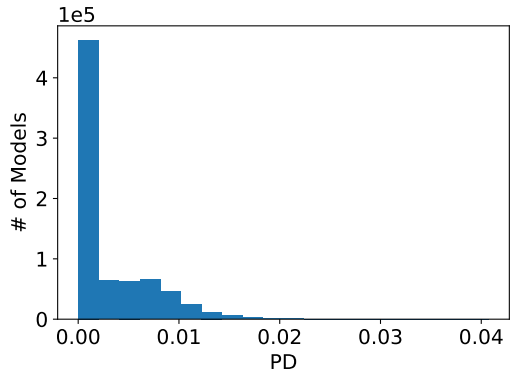

Figure 8. We show the histogram of the test samples of Swing dataset according to PD, which illustrates the long-tail characteristic.

B. Selection of N_{init}

The N_{init} for our method is setting as the "elbow point" of the accuracy vs. the sample size of the baseline method ***Supv + bd***, which is detected using (Satopaa et al., 2011). The parameters we use for (Satopaa et al., 2011): curve as concave, direction as increasing, sensitivity as 1 and interpolation method as polynomial. We show an example on Hand dataset in Fig. 9.

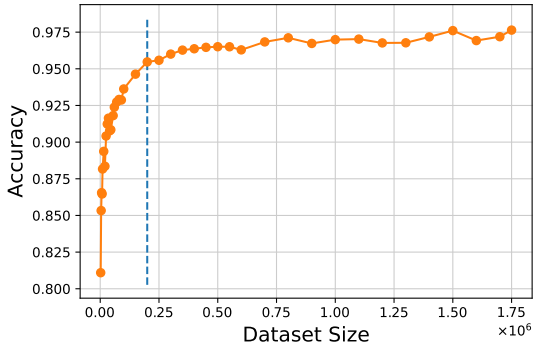

Figure 9. We plot the accuracy of ***Supv + bd*** against the dataset size from 2000 to 1.75×10^6. The "elbow point" detected by (Satopaa et al., 2011) in this plot is 200000 as shown by the dashed line.

C. Overall Training Algorithm for θ_C

To help readers get a better understanding how we use active learning, our overall training method for the parameters θ_C of the collision detector is illustrated in Algorithm 1.

Algorithm 1 Learning θ_C

Prepare initial data set $\mathcal{D}$
Learn a latent space using autoencoder or directly use the SMPL model
Initialize $\mathcal{D}_C$ by drawing N_{init} samples from $U(\mathcal{Z})$
for $Z_{\text{all}} \in \mathcal{D}_C$ **do**
  Compute PD and $I_c^*(Z_{\text{all}})$
Update θ_C using $\mathcal{D}_C$
while Not converged **do**
  $\Delta\mathcal{D}_{C1} \leftarrow$Draw $N_{\text{aug}}/2$ samples from $\mathcal{D}_C$
  for $Z_{\text{all}} \in \Delta\mathcal{D}_{C1}$ **do**
    $Z_{\text{all}}^{\text{last}} \leftarrow Z_{\text{all}}$
    while True **do**
      Update Z_{all} using Eq. 4
      if $\|Z_{\text{all}} - Z_{\text{all}}^{\text{last}}\|_\infty$ sufficiently small **then**
        Break
      $Z_{\text{all}}^{\text{last}} \leftarrow Z_{\text{all}}$
    $\mathcal{D}_C \leftarrow \mathcal{D}_C \bigcup \{\langle Z_{\text{all}}^{\text{last}}, I_c^*(Z_{\text{all}}^{\text{last}})\rangle\}$
  $\Delta\mathcal{D}_{C2} \leftarrow$Draw $N_{\text{aug}}/2$ samples from $U(\mathcal{Z})$
  for $Z_{\text{all}} \in \Delta\mathcal{D}_{C2}$ **do**
    $\mathcal{D}_C \leftarrow \mathcal{D}_C \bigcup \{\langle Z_{\text{all}}, I_c^*(Z_{\text{all}})\rangle\}$
  Update θ_C using $\mathcal{D}_C$