CharacterMixer: Rig-Aware Interpolation of 3D Characters

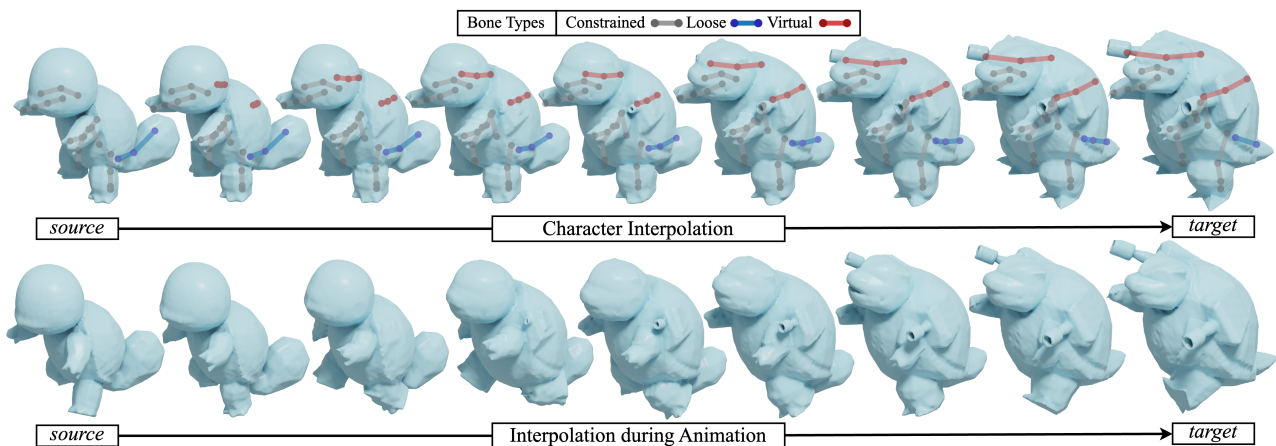Xiao Zhan[1]   Rao Fu[1]   Daniel Ritchie[1]

[1]Brown University, USA

Figure 1: *CharacterMixer enables interpolation between two 3D models that have different surface mesh topologies and rig skeletons. It preserves a posable rig throughout interpolation. In the top row, we show an example of interpolating source to target with a fixed pose. CharacterMixer constructs three types of bones to handle different skeleton topologies: 1-to-1 matched bones are Constrained, 1-to-many matches are Loose (tail), and 1-to-void matches are Virtual (shoulder-mounted cannons). The bottom row shows the same interpolation, but where the character's pose changes continuously during a run cycle (posed rigs are omitted for a clear visual of the geometry).*

Abstract

We present CharacterMixer, a system for blending two rigged 3D characters with different mesh and skeleton topologies while maintaining a rig throughout interpolation. CharacterMixer also enables interpolation during motion for such characters, a novel feature. Interpolation is an important shape editing operation, but prior methods have limitations when applied to rigged characters: they either ignore the rig (making interpolated characters no longer posable) or use a fixed rig and mesh topology. To handle different mesh topologies, CharacterMixer uses a signed distance field (SDF) representation of character shapes, with one SDF per bone. To handle different skeleton topologies, it computes a hierarchical correspondence between source and target character skeletons and interpolates the SDFs of corresponding bones. This correspondence also allows the creation of a single "unified skeleton" for posing and animating interpolated characters. We show that CharacterMixer produces qualitatively better interpolation results than two state-of-the-art methods while preserving a rig throughout interpolation. Project page: https://seanxzhan.github.io/projects/CharacterMixer.

CCS Concepts
• *Computing methodologies → Shape analysis; Shape modeling;*

1. Introduction

Interpolation is a fundamental operation in 3D shape modeling and editing. Producing smooth blends between shapes can be used to create animations [The23], to "fill in gaps" between shapes in a collection [MCA*22], or to create new hybrid shapes [ALX*14]. One of the most common types of 3D shape is a *3D character*:

an articulated body that is animated in some film, game, or other 3D graphics experience. Interpolation between 3D characters can be used for pose matching [ENK*21] or for creating a range of blended characters from a smaller set of hand-modeled ones (e.g. for creating crowds of background characters) [CG 23].

When the shapes to be interpolated are 3D characters, the *rigs*,

or the articulated skeletons that allow characters to be animated, for those characters should be taken into account, which complicates the problem. In practice, most systems which can interpolate between rigged characters are based on parametric models which can produce variations of the character's body shape but always keep the same surface mesh and rig topologies, limiting the range of characters that can be interpolated [LMR*15]. Methods exist for interpolating between different 3D shapes, but when applied to rigged characters, they ignore the rigs, leading to intermediate shapes that are no longer rigged and thus not directly posable [ENK*21].

In this paper, we present CharacterMixer, the first system for interpolating between two rigged characters with different mesh and skeleton topologies, such that a rig is preserved. With the preserved rig, not only does CharacterMixer allow interpolated characters to be posable, but it can also generate animation sequences where interpolation happens at the same time (Fig. 1). Handling different mesh and skeleton topologies is crucial for interpolation tasks involving characters not created by the same artist, or when animators do not control characters' sources, such as uploaded assets in the online gaming community. To handle characters with different mesh topologies, CharacterMixer uses signed distance field (SDF) representations of the source and target geometries. To make the system rig-aware, a character is represented as a union of SDFs, one per each bone of the rig. Mesh-based methods such as NeuroMorph [ENK*21] are unable to interpolate the identities of characters; they deform the source mesh to match the shape of the target, keeping the source topology unchanged. In contrast, the SDF representation allows our method to interpolate geometry and produce intermediate characters of different identities (Fig. 1). To interpolate between two rigged characters with different skeletal topologies, CharacterMixer computes a hierarchical correspondence between two skeletons. This correspondence allows it to create a single "unified skeleton" whose pose can drive the pose of both the source and target characters. Given the unified skeleton, CharacterMixer interpolates between the two characters' geometries by linear interpolation of the SDFs of corresponding bones.

We evaluate CharacterMixer by comparing to a state-of-the-art optimal transport approach for shape interpolation [SdGP*15] and a mesh-based data-driven method for shape correspondence and interpolation [ENK*21], showing that CharacterMixer generates intermediate shapes with higher visual fidelity while also maintaining a posable rig. In summary, our contributions are:

- A method for computing hierarchical correspondence between two skeletons and producing unified intermediate skeletons
- A technique for posing and animating interpolated characters using the unified skeletons
- An interpolation approach for blending between two characters' geometries while preserving a rig

2. Related Works

Shape Interpolation. There is a significant body of prior work on shape interpolation and blending. One family of work uses optimal transport, treating the source and target shapes as probability distributions and finding a transformation of the source to the target that moves as little probability mass as possible [SdGP*15, JCG20, MDZ*21]. Another work interpolates the

interiors of shapes in an as-rigid-as-possible manner, restricting local volumes to be least-distorting [ACOL00]. There are also data-driven approaches, interpolating from a source shape to a target shape by finding a path through a large collection of related shapes [AS21, GCLX16, GLHH13] or using the structures of manufactured shapes [YML*22, GYW*19]. Most recently, several works train neural networks to produce deformations from a source to target shape [ENK*21, YAK*20, JHTG20]. Deep generative models can also be viewed as interpolators, as their latent spaces allow interpolation between shapes in the generator's output domain [ADMG17, LLHF21, YHH*19, CZ19, ZLWT22]. These methods are all oblivious to character rigs and thus intermediate interpolated shapes would not be posable, making it impossible to interpolate throughout an animation.

To the best of our knowledge, no prior work focuses on rig-aware character interpolation. Parametric body models, such as SMPL [LMR*15], support interpolation between body shapes with the same rig; these shapes all have the same mesh and skeleton topology. Our method supports interpolation between characters with different mesh and skeleton topologies.

Automated Character Rigging. Our system interpolates between 3D characters such that the intermediate characters are still animatable. One could instead use a rig-oblivious shape interpolation method and then attempt to automatically compute a rig for the new intermediate shape. Several automated rigging methods exist: some are restricted to characters created via a specialized sketch-based modeling interface [BJD*12, DSC*20], whereas others can take arbitrary shapes as input and produce a skeleton [XZKS19], potentially with skinning weights [XZK*20]. These methods can sometimes fail to predict usable rigs, and they would produce different rigs for each step in an interpolation sequence. Our method produces a single rig that can animate all intermediate characters over an interpolation sequence. Alternatively, one may opt not to use a skeleton to pose an intermediate character. This would require posing one of the two input characters and using techniques such as [LMHM18, HRE*08, VCH*21, ZYD*21] to transfer the pose to the other character before blending the two characters. However, this offers no direct rigging control over the intermediate character.

Tree Correspondence. CharacterMixer computes a hierarchical correspondence between two rig skeletons. This is related to prior work on computing hierarchical correspondences between 3D shapes [ZYL*17]. One work in this space uses these correspondences to "interpolate" between shapes [ALX*14], though they focus on manufactured objects and produce transitions that involve discrete structural switches; we instead focus on continuous blends.

Part-based character blending. One way to produce transitions or blends between two 3D shapes is by gradually swapping their constituent parts. Modeling-by-assembly could be used to do this, albeit with considerable user interaction [FKS*04, KJS07]. Some prior work can produce such transitions automatically [JTRS12]. There also exists a system which can mix and match parts from rigged character models such that the resulting chimera is also rigged [NPC*22]. Our goal is to produce a qualitatively different kind of interpolation between characters: a continuous morph from one to the other.
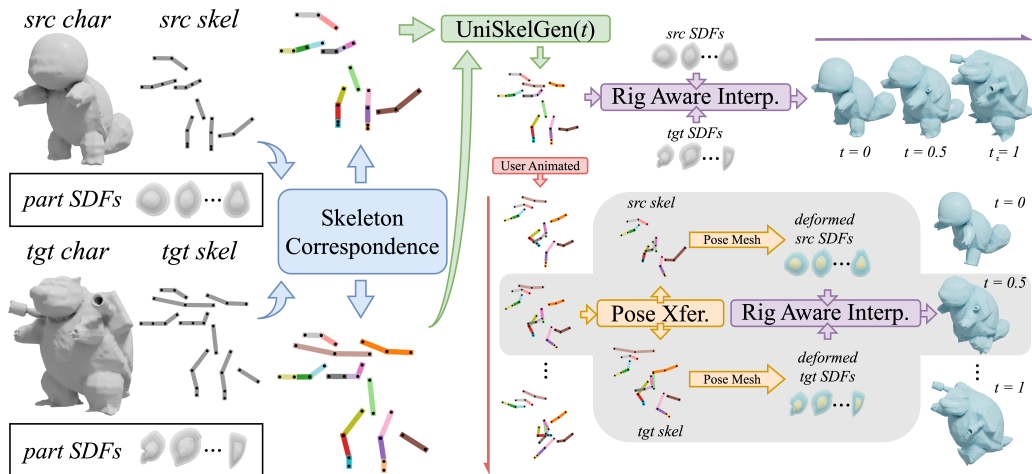
Figure 2: *The overall pipeline of CharacterMixer. Given a source and target character (represented as surface meshes + skeletal rigs), CharacterMixer uses rig skinning weights to segment each character's geometry into a set of **parts**. It also computes a correspondence between the source and target skeletons, which it uses to create a single **unified skeleton** given a time step. This unified skeleton is used to guide interpolation between the geometries of corresponding parts. Given a posed unified skeleton, CharacterMixer transfers the pose to source and target characters and interpolates the deformed geometries of the two posed characters. CharacterMixer enables interpolation during animation, where poses change (red arrow) concurrently with interpolation time steps (purple arrow) as shown in the bottom right.*

3. Approach

Since prior shape interpolation methods [SdGP*15, ENK*21] ignore the underlying structure of shapes, the intermediate results cannot be manipulated. When the shapes are 3D characters, the interpolated characters are not posable. CharacterMixer resolves this issue by rig-aware interpolation; it maintains a rig throughout the interpolation process. Users can pose an intermediate rig to animate an interpolated character. Furthermore, while the geometry of the unified rig changes with varying interpolation time steps, its topology remains the same. Thus, characters can be interpolated for an animation sequence of a single unified skeleton, such that intermediate characters' poses and identities vary at the same time.

Fig. 2 illustrates CharacterMixer's pipeline. Given a pair of source and target characters in rest-pose and their rigs, CharacterMixer produces animatable interpolated characters. CharacterMixer first finds hierarchical correspondences between the skeletons of the two characters using recursively defined cost functions (Section 4). It then creates a *unified skeleton* using the corresponding pairs (Section 5). The unified skeleton serves as a proxy that guides geometry interpolation between topologically-different characters and supports user interaction. Users may animate the unified skeleton, and CharacterMixer transfers the poses to source and target skeletons by propagating bone transformations (Section 6). If there is no pose input, the source and target characters remain in their rest poses. Lastly, CharacterMixer generates geometry for each bone in the unified skeleton by interpolating between the corresponding character part SDFs (Section 7).

4. Skeleton Correspondence

Character interpolation should preserve semantics of body parts: legs should be interpolated with legs, and arms should be in-

terpolated with arms, etc. This calls for a method to find corresponding body parts. While existing methods find surface correspondence using functional maps [OBCS*12] or neural networks [ETLC20, ENK*21], their interpolation ignores rigs such that intermediate results are not posable. In contrast, we seek to maintain a rig throughout the interpolation process. Thus, CharacterMixer finds hierarchical correspondence between input *skeletons*. As part of preprocessing, CharacterMixer segments rest-pose characters' meshes into surface patches representing body parts using skinning weights, where we assign each vertex to the bone that has the highest skinning weight for that vertex. If two bones are corresponded, their segmented body parts are also corresponded. We then convert segmented surface patches of characters in rest-pose to SDFs, which is further discussed in Section 7.

The process of finding bone correspondences is identified as the "Skeleton Correspondence" module in Figure. 2. A bone within a skeleton hierarchy is defined as $(\mathbf{h}_i, l_i, \mathbf{b}_i, M(x_i, y_i, z_i))$, where $\mathbf{h}_i$ is the world position of the head of the bone, l_i is the bone length, $\mathbf{b}_i$ is the tightest axis-aligned bounding box around the part surface geometry in bone local space, and M transforms from bone local space to world space given by the bone's x, y, z axes. Note that the y-axis of bone local space is along the direction of the bone from head to tail, and the x, z axes are computed such that the rotation matrix from the y-axis satisfies the damped track constraint [Ble23]. $\mathbf{b}_i$'s y-axis is aligned along the bone's y-axis. The need for $\mathbf{b}_i$ is explained in Section 7.1. We define source skeleton as bones $\mathbb{S} = \{s_i\}$ and target skeleton as $\mathbb{D} = \{d_j\}$.

CharacterMixer first produces initial bone correspondences, using Xu et al. [XLY*22]'s hierarchical correspondence algorithm with our custom heuristics suitable for 3D bone matching. The algorithm outputs 1-to-1 and 1-to-void pairs, where one bone can
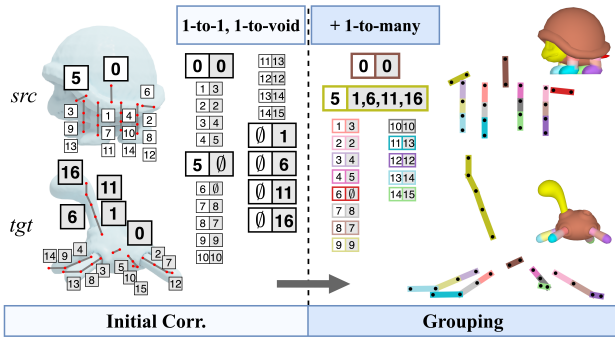
Figure 3: *Our pipeline for computing correspondences between skeletons and constructing an unified skeleton. Left: All the bones from both source and target skeletons. Middle: The initial correspondence phase which takes the source skeleton (white) and the target skeleton (gray) and produces 1-to-1 and 1-to-void matches, where void is denoted by $\emptyset$. Right: Grouping 1-to-void pairs to create 1-to-many correspondences.*

match with another bone or none. Having only these two types of pairs is undesirable for 3D skeleton correspondence, as some matching body parts may have different numbers of bones. CharacterMixer addresses this issue by grouping as many 1-to-void correspondences as possible into *1-to-many correspondences*, where one bone is matched to multiple bones. In this way, semantically matching body parts with different numbers of bones can be corresponded correctly. To illustrate, in Fig. 3, the heads of source and target shapes are correctly corresponded after grouping.

4.1. Producing Initial Skeleton Correspondences

CharacterMixer finds bone correspondences between two input skeletons to establish part correspondences, and it produces an intermediate character by interpolating geometries of two corresponding parts. We adapt Xu et al. [XLY*22]'s hierarchical correspondence algorithm to find 1-to-1 and 1-to-void bone mappings between two input hierarchies. Note that a bone in the skeleton hierarchy can be either a leaf bone or a branch bone. There are five correspondence cases: leaf-to-leaf, leaf-to-void, branch-to-void, branch-to-leaf, and branch-to-branch. Xu et al.'s algorithm defines cost functions for leaf-to-leaf and leaf-to-void correspondences, and the costs for the latter three cases are recursively computed from the first two. A matrix encoding the cost of matching any source bone to any target bone is then constructed, and the Hungarian algorithm is used to solve for optimal 1-to-1 and 1-to-void correspondences [Kuh55]. The cost function heuristics used in Xu et al.'s algorithm are designed for 2D layouts, so we have developed custom heuristics suitable for 3D skeletons. In the supplemental material, we enumerate our heuristics and perform an ablation study that validates them.

4.2. Post-Processing Initial Skeleton Correspondences

Source and target characters may have semantically corresponding body parts with different numbers of bones. Fig. 3 shows a

source character having a head with one bone (5) and a target character having a head with four bones (1, 6, 11, 16). When interpolating, since the head bones have 1-to-void correspondence, the source head would shrink and the target head would grow instead of interpolation. This is undesirable, so we post-process the initial pairings by introducing 1-to-many correspondences. A source set and a target set of 1-to-void correspondences can be grouped into 1-to-many correspondence if the lowest common ancestor of the source nodes is in 1-to-1 correspondence with the lowest common ancestor of the target nodes. In Fig. 3, the four target bones (1, 6, 11, 16) are grouped together to correspond to the one source bone (5) because all bones satisfy 1-to-void correspondence and the source ancestor (0) is 1-to-1 corresponded with the target ancestor (0). In the supplemental material, we provide detailed pseudocode for an algorithm that eliminates as many 1-to-void correspondences as possible in favor of 1-to-1 and 1-to-many mappings. We define correspondence as $P = (r_s, r_d)$, where P has five cases: $(s,d), (S,d), (s,D), (s,\emptyset), (\emptyset,d), S \subset \mathbb{S}, D \subset \mathbb{D}, s \in \mathbb{S}, d \in \mathbb{D}$.

5. Unified Skeleton Generation

We seek to preserve a rig throughout the interpolation process such that the intermediate characters are posable. Given 1-to-1, 1-to-many, 1-to-void skeleton correspondences (Section 4), CharacterMixer generates a *unified skeleton* at a time step t (Fig. 4). This process is identified as "UniSkelGen" in Fig. 2. The geometries of unified skeletons vary depending on t, but their topologies remain the same. Thus, animators can interact with a single unified skeleton and specify interpolation steps for each frame to generate an animation sequence where motion and interpolation happen simultaneously (Figs. 1, 9). A unified bone inherits the properties of a regular skeleton bone, and it additionally references source and target bones, a source bone and void, or a target bone and void. Define reference $R = (r_s, r_d)$, where R is one of $(s,d), (s,\emptyset), (\emptyset,d), s \in \mathbb{S}, d \in \mathbb{D}$. A unified bone also carries t, denoting the interpolation time step. Thus, a unified bone is defined as $(\mathbf{h}_k, l_k, \mathbf{b}_k, R_k, t), k \in \mathbb{K}$, where $\mathbb{K}$ is the set of all bones in an unified skeleton. In this section, we will first discuss the three types of unified bones then explain how to construct a unified bone. We define three types of unified bones displayed in Fig. 4:

- **Constrained**. Reference $R = (s,d)$, where s,d satisfy 1-to-1 correspondence $P = (s,d)$.
- **Loose**. Reference $R = (s,d)$, where s,d are associated via 1-to-many correspondence $P = (S,d)$ or (s,D), $s \in S, d \in D$.
- **Virtual**. Reference $R = (s,\emptyset)$ or $(\emptyset,d)$, where s,d satisfy 1-to-void correspondence $P = (s,\emptyset)$ or $(\emptyset,d)$.

Given a 1-to-1 pair $P = (s,d)$, CharacterMixer constructs a **constrained** unified bone (Fig. 4, Constrained). Since we have 1-to-1 mapping, it's straightforward to make an intermediate bone; we linearly interpolate each attribute of s,d. $\mathbf{h}_k = \text{lerp}(\mathbf{h}_s, \mathbf{h}_d, t)$, $l_k = \text{lerp}(l_s, l_d, t)$, $\mathbf{b}_k = \text{lerp}_{\text{box}}(\mathbf{b}_s, \mathbf{b}_d, t)$, and $M_k = \text{slerp}(M_s, M_d, t)$, where lerp_{box} linearly interpolates the eight corners of two bounding boxes. The constrained unified bone references s and d.

Given a 1-to-many correspondence pair $P = (S,d)$, CharacterMixer constructs $|S|$ number of **loose** unified bones (Fig. 4, Loose). CharacterMixer first splits up d into $|S|$ parts whose lengths are
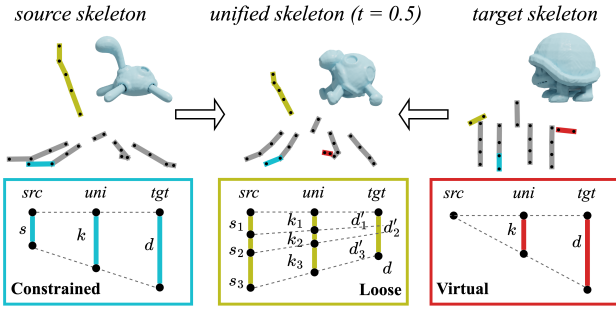
Figure 4: *Generating unified bones at t = 0.5. The blue claw is a constrained bone made from 1-to-1 correspondence. Each yellow head bone is a loose bone made from 1-to-many correspondence. The red tail is a virtual bone made from 1-to-void correspondence.*

proportional to $l_{s_i}, s_i \in S$. $\mathbf{b}_{k_i}$ is generated similarly by splitting $\mathbf{b}_d$ into parts proportional to the geometries of $\mathbf{b}_{s_i}$ (Section 7.2). We have now converted the problem into 1-to-1 interpolation. Each k_i can then be constructed in the same way as a constrained bone. For example, in Fig. 4, k_2 is linearly interpolated between s_2 and d'_2. The loose unified bone k_i references s_i and d, where $s_i \in S$.

Given a 1-to-void correspondence pair $P = (\emptyset, d)$, CharacterMixer constructs a **virtual** unified bone (Fig. 4, Virtual). CharacterMixer uses *bounding box mapping* to map $\mathbf{h}_d$ to the source character, where the details of bounding box mapping are explained in Section 7.1. Note that the center of the bounding box sits at the head of the root bone, and the axes of the box are aligned with those of the root bone. Then, CharacterMixer linearly interpolates $\mathbf{h}_d$ and the projected point to acquire $\mathbf{h}_k$. The virtual unified bone k's length and bounding box is computed by $l_k = \text{lerp}(0, l_d, t)$, and $\mathbf{b}_k = \text{lerp}_{\text{box}}(\mathbf{b}_0, \mathbf{b}_d, t)$, where $\mathbf{b}_0$ denotes a bounding box whose height, width, and length are all 0. The virtual unified bone references $\emptyset$ and d. The supplemental material contains more detailed pseudocode for the unified skeleton construction process.

6. Pose Transferring

Users can interact with the unified skeletons to pose or animate interpolated characters. CharacterMixer also produces animation sequences where interpolation happens simultaneously as shown in Fig. 1. Given animation input for a unified skeleton at t, CharacterMixer transfers poses to source and target characters' skeletons and blends their geometries to generate a posed or animated interpolated character at interpolation time t. Note that by construction, even though the geometry of the unified skeleton varies depending on time step t, its topology remains the same throughout the interpolation. Thus, users can animate any unified skeleton and also vary the interpolate time step such that characters are interpolated during an animation sequence. In this section, we will discuss how pose is transferred from a unified skeleton to source and destination skeletons. This process is identified as "Pose Xfer." in Fig. 2.

CharacterMixer transfers the unified skeleton's pose to source and target skeletons by propagating bone transformations. We will explain how CharacterMixer transfers poses from constrained, loose, and virtual unified bones.

A **constrained** unified bone k references one source bone s and one target bone d, where s and d have 1-to-1 correspondence. In this case, the source and target nodes should transform in the same way as the unified bone. Given user input joint angles for the unified bone, CharacterMixer computes a local rotation matrix Rot_k and sets $Rot_s = Rot_d = Rot_k$.

Loose unified bones are created from 1-to-many correspondence $P = (S, d)$, and each unified bone k_i references $R_i = (s_i, d), s_i \in S$. When users pose k_i, the behavior of s_i should be the same; $Rot_{s_i} = Rot_{k_i}$. To rotate bone d, one way would be to compute the vector from the head of the S linkage to its tail and rotate bone d to align with that vector. However, this would ignore the rotation around the axis along bone d. Thus, CharacterMixer averages the joint angles of $\{k_i\}$ to construct rotation matrix Rot_d.

A **virtual** unified bone k references $R = (s, \emptyset)$ or $R = (\emptyset, d)$. If $R = (s, \emptyset)$, rotating k only affects s and has no impact on the target skeleton. Thus, CharacterMixer sets $Rot_s = Rot_k$ or $Rot_d = Rot_k$.

7. Rig-Aware Geometry Interpolation

We have discussed how CharacterMixer computes skeleton correspondence and constructs a unified skeleton. It is established that bone correspondence implies part correspondence (Section 4), and each unified bone refers to a source and/or a target bone in either 1-to-1, 1-to-many, or 1-to-void correspondence (Section 5). The unified skeleton allows for rig-aware character interpolation; a rig is maintained throughout interpolation, and CharacterMixer builds geometries around the unified skeleton. Given user pose or animation input for a unified skeleton, CharacterMixer transfers the pose to source and target characters' skeletons and interpolates their posed geometries. If there is no pose input, CharacterMixer interpolates their rest-pose geometries. Note that although the geometries of the unified skeletons vary depending on the input interpolation time step t, their topologies remain the same. Thus, users can pose a single unified skeleton generated at an arbitrary t and pass different interpolation steps to CharacterMixer's rig-aware geometry interpolation algorithm for each frame of animation. CharacterMixer then produces an animation sequence where interpolation and motion happen at the same time (Fig. 1 bottom row). In Fig. 2, a user animates the unified skeleton generated at $t = 0.5$ (center) and passes different interpolation steps for each animated frame to CharacterMixer and produces a run cycle sequence where the source character is interpolated to the target character.

Since interpolation of source and target geometries should preserve part-level semantics (e.g. legs should be interpolated with legs), CharacterMixer employs part-based representation of characters, and CharacterMixer uses each unified bone's reference information R to identify and interpolate source and target parts. We choose signed distance fields (SDFs) to represent characters' body parts, as SDFs allow easy interpolation between geometries with different mesh topologies. Furthermore, CharacterMixer's geometry interpolation is aware of the rig state, which allows for interpolating character geometries during animation,; CharacterMixer keeps track of bone local frames and interpolates parts in bone local spaces where SDFs are defined. To obtain an SDF for a segmented body part, CharacterMixer converts the part to voxels on a 128^3
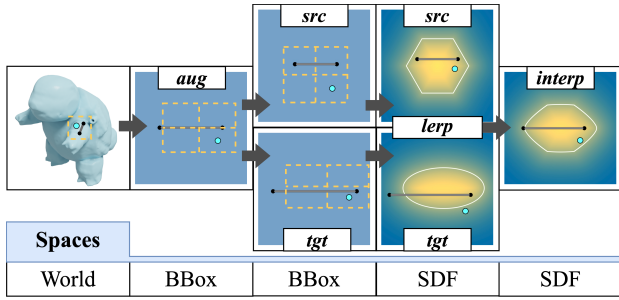
Figure 5: *Interpolating part geometries for a constrained unified bone. With part-based SDFs, interpolating between two characters amounts to taking a query point (cyan dot) within the bounding box of a unified skeleton bone in the bone's local space, finding which point in the source and target boxes that it maps to, looking up SDF values at those points, and linearly interpolating between them.*
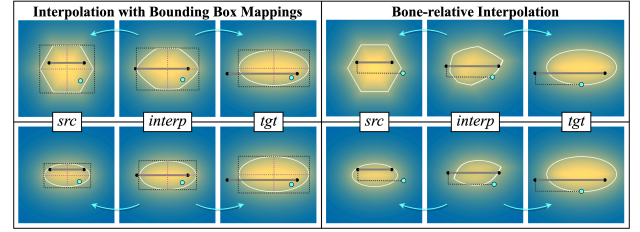

Figure 6: *Comparison between interpolation using bounding box mapping and bone-relative mapping with 2D SDF visualization, where white outlines geometries and grey denotes bones. The interpolated geometry ("interp") is generated by mapping a point (cyan dot) in the unified bone's space to source and target spaces and then interpolating the queried SDFs. Interpolation with bounding box mapping preserves characteristics of input geometries while the other method does not.*

regular grid then to SDFs using distance transform. When interpolating part SDFs, CharacterMixer uses *bounding box mapping* to ensure that the interpolated geometry preserves characteristics from both parts. Fig. 5 shows how CharacterMixer acquires SDF values for a blended part by mapping a query point to the source and target spaces then interpolating. A query point is evaluated by each unified bone $k \in \mathbb{K}$ to produce SDF value v_k; CharacterMixer defines the final SDF value for that point as $V_k = \min\{v_k\}$, which can be interpreted as a union operation. In the following subsections, we explain bounding box mapping, discuss shape interpolation for different types of unified bones, and show how to query deformed SDFs given character poses.

7.1. Bounding Box Mapping

When interpolating between corresponding parts, the intermediate result should preserve characteristics from both parts. Character-Mixer achieves this by re-localizing query points with bounding box centers and scaling them when transforming to source and target spaces. Fig. 6 shows two methods for part interpolation. If we map a query point p in the unified bone space to source and target spaces relative to its position to the bone heads, interpolating a hexagon and an ellipse results in an unexpected shape. In contrast, *bounding box mapping* outputs a rounded hexagon. CharacterMixer first constructs bounding boxes B_s, B_d around source and target part mesh geometries and interpolate to obtain an intermediate bounding box $B_k = \text{lerp}_{\text{box}}(\mathbf{b}_s, \mathbf{b}_d, t)$ for a constrained or loose unified bone k. Then CharacterMixer finds p's location in the intermediate bounding box before scaling it to source and target bounding box spaces. Let $\mathbf{b}^c$ denotes the center of $\mathbf{b}$, and $\mathbf{b}^x, \mathbf{b}^y, \mathbf{b}^z$ denotes the x, y, z axes lengths of $\mathbf{b}$. Point p in the source bounding box space is given by $(p - \mathbf{b}_k^c) \cdot \left(\frac{\mathbf{b}_s^x}{\mathbf{b}_k^x}, \frac{\mathbf{b}_s^y}{\mathbf{b}_k^y}, \frac{\mathbf{b}_s^z}{\mathbf{b}_k^z}\right)$. Note that a query point may land outside of the interpolated bounding box, but the transformation still applies as bounding box geometries provide scaling.

7.2. Interpolation with Unified Bones

CharacterMixer uses a unified bone k's reference information R to interpolate between source and destination part geometries.

It's simple to compute geometry for a **constrained** unified bone k which references one source bone s and one target bone d. Fig. 5 shows how to obtain an interpolated SDF value given a query point p in world space. CharacterMixer first transforms p into the interpolated bounding box $\mathbf{b}_k = \text{lerp}_{\text{box}}(\mathbf{b}_s, \mathbf{b}_d, t)$. Using bounding box mapping, CharacterMixer finds the position of p in $\mathbf{b}_s$, $\mathbf{b}_d$. SDF values of source and target parts are defined in bone local space, so CharacterMixer transforms p from bounding box space to bone local space to query v_s, v_d, where the point is translated by $\mathbf{b}^c$, the bounding box center. The final SDF value is given by $v_k = \text{lerp}(v_s, v_d, t)$.

When source and target bones are in 1-to-many correspondence $P = (S, d)$, CharacterMixer constructs $|S|$ number of **loose** unified bones k_i, each referencing $R_i = (s_i, d), s_i \in S$, where each s_i has bounding box $\mathbf{b}_{si}$. Similar to how CharacterMixer proportionally splits up bone d when generating k_i, CharacterMixer splits d's bounding box $\mathbf{b}_d$ into sub-boxes $\mathbf{b}'_{d_i}$ whose y-axis lengths are proportional to each of $\mathbf{b}_{s_i}$. In this way, an interpolated bounding box can be generated for each loose unified bone k_i, where $\mathbf{b}_{k_i} = \text{lerp}_{\text{box}}(\mathbf{b}_{s_i}, \mathbf{b}'_{d_i}, t)$. CharacterMixer then proceeds to interpolate geometries as described in the previous paragraph.

A **virtual** unified bone k references $(s, \emptyset)$ or $(\emptyset, d)$. Interpolation for virtual bones works similarly to constrained bones. When either the source or destination bounding box is $\mathbf{b}_0$, CharacterMixer doesn't attempt to transform point p into $\mathbf{b}_0$ but sets $v_s = 0$ or $v_d = 0$ then interpolates for v_k.

7.3. Querying Deformed SDFs

Given a posed unified skeleton and an interpolation time step, CharacterMixer queries deformed SDFs of source and target characters to generate an interpolated character. After transferring the unified skeleton's pose to source and target skeletons, Character-Mixer poses the two input characters using their respective skinning weights. CharacterMixer then segments the characters into parts using the procedure described in the beginning of Section 7, converts them to SDFs, and uses rig-aware interpolation (Section 7.2) to generate geometry for the posed unified skeleton.
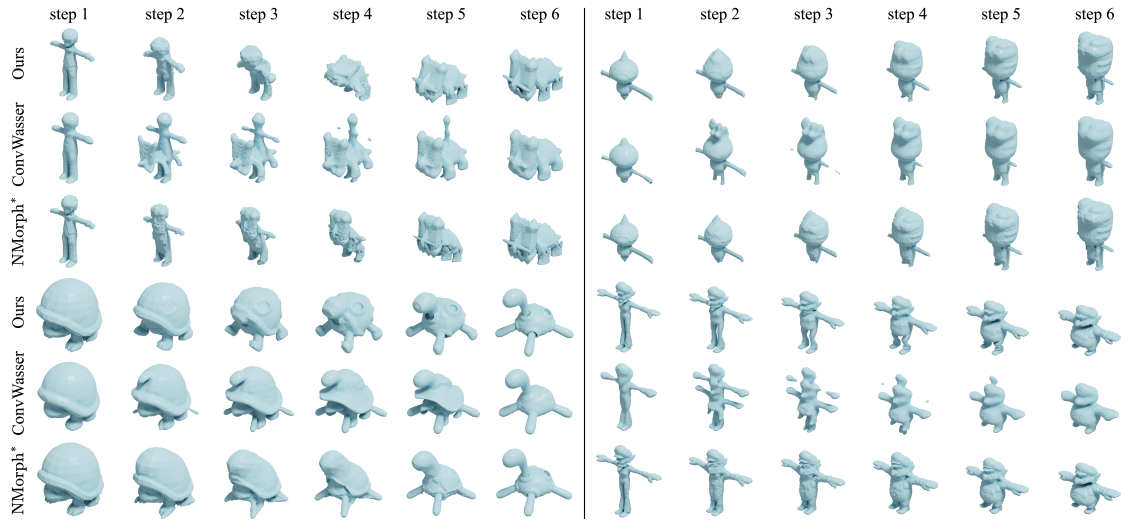
Figure 7: *Qualitative comparisons between CharacterMixer, ConvWasser [SdGP*15], and NeuroMorph [ENK*21] ("NMorph*"). The "NMorph*" rows are produced using the method in Section 8. CharacterMixer smoothly interpolates characters while the others do not. ConvWasser is agnostic to correspondences, and NeuroMorph produces many artifacts (arms at steps 3, 4 of top left; head and paws at steps 3, 4 of bottom left; legs at steps 2, 3, 4 of top right; arms and head at steps 2, 3, 4, 5 of bottom right). In the supplemental document, we show two sets of NeuroMorph interpolations generated from both directions for each pair. See supplementary video for animated comparisons.*
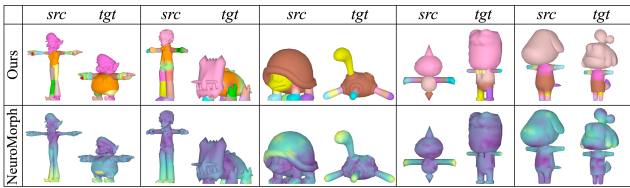

Figure 8: *Qualitatively comparing our correspondences with NeuroMorph. We visualize CharacterMixer's skeleton correspondence by segmenting meshes using skinning weights and assigning different colors to surface patches associated with corresponding bones. NeuroMorph colors denote vertex-level correspondence.*

8. Results and Evaluation

In this section, we present results of interpolating a variety of characters with CharacterMixer. We compare CharacterMixer with ConvWasser [SdGP*15], a rig-oblivious method for shape interpolation with optimal transport, and NeuroMorph [ENK*21], a learning-based approach that produces surface correspondence and interpolation given two input meshes. We trained each character pair with NeuroMorph for 5000 epochs to obtain results. For our experiments, we used 36 character pairs from the RigNet dataset [XZK*20], sourced from Models Resource [VG 19], a publicly-available dataset of rigged video game characters. Experiments were run on 6-core Intel i7 machine with 32GB RAM and a NVIDIA GTX 1080Ti GPU. We strongly encourage readers to watch our supplementary video for animated results.

Fig. 8 compares CharacterMixer and NeuroMorph correspondences. Leveraging characters' rigs, CharacterMixer produces more fine-grained correspondences and correctly identifies corre-

sponding body parts. For instance, in Fig. 8 column 3, we match the two heads of the character pair, while NeuroMorph matches the shell of the first character to the head of the second character. See the supplemental material for more of our correspondences.

Fig. 7 compares how well CharacterMixer interpolates rest-pose characters compared to ConvWasser and NeuroMorph*. Neuro-Morph changes the pose of one shape to match the other while leaving its identity unchanged. We produced NeuroMorph's interpolation for both directions from source s to target d and d to s to obtain $s_{t=0}, ... s_{t=1}$ and $d_{t=0}, ... d_{t=1}$. Intermediate results, labeled NeuroMorph*, are computed by $\text{lerp}(\text{SDF}(s_t), \text{SDF}(d_{1-t}), t)$ where $\text{SDF}(\cdot)$ converts a mesh to SDF. Our approach produces higher-quality and semantic-preserving interpolations. For example, in the bottom right of Fig. 7, our approach smoothly interpolates the heads, while ConvWasser ignores correspondence and NeuroMorph* has many artifacts around the intermediate characters' head and arms. Furthermore, our method allows intermediate characters to be posed, while ConvWasser and NeuroMorph do not.

Fig. 9 shows interpolation during animation. For each sequence, an artist has animated the unified skeleton generated at $t = 0.5$ (Section 5) and specified interpolation time steps for each frame. CharacterMixer then transfers the poses to source and target characters (Section 6) and performs rig-aware interpolation (Section 7). Since CharacterMixer maintains a rig with the same topology throughout interpolation, it allows for interpolation while the intermediate characters perform a motion sequence.

In terms of timing, CharacterMixer produces a posed interpolated character in 83 seconds on average. We experimented with another approach to generate geometry for a posed intermediate character that is 17% faster with negligible cost in quality (see the supplemental material).
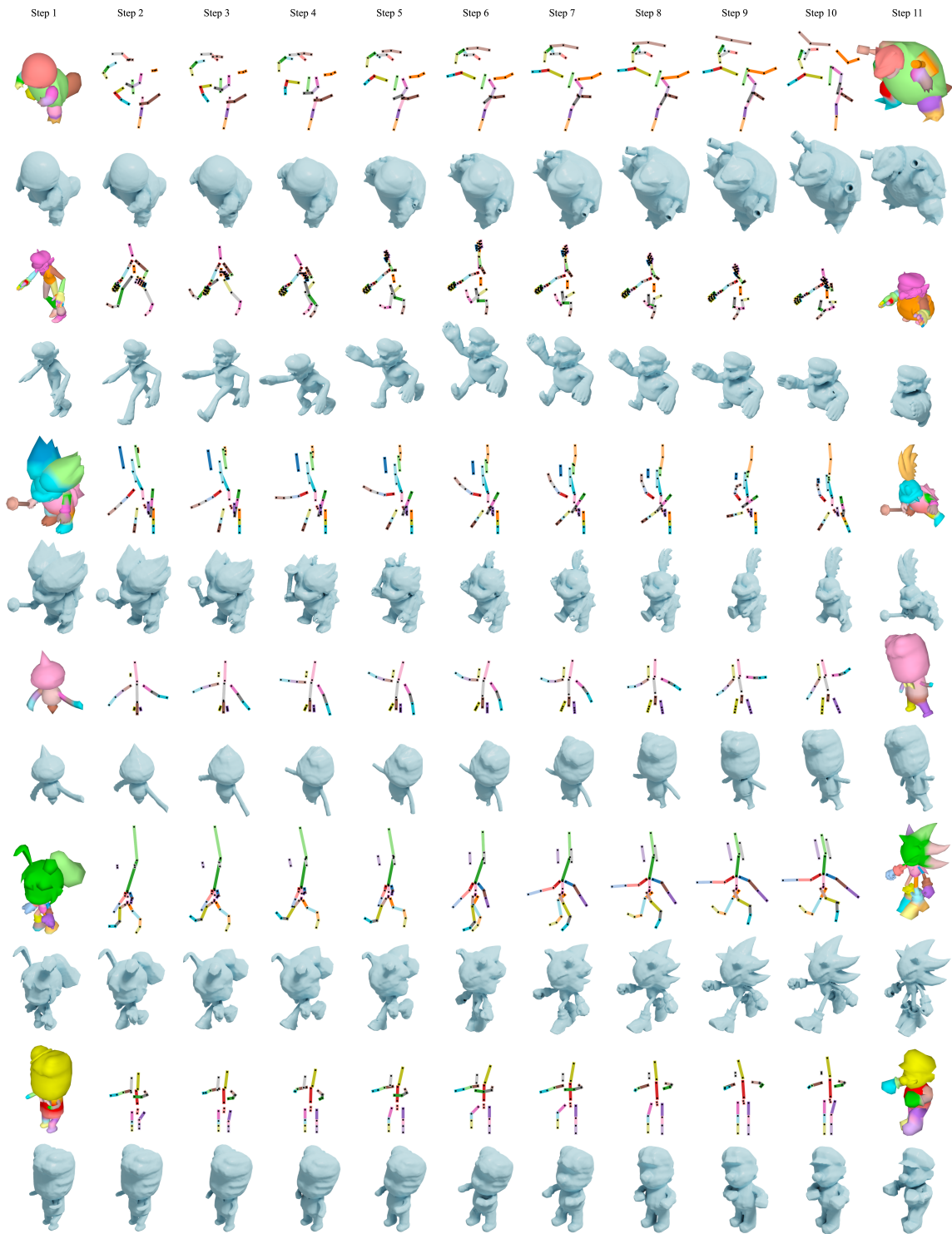
X. Zhan, R. Fu, D. Ritchie / CharacterMixer: Rig-Aware Interpolation of 3D Characters

Figure 9: *Interpolation during animation. For each character pair, the top row shows correspondence and posed unified skeletons, and the row below shows a motion sequence where character poses and interpolation steps vary at the same time. During interpolation, Character-Mixer preserves a "unified skeleton" whose geometry varies depending on time steps and topology remains the same. Thus, animators can interact with a single unified skeleton and specify interpolation steps for each frame to achieve interpolation during animation. Pairs 1, 4 include 1-to-1, 1-to-void, and 1-to-many correspondences, and pairs 2, 3, 5, 6 include 1-to-1 and 1-to-void correspondences. Please watch our supplementary video for animated sequences.*

9. Conclusion

We presented CharacterMixer, a method to interpolate between 3D characters with different mesh and rig topologies, such that users can pose the intermediate interpolated characters. It also enables interpolation during animation. To the best of our knowledge, CharacterMixer is the first system that tackles this novel challenge. CharacterMixer achieves this goal by maintaining a unified rig throughout interpolation, where the unified rig is built from skeleton correspondences between two input rigs of potentially different topologies. CharacterMixer is agnostic to mesh topologies as it represents characters as a union of sign distance fields, one per each bone of the character's rig. We showed how to perform rig-aware interpolation of characters and pose any intermediate interpolated character. Our experiments show that CharacterMixer produces higher-quality character interpolations than rig-oblivious shape interpolation methods [SdGP*15, ENK*21].

CharacterMixer is not without its limitations. Similar to Xu et al. [XLY*22]'s work, our skeleton correspondence algorithm can sometimes produce incorrect correspondences that may not satisfy users. In this case, an interactive system built on CharacterMixer could simply allow users to manually correct the automatically-produced correspondences. Moreover, CharacterMixer can struggle to produce a good correspondence between characters with drastically different skeletons—indeed, in some cases, a meaningful correspondence might not exist. In the supplemental material, we provide some correspondence failure cases. Although we have shown that NeuroMorph [ENK*21], a mesh-based method, creates severe artifacts and does not support interpolation during animation, CharacterMixer's SDF-based representation has a lack of control over output surfaces such as loss of original mesh topology and texture. In terms of posing intermediate results by propagating poses to source and target characters, the current pose transferring method works well when the local frames of matching bones align well. It is an interesting future direction to implement more dedicated pose retargeting to refine the intermediate characters' motion. Nevertheless, such dedicated techniques only need to deal with local motion retargeting as we have simplified the problem to three categories – constrained, loose, and virtual. Additionally, more work is needed to optimize CharacterMixer for real-time use. To pose an interpolated character, 70% of reconstruction time is spent on converting part-level mesh geometries to SDFs with voxelization and distance transform. The runtime can be improved by employing faster methods to convert from mesh to SDFs, such as fast winding numbers [BDS*18]. Neural SDFs [DNJ20] may further increase the speed of interpolation.

10. Acknowledgements

References

[ACOL00] ALEXA M., COHEN-OR D., LEVIN D.: As-rigid-as-possible shape interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 157–164.

[ADMG17] ACHLIOPTAS P., DIAMANTI O., MITLIAGKAS I., GUIBAS L. J.: Learning representations and generative models for 3d point clouds. In *International Conference on Machine Learning* (2017).

[ALX*14] ALHASHIM I., LI H., XU K., CAO J., MA R., ZHANG H.: Topology-varying 3d shape creation via structural blending. *ACM Trans. Graph. 33*, 4 (jul 2014).

[AS21] AYDINLILAR M., SAHILLIOĞLU Y.: Part-based data-driven 3d shape interpolation. *Computer-Aided Design 136* (2021), 103027.

[BDS*18] BARILL G., DICKSON N. G., SCHMIDT R., LEVIN D. I. W., JACOBSON A.: Fast winding numbers for soups and clouds. *ACM Trans. Graph. 37*, 4 (jul 2018).

[BJD*12] BOROSÁN P., JIN M., DECARLO D., GINGOLD Y., NEALEN A.: Rigmesh: Automatic rigging for part-based shape modeling and deformation. *ACM Trans. Graph. 31*, 6 (nov 2012).

[Ble23] BLENDER 3.5 MANUAL: Damped Track Constraint. `https://docs.blender.org/manual/en/latest/animation/constraints/tracking/damped_track.html`, 2023. Accessed: 2023-05-19.

[CG 23] CG MEETUP: How Pixar Created the Background Creatures for the Monster University Movie. `https://www.dailymotion.com/video/x16wrpj`, 2023. Accessed: 2023-01-24.

[CZ19] CHEN Z., ZHANG H.: Learning implicit fields for generative shape modeling. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[DNJ20] DAVIES T., NOWROUZEZAHRAI D., JACOBSON A.: Overfit neural networks as a compact shape representation. *CoRR abs/2009.09808* (2020).

[DSC*20] DVOROŽŇÁK M., SÝKORA D., CURTIS C., CURLESS B., SORKINE-HORNUNG O., SALESIN D.: Monster mash: A single-view approach to casual 3d modeling and animation. *ACM Trans. Graph. 39*, 6 (nov 2020).

[ENK*21] EISENBERGER M., NOVOTNY D., KERCHENBAUM G., LABATUT P., NEVEROVA N., CREMERS D., VEDALDI A.: Neuromorph: Unsupervised shape interpolation and correspondence in one go, 2021.

[ETLC20] EISENBERGER M., TOKER A., LEAL-TAIXÉ L., CREMERS D.: Deep shells: Unsupervised shape correspondence with optimal transport. *CoRR abs/2010.15261* (2020).

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. In *ACM SIGGRAPH 2004 Papers* (2004).

[GCLX16] GAO L., CHEN S.-Y., LAI Y.-K., XIA S.: Data-driven shape interpolation and morphing editing. *Computer Graphics Forum 36* (09 2016).

[GLHH13] GAO L., LAI Y.-K., HUANG Q.-X., HU S.-M.: A data-driven approach to realistic shape morphing. *Computer Graphics Forum 32*, 2pt4 (2013), 449–457.

[GYW*19] GAO L., YANG J., WU T., YUAN Y.-J., FU H., LAI Y.-K., ZHANG H.: Sdm-net: Deep generative network for structured deformable mesh, 2019.

[HRE*08] HECKER C., RAABE B., ENSLOW R. W., DEWEESE J., MAYNARD J., VAN PROOIJEN K.: Real-time motion retargeting to highly varied user-created morphologies. In *ACM SIGGRAPH 2008 Papers* (New York, NY, USA, 2008), SIGGRAPH '08, Association for Computing Machinery.

[JCG20] JANATI H., CUTURI M., GRAMFORT A.: Debiased sinkhorn barycenters, 2020.

[JHTG20] JIANG C., HUANG J., TAGLIASACCHI A., GUIBAS L.: Shapeflow: Learnable deformations among 3d shapes. In *Advances in Neural Information Processing Systems* (2020).

[JTRS12] JAIN A., THORMÄHLEN T., RITSCHEL T., SEIDEL H.-P.: Exploring shape variations by 3d-model decomposition and part-based recombination. *Comput. Graph. Forum 31*, 2pt3 (May 2012), 631–640.

[KJS07] KREAVOY V., JULIUS D., SHEFFER A.: Model composition from interchangeable components. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (USA, 2007), PG '07, IEEE Computer Society, p. 129–138.

[Kuh55] KUHN H. W.: The hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*, 1-2 (1955), 83–97.

[LLHF21] LI R., LI X., HUI K.-H., FU C.-W.: Sp-gan:sphere-guided 3d shape generation and manipulation. *ACM Transactions on Graphics (Proc. SIGGRAPH) 40*, 4 (2021).

[LMHM18] LIU Z., MUCHERINO A., HOYET L., MULTON F.: Surface based motion retargeting by preserving spatial relationship. In *Proceedings of the 11th ACM SIGGRAPH Conference on Motion, Interaction and Games* (New York, NY, USA, 2018), MIG '18, Association for Computing Machinery.

[LMR*15] LOPER M., MAHMOOD N., ROMERO J., PONS-MOLL G., BLACK M. J.: SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia) 34*, 6 (Oct. 2015), 248:1–248:16.

[MCA*22] MURALIKRISHNAN S., CHAUDHURI S., AIGERMAN N., KIM V., FISHER M., MITRA N.: Glass: Geometric latent augmentation for shape spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022).

[MDZ*21] MA P., DU T., ZHANG J. Z., WU K., SPIELBERG A., KATZSCHMANN R. K., MATUSIK W.: Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. *ACM Trans. Graph. 40*, 4 (jul 2021).

[NPC*22] NUVOLI S., PIETRONI N., CIGNONI P., SCATENI R., TARINI M.: Skinmixer: Blending 3d animated models. *ACM Trans. Graph. 41*, 6 (nov 2022).

[OBCS*12] OVSJANIKOV M., BEN-CHEN M., SOLOMON J., BUTSCHER A., GUIBAS L.: Functional maps: A flexible representation of maps between shapes. *ACM Trans. Graph. 31*, 4 (jul 2012).

[SdGP*15] SOLOMON J., DE GOES F., PEYRÉ G., CUTURI M., BUTSCHER A., NGUYEN A., DU T., GUIBAS L.: Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Trans. Graph. 34*, 4 (jul 2015).

[The23] THE BLENDER FOUNDATION: Animation & Rigging / Shape Keys / Introduction. https://docs.blender.org/manual/en/latest/animation/shape_keys/introduction.html, 2023. Accessed: 2023-01-24.

[VCH*21] VILLEGAS R., CEYLAN D., HERTZMANN A., YANG J., SAITO J.: Contact-aware retargeting of skinned motion. *CoRR abs/2109.07431* (2021).

[VG 19] VG RESOURCE: The Models Resource. https://www.models-resource.com/, 2019.

[XLY*22] XU P., LI Y., YANG Z., SHI W., FU H., HUANG H.: Hierarchical layout blending with recursive optimal correspondence. *ACM Trans. Graph. 41*, 6 (nov 2022).

[XZK*20] XU Z., ZHOU Y., KALOGERAKIS E., LANDRETH C., SINGH K.: Rignet: Neural rigging for articulated characters. *ACM Trans. on Graphics 39* (2020).

[XZKS19] XU Z., ZHOU Y., KALOGERAKIS E., SINGH K.: Predicting animation skeletons for 3d articulated models via volumetric nets. In *2019 International Conference on 3D Vision (3DV)* (2019).

[YAK*20] YIFAN W., AIGERMAN N., KIM V. G., CHAUDHURI S., SORKINE-HORNUNG O.: Neural cages for detail-preserving 3d deformations. In *CVPR* (2020).

[YHH*19] YANG G., HUANG X., HAO Z., LIU M.-Y., BELONGIE S., HARIHARAN B.: Pointflow: 3d point cloud generation with continuous normalizing flows, 2019.

[YML*22] YANG J., MO K., LAI Y.-K., GUIBAS L. J., GAO L.: Dsgnet: Learning disentangled structure and geometry for 3d shape generation, 2022.

[ZLWT22] ZHENG X.-Y., LIU Y., WANG P.-S., TONG X.: Sdf-stylegan: Implicit sdf-based stylegan for 3d shape generation. In *Comput. Graph. Forum (SGP)* (2022).

[ZYD*21] ZHU W., YANG Z., DI Z., WU W., WANG Y., LOY C. C.: Mocanet: Motion retargeting in-the-wild via canonicalization networks. *CoRR abs/2112.10082* (2021).

[ZYL*17] ZHU C., YI R., LIRA W., ALHASHIM I., XU K., ZHANG H.: Deformation-driven shape correspondence via shape recognition. *ACM Trans. Graph. 36*, 4 (jul 2017).