Appendix

A. Video comparisons

Please see our supplementary video, `ICCV2021.mp4`, for video comparisons. The video playback has been tested using VLC player and Apple's Quicktime.

A.0.1 Ablation study

Method	Geometry evaluation		Motion evaluation	
	Inter-Penetrations ↓	Vertex Contact MSE ↓	Foot Contact Accuracy ↑	Global Position MSE ↓
E_{skel}	1.67	4.23	0.71	0.56
$E_{skel} + E_{foot}$	1.59	4.21	0.80	0.66
$E_{skel} + E_{foot} + E_{int}$ without $w_{r,i}$	**0.35**	7.94	0.82	1.00
$E_{skel} + E_{foot} + E_{int}$ with $w_{r,i}$	0.56	6.79	0.81	0.97
E_{full}	1.18	4.52	0.76	1.54
E_{full}+ ESO	0.81	**3.87**	**0.97**	**0.48**

Table 3. We show the effects of the different energy terms in our contact-aware motion retargetting method.

In order to demonstrate the effectiveness of the different terms of our energy function (Equation 1), we provide a detailed ablation study in Table 3.

We first evaluate the performance of our network trained with the skeleton-level motion modeling term, E_{skel} which includes the local and global motion terms and the end-effector motion terms. While using only the skeleton term results in lower global position error, we observe worse foot contact modeling and interpenetration in comparison to the other baselines.

We next evaluate the different terms involved in the geometry based energy. First, we add the foot contact preservation term, E_{foot}. This version results in a significant boost in terms of handling foot contacts. Then, we include the interpenetration term E_{int} without the geodesic based vertex weighting, $w_{r,i}$, which results in the best interpenetration reduction, but worst vertex contact MSE. This is because the retargeted motion tries to avoid all interpenetration, including those that involve nearby vertices that are often not noticeable by viewers. This turns out to result in a stiff motion where most self-contacts are avoided. Adding the weight, $w_{r,i}$, addresses this issue by relaxing the range of the character motion. We observe that slight interpenetration occurs, but the overall motion quality and contact handling improve. We then include the vertex contact modeling term, E_{j2j}, effectively using our full energy function E_{full}. This results in a slight degradation in terms of interpenetration and motion modeling, but better self-contact handling. Finally, incorporating the Encoder-Space Optimization (ESO) to satisfy hard constraints achieves the best overall performance both in terms of motion quality, foot contact, and self-contact handling.

For sake of completeness, we also check whether we can achieve accurate foot contacts by simply applying the IK post-optimization step in [2], E_{full} + *IK post-process Optim.*, and find that it improves on foot contacts accuracy, as easier task than full energy, but does not outperform our encoder-state optimization.

B. Geometry-Conditioned RNN Details

The input to the network is the source skeletal motion $m^A_{1:T} \in \mathbb{R}^{267}$ and the target character represented as a set of skeleton joint offsets $\bar{s}^B \in \mathbb{R}^{J\times 3}$ in reference pose (i.e., T-pose) and a skin geometry with vertices $\bar{v}^B \in \mathbb{R}^{V\times 3}$. The source motion (and similarly the retargeted motion) can be decomposed into (i) $p^A_{1:T} \in \mathbb{R}^{J\times 3}$, local joint coordinates with respect to the hip (root) joint, (ii) $\theta^A_t \in \mathbb{R}^{9J}$, local joint rotations with respect to each parent joint, and (iii) $o^A_t \in \mathbb{R}^3$, the global root velocity.

As illustrated in Figure 3, at each frame t, the network retargets the motion by the following:

$$h_t^{\text{enc}} = f^{\text{enc}}(m_t^A, h_{t-1}^{\text{enc}}), \tag{12}$$

$$h_t^{\text{dec}} = f^{\text{dec}}(p_{t-1}^B, o_{t-1}^B, \hat{s}^B, e^B, h_t^{\text{enc}}, h_{t-1}^{\text{dec},}), \tag{13}$$

$$\theta_t^B = f^{\theta}(h_t^{\text{dec}}), \tag{14}$$

$$p_t^B = \text{FK}(\theta_t^B, \hat{s}^B), \tag{15}$$

$$\hat{v}_t^B = \text{SK}(\theta_t f^B, p_t^B, \hat{s}^B), \tag{16}$$

$$o_t^B = f^{o}(h_t^{\text{dec}}), \tag{17}$$

$$g_t^B = p_t^B + o_{t-1}^B, \tag{18}$$

where f^{enc} is an encoder RNN, f^{dec} is a decoder RNN, and f^{θ} and f^{o} are linear layers that output rotations and the root velocity, respectively. FK$(.,.)$ is the forward kinematics layer that reposes the skeleton based on the joint rotations and SK$(.,.,.)$ is the skinning layer that deforms the skin geometry accordingly. $h_t^{\text{enc}} \in \mathbb{R}^d$ is state of the encoder RNN at frame t, $h_{t-1}^{\text{enc}} \in \mathbb{R}^d$ and $h_{t-1}^{\text{dec}} \in \mathbb{R}^d$ are the states of the encoder and decoder RNNs in the previous frame. θ_t^B are the joint rotations retargeted into character B, p_t^B and p_{t-1}^B are the resulting local joint coordinates after applying forward kinematics for frame t and $t-1$, and o_t^B and o_{t-1}^B are the root velocities retargeted to character B for frame t and $t-1$. $\hat{v}_t^B$ is the skinned retargeted motion output by the skinning layer and g_t^B are the retargeted motion joints in global coordinate system. Finally, e^B is an embedding computed from the target geometry by:

$$e^B = \max_{\bar{v}^B} \quad f^{\text{vert}}(\bar{v}^B, w^B) \tag{19}$$

where f^{vert} is implemented as a two layer neural network that maps the concatenated vertices $\bar{v}^B \in \mathbb{R}^{V' \times 3}$ in the t-pose and the skinning weights $w^B \in \mathbb{R}^{V' \times J}$ into feature vectors for each vertex which are then max pooled over all extracted vertex features following PointNet. Please note that our network architecture is independent of the choice of the vertex encoder and we use PointNet since it is independent of the underlying mesh topology.

B.1. Architecture, training and testing details

We train our method on a single NVIDIA Tesla V100 GPU (16GB). Our encoder f^{enc} and decoder f^{dec} are implemented with a two-layer GRU with 512 hidden units each. Our output functions f^{θ} and f^{o} are implemented with single linear layers. FK$(.,.)$ and SK$(.,.,.)$ are implemented with standard Forward Kinematics (FK) and Linear Blend Skinning (LBS) formulations. For our encoder f^{vert}, we use a two-layer MLP with ReLU activation and 256 hidden units in each layer. During training, we use the Adam solver with $\beta_1 = 0.5, \beta_2 = 0.999, \epsilon = 1e-8$ and learning rate of 0.0001. We train for 10 epochs with dropout out rate of 0.1, batch size of 256 sequences, and learning rate decay of 0.95 per epoch. At training time, our energy function hyper-parameters are set as $\lambda = 1000$, $\beta = 100$, $\gamma = 0.1$, $\rho = 1000$, and $\omega = 1000$. During training, we gradually introduce the foot contact loss E_{foot} by multiplying γ times a weigh factor defined as $currentepoch/totalepochs$. During the first epoch the training focuses on moving the character limbs without penalizing ground contacts. LBS is GPU memory heavy, therefore, we use a sampling strategy during training to minimize GPU usage while modeling longer motion sequences (60 frames at training time). We uniformly sample a single motion frame per motion sequence and apply interpenetration and self-contact penalties the chosen frame. By randomly sampling, our network still need to learn to model the penalties in order to minimize them for any frame in the motion. Using larger accelerators (GPU or TPU) can enable our method to apply the penalties for the same sequence length used in our experiments, and so, improve its performance. In order to easily batch multiple character meshes during training, we perform a preprocessing step where we sub-sample all our training meshes to 3000 vertices. Nevertheless, our network can read in an arbitrary number of vertices via f^{vert} at test time. During our ESO step, we optimize $h_{t,n+1}^{\text{enc}}$ and $o_{t-1,n+1}$ by using the Adam solver with $\beta_1 = 0.5, \beta_2 = 0.999, \epsilon = 1e-8$ and learning rate of $\alpha = 0.01$. In addition, we relax the interpenetration and self-contact penalties hyper-parameters to $\beta = 1.0$ and $\lambda = 100.0$ to allow the motion to be more dynamic. In addition, we only optimize self-contacts if the initial output from our network is within 15cm of the target contact location. We do this because the initial outputs from our network should respect the target character's shape. Therefore a large distance to a contact point means it is likely not possible for the character to reach the contact point without performing non-natural motion. Moreover, we encourage hands to move smoothly from the initial output during ESO by aggregating 0.7 of the distance from the initial vertex position and 0.3 of the target contact vertex position as the self-contact penalty. Finally, we only allow hands to move freely by adding a distance penalty for the joints the hands reach during contact to stay close to their original positions.

C. Test set details

We downloaded source 30 motions of variable length where the shortest motion is 27 frames and the longest motion is 455 frames. In addition, we download a total of 180 target motions for evaluation which are composed of the original 30 motions retargeted into 6 different characters where 3 are skinny and 3 are bulky. We make sure all of our test characters contain the same motions so that we can test the behavior of each of those motions retargeted to characters with different geometries. We chose motions that have 3 key properties such as self-contacts, potential interpenetration, and also self-contact free motions. Within those key properties we can test whether our algorithm is addressing the problems we are trying to solve involving geometry constraints and skeletal motion constraints. We will release the test set upon acceptance of our paper.

D. Skinny versus bulky character evaluation

In this section, we present a split of our quantitative evaluation into skinny and bulky characters. In Tables 4 and 5, we note the difficulty of modeling bulky characters versus skinny characters. We can see that Interpenetration and Vertex Contact MSE roughly two times worse in the bulky characters results (Table 5). From our observations, we conjecture that our bulky characters have a harder time mimicking the source motion which in our experiments comes from a skinny character (Ybot) without violating their geometric constraints. This is observed in our Global Position MSE results which are worse than the baseline which does not model geometric constraints, SAN. In contrast, our Global Position MSE and all geometry evaluation metrics from skinny character motion retargeting are the best amongst all methods (Table 4). This happens because it is easier to mimic Ybot's motion due to the target characters also having a thin body geometry.

Skinny characters evaluation				
Method	Geometry evaluation		Motion evaluation	
	Inter-Penetrations ↓	Vertex Contact MSE ↓	Foot Contact Accuracy ↑	Global Position MSE ↓
Ours	**0.49**	**2.24**	**0.98**	**0.31**
E_{full} + IK	0.82	2.81	0.81	1.97
SAN [2]	1.21	4.50	0.61	1.23
SAN + IK [2]	1.17	4.22	0.71	1.13
PMnet [20]	2.94	18.31	0.71	4.38
NKN [31]	2.11	13.00	0.72	13.95

Table 4. Skinny characters evaluation. We evaluate the retargetted motion at the geometry and skeletal level. We evaluate the amount of self-penetrations, geometry contacts distance, foot contacts with the floor, and global joint positions in the retargetted motion.

Bulky characters evaluation				
Method	Geometry evaluation		Motion evaluation	
	Inter-Penetrations ↓	Vertex Contact MSE ↓	Foot Contact Accuracy ↑	Global Position MSE ↓
Ours	**1.14**	**5.50**	**0.96**	0.65
E_{full} + IK	1.56	6.24	0.84	1.19
SAN [2]	1.66	6.02	0.64	0.42
SAN + IK [2]	1.48	7.34	0.74	**0.35**
PMnet [20]	4.01	27.92	0.70	2.97
NKN [31]	4.29	16.72	0.70	2.34

Table 5. Bulky characters evaluation. We evaluate the retargetted motion at the geometry and skeletal level. We evaluate the amount of self-penetrations, geometry contacts distance, foot contacts with the floor, and global joint positions in the retargetted motion.

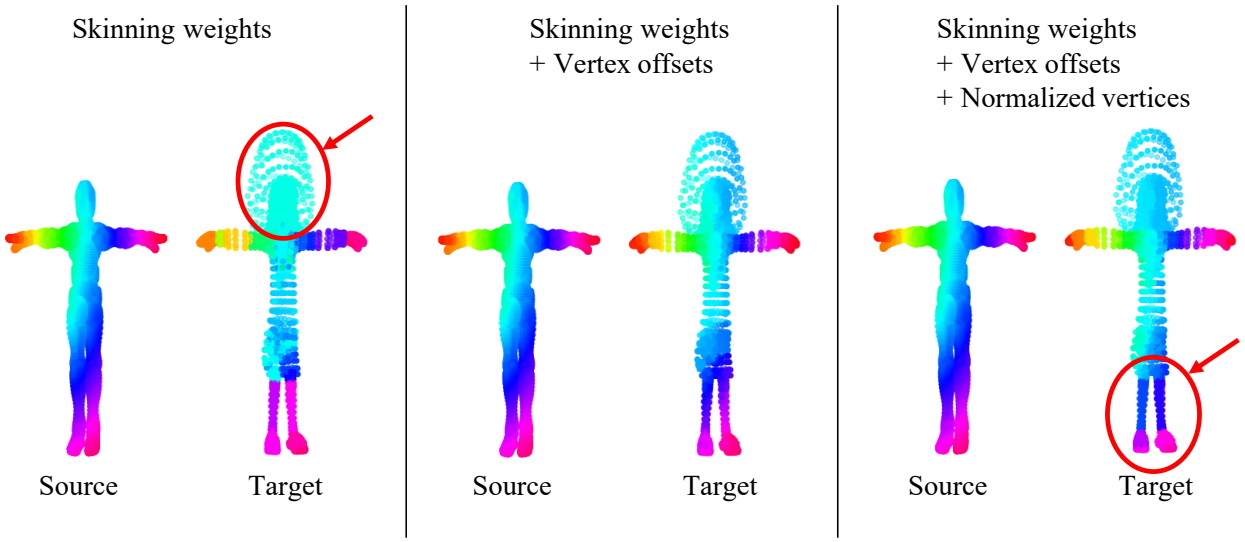

Figure 7. Correspondences experiments. We show color-coded results for using different features while computing vertex correspondences. We experimented with 1) only using skinning weights, 2) adding vertex offset from parent joint, and 3) adding normalized vertex coordinates. As pointed by the circles and arrows, feature combinations 1) and 3) result in inaccurate correspondence mapping due to ambiguity of skinning weights alone and noise added by the normalized vertices.

E. User study details

The SAN baseline we compare against is the full method presented in their original paper using IK as post-processing. We do this to take advantage of the slight foot sliding fix performed by their standard IK post optimization method. We independently sample 20 sequences without replacement for each subject in our study using a uniform distribution over the 180 available sequences. For each of the 20 samples, we choose whether our result is placed on the left or right side of the source motion with a probability of 0.5. Finally, we label the video on the left as *A)* and the video on the right as *B)*. Therefore, each of our subjects get a unique set of 20 videos with the source motion in the middle, a video labeled as *A)* on the left and a video labeled as *B)* on the right. They are asked choose the video that looks closest to the source motion and provide 20 answers where each answer is either A or B. We then get their answers, check whether they picked our motion or the baseline, and compute the numbers presented in the main text.

F. Correspondence feature vector details

For our vertex correspondence computation, we perform nearest neighbors search using the KDTree algorithm on a feature vector based on each vertex skinning weights and offset from parent joint. We next describe how we decided the use of that specific feature vector. We experimented with 3 different sets of feature vectors for computing correspondences between source and target mesh vertices. We started by simply using skinning weight as feature vectors, and found that it resulted in inaccurate correspondences where large portions of the target mesh map to a similar vertex in the source mesh as see on the left column in Figure 7. We hypothesize that this is due to ambiguities in the skinning weights were multiple vertices are transformed with the same skinning weight values. Next, we move onto using skinning weights and offsets going from a vertex to its parent joint in the character skeleton. We find the mix of the two features doing a reasonable job as seen on the middle column in Figure 7. Finally, we experimented with the normalized mesh vertices and find that it negatively affects the correspondence results. We hypothesize that because the source and target meshes are different in shape, using their vertices adds noise to the correspondence computation. Therefore, in our final method, we use a feature vector constructed from each vertex skinning weights and offset from its parent joint.

G. Data processing details

The motion data used in our experiments consists of the following joint set: Root, Spine, Spine1, Spine2, Neck, Head, LeftUpLeg, LeftLeg, LeftFoot, LeftToeBase, RightUpLeg, RightLeg, RightFoot, RightToeBase, LeftShoulder, LeftArm,

LeftForeArm, LeftHand, RightShoulder, RightArm, RightForeArm, and RightHand. These 22 joints are shared across all Mixamo characters, and represent the main motion present in Mixamo animations.