SkinMixer: Blending 3D Animated Models

STEFANO NUVOLI, ISTI-CNR, Italy, University of Technology Sydney, Australia, and University of Cagliari, Italy
NICO PIETRONI, University of Technology Sydney, Australia
PAOLO CIGNONI, ISTI-CNR, Italy
RICCARDO SCATENI, University of Cagliari, Italy
MARCO TARINI, University of Milan, Italy

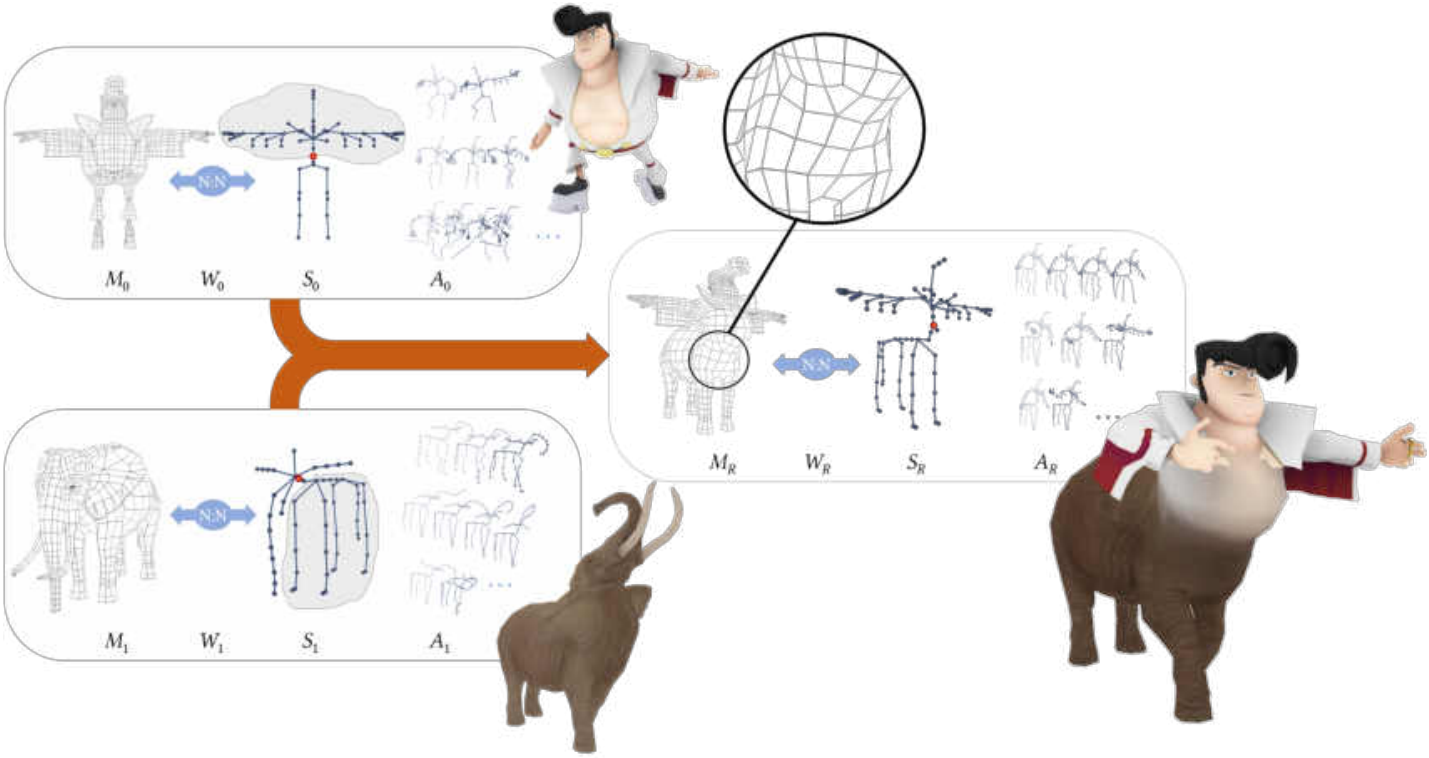

Fig. 1. Our technique in action to mix-and-match animated skinned models. Two production-ready videogame models, comprising of a semi-regular quad-dominant mesh $M_{0,1}$, a set of skinning weights $W_{0,1}$, a skeleton $S_{0,1}$, and a set of keyframe animations $A_{0,1}$, are automatically blended into a fully equipped model comprised of a semi-regular unified meshing M_R, a skeleton S_R, skinning weights W_R, and a new set of compatible animations A_R. A user intuitively specifies this process by simply selecting the portions of the skeleton to be merged (the subtrees in the gray areas rooted at the red dot).

We propose a novel technique to compose new 3D animated models, such as videogame characters, by combining pieces from existing ones. Our method works on production-ready rigged, skinned, and animated 3D models to reassemble new ones. We exploit *mix-and-match* operations on the skeletons to trigger the automatic creation of a new mesh, linked to the new skeleton by a set of skinning weights and complete with a set of animations. The resulting model preserves the quality of the input meshings (which can be quad-dominant and semi-regular), skinning weights (inducing believable deformation), and animations, featuring coherent movements of the new skeleton.

Our method enables content creators to reuse valuable, carefully designed assets by assembling new ready-to-use characters while preserving most of the hand-crafted subtleties of models authored by digital artists. As shown in the accompanying video, it allows for drastically cutting the time needed to obtain the final result.

CCS Concepts: • **Computing methodologies** → **Animation**; **Mesh models**.

Additional Key Words and Phrases: model composition, skinning weights

Authors' addresses: Stefano Nuvoli, ISTI-CNR, Pisa, Italy , University of Technology Sydney, Sydney, Australia , Dept. of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy, stefano.nuvoli@isti.cnr.it; Nico Pietroni, University of Technology Sydney, Sydney, Australia, nico.pietroni@uts.edu.au; Paolo Cignoni, ISTI-CNR, Pisa, Italy, paolo.cignoni@isti.cnr.it; Riccardo Scateni, Dept. of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy, riccardo@unica.it; Marco Tarini, University of Milan, Milan, Italy, marco.tarini@unimi.it.

ACM Reference Format:
Stefano Nuvoli, Nico Pietroni, Paolo Cignoni, Riccardo Scateni, and Marco Tarini. 2022. SkinMixer: Blending 3D Animated Models. *ACM Trans. Graph.* 41, 6, Article 250 (November 2022), 15 pages. https://doi.org/10.1145/3550454.3555503

1 INTRODUCTION

Applications such as 3D Videogames or Virtual Reality require using a variety of 3D assets to recreate vibrant, realistic, reactive 3D virtual worlds. Producing these 3D assets beforehand is one of the main challenges for the developers, absorbing a considerable effort by highly specialized digital artists. One particularly work-intensive asset class is *3D animated models*, used for digital characters. They typically consist of skinned, animated, and textured polygonal meshes. A 3D animated model efficiently encodes not only the 3D shape and visual aspect of the digital character but also their behavior, that is, the animations they are capable of and the specific deformations their body will undergo.

Traditionally, authoring this kind of asset requires a cascade of interrelated complex tasks, often performed by different artists, such as shape modeling, texturing, remeshing, rigging, skinning, and animation. The details vary from one production pipeline to another. Still, many techniques are invariably leveraged throughout the phases, including digital sculpting, direct low poly modeling, automatic meshing tools, texture baking, automatic skinning, motion capture, manual keyframe editing, and others. An array of State of the Art tools exist that strive to assist (rarely, even replace) the work required by digital artists in each phase. Despite the advancements in many fields, including the recent adoption of Machine Learning techniques, creating 3D animated models remains a complex, expensive, and time-consuming task requiring specialized artistic skills.

An alternative strategy for content creation consists of mix-and-matching existing ready-made assets to produce new ones, leveraging partial reuse rather than costly authoring from scratch. This strategy is highly appealing whenever the context allows for it. For example, it has been successfully explored with other classes of difficult-to-produce assets, such as low-poly semiregular quadmeshes. Unfortunately, this is not currently possible for animated 3D models due to their nature.

In this work, we propose the first mix-and-match method to produce new 3D animated models from existing ones. Starting from existing animated models, the artist selects portions of ready-made 3D animated models and freely recombines them into a new one. The resulting asset is ready-to-use and inherits, as much as possible, the original pieces' characteristics and qualities, both in terms of shape (geometry and meshing) and behavior (animations, articulation, and deformations).

Examples of intended usages include: assembling chimeras, such as a centaur obtained by joining a human torso with an elephant body, or a mermaid obtained by assembling a woman torso with a fishtail; modifying a body plan by adding extra appendages, such as constructing a four-armed creature stacking different (or multiple replicas of the same) torsos, or adding pairs of wings, or a tail, to a human figure; replacing an existing appendage with a new one, fulfilling the same purpose, from a different character, such as swapping humans legs with dinosaur legs, tucano wings grafted on a cat body, or even a set of human legs with another. See Section 9 for a gallery of actual results.

The generated result is a complete and independent new asset, ready to be used in the downstream application (or further recombined in mixing operations).

1.1 Problem definition and challenges

In our context, an *animated model* consists of a tuple (M, W, S, A):

- a polygonal mesh M, describing the geometry of the rest shape of the object;
- a skinning W, i.e., a set of weighted links between vertices of M and bones of S;
- a skeleton S, composed of a certain number of bones (or joints) interconnected in a hierarchical tree structure;
- a set of animations $A = \{a_1, ..., a_n\}$, each consisting of a temporal sequence of time-stamped keyframes for S.

Our mixing strategy takes as input two tuples (M_0, W_0, S_0, A_0) and (M_1, W_1, S_1, A_1) and recombines selected pieces of them to create a new one (M_R, W_R, S_R, A_R).

A key insight is that none of the components of the tuple can be manipulated in isolation, and each makes sense only as a part of the whole. For example, constructing M_R from M_0 and M_1 alone (e.g., via [Nuvoli et al. 2019]) would only result in a static pose that would need to be rigged, skinned, and animated anew, defying our purpose; neither sets of animations A_0, A_1 nor sets of weights W_0 or W_1, could be trivially transferred to M_R because they are defined only for the respective original skeleton S_0 and S_1. Instead, we design a unified strategy that uses every item of the input tuples to inform the process of constructing the output tuple.

To do this, we must fulfill several requirements and face corresponding challenges.

(1) **Global consistency**: clearly, we need the output tuple to be consistent; for example, W_R must link vertices of M_R to bones of S_R (despite being assembled from pieces referring to a different skeleton and a different mesh);
(2) **Blended shape**: we want M_R to represent a plausible shape that blends the M_0 and M_1 in a natural way;
(3) **Meshing quality**: M_R must feature a meshing that, far from the merging zone, matches the original one of M_0 and M_1; while, near the new conjunctions, it must roughly match the tessellation density, regularity, and other characteristics (see input assumptions below);
(4) **Deformation quality**: in a skinned model, skin deformations are determined by the set of weights W, which are very carefully designed by "skinning" artists. Therefore, these characteristics must be preserved in the output;
(5) **Animation completions**: we cast the problem of constructing the output set of animations A_R as the task of automatically *completing* individual animations originally defined in one or the other input tuple in some appropriate way. The context determines which animations from the original sets must be completed (by default, we complete them all). For example, a newly assembled "centaur" must inherit the gait

animations from the input animated horse model, accompanying each with some appropriate default movement of the human spine and shoulder; or, the added dog tail to a human model must feature some movement during each of the movements of the human. Observe that the animations of the resulting set A_R, once defined over the common skeleton S_R, can be blended or "overlayed" in real-time, as commonly done by the downstream application (e.g., the game engine).

Assumptions on the input. We assume input meshes M_0 and M_1 to feature characteristics typical of the current videogame context: reasonably low resolution (with finer details handled by textures); triangle or quad-dominant structure (but usually not pure quad); notably, we do not assume meshes to be single connected watertight models: open mesh boundaries are common, for example for representing clothes and hair. Similarly, we do not assume meshes to consist of only one single connected component: character models often feature topologically disconnected parts for props like pieces of armors, weapons, decorations, etc. We want our output meshes to preserve all these characteristics whenever they are in the input. Skinning S_0 and S_1 and animations A_0 and A_1 are assumed to be applied with either plain Linear Blend Skinning or Dual Quaternion Skinning [Kavan et al. 2007], which are still the de-facto standards of the industry. A texture and a UV-map typically enrich meshes. While we strive to preserve these components too, to some extent (Sec. 7.2), that is not the focus of our work.

Opportunities. Compared to the case of mixing-and-matching static shapes, our task presents not only unique challenges but also profitable opportunities, which we readily exploit. For example, the skeleton S doubles as an ideal high-level, semantically meaningful abstraction of the shape, which is convenient to let a user *define* the intended mix-and-match operation (Section 4); we use the skinning $W_{0,1}$ to drive the *segmentation* of the meshes into pieces to be assembled (Section 6); we exploit topological and geometrical similarities in $S_{0,1}$ (Section 5) to retarget $W_{0,1}$, producing W_R (Section 7); we take advantage of $W_{0,1}$ to apply to $M_{0,1}$ a preliminary morphing to ease their fusion into M_R (Sec. 4.1).

2 RELATED WORK

To our knowledge, SkinMixer is the first method to produce new high-quality animated assets by compositing existing ones, while extensive literature focused on automatizing or assisting the generation of this kind of asset, specifically: the meshing, the skinning weights, or the animations. The following discussion focuses on the aspects of this literature that are more closely relevant to our proposed solution.

2.1 Mix-and-match of Geometries

Compositing multiple *static* shapes into a new shape is a well-studied topic. Various interactive tools have been designed to enable an artist to mix different meshed surfaces in real-time [Funkhouser et al. 2004; Kreavoy et al. 2007; Schmidt and Singh 2010; Sharf et al. 2006; Yin et al. 2020; Zhang et al. 2010]. The fundamental step of these approaches is computing the fusion of input surfaces and then performing a local re-meshing in correspondence with the intersection area [Bischoff and Kobbelt 2005; Pavic et al. 2010]. The new geometry can be obtained by fusing automatically-derived implicit representations [Singh and Parent 2001], or by merging the volumes enclosed by the triangular surfaces through exact Boolean operations [Campen and Kobbelt 2010; Cherchi et al. 2020; Jacobson et al. 2013], or other, more expressive volumetric operations [Angles et al. 2017]. Additionally, many algorithms allow transporting attributes between different surfaces [Melzi et al. 2020], including skinning weights, but only assuming that a common skeleton is shared.

2.2 Mix-and-match of Semi-regular Meshes

Our objective partially overlaps with the recent works that strive to perform a blend of input semi-regular meshings while preserving their regularity. QuadMixer [Nuvoli et al. 2019] combines semi-regular quadrangulated meshes to preserve the initial tessellation and smoothly blend the edge flow in the intersection regions. In a similar spirit, among commercial software packages, the Modo suite [Visionmongers 2018] provides a tool, called *MeshFusion*, able to combine quad-based mesh representations with boolean operations while partially preserving their original meshing. The main difference is, of course, that we target animated models (which, as discussed, allows us to exploit more information but pose additional challenges). In addition, we improve over [Nuvoli et al. 2019] by allowing for a broader range of input, which is a necessity in our context: differently from QuadMixer, our method can be applied over non-pure-quad, non-watertight, or even locally non-manifold surfaces, and models consisting of different connected components. We are motivated by these characteristics' prevalence in video games' animated assets. Construction of semi-regular meshes from a *single* input model is a highly well-studied topic, and a few recent works focus on the case of meshes that will be animated as blend-shapes [Marcias et al. 2013; Zhou et al. 2018].

2.3 Data-Driven Mesh Synthesis

Data-driven synthesis is a different class of approaches that creates new content from pre-existing assets like ours. These methods first extract the information from a predefined shape database, then translate it into a compact representation that allows recombination and synthesis from recombination and local modifications. Procedural methods generate shapes by applying basic rules or using predefined grammar. The space of possible shapes is spanned by applying different sequences of derivation rules and variating their parameters. These methods offer high-quality and editable results and allow for composition and partial re-utilization of the portion of meshes. There is a long tradition of procedural model generation in computer graphics. Usually, those methods are specialized and contextualized to specific application domains such as urban environments [Müller et al. 2006; Parish and Müller 2001]. Other approaches rely on more sophisticated inverse procedural modeling [Jones et al. 2020; Martinovic and Van Gool 2013; Ritchie et al. 2018]. However, those methods tend to be very specialized in the class of shapes they can generate. Methods based on deep learning offer a more general expressive power [Chaudhuri et al. 2019]. Those methods might use 3D occupancy grids [Wu et al. 2016] or work on

implicit surface representation [Chen and Zhang 2019; Park et al. 2019]. Other works follow the philosophy of mix-and-match using deep learning [Zhu et al. 2018].

2.4 Automatic and Assisted Creation of Skinning Weights

There is a substantial interest in the industry in defining methods for the automatic derivation of skinning weights. Most of the existing 3D modeling tools like Blender or Maya [Autodesk 2019; Community 2018] offer automatic or semi-automatic rigging tools. Mixamo offers semi-automated tools to derive skinning weights for humanoid characters. In academia, the first method to derive automatic rigging of input 3D models using a combination of discrete and continuous optimization is "Pinocchio" [Baran and Popović 2007]. Skinning weights have been automatically derived, among other ways, as minimizers of higher-order surface smoothness functionals [Jacobson et al. 2011] or elastic energy functionals [Kavan and Sorkine 2012], by proximity-based methods [Dionne and de Lasa 2013], or by maximizing the physical plausibility of resulting deformations (as predicted with position based dynamics) [Pan et al. 2018]. A recent trend [Liu et al. 2019; Xu et al. 2020, 2019] uses deep-learning to derive associated skinning weights, often in conjunction with the skeleton.

Despite multiple attempts, deriving proper skinning weights is still perceived as an art. Artists often prefer to customize weights to recreate the desired deformations, traditionally, by direct painting; innovative methods have been designed to assist and ease this task [Bang and Lee 2018; Borosán et al. 2012].

In contrast to all these methods, which do not allow for the preservation of some previously modeled and skinned assets, our work stems from the idea of preserving the original artist's work as much as possible.

2.5 Animation Synthesis

There is a vast literature on methods to support the artist in producing plausible animations. Some recent approaches use deep learning methods to synthesize natural motion or interpolate different animations in real time. These methods are usually specialized in some subcategory of movements such as speech animation [Taylor et al. 2017] or martial arts [Starke et al. 2021]. Other methods based on neural networks are more targeted to generate complex humanoid motions, real-time control, and adaptation to different boundary constraints [Holden et al. 2016]. Older data-driven approaches are based on Gaussian processes [Grochow et al. 2004]. Despite the realism of the produced animations, each method targets a specific skeleton class and its relative motions set. Moreover, as opposed to the presented method, most cited approaches are designed to mix two movements temporally rather than combining different sets of bones with their relative animations.

2.6 Animation Retargeting

Animation retargeting attracted significant attention in Computer Graphics, starting from the proliferation in the 90s of systems for transfer movements from live actors to rigged characters [Gleicher 1998]. In the beginning, these techniques have been employed to transfer the motion between skeletons or interpolate between sparse poses, solving the kinematic constraints to optimize the continuity of the resulting motion [Choi and Ko 2000]. Advanced tools use more sophisticated representations to avoid self-intersections [Molla et al. 2018]. The approach presented in [Hecker et al. 2008], used in the videogame Spore, is fascinating but inherently different from our solution. It uses predefined building blocks like limbs, heads, and torsos to assemble new creatures, while our method can take any part of existing characters and mix them to generate new ones.

However, techniques based solely on the skeleton might fail to transfer surface features effectively. A class of methods tries to bypass this problem by transferring poses directly between surfaces. Usually, these techniques rely on a compact representation of the deformation to be transferred between meshes [Sumner and Popović 2004]. An alternative approach consists in applying forces on the target characters and letting a physics simulation derive the correct deformation on the target shape [Borno et al. 2018].

Data-driven approaches have become more popular with the widespread diffusion of pre-made assets. They learn the semantic correspondences between different shapes to transfer motions plausibly [Boukhayma et al. 2017] or to automatically combine motions from different animations into new ones [Jang et al. 2008]. Data-driven methods are often used to transfer facial expressions [Bouaziz et al. 2013] or to transfer animation in a more general setup [Aberman et al. 2020; Gao et al. 2018]. Recently, deep learning methods for retargeting have produced impressive results [Villegas et al. 2018; Won and Lee 2019]. A recent method also allows us to retarget motions extracted from video to a 3D model [Aberman et al. 2020]. However, they require significant datasets for the initial training phase.

In general, while animation transfer can be a powerful aid to content creators, it cannot be expected to work reliably on drastically different skeletons.

3 METHOD OVERVIEW

As observed, the skeleton serves as an abstract description of the 3D model, which is ideal for this task. Our method starts with a given mix-and-match operation, issued by the user by manipulating the input skeletons S_0 and S_1, with actions such as the substitution or the addition of subtrees (Section 4). The operation defined by the user on the two input skeletons already produces the final output skeleton S_R (see Fig. 2). The subsequent steps blend with the other elements to construct the output tuple.

The first step consists in constructing two weighted mappings $\mathcal{B}_0$ and $\mathcal{B}_1$ from the original bones in S_0 and S_1 to the bones in S_R (Section 5). These mappings are generally neither injective nor surjective. Conceptually, each bone of the input skeletons is mapped to the "semantically equivalent" bone in the output skeleton, even if this equivalence can be only loosely defined and uncertain for many bones, especially far away from the junction bones. The construction of this mapping is informed by the geometric and topological similarity between the input and the output skeletons.

Then, a new mesh M_R is constructed (Section 6 and Fig. 9). This task fuses appropriately selected sub-parts of M_0 and M_1 with a new surface, which smoothly interconnects the two without interrupting their edge-flows. This is done by exploiting the information in

$W_{0,1}$. The interconnecting shape is created using voxelized implicit representations.

Then, a new set of skinning weights W_R is constructed, to link the vertices of M_R to the bones of S_R (Section 7). Thanks to the mappings $\mathcal{B}_0$, this task can be cast as a transfer of vertex attributes of M_0 and M_1, i.e. $\mathcal{B}_0(W_0)$ and $\mathcal{B}_1(W_1)$ (see Fig. 11).

The final assets are not ready for the downstream application until they are completed with behavioral information: we construct a set of animations A_R defined for the new skeleton S_R, by completing individual animations from the either input animations sets A_0 and A_1 (see Fig. 12 and Section 8).

4 DEFINING THE OPERATION

A mix-and-match operation can be intuitively performed by directly working on the skeletons S_0 and S_1 (Fig. 2). In our system, an operation is just a sequence of various attach/detach actions, where we can take a node b_i and all its subtree from a skeleton and graft it onto a node $b_j \in S_1$ (or vice-versa), either adding or replacing the current subtree of b_j. Multiple such actions can be issued in one operation (e.g., when "transplanting" both arms from one character to another). One operation directly defines the final skeleton S_R and automatically triggers the steps to generate the rest of the (M_R, W_R, S_R, A_R) tuple: the meshing, the skinning weights, and the new set of animations.

4.1 Preliminary positioning and deformation

Any blending operation of 3D shapes expects a user-defined rotation, translation, and scaling of the two models to reposition and resize them appropriately with respect to each other, before their fusion.

In our framework, we consider this as just a particular case of a more general preliminary operation, where M_0 or M_1 can be posed freely as a convenient way to customize the resulting blended shape by allowing the user to deform either shape before merging (see for example Fig. 3). However, unless otherwise specified, all our examples use only the default per bone translation, which we automatically set as follows.

As standard, a pose consists of a rotation defined by each joint in $S_{0,1}$, optionally accompanied by an additional translation (thus stretching or shrinking the bones), redefining the ones defined on the rest pose. The default pose for a given operation consists solely of an additional per-bone translation over S_0. When an operation merges a node $n_i \in S_0$ with $n_j \in S_1$, we define a new translation, at node n_i, that brings its spatial position into the position of n_j, in the respective rest poses. Different translations are defined in each node when multiple operations are issued, matching multiple nodes of S_0 into multiple nodes of S_1, such as a "transplant" of numerous limbs.

Then, we smoothly propagate the defined translations over all other nodes of S_0. We do this by applying a sequence of Laplacian smoothing operations over S_0 (considering it a graph) while keeping all the assigned nodes fixed until convergence. When a single node is merged, the smoothing trivially results in the translation assigned initially to that node being propagated unaltered over the entire S_0, and thus in the mesh M_0 being rigidly translated into the appropriate position of M_1; when multiple nodes are matched, the smoothing has the effect of deforming M_0, such as the case shown in Fig. 4, where the hip region of M_0 is widened, and its torso region shortened, improving the subsequent geometric fusion with M_1. We then apply a Laplacian smoothing of the translations.

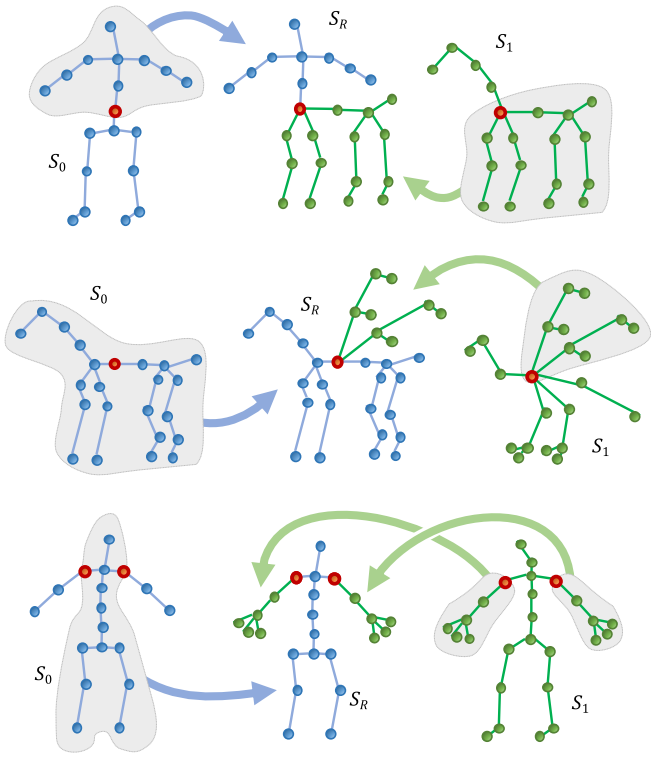

Fig. 2. Examples of the types of operations that can be easily defined on the input skeletons S_0 and S_1, obtaining the output skeleton S_R. Joint nodes are pictured in red. Top: an example of *composition*, where a portion of S_1 is added to a portion of S_0 (e.g., to construct a "centaur" figure assembling a human torso on a quadruped, like in the result visible in Fig. 1). Middle: an *addition* where a portion of S_1 is grafted on S_0 (e.g., to get a winged animal by grafting wings from a bird into the spine of a quadruped, like in Fig. 12). Bottom: a *multiple composition* where more than one portion of S_1 is assembled with a portion of S_0 (e.g., to substitute several limbs from a humanoid character to another, like in Fig. 4).

5 DETERMINING A BONE-TO-BONE MAPPING

A central phase of our method consists in identifying for each bone $b_i \in S_0$ the corresponding bone in the output skeleton $\mathcal{B}_0(b_i) \in S_R$, associated with some scalar confidence value (between 0 and 1), and likewise for S_1. We construct mappings $\mathcal{B}_0$ and $\mathcal{B}_1$ as follows (Fig. 5), and use them in subsequent phases. Fig. 6 shows an example of the resulting mapping.

Initialization. $\mathcal{B}_{0,1}(b_i)$ are initialized, for all trivial cases, with confidence one (Fig. 5, left). Specifically: for an operation prescribing to fuse node $b_i \in S_0$ with $b_j \in S_1$, we pick the node $b_k \in S_R$ resulting from their fusion and prescribe $\mathcal{B}_0(b_i) = \mathcal{B}_1(b_j) = b_k$; for any bone b_j in S_R that was created as a copy of a bone b_i, if $b_i \in S_0$, we set $\mathcal{B}_0(b_i) = b_j$, and if $b_k \in S_1$ we set $\mathcal{B}_1(b_i) = b_j$.

Propagation. We then extend the incomplete $\mathcal{B}_{0,1}$ over all other bones of $S_{0,1}$, using an heuristic based on skeleton similarity, in terms of both topology and geometry (Fig. 5, right). Specifically, in each step, we process one edge in S_0 or S_1, connecting nodes b_i and b'_i, such that b'_i is currently assigned to $\mathcal{B}(b'_i) = b'_j \in S_R$ with some confidence c, and b_i is not assigned yet. We assign it by evaluating all bones in S_R: for each candidate, we estimate its similarity score with b_i, between 0 and 1 (see below), and pick the node b_j with the highest score s. We then define $\mathcal{B}(b_i)$ to be b_j, with confidence $c \cdot s$, and proceed with the next edge, until all nodes in S_0 and S_1 are assigned.

We are aware that the semantic of $\mathcal{B}$ is only lousily defined, especially far from the junction node(s), where the confidence is low. In many cases, no close semantical counterpart even exists for a given node (for example, if a mermaid is obtained as a mix of a maid and a dolphin, there is no "real" correspondent of the bone relative to the left foot). Our method uses $\mathcal{B}$ accordingly.

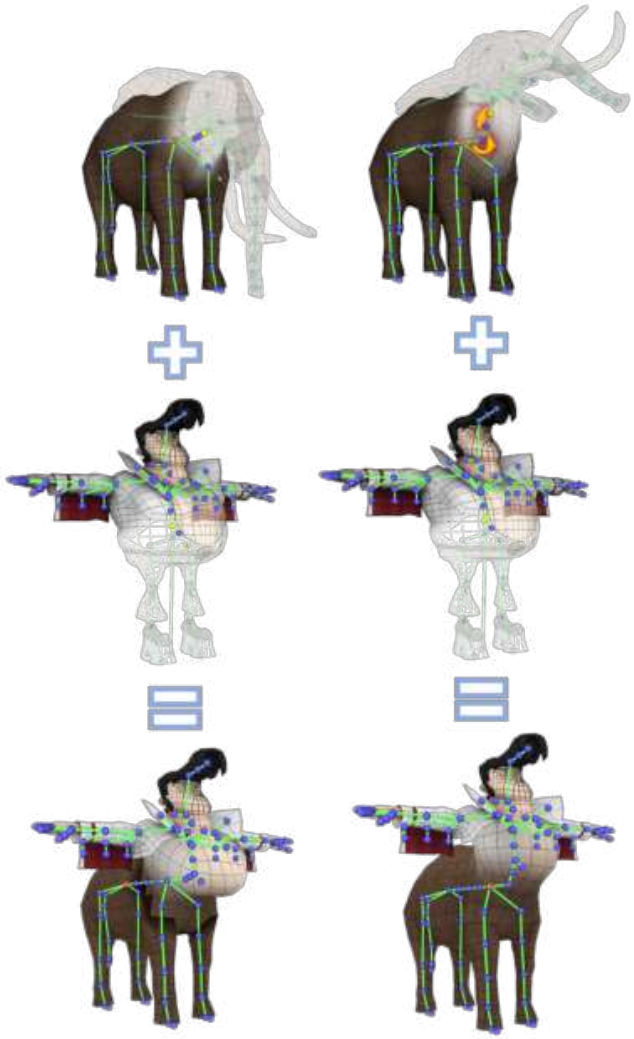

Fig. 3. An example of posing the models as a way to control the blend results. Using the *elephant* model in rest pose (left) results in a bulged abdomen in the final model, while posing the elephant with the head rotated up, and its neck slightly stretched produces a preferable surface (right).

Fig. 4. An example of a composite operation, where both arms and legs are transferred to from M_0 (*"Timmy"* model, top left) to M_1 (*"Mouse"* model, top center). Bottom left: M_0, after a skinning deformation triggered by the default per bone translations, which bring the four connected bones to match the position of the respective receiving bone in S_1; this induces some unwanted deformations in the shoulder region. Bottom center: M_0 after smoothing the per-bone translations, which fixes the issue. Right: M_R.

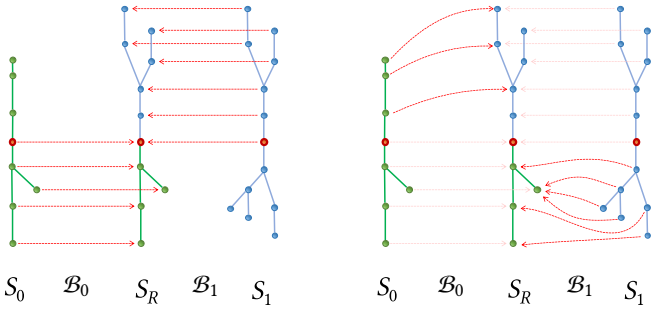

Fig. 5. The constructions of bone-to-bone mappings $\mathcal{B}_0 : S_0 \to S_R$ and $\mathcal{B}_1 : S_1 \to S_R$. Left, after initialization. Right: after all propagation steps. Observe that neither mapping is either injective or surjective, and they are less accurately defined further away from the joining node (in red).

5.1 Bone-to-bone similarity measure

We detail here the similarity measure between a node in S_0 or S_1 and a node in S_R that we use to heuristically build the rough mapping $\mathcal{B}$. We consider each skeleton as a 3D embedded tree, each joint b_i having the 3D position in rest pose $R(b_i)$. Additionally, we define, for each node, a *parametric position* $P(b_i)$, as a scalar value between 0 (for roots) to 1 (for leaves), as follows: for each leaf in the tree, we traverse the path from the root to that leaf, and assign to each node along the path its parametric position in that path (defined as the ratio between the traversed distance up that node, over the length of the path). The parametric position of a node is then the average of its parametric positions for all paths that include it.

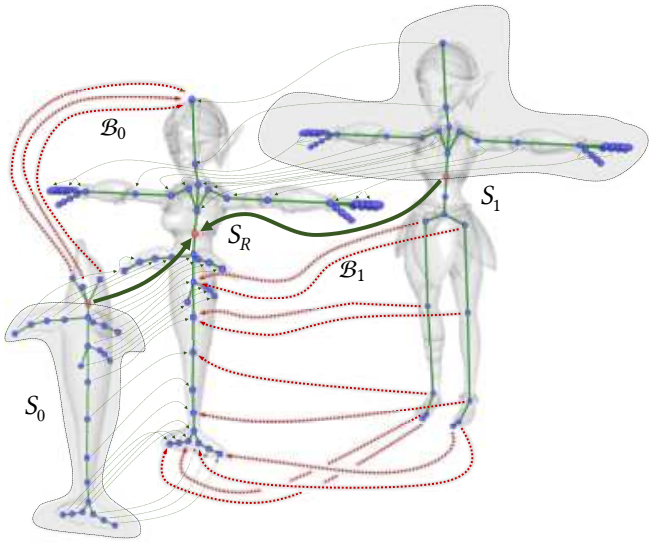

Fig. 6. An example of bone-to-bone mappings $\mathcal{B}_0 : S_0 \to S_R$ and $\mathcal{B}_1 : S_1 \to S_R$. The red node represents the only joint node. In thin green lines are the relationships defined in the initialization phase, in red lines are the ones inferred by our heuristics (Section 5).

The similarity between nodes $b_j \in S_R$ and b_i, which is directly connected to a node b'_i, with $b'_j = \mathcal{B}(b'_i)$ is given by a weighted sum of four criteria. While not all criteria are valid in all cases, their aggregation is sufficiently reliable for our purposes. Specifically, we use the following criteria and weights (determined empirically, but different choices lead to similar results):

a) *geometrical similarity* (weight 0.3), defined as the average between the *local* directional similarity:

$$\frac{R(b_i) - R(b'_i)}{\|R(b_i) - R(b'_i)\|_2} \cdot \frac{R(b_j) - R(b'_j)}{\|R(b_j) - R(b'_j)\|_2}$$

and the *global* directional similarity:

$$\frac{R(b_i) - R(r_i)}{\|R(b_i) - R(r_i)\|_2} \cdot \frac{R(b_j) - R(r_j)}{\|R(b_j) - R(r_j)\|_2}$$

where r_i and r_j are the two roots of the respective tree, and $\cdot$ denotes the dot product;

b) *topological similarity* (weight 0.3), is defined as $1/k$, with k the number of edges separating b_j from b'_j, or 2 if $b_j = b'_j$ (it is maximal when $k = 1$, that is, when b_j is directly connected to b'_j, matching the fact that b_i is directly connected to b'_i);

c) *parametric similarity* (weight 0.3): defined as $|P(b_i) - P(p_j)|$;

d) *valence similarity* (weight 0.1), is the similarity between the number of connections of b_i and b_j, defined as $\min(b_i, b_j)/\max(b_i, b_j)$.

Injectivity bonus. Additionally, we add a "injectivity bonus" to the similarity score of a candidate p_j to favor the construction of injective mappings (without enforcing it). Whenever a candidate p_j has not been selected yet, we add 0.5 to its score and 0.1 if the p_j was already picked, but with confidence < 1.

6 PRODUCING THE NEW SURFACE

The task of producing the unified mesh M_R is central to our pipeline. Our objective is to build a new (rest) shape that plausibly blends the two input shapes while preserving and extending the meshing characteristics (in terms of resolution, regularity, and edge flows).

This task is broken into a sequence of operations:

Automatic vertex selection. First, for each of the two input meshes $M_{0,1}$, vertices are labeled with a "presence" value in [0..1], indicating whether the vertex should appear in the output or not. This labeling is fuzzy, and it is easily initialized by exploiting the information on the respective skeleton $S_{0,1}$ and skinning $W_{0,1}$: the *presence* value for a vertex is the sum of the weights linking it to bones of $S_{0,1}$ that are directly used in S_R. After the assignment, we smooth the presence values over $M_{0,1}$ with an iteration of Laplacian smoothing, but without affecting any value at 0 or 1 (within a small tolerance of 0.02).

Automatic face selection. Then, we select a subset of the mesh polygonal faces $M'_0 \subseteq M_0$ and $M'_1 \subseteq M_1$ that can be preserved *unmodified* in the output mesh, defined as the faces with a uniform presence value of 1. To regularize the region's boundaries formed by the selected faces, we apply a morphological opening-closing operator [Roessl et al. 2000].

Removal of secondary components. Many input models in, for example, videogames, feature secondary disconnected components used to model feathers, fur "flaps", armor pieces, and so on. For increased robustness, we identify any disconnected components with a roughly constant presence value (we used variance smaller than 0.04 as a criterion) and ignore them in the subsequent geometric fusion process (Fig. 7).

Fig. 7. Automatic selection of secondary connected components. An input model features multiple connected components that are topologically separated from the main component body (highlighted in the top image). The per-face selection can be made aware of this, identifying and removing them from the subsequent mesh fusion phase (bottom right).

The rationale is that these elements should not be blended in the results but either be fully preserved or entirely removed. We decide based on the averaged presence value being smaller or larger than 0.5. This simple countermeasure allows us to increase automatism with a more significant number of existing real-world assets (see Fig. 8).

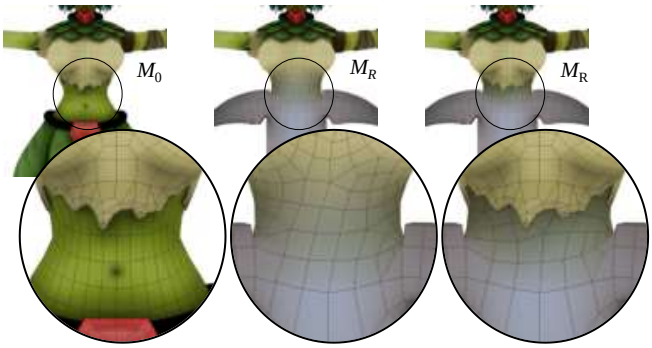

Fig. 8. Results of automatic handling of secondary connected components. In this example, the shirt of the input *elf* model is a separate, connected component (left), which is automatically identified as almost unaffected by the geometrical fusion (as its "presence" value is almost uniformly 1). For this reason, our method excludes it from the fusion operation and reinserts it after the fusion. The final result obtained in this way (right) can be preferable to the otherwise obtained (middle).

6.1 Definition of the interconnecting implicit surface

Next, we generate the shape of the connecting geometry that will be used to smoothly join M_0' and M_1'. We identify the affected volume region as the minimal Axis Aligned Bounding Box encapsulating all vertices that are not either fully 1 or fully 0 (within a small tolerance of 0.01) and sample each voxel inside this volume on a regular lattice at a fixed resolution (the voxel size is defined as 0.8 times the minimum of the average edge lengths in M_0 and M_1).

At each voxel, we compute the Signed Distance Function (SDF) [Curless and Levoy 1996] values δ_0 and δ_1 from M_0 and M_1 respectively, and record the presence values p_0 and p_1 of the two closest point on M_0 and M_1, found by interpolating the per-vertex values inside triangular faces (splitting non triangular faces into triangles). To compute the values $\delta_{0,1}$, we use the Generalized Winding Number [Jacobson et al. 2013] over M_0 and M_1 respectively, using [Barill et al. 2018], which extends the definition of the sign of the SDF to open meshes.

We then produce a resulting SDF value δ_R for that voxel as a function of $\delta_{0,1}$ and $p_{0,1}$. Many ways to do this are explored in literature [Angles et al. 2017; Bernhardt et al. 2013; Gourmel et al. 2013], resulting in different transition shapes. In our case, we obtain good results using just two different basic solutions chosen according to the semantics of the skeleton operation.

$$\delta_R = \begin{cases} \frac{p_0\,\delta_0 + p_1\,\delta_1}{p_0+p_1} & \text{for replacement operations} \\ \min(\delta_0, \delta_1) & \text{for addition operations} \end{cases} \quad (1)$$

In words, we perform a simple implicit union operator for operations involving the addition of subtrees to a skeleton (see Fig. 2, middle). We use a simple implicit blend operator for operations involving the substitution of subtrees, i.e., addition and removal of subtrees (see Fig. 2, top and bottom). While prone to some limitations (discussed in Sec. 10.1), this simple solution works well in practice.

The resulting voxelized field describes the shape of the piece of the surface, which will be used to interconnect M_0' and M_1'.

Construction of the interconnecting meshing. We then produce a meshing for the connecting geometry. We construct a new meshing M_J that seamlessly fuses with the open boundaries of $M_{0,1}'$, extending their edge flows. The final mesh M_R is the defined as $M_0' \cup M_1' \cup M_J$. This operation follows the spirit of [Nuvoli et al. 2019] but sets apart from it by dropping the requirement on the closed mesh, two-manifold, and pure-quad to match our typical scenario (videogame assets).

Using the presence field, we first determine the part of the original meshes that need to be preserved. Then, we mesh the result of the composition of the two distance fields in the region where the two meshes are not maintained. At this point, we want to attach the two preserved parts to the new implicit surface that blends them. We first associate each border of the preserved mesh with the borders of the blending surface. Then we use an approach similar to [Duncan et al. 2016] to blend and merge between borders, and we perform some simple local remeshing operations on the borders to eliminate degenerate triangles. Finally, we tessellate the newly created surfaces using the approach of [Pietroni et al. 2021], which merges between the original flows and creates a high-quality quadrilateral mesh. This process is overviewed in Fig. 9.

The method we defined for a single operation on two models can be easily extended to composite operations on more than two models (see Fig. 10 for an example). We define a single SDF for the entire operation, and, in each voxel, we compute σ_R according to the single sub-operation involving the two highest presence factors.

7 DETERMINING THE NEW SKINNING WEIGHTS AND OTHER PER-VERTEX QUANTITIES

Our strategy to construct M_R allows us to transfer any per-vertex attribute (e.g. colors) originally stored in the input meshes $M_{0,1}$.

For any vertex in M_0' or M_1', we simply copy the attribute from M_0 or M_1. For each vertex v_J in M_J, we take the two vertices v_0 and v_1 in M_0 and M_1 respectively, that are closest to v_J, and interpolate their attributes using, as interpolating factor, δ_R in equation (1) computed in the volumetric structure at position v_J. Vertices v_0 and v_1 and the delta values are quickly recovered leveraging on the volumetric voxel-based structure used to construct M_J.

7.1 Skinning weights

The skinning weights W_0 and W_1 cannot be transferred directly from vertices of $M_{0,1}$ to M_R, because they are defined over the respective skeleton $S_{0,1}$. In order to transfer them, we first retarget them, simply as $W_0' = \mathcal{B}_0(W_0)$, and $W_1' = \mathcal{B}_1(W_1)$ (this operation denotes a re-indexing of the bone weights, including when necessary the sum of the weights associated to pairs of bones of $S_{0,1}$ that are mapped by

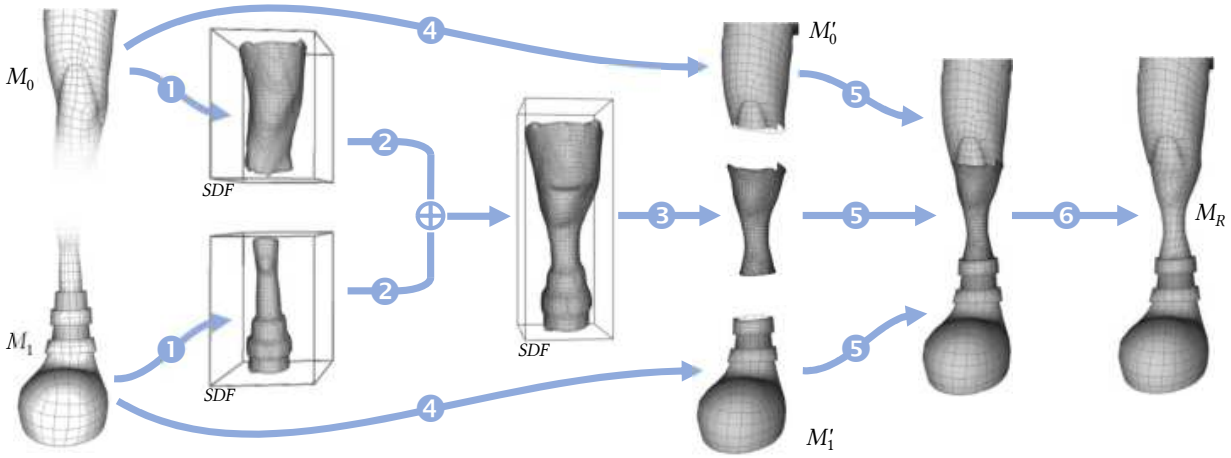

Fig. 9. The pipeline to produce a meshing M_R from the input meshes $M_{0,1}$ (transparency alpha values reflect the per-vertex presence value). The two meshes are first sampled into two Signed Distance Fields (1), which are blended into one unified blending field (2), and extracted isosurface is cut (3) to match the two sub-meshes $M'_{0,1}$ selected from $M_{0,1}$ (4). The results are fused into one mesh (5), and the final results are re-meshed (5).

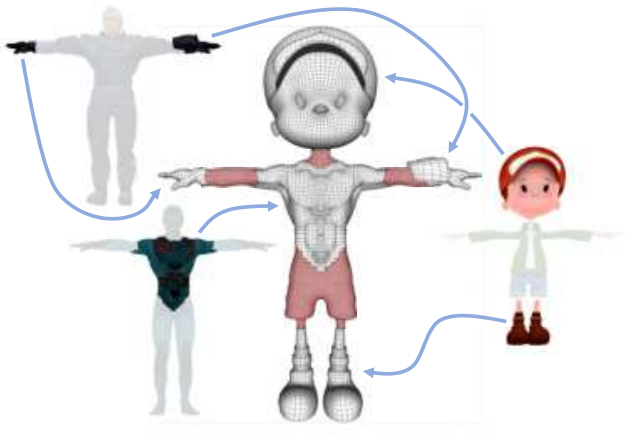

Fig. 10. We compose three different models to obtain the final result: the head and the feet from one of them, the hands from another, and the torso from the third. In the composite model, we highlight in red the junctions produced by our method.

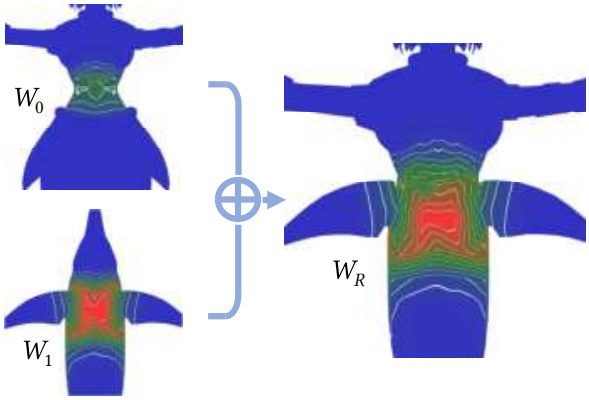

Fig. 11. An example of the set of skinning weights W_R resulting from blending the two input sets W_0 and W_1. The color coding shows the zone of influence of one abdominal bone, which is the connecting bone of the operation, present in all three models). Weight isolines (white) reveal that the influence matches the two models in the respective areas.

$\mathcal{B}_{0,1}$ into the same bone of S_R). This process gives good results in spite of the mappings $\mathcal{B}_{0,1}$ being possibly inaccurate far from the joining node.

Next, we transfer the two sets of retargeted skinning weights W'_0 and W'_1 from M_0 and M_1 into M_R, treating them as any other attribute.

Finally, we do a pass over the obtained weights to re-enforce sparsity, that is, to enforce that only up to N weights are non-zero. We do this simply by setting to zeros all but the $N = 4$ largest weights and re-normalizing the results so that it sums up to 1.

7.2 Colors and texture

In our work, we do not focus on texturing or UV-maps, which are usually part of the inputs meshes in a videogame context. Ideally, the newly produced M_R can be enriched with a novel texture, obtained by resampling the input textures, with, possibly, using texture-synthesis strategies [Akl et al. 2018] to produce an automatic transition between these original textures. This would also require, as a preliminary step, the production of a UV-map for M_R, either by creating a new one from scratch (which is a well studied, but open, problem), or by adapting the combined UV-maps of M_0 and M_1 (e.g. after [Maggiordomo et al. 2021]). We consider these tasks as orthogonal to our objective, and leave them to future studies. As a cheaper fallback strategy (used in all the renderings in this paper), we apply the original UV-maps and textures verbatim over

M'_0 and M'_1, and we color the junction piece M_J using per-vertex colors attributes instead. To do so, we first bake texture colors into per-vertex color attributes in both M_0 and M_1, and then interpolate them into M_J, like any other standard attribute (resulting, as a drawback, in blurred colors in M_J).

8 DETERMINING THE NEW ANIMATIONS

Lastly, we need to produce a new set of animations A for the skinned model. Automatically production of new animations, or re-targeting of existing ones, are exceedingly complex tasks, deeply studied in literature from a variety of angles (see Sections 2.5 and 2.6), and are ultimately open problems. In our scenario, we can, instead, rely on the existence of high-quality animations already designed for and consistent with every sub-part of the final skeletons. We decide to cast the problem as the task of *completing* existing animations from either input model with data coming from the set of animations of the other, without "inventing" any new animation data; this also takes full advantage of the intrinsic mixability of skeletal animations.

We repeatedly take a given animation a from either sets A_0 or A_1 and complete it by using the information extracted from the whole other set by carefully selecting any frame in the latter for each frame in a. This process can be applied to any animation in $A_0 \cup A_1$; which one is useful depends on the context and the semantics. In our prototype, we let the user choose or, by default, complete all animations.

8.1 Completing an animation

One animation a is a sequence of timed keyframes $(k_0, k_1, ...)$, each keyframe defining a per-bone local rotation (occasionally, a translation, modifying the ones otherwise inherited by the rest pose) concerning its parent node. As a preliminary step, we re-sample the animations by interpolating keyframes at a regular time interval (we used 30 frames per second) to ensure that the process is robust to both sparse and dense input keyframe sequences.

Consider, for example, the generation of an output animation a for skeleton S_R from an input animation $a' \in A_0$, which is initially defined for skeleton S_0.

We produce a keyframe k in a for each keyframe k' in a'. A trivial idea would be to define mapping $k = \mathcal{B}_0(k')$, i.e. to copy the local rotations of each bone $b_i \in S_0$ into the local rotation of the corresponding bone $\mathcal{B}_0(b_i) \in S_R$. However, this gives terrible results because, as noted, the mapping $\mathcal{B}_0$ can be very approximate for many bones and is not even necessarily surjective (not all bones in S_R would receive a transformation).

Instead, our strategy is to only use $\mathcal{B}_{0,1}$ to identify a proper keyframe k'' (defined over skeleton S_1) from *any* animations in A_1 (a task for which $M_{0,1}$ needs only to be minimally accurate on average). Then, we assign all per-bone rotations of k, picking them from k' or k'' (see Figure 12).

The crucial part of this process is the selection of k'' from a given k'. We choose not to limit the choice to keyframes of a single animation in A_1, nor to pick consecutive keyframes from a given animation. While this would be an easy route to enforce temporal continuity in the final animation, we want to exploit the entire range of behaviors captured by the whole input set of animations. Instead, we pursue temporal continuity explicitly as just one criterion to be balanced against others (as discussed later).

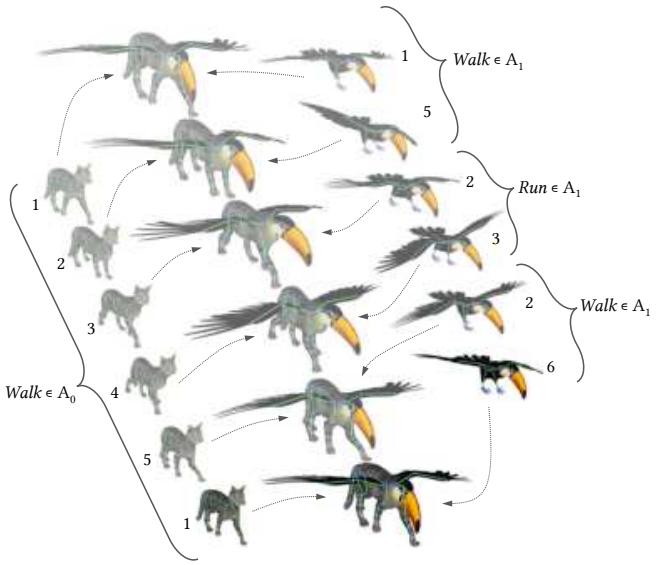

Fig. 12. An example of the construction of one animation in the output set A_R. Given a target animation $a \in A_0$, originally defined for skeleton S_0, we produce a corresponding animation $a' \in A_R$, for skeleton S_R, by electing, for each keyframe of a (left), one appropriate best matching keyframe (right), choosing among all keyframes in any animations in A_1, according to several criteria – see text. Then, the two keyframes are fused into a new keyframe of a' (center) via the bone-to-bone mapping $\mathcal{B}$. See also attached videos for this and other examples.

Our selection strategy relies upon the notion of keyframe-to-keyframe distance. We evaluate all the candidate keyframes (i.e., all keyframes of all animations in A_1), assign them to a score measuring similarity and continuity, and select the highest scorer.

Keyframe-to-keyframe distance. To compare keyframe k' (for S_0) and k'' (for S_1), we first apply our bone-to-bone mapping $\mathcal{B}$ to both and then evaluate the similarity of $\mathcal{B}(k')$ with $\mathcal{B}_1(k'')$. To apply a $\mathcal{B}$ to a keyframe k means to inject all local rotations defined for a bone $b_i \in S_0$ to the bone $\mathcal{B}(b_i) \in S_R$. The bones in S_R that are not reached by any bone (as $\mathcal{B}$ is not surjective) are assigned to the identity, and the bones in S_R that are reached by multiple bones (as $\mathcal{B}$ is not injective) cumulate the rotations.

Keyframes $\mathcal{B}_0(k')$ and $\mathcal{B}_1(k'')$ can then be compared bone to bone by computing their global rotations (with direct kinematics, as commonly done during the execution of a skinning animation) and adding together the similarities between the resulting global rotations. The similarity between two global rotations is defined as the absolute value of the dot product of their quaternionic representations (using the absolute value produces the correct similarity measure, despite a rotation being potentially expressed by two opposite quaternions). Each contribution is weighted by a confidence value derived from $\mathcal{B}$, defined, for each node in S_R, as the sum of all confidence values mapped on that node by $\mathcal{B}$.

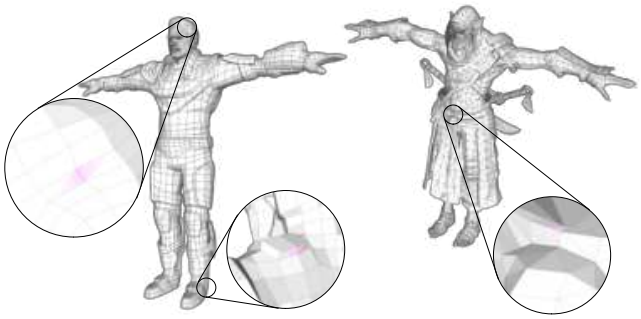

Fig. 13. Input models are not required to be watertight and can contain meshing artifacts such as non-manifold edges, as shown in the insets.

The rationale of this procedure is to favor the selection of a frame where the parts that have been substituted have a behavioral similarity with the parts that substituted them.

Temporal continuity. We favor the selection of keyframes resulting in temporal continuity by evaluating the keyframe-to-keyframe distance between the currently processed frame k_i and its candidate k_j, and between the few subsequent and precedent keyframes from the respective animations. In our experiments, we add the distances between the two keyframes at temporal distance 1 step with a weight of 0.2 and the two keyframes at temporal distance two steps with a weight of 0.1. We improve the temporal continuity of the results by performing a light final temporal smoothing in the keyframes of the resulting animations. Standard keyframe interpolation in the downstream application mechanisms improves temporal continuity in the final results.

9 RESULTS

We integrated our system into an interactive editor that allows the artist to control the result of the mixing operations (see attached videos). The editing system offers the artist an instrument to *attach* new portions of existing skeletons on top of others or to *replace* a part of an existing skeleton with the one taken from a second model. The user can combine multiple such mixing operations; the system quickly previews the final result. This preview does not include any mixing steps, yet it is still a valuable tool to promptly grasp the final result. After the preview, the system produces the final result, consisting of a rigged, skinned, and animated high-quality mesh, which can be inspected (using any newly produced animations).

Our interactive approach allows the user to easily navigate among similar possibilities. For example, the accompanying video shows a sequence where the user tries and discards a small succession of attempts, resulting from slightly different edits (such as the selection of nearby bones for the attaching point) before landing on the preferred solution. All such operations generate models with a comparable quality (e.g., in terms of meshing quality), so user discretion is the deciding factor.

We performed our tests on a laptop computer with a Ryzen9, 3.3 GHz processor with 16GB of RAM. We used OpenVDB [Museth et al. 2013] for the operations and Gurobi [Gurobi Optimization 2018] to solve the linear-integer problems involved in the phase of remeshing (see [Nuvoli et al. 2019] for details). Our implementation used also other external processing libraries including VCG Library [CNR 2013], CG3Lib [Muntoni and Nuvoli 2021], libigl [Jacobson et al. 2016] and Eigen [Guennebaud et al. 2014].

Fig. 14 shows the input models and their recombination, showing, for each result, one final pose captured from one of the synthesized animations. Fig. 15 shows an additional model in different poses. Differently from [Nuvoli et al. 2019], our system does not make any assumptions on the input mesh and can manage open, non-manifold meshes, multiple components, and non-quadrilateral elements. In the additional materials, we provide 35 animations showing the animations of 9 models that have been obtained, combining 16 different starting characters, featuring many different characteristics from models that have non-manifold artifacts (see Fig. 13).

The generation of these results required, on average, about 12 seconds, with a minimum and maximum running time of respectively 3 and 42 seconds. The running time mainly depends on the number of operations performed, the voxel size, and the intersected area's size. The most expensive procedures concerning time and memory footprints are extracting and blending the fields.

Evaluation of produced skinning weights. Fig. 16 shows a comparison of the skinning weights obtained with our *blending* method and the skinning weights generated by the automatic tool provided in Maya [Autodesk 2019]. In each figure, we used Dual Quaternion Skinning [Kavan et al. 2007] to compute the mesh pose. As expected, our method preserves as much as possible the original skinning weights provided by the artist and smoothly blends in between, generating no artifacts. We believe that the advantages provided by our method will be even more evident in the cases where the skinning weights are not directly related to the distance between the surface and the closest point on the skeleton, which is a common scenario for video game assets. Currently, no method is capable of a context-aware mix between weights and mesh, so the complete re-assignment of skinning weights is the only available option.

9.1 Comparison with traditional asset production pipelines

To get a first indication of the potential impact of our integrated semiautomatic technique, we hired a professional digital artist to obtain a comparable result with standard techniques. The example chosen for this experiment is the one visible in Fig. 1 and Fig. 3. The artist (credited at the end) has been provided the same initial assets and tasked to obtain the result visible in the images described verbally. The entire session has been recorded, and a time-lapse is provided in additional material. Using a standard digital production suite [Autodesk 2019], the artist proceeded to manually fuse and retopologize the mesh, combine the two skeletons, refine the skinning weights, and finally, key-frame edit one single walking animation. The process took 5.5 hours of high-paced work, including 2.5 hours of interaction with the editing system. In comparison, with our system, around 2 minutes of user interaction, plus less than 15 minutes of automatic computation, produced a comparable result, endowed with an entire set of animations (the union of the complete sets of animations from both original models). The processing time was mostly spent on producing the set of animations, with the re-meshed and skinned model available for inspection after

Fig. 14. Models obtained by mixing different inputs in different ways. For each model, we show the inputs in rest pose and the result in one animated pose.

about 10 seconds. The interaction time included a few attempted, previewed, and rejected mixing operations in the search for the desired effect. A preliminary assessment by the artist indicates that our result is directly usable.

10 CONCLUSIONS

Given the high quality of the generated results, we believe our methods could significantly impact the entertainment industry. Integrated into current software for 3D modeling, it would provide a powerful tool for artists to reuse parts of their well-designed characters automatically.

Our method takes as input a typical asset used in the videogames or animation industry, with no assumption on the structure of the input meshes, their animation, or the details of their creation (see Fig. 13). We defined a new operation that considers both rigged models and their animation as compact tuples that can be recombined to create new ones. The result is a new complete asset, including geometry, rigging, and a new animation set, ready to be integrated into games or other immersive applications.

Despite current limitations, our system can lead to a new direction for creating models in the industry, significantly reducing the entire process's cost.

We believe that modeling-by-composition will soon consolidate as a new strategy for content creation as it is the perfect application domain for deep learning techniques.

10.1 Limitations and future work

Our method suffers from various limitations.

Loss of meshing symmetry. Our approach is not guaranteed to reproduce input symmetries (e.g., bilateral symmetries) in the output meshing, as visible, for example, in the closeup in Fig. 1. Loss of symmetry is a recurring limitation in many automatic remeshing algorithms [Bommes et al. 2013], even if countermeasures have been proposed and could be employed [Panozzo et al. 2012].

Geometric problems. Our geometric fusion and skinning transfer strategy work best when blending surfaces with similar orientations. When the orientation differs drastically, the computation of the SDF

Fig. 15. We can create a new hybrid animated model – a mermaid – using the upper part and upper limbs of a female humanoid and the tail and fins of a dolphin. The final result, available in the supplementary material video, is a mermaid with smooth animations ready to be used.

might produce unexpected geometry and the associated skinning weights (see Fig. 17). While a user can easily circumvent this by controlling the shape of the fused figure using the preliminary posing (Sec. 4.1 and Fig. 3), this observation can be exploited to attempt to automatize this phase. Another countermeasure could be to adopt more sophisticated gradient-based operators to composite SDFs [Angles et al. 2017].

Limitations of animations. One strength of our unified approach is that it produces ready assets, which include animations for the new rig. Specifically, the new animations are obtained by *completing* existing animations, initially designed for either rig, with accompanying motions extracted from an animation initially designed for the other rig; in other words, instead of attempting to synthesize any new motion, we only recombine the input hand designed,

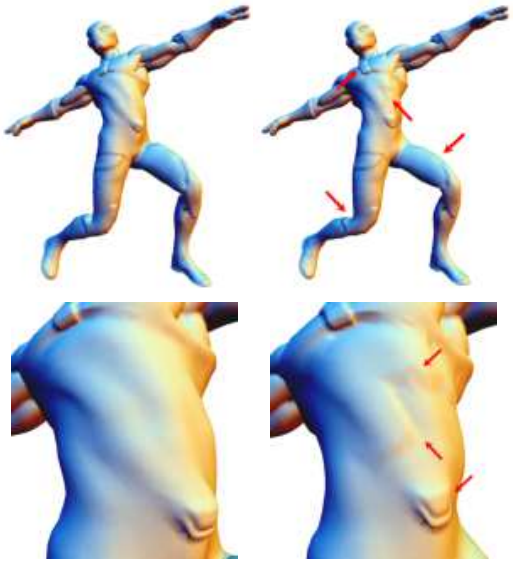

Fig. 16. The figure shows a comparison between the rig recombined with our method (left) with the automatic skinning made by commercial software (right). Recomputing skinning weights from scratch can reveal visible artifacts (we highlighted them with arrows). These artifacts might be even more extreme in the case of non-humanoid characters.

high-quality animations. This approach has both obvious benefits and limitations. In many cases, the produced animations are usable

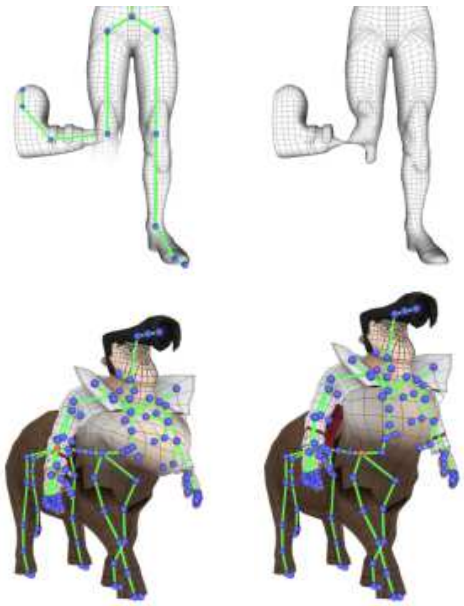

Fig. 17. Blending between two parts having significantly different orientations may produce artifacts in the interconnecting surface (top) and the model deformation (bottom).

Fig. 18. A few frames of a swimming animation produced by our method for a creature that we obtained by attaching the upper torso of a model to another one. The method fails to produce a viable swim animation for the resulting six-limbed creature, despite one of the input assets (but not the other) coming with its swim animation.

either directly or as a starting point for further editing (see attached video); it cannot, however, be expected to divine new complex behavior of the new creature when they cannot be extracted from either set of motions. We depict one example of such failure in Figure 18, where the system fails to believably complete a swimming animation of the output creature (as the second model had no swimming animation, and a six-limbed creature would probably swim in a completely different way).

Self-intersections during animations. Even if the initial animation does not self-intersect, we cannot guarantee that the final result will be free of self-intersection (see the inset). Indeed we do not perform any check nor attempt to solve that issue. A possible solution to this issue could be choosing only the frames that do not generate self-intersecting models.

Processing speed. The implementation is currently not real-time. Making our method interactive by improving our algorithms' efficiency or engineering the implementation would ease the integration on real industrial pipelines. The current implementation offers considerable space for distance field representation and manipulation optimization.

ACKNOWLEDGMENTS

We thank Alexandra Neville for the professional expertise and the help in performing the experiment outlined in Section 9.1. Stefano Nuvoli gratefully acknowledges Sardinia Regional Government for the financial support of his Ph.D. scholarship (P.O.R. Sardegna F.S.E.1 Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.).

REFERENCES

Kfir Aberman, Yijia Weng, Dani Lischinski, Daniel Cohen-Or, and Baoquan Chen. 2020. Unpaired Motion Style Transfer from Video to Animation. *ACM Trans. Graph.* 39, 4, Article 64 (2020), 12 pages. https://doi.org/10.1145/3386569.3392469

Adib Akl, Charles Yaacoub, Marc Donias, Jean-Pierre Da Costa, and Christian Germain. 2018. A survey of exemplar-based texture synthesis methods. *Computer Vision and Image Understanding* 172 (2018), 12–24. https://doi.org/10.1016/j.cviu.2018.04.001

Baptiste Angles, Marco Tarini, Brian Wyvill, Loïc Barthe, and Andrea Tagliasacchi. 2017. Sketch-Based Implicit Blending. *ACM Trans. Graph.* 36, 6, Article 181 (2017), 13 pages. https://doi.org/10.1145/3130800.3130825

Autodesk. 2019. Maya.

Seungbae Bang and Sung-Hee Lee. 2018. Spline Interface for Intuitive Skinning Weight Editing. *ACM Trans. Graph.* 37, 5, Article 174 (2018), 14 pages. https://doi.org/10.1145/3186565

Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. *ACM Trans. Graph.* 26, 3 (2007), 72—-es. https://doi.org/10.1145/1276377.1276467

Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. 2018. Fast Winding Numbers for Soups and Clouds. *ACM Trans. Graph.* 37, 4, Article 43 (2018), 12 pages. https://doi.org/10.1145/3197517.3201337

Adrien Bernhardt, Loïc Barthe, Marie-Paule Cani, and Brian Wyvill. 2013. Implicit Blending Revisited. *Comput. Graph. Forum* 29, 2 (2013), 367–376. https://doi.org/10.1111/j.1467-8659.2009.01606.x

Stephan Bischoff and Leif Kobbelt. 2005. Structure Preserving CAD Model Repair. *Comput. Graph. Forum* 24, 3 (2005), 527–536. https://doi.org/10.1111/j.1467-8659.2005.00878.x

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Cláudio T. Silva, Marco Tarini, and Denis Zorin. 2013. Quad-Mesh Generation and Processing: A Survey. *Comput. Graph. Forum* 32, 6 (2013), 51–76. https://doi.org/10.1111/cgf.12014

Mazen Borno, Ludovic Righetti, Michael Black, Scott Delp, Eugene Fiume, and Javier Romero. 2018. Robust Physics-based Motion Retargeting with Realistic Body Shapes. *Comput. Graph. Forum* 37, 8 (2018), 81–92. https://doi.org/10.1111/cgf.13514

Péter Borosán, Ming Jin, Doug DeCarlo, Yotam Gingold, and Andrew Nealen. 2012. RigMesh: Automatic Rigging for Part-Based Shape Modeling and Deformation. *ACM Trans. Graph.* 31, 6, Article 198 (2012), 9 pages. https://doi.org/10.1145/2366145.2366217

Sofien Bouaziz, Yangang Wang, and Mark Pauly. 2013. Online Modeling for Realtime Facial Animation. *ACM Trans. Graph.* 32, 4, Article 40 (2013), 10 pages. https://doi.org/10.1145/2461912.2461976

Adnane Boukhayma, Jean-Sébastien Franco, and Edmond Boyer. 2017. Surface Motion Capture Transfer with Gaussian Process Regression. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3558–3566. https://doi.org/10.1109/CVPR.2017.379

Marcel Campen and Leif Kobbelt. 2010. Exact and Robust (Self-)Intersections for Polygonal Meshes. *Comput. Graph. Forum* 29, 2 (2010), 397–406. https://doi.org/10.1111/j.1467-8659.2009.01609.x

Siddhartha Chaudhuri, Daniel Ritchie, Kai Xu, and Hao (Richard) Zhang. 2019. Learning Generative Models of 3D Structures. In *Eurographics 2019 - Tutorials*, Wenzel Jakob and Enrico Puppo (Eds.). The Eurographics Association. https://doi.org/10.2312/egt.20191038

Zhiqin Chen and Hao Zhang. 2019. Learning Implicit Fields for Generative Shape Modeling. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5939–5948. https://doi.org/10.1109/CVPR.2019.00609

Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene. 2020. Fast and Robust Mesh Arrangements Using Floating-Point Arithmetic. *ACM Trans. Graph.* 39, 6, Article 250 (2020), 16 pages. https://doi.org/10.1145/3414685.3417818

Kwang-Jin Choi and Hyeong-Seok Ko. 2000. Online motion retargetting. *The Journal of Visualization and Computer Animation* 11, 5 (2000), 223–235.

CNR. 2013. The Visualization and Computer Graphics Library. http://vcg.isti.cnr.it/vcglib/

Blender Online Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam.

Brian Curless and Marc Levoy. 1996. A Volumetric Method for Building Complex Models from Range Images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 303–312. https://doi.org/10.1145/237170.237269

Olivier Dionne and Martin de Lasa. 2013. Geodesic Voxel Binding for Production Character Meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Anaheim, California) *(SCA '13)*. Association for Computing Machinery, New York, NY, USA, 173–180. https://doi.org/10.1145/2485895.2485919

Noah Duncan, Lap-Fai Yu, and Sai-Kit Yeung. 2016. Interchangeable Components for Hands-on Assembly Based Modelling. *ACM Trans. Graph.* 35, 6, Article 234 (2016), 14 pages. https://doi.org/10.1145/2980179.2982402

Thomas A. Funkhouser, Michael M. Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David P. Dobkin. 2004. Modeling by example. *ACM Trans. Graph.* 23, 3 (2004), 652–663.

Lin Gao, Jie Yang, Yi-Ling Qiao, Yu-Kun Lai, Paul L. Rosin, Weiwei Xu, and Shihong Xia. 2018. Automatic Unpaired Shape Deformation Transfer. *ACM Trans. Graph.* 37, 6, Article 237 (2018), 15 pages. https://doi.org/10.1145/3272127.3275028

Michael Gleicher. 1998. Retargetting Motion to New Characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*. Association for Computing Machinery, New York, NY, USA, 33–42. https://doi.org/10.1145/280814.280820

Olivier Gourmel, Loic Barthe, Marie-Paule Cani, Brian Wyvill, Adrien Bernhardt, Mathias Paulin, and Herbert Grasberger. 2013. A Gradient-Based Implicit Blend. *ACM Trans. Graph.* 32, 2, Article 12 (2013), 12 pages. https://doi.org/10.1145/2451236.2451238

Keith Grochow, Steven L. Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-Based Inverse Kinematics. *ACM Trans. Graph.* 23, 3 (2004), 522—-531. https://doi.org/10.1145/1015706.1015755

Gael Guennebaud, Benoit Jacob, et al. 2014. Eigen: a C++ linear algebra library. http://eigen.tuxfamily.org

LLC Gurobi Optimization. 2018. Gurobi Optimizer Reference Manual.

Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. 2008. Real-Time Motion Retargeting to Highly Varied User-Created Morphologies. In *ACM SIGGRAPH 2008 Papers* (Los Angeles, California) *(SIGGRAPH '08)*. Association for Computing Machinery, New York, NY, USA, Article 27, 11 pages. https://doi.org/10.1145/1399504.1360626

Daniel Holden, Jun Saito, and Taku Komura. 2016. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.* 35, 4, Article 138 (2016), 11 pages. https://doi.org/10.1145/2897824.2925975

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Trans. Graph.* 30, 4, Article 78 (2011), 8 pages. https://doi.org/10.1145/2010324.1964973

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.* 32, 4, Article 33 (2013), 12 pages. https://doi.org/10.1145/2461912.2461916

Alec Jacobson, Daniele Panozzo, et al. 2016. libigl: A simple C++ geometry processing library. https://libigl.github.io

Won-Seob Jang, Won-Kyu Lee, In-Kwon Lee, and J. Lee. 2008. Enriching a motion database by analogous combination of partial human motions . *Vis. Comput.* 24 (2008), 271–280. https://doi.org/10.1007/s00371-007-0200-1

R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2020. ShapeAssembly: Learning to Generate Programs for 3D Shape Structure Synthesis . *ACM Trans. Graph.* 39, 6, Article 234 (2020), 20 pages. https://doi.org/10.1145/3414685.3417812

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2007. Skinning with Dual Quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (Seattle, Washington) *(I3D '07)*. Association for Computing Machinery, New York, NY, USA, 39–46. https://doi.org/10.1145/1230100.1230107

Ladislav Kavan and Olga Sorkine. 2012. Elasticity-Inspired Deformers for Character Articulation. *ACM Trans. Graph.* 31, 6, Article 196 (2012), 8 pages. https://doi.org/10.1145/2366145.2366215

Vladislav Kreavoy, Dan Julius, and Alla Sheffer. 2007. Model Composition from Interchangeable Components. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)*. 129–138. https://doi.org/10.1109/PG.2007.40

Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. 2019. NeuroSkinning: Automatic Skin Binding for Production Characters with Deep Graph Networks . *ACM Trans. Graph.* 38, 4, Article 114 (2019), 12 pages. https://doi.org/10.1145/3306346.3322969

Andrea Maggiordomo, Paolo Cignoni, and Marco Tarini. 2021. Texture Defragmentation for Photo-Reconstructed 3D Models. *Comput. Graph. Forum* 40, 2 (2021), 65–78. https://doi.org/10.1111/cgf.142615

Giorgio Marcias, Nico Pietroni, Daniele Panozzo, Enrico Puppo, and Olga Sorkine-Hornung. 2013. Animation-Aware Quadrangulation. *Comput. Graph. Forum* 32, 5 (2013), 167–175. https://doi.org/10.1111/cgf.12183

Andelo Martinovic and Luc Van Gool. 2013. Bayesian Grammar Learning for Inverse Procedural Modeling. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 201–208. https://doi.org/10.1109/CVPR.2013.33

Simone Melzi, Riccardo Marin, Pietro Musoni, Filippo Bardon, Marco Tarini, and Umberto Castellani. 2020. Intrinsic/extrinsic embedding for functional remeshing of 3D shapes. *Computers & Graphics* 88 (2020), 1–12. https://doi.org/10.1016/j.cag.2020.02.002

Eray Molla, Henrique Galvan Debarba, and Ronan Boulic. 2018. Egocentric Mapping of Body Surface Constraints. *IEEE Transactions on Visualization and Computer Graphics* 24, 7 (2018), 2089–2102. https://doi.org/10.1109/TVCG.2017.2708083

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural Modeling of Buildings. *ACM Trans. Graph.* 25, 3 (2006), 614—-623. https://doi.org/10.1145/1141911.1141931

Alessandro Muntoni and Stefano Nuvoli. 2021. *CG3Lib: A C++ geometry processing library.* https://doi.org/10.5281/zenodo.4431777

Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. 2013. OpenVDB: An Open-Source Data Structure and Toolkit for High-Resolution Volumes . In *ACM SIGGRAPH 2013 Courses* (Anaheim, California) *(SIGGRAPH '13)*. Association for Computing Machinery, New York, NY, USA, Article 19, 1 pages. https://doi.org/10.1145/2504435.2504454

Stefano Nuvoli, Alex Hernandez, Claudio Esperança, Riccardo Scateni, Paolo Cignoni, and Nico Pietroni. 2019. QuadMixer: Layout Preserving Blending of Quadrilateral Meshes. *ACM Trans. Graph.* 38, 6, Article 180 (2019), 13 pages. https://doi.org/10.1145/3355089.3356542

Junjun Pan, Lijuan Chen, Yuhan Yang, and Hong Qin. 2018. Automatic Skinning and Weight Retargeting of Articulated Characters Using Extended Position-Based Dynamics . *Vis. Comput.* 34, 10 (2018), 1285—-1297. https://doi.org/10.1007/s00371-017-1413-6

Daniele Panozzo, Yaron Lipman, Enrico Puppo, and Denis Zorin. 2012. Fields on Symmetric Surfaces. *ACM Trans. Graph.* 31, 4, Article 111 (2012), 12 pages. https://doi.org/10.1145/2185520.2185607

Yoav I. H. Parish and Pascal Müller. 2001. Procedural Modeling of Cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 301–308. https://doi.org/10.1145/383259.383292

Jeong Joon Park, Peter R. Florence, Julian Straub, Richard A. Newcombe, and S. Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation . In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 165–174. https://doi.org/10.1109/CVPR.2019.00025

Darko Pavic, Marcel Campen, and Leif Kobbelt. 2010. Hybrid Booleans. *Comput. Graph. Forum* 29, 1 (2010), 75–87. https://doi.org/10.1111/j.1467-8659.2009.01545.x

Nico Pietroni, Stefano Nuvoli, Thomas Alderighi, Paolo Cignoni, and Marco Tarini. 2021. Reliable Feature-Line Driven Quad-Remeshing. *ACM Trans. Graph.* 40, 4, Article 155 (2021), 17 pages. https://doi.org/10.1145/3450626.3459941

Daniel Ritchie, Sarah Jobalia, and Anna Thomas. 2018. Example-based Authoring of Procedural Modeling Programs with Structural and Continuous Variability . *Comput. Graph. Forum* 37, 2 (2018), 401–413. https://doi.org/10.1111/cgf.13371

Christian Roessl, Leif Kobbelt, and Hans-Peter Seidel. 2000. Extraction of feature lines on triangulated surfaces using morphological operators . In *Papers from 2000 AAAI Spring Symposium: Smart Graphics*. AAAI, 71–75.

Ryan Schmidt and Karan Singh. 2010. Meshmixer: An Interface for Rapid Mesh Composition. In *ACM SIGGRAPH 2010 Talks* (Los Angeles, California) *(SIGGRAPH '10)*. ACM, New York, NY, USA, Article 6, 1 pages.

Andrei Sharf, Marina Blumenkrants, Ariel Shamir, and Daniel Cohen-Or. 2006. SnapPaste: an interactive technique for easy mesh composition. *Vis. Comput.* 22, 9-11 (2006), 835–844.

Karan Singh and Richard E. Parent. 2001. Joining polyhedral objects using implicitly defined surfaces. *Vis. Comput.* 17, 7 (2001), 415–428. https://doi.org/10.1007/s003710100115416

Sebastian Starke, Yiwei Zhao, Fabio Zinno, and Taku Komura. 2021. Neural Animation Layering for Synthesizing Martial Arts Movements. *ACM Trans. Graph.* 40, 4, Article 92 (2021), 16 pages. https://doi.org/10.1145/3450626.3459881

Robert W. Sumner and Jovan Popović. 2004. Deformation Transfer for Triangle Meshes. *ACM Trans. Graph.* 23, 3 (2004), 399—-405. https://doi.org/10.1145/1015706.1015736

Sarah Taylor, Taehwan Kim, Yisong Yue, Moshe Mahler, James Krahe, Anastasio Garcia Rodriguez, Jessica Hodgins, and Iain Matthews. 2017. A Deep Learning Approach for Generalized Speech Animation. *ACM Trans. Graph.* 36, 4, Article 93 (2017), 11 pages. https://doi.org/10.1145/3072959.3073699

Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. 2018. Neural Kinematic Networks for Unsupervised Motion Retargetting. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 8639–8648. https://doi.org/10.1109/CVPR.2018.00901

The Foundry Visionmongers. 2018. Modo 12.1.

Jungdam Won and Jehee Lee. 2019. Learning Body Shape Variation in Physics-Based Characters. *ACM Trans. Graph.* 38, 6, Article 207 (2019), 12 pages. https://doi.org/10.1145/3355089.3356499

Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. 2016. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., Red Hook, NY, USA, 82–90.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RigNet: Neural Rigging for Articulated Characters. *ACM Trans. Graph.* 39, 4, Article 58 (2020), 14 pages. https://doi.org/10.1145/3386569.3392379

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, and Karan Singh. 2019. Predicting Animation Skeletons for 3D Articulated Models via Volumetric Nets . In *2019 International Conference on 3D Vision (3DV)*. 298–307. https://doi.org/10.1109/3DV.2019.00041

Kangxue Yin, Zhiqin Chen, Siddhartha Chaudhuri, Matthew Fisher, Vladimir G. Kim, and Hao Zhang. 2020. COALESCE: Component Assembly by Learning to Synthesize Connections. In *2020 International Conference on 3D Vision (3DV)*. 61–70. https://doi.org/10.1109/3DV50981.2020.00016

Juyong Zhang, Chunlin Wu, Jianfei Cai, Jianmin Zheng, and Xue-Cheng Tai. 2010. Mesh Snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow . *Comput. Graph. Forum* 29, 2 (2010), 517–526. https://doi.org/10.1111/j.1467-8659.2009.01621.x

Jiaran Zhou, Marcel Campen, Denis Zorin, Changhe Tu, and Cláudio T. Silva. 2018. Quadrangulation of non-rigid objects using deformation metrics. *Computer Aided Geometric Design* 62 (2018), 3–15. https://doi.org/10.1016/j.cagd.2018.03.003

Chenyang Zhu, Kai Xu, Siddhartha Chaudhuri, Renjiao Yi, and Hao Zhang. 2018. SCORES: Shape Composition with Recursive Substructure Priors. *ACM Trans. Graph.* 37, 6, Article 211 (2018), 14 pages. https://doi.org/10.1145/3272127.3275008