





Real-Time Deformation with Coupled Cages and Skeletons

F. Corda¹ , J. M. Thiery², M. Livesu³ , E. Puppo⁴ , T. Boubekeur^{5,2} and R. Scateni¹ 

¹University of Cagliari, Cagliari, Italy
cordafab@gmail.com, riccardo@unica.it

²Télécom Paris, LTCI, IPP - Institut Polytechnique de Paris, Paris, France
jean-marc.thiery@telecom-paristech.fr

³CNR IMATI, Pavia, Italy
marco.livesu@gmail.com

⁴University of Genoa, Genoa, Italy
Enrico.Puppo@unige.it

⁵Adobe, Paris, France
tamy.boubekeur@telecom-paris.fr

Abstract

Skeleton-based and cage-based deformation techniques represent the two most popular approaches to control real-time deformations of digital shapes and are, to a vast extent, complementary to one another. Despite their complementary roles, high-end modelling packages do not allow for seamless integration of such control structures, thus inducing a considerable burden on the user to maintain them synchronized. In this paper, we propose a framework that seamlessly combines rigging skeletons and deformation cages, granting artists with a real-time deformation system that operates using any smooth combination of the two approaches. By coupling the deformation spaces of cages and skeletons, we access a much larger space, containing poses that are impossible to obtain by acting solely on a skeleton or a cage. Our method is oblivious to the specific techniques used to perform skinning and cage-based deformation, securing it compatible with pre-existing tools. We demonstrate the usefulness of our hybrid approach on a variety of examples.

Keywords: animation, modelling, deformations

ACM CCS: • Computing methodologies → Shape modelling

1. Introduction

Interactive shape deformation is a fundamental building block in 3D modelling and many other applications. The various techniques available in the literature rely on simplified control structures: the user interacts with them, and the changes smoothly transfer to the high resolution controlled object (the *skin*) through a set of weights, which establish a relation between each point of the model and the handles of the control structure [JDKL14].

The two most widely used controllers for real-time deformation, namely *skeletons* and *cages*, support complementary tasks: skeleton-based techniques are adequate to control rigid parts and pose articulated bodies; conversely, cage-based methods are best for smooth volumetric deformations. Each control structure becomes unwieldy and overly complicated to use where the other excels, thus pushing practitioners to use them together on the same skin.

Numerous discussions on how to combine skeletons and cages can be found on specialized forums, blogs and other online resources. Despite this interest from the community of practitioners, the problem of keeping them in sync and finding consensus between the deformations they induce is still open. To this end, an important issue is that there exists a plethora of alternative skeleton and cage techniques, which are already implemented in almost any available deformation software. Therefore, to promote a seamless integration of hybrid deformation approaches into well-established software packages, it becomes crucial to guarantee some sort of flexibility and back-compatibility.

The complexity in combining skeletons and cages comes from the fact that they achieve deformation in substantially different ways. Skeleton-based techniques perform deformations that are *relative* to a particular pose of the skin, often called the *rest* pose: the deformed skin is always a combination of its shape in the rest pose with the

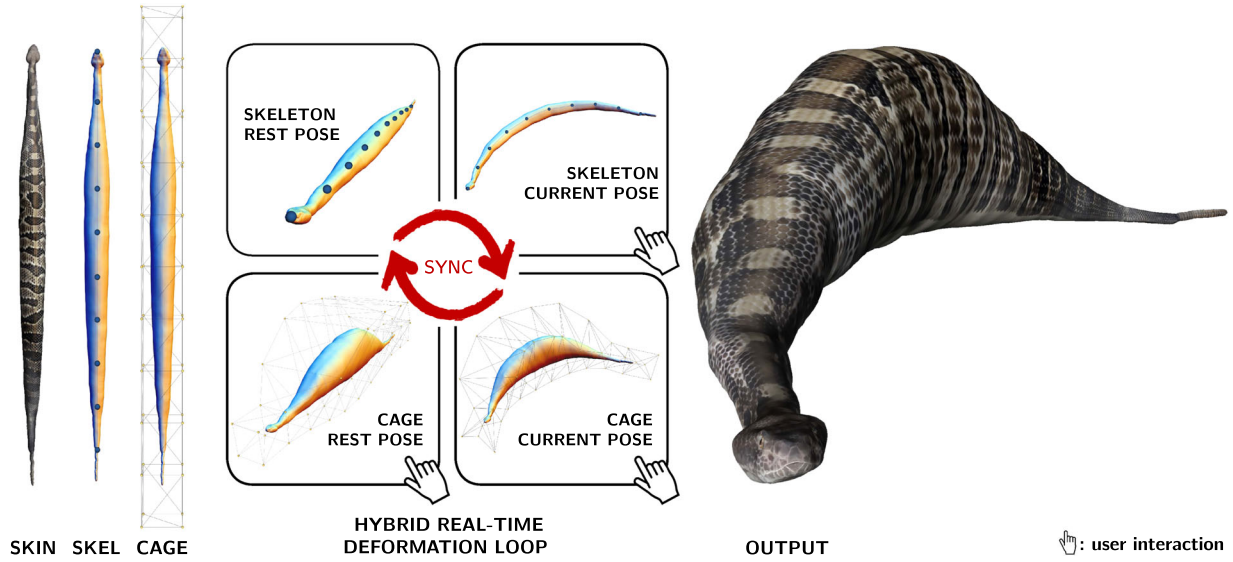


Figure 1: Given an input 3D shape equipped with its deformation skeleton and cage (left), our framework seamlessly combines both structures, merging their associated deformation spaces. We achieve this result by using rest and current pose both for the skeleton and cage (middle). User can deform the shape with any interlaced combination of the two control structures, editing the skeleton in the current pose or the cage both in the current and rest pose. Our novel operators automatically maintain all these entities in sync in real-time.

current position of the skeleton. Conversely, cage-based deformation follows an *absolute* approach: the current position of the control cage entirely determines the skin it envelopes, with no particular reference pose existing.

Previous attempts to combine skeletons and cages fall short, either because they are not general enough, or because they break back-compatibility. Blender [Ble18] allows to link a control cage to a skeleton and move the cage through it. The communication is only mono-directional: edits performed on the cage do not reflect on the skeleton, thus requiring complex manual edits to reposition the centres of rotation (CoR) of each bone. Jacobson and colleagues [JBPS11] combined skeleton, cage and point handles into a unified deformation meta-structure, but impose the simultaneous definition of all the handles and relative weights, and do not keep the sync between them. Moreover, their system implements a customized pipeline, which is not compatible with standard techniques.

We propose a hybrid deformation paradigm that seamlessly combines skeletons and cages, providing a real-time framework where the user can operate using any interlaced combination of the two control structures (Figure 1). Our method is compatible with classical skeleton-based and cage-based deformation techniques: it just acts as middleware to reach consensus between the two control structures. In particular, skeleton deformations can be transferred to the skin using the popular LBS [MTLT89], DQS [KCZO08], CoR [LH16] or any alternative approach (Figure 2). Linking weights can be either the result of an automatic computation (e.g. [BP07]) or hand painted by a digital artist, as it often happens in the industry. Similarly, cage-based deformation admits the use of any type of generalized barycentric coordinates that produce deformation through a linear relation between the cage handles and the vertices of the skin [NS13]. To achieve consensus between the two control

structures, we retain the relative nature of skeletons and extend it to the cages, which therefore exist both in the rest and current pose. At modelling time, the user guides a deformation by acting on one

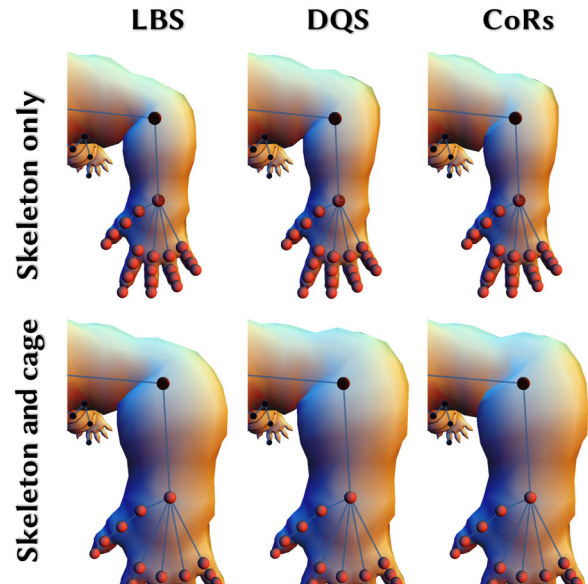


Figure 2: Results obtained with various alternative skinning methods implemented in our framework. The top row shows deformations obtained with a skeleton edit only (a 90 degrees rotation of the elbow). The bottom row shows results with an additional cage edit (a uniform stretch of the arm). Our joints and CoRs dynamic repositioning method handles both transformations in a natural manner.

controller, and the system automatically updates all the other poses accordingly. Our approach makes it possible to edit shapes both in current and rest poses, and occurs in real-time, with negligible overhead to classical skeleton- and cage-based deformation workload. Figure 3 provides an overview of our framework, depicting the paths of user interactions.

Contribution. Our main contribution is a deformation system that combines the deformation spaces of skeletons and cages, revealing a much larger deformation space, which contains configurations that cannot be achieved using solely a skeleton or a cage. From a technical point of view, our contribution consists of a collection of synchronization operators that maintain the pose set up-to-date during the editing session in real-time. We demonstrate our hybrid deformation approach on a variety of examples, including shape modelling and digital animation (Section 5). Our software prototype already implements several skeleton- and cage-based techniques, and can potentially embrace most existing techniques based on these control structures, thus demonstrating a great flexibility and back-compatibility.

2. Related Works

Our approach bridges skeleton-based and cage-based deformations. While works of this kind are quite rare, the literature offers a wide variety of approaches that focus on one deformation paradigm or the other. We refer the reader to the work of Jacobson *et al.* [JDKL14] for a recent survey, and focus here on the articles most relevant to our work.

Skeletons. Deformations defined by linearly blending transformation matrices associated with the bones of a control skeleton, also known as linear blend skinning (LBS), appeared long time ago in the literature [MTLT89] and have become extremely popular ever since. LBS suffers from a number of artefacts, most of which come from singular transformation matrices generated by the linear interpolation of rotations. A typical artefact of this kind is the well-known *candy wrapper*, which arises at the shoulders of a character when big torsions are applied to its arms. Recent literature has proposed more robust ways to combine transformation matrices. In particular, Kavan and colleagues proposed to blend rotations in the space of dual quaternions (DQS), avoiding the generation of candy wrapper artefacts at the cost of a minimal overhead in the real-time deformation pipeline [KCŽO08]. Due to their simplicity and intuitiveness, LBS and DQS are *de facto* standards in skeleton-based deformation. More recent non-linear skinning methods, such as stretchable and twistable bones [JS11], differential blending [OBP*13] and elasticity inspired deformer [KS12], introduce more sophisticated techniques that act on the same basic ingredients, namely rigging weights and affine transformations, to define the pose of the skeleton. Finally, Le and Hodgins [LH16] introduced an efficient method designed to avoid the respective artefacts of LBS and DQS, through the definition of a per-vertex CoR derived from an analysis of the skinning weights, and resulting from an optimization targeting as-rigid-as-possible deformations. In our software prototype, we incorporated LBS, DQS and CoRs (Figure 2), but our framework can support all previously cited skinning methods, thus promoting a seamless integration with available implementations. Critical to the pre-

viously cited methods is the definition of the so-called skinning weights. Various automatic methods aim at defining smooth weights [BP07, WL08, JBPS11, JWS12, DdL13] or target as-rigid-as-possible deformations under training [TE18]. While automatic methods generally provide satisfactory results on challenging inputs, artists often need to tune the skinning weights as they target specific effects, and methods allowing for high-level skinning weights editing have been developed recently [BCBiR*15, BL18]. Our method is quite general and has no restrictions on the skinning weights used for deformation, thus allowing for great artistic flexibility.

Cages. Cage-based methods derive directly from the free form deformation [SP86] and allow to deform a volume bounded by a control mesh. Each point within such volume is defined as a weighted sum of cage vertices, hence its position can be efficiently updated each time the cage is deformed by the user. Control weights are generalized barycentric coordinates, and differ to each other mainly for their smoothness and locality. Several alternatives have been proposed in the literature [TMB18, TTB13, JSW05, JMD*07, LKCOL07, LLCO08, ZDL*14, HS08]. All methods, with the exception of [LLCO08, BCWG09], compute skin coordinates as a linear combination of cage vertices, and are seamlessly supported by our framework. Similarly to skeletons, cages can be either automatically computed or manually crafted, thus ensuring a seamless integration with most available implementations. For brevity, we do not review methods for cage generation. We point the reader to the survey of [NS13] for classical literature in the field, and to [CLM*19] and references therein for a list of more recent algorithms.

Hybrid approaches. Some methods depart from the classical skeleton-based and cage-based deformation paradigms, trying to improve on them on some aspect. Garcia *et al.* [GPCP13] proposed a hybrid system that seamlessly combines multiple cages and barycentric coordinates. The system is completely devoted to cages only, and does not take into consideration interactions with a skeleton. Mukai and Kuriyama [MK16] propose the use of automatically generated bone helpers to enrich the space of deformations of LBS with secondary motions, enabling the animation of muscles and soft tissues. The system focuses on a very specific problem and does not offer the flexibility granted by a real control cage. Being compatible with LBS, their bone helpers could also be incorporated in our framework. Ju *et al.* [JZvdP*08] combined the use of cages and skeletons to avoid the candy wrapper artefacts of LBS. Opposite to ours, their system works as an open loop, using the skeleton to pose the cage, and the cage to pose the skin. Similar systems are currently supported by commercial software (e.g. Blender [Ble18]), but do not really offer the possibility to seamlessly combine deformations defined on the skeleton with others defined on the cage in arbitrary order. The combination of skeletons and point handles was explored in [WJBK15]. Jacobson *et al.* [JBPS11] proposed a system where skeleton, cage and point handles are all integrated into the same framework. Their system requires the simultaneous definition of all structures (see Equation 1 in the original paper), and is based on the use of the same coordinates at all levels, without permitting the use of manually painted weights, or different weights for different handles. Moreover, the skeleton and cage are part of the same meta-structure and must be jointly animated: manipulating

the skeleton (respectively, the cage) will not induce a deformation of the cage (respectively, the skeleton), contrary to what we aim at. All in all, previous techniques cannot offer the flexibility of our technique, that is switching seamlessly from one structure to the other while always relying on the optimization framework to update the other structure appropriately. Finally, combining deformation rigs has been used for a long time in specialized industrial scenarios. In particular, on-surface facial rigs are often superimposed on the skeleton rigs that control the head orientation. However, these scenarios typically come with a fixed prioritization policy, e.g. facial rigs defined in the local frame of the face, which itself undergoes a single rigid transform stemming from the neck bone. The scenarios we address are more challenging, as both structures compete to control global and non-rigid deformations that happen simultaneously.

Exploration of shape space. Our method is loosely related to methods based on the analysis of 3D shapes and subsequent structure aware deformation, which have the main intent to explore the space of shapes similar to a reference mesh (Figure 8). To this end, our real-time deformation tool complements classical methods based on feature curves [GSMCO09], bounding envelopes [ZFCO*11] or semantic handles [YCHK15], without exposing any significant technical overlap.

3. Background

We take as input a polygonal mesh M_0 , together with a skeleton S_0 rigged to M_0 , and a cage C_0 surrounding M_0 . We set our working structures at rest pose by initializing $M \equiv M_0$, $S \equiv S_0$ and $C \equiv C_0$; note that M , C and S can evolve because of editing, as discussed in the next section. Our method uses only the set of vertices of meshes and is oblivious of their connectivity. For this reason, whenever no ambiguity arises, we will overload the same symbol to denote both a mesh and its set of vertices.

3.1. Skeleton-based deformation

As customary, instead of representing the skeleton S explicitly, we consider the set of rigid transformations $\mathcal{T} = \{\mathbf{T}_1, \dots, \mathbf{T}_s\}$ that determine a given pose. Each transformation \mathbf{T}_j is associated with the j -th bone of S and represents a rotation around one of its end-points, which affects all the sub-skeleton below the given joint in the bones hierarchy. At rest pose, all transformations are set to identity. The skeleton S is rigged to M through an $m \times s$ (sparse) matrix of weights Ω , where each entry $\omega_{i,j}$ defines the influence of the j -th bone of S on i -th vertex of M .

A general skeleton-based deformation (a.k.a. skinning) has the following form:

$$M' = F(\mathcal{T}, \Omega, M), \quad (1)$$

where M' is the deformed (current) mesh. For instance, LBS is encoded vertex-wise by

$$\mathbf{v}'_i = \sum_{j=1}^s \omega_{i,j} \mathbf{T}_j \mathbf{v}_i, \quad (2)$$

which is often presented in the following form where the linear part (\mathbf{T}_j^r) applied to the vertex is separated from the translation part (\mathbf{T}_j^t)

$$\mathbf{v}'_i = \left(\sum_{j=1}^s \omega_{i,j} \mathbf{T}_j^r \right) \cdot \mathbf{v}_i + \left(\sum_{j=1}^s \omega_{i,j} \mathbf{T}_j^t \right), \quad (3)$$

while DQS is encoded similarly

$$\mathbf{v}'_i = \text{DQblend}(\mathcal{T}_i, \Omega_i) \mathbf{v}_i, \quad (4)$$

where DQblend is the proper function to blend transformations represented via DQS, while \mathcal{T}_i and Ω_i denote the i -th rows of \mathcal{T} and Ω , respectively.

The recently introduced CoR method [LH16] is slightly different from other methods, in the sense that it makes the use of an additional parameter derived from a cross-analysis of the mesh geometry and the skinning weights: a per-vertex CoR. This CoR p_i associated with vertex i is computed as:

$$p_i := \int_{x \in M} \delta(\omega_{i,\cdot}, \omega_{x,\cdot}) x \, dx / \int_{x \in M} \delta(\omega_{i,\cdot}, \omega_{x,\cdot}) dx, \quad (5)$$

$\delta(\omega_{i,\cdot}, \omega_{x,\cdot})$ denoting a distance between the sets of weights of vertex i and vertex x .

Given the CoR p_i associated with vertex i , and computing the linear part $R(i)$ applied to the mesh vertex using quaternion blending of the bone rotations $\{\mathbf{T}_j^r\}_j$, the translation applied to the vertex is computed as:

$$t(i) = \sum_{j=1}^s \omega_{i,j} (\mathbf{T}_j^r \cdot p_i + \mathbf{T}_j^t) - R(i) \cdot p_i.$$

Since the CoR computation depends only on the skinning weights, vertices with similar skinning weights have similar CoRs and are therefore transformed by the same rigid transformation.

Overall, the (run time) deformation of vertex i by the CoR method can be summarized as:

$$\begin{cases} \mathbf{v}'_i = R(i) \cdot \mathbf{v}_i + t(i) & \text{(deformation),} \\ R(i) = \text{DQBlendRot}(\{\omega_{i,j}, \mathbf{T}_j^r\}_j) & \text{(linear part),} \\ t(i) = \tilde{p}_i - R(i) \cdot p_i & \text{(translation part),} \\ \tilde{p}_i = \sum_{j=1}^s \omega_{i,j} (\mathbf{T}_j^r \cdot p_i + \mathbf{T}_j^t) & \text{(transformed CoR),} \end{cases} \quad (6)$$

where DQBlendRot returns the linear part of the matrix DQBlend as defined above in equation 4.

Our method is compliant with all skinning methods mentioned in Section 2, and any which requires only the rest pose location of the joints of the skeleton (or CoR derived from the rest pose mesh as done in the CoR method).

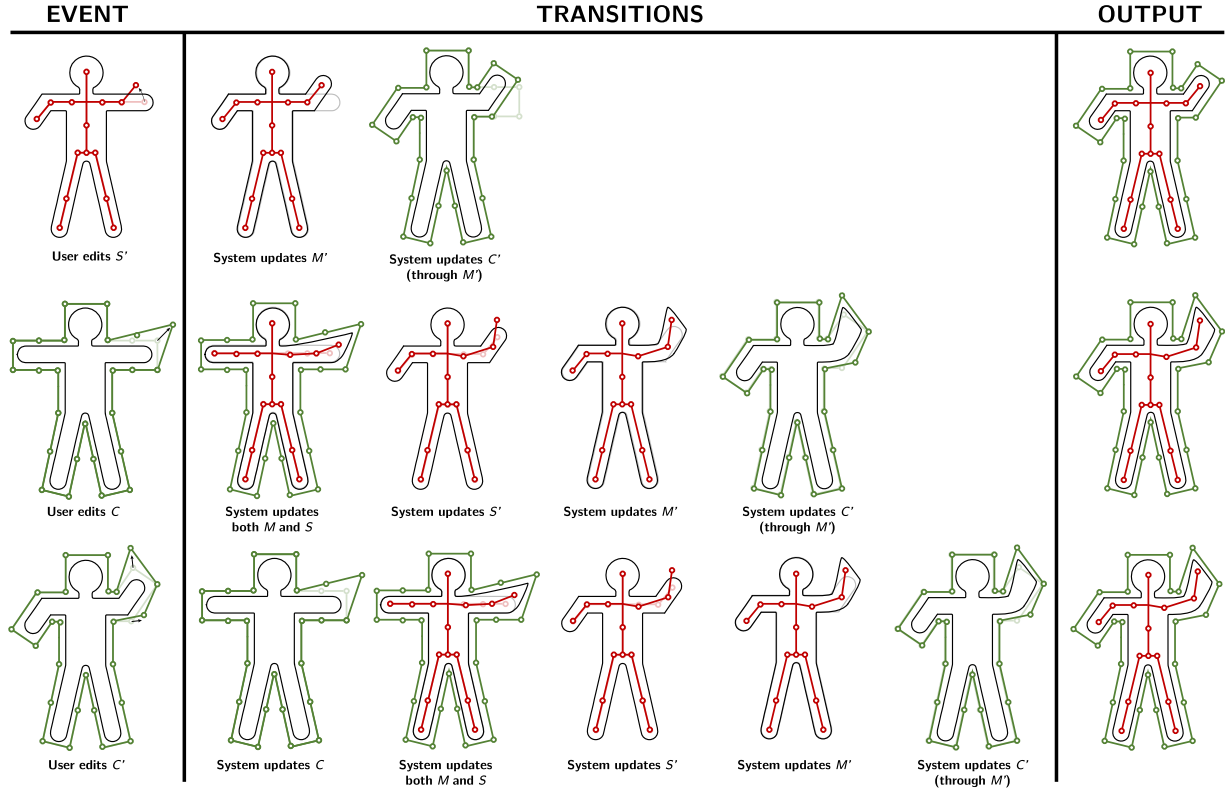


Figure 3: Interactions with our hybrid deformation system: the user can edit the skeleton in the current pose (S') or the cage, both in the rest (C) and current (C') pose. The diagram illustrates the chain of reactions that automatically update the system, maintaining the sync among the various entities involved in the deformation. All interactions occur in real-time.

3.2. Cage-based deformation

The cage C controls M via an $m \times c$ (sparse) matrix of barycentric coordinates Φ , where entry $\phi_{i,k}$ is the barycentric coordinate of mesh vertex \mathbf{v}_i with respect to cage vertex \mathbf{c}_k . We require deformation to be given vertex-wise by the standard linear equation:

$$\mathbf{v}'_i = \sum_{k=1}^c \phi_{i,k} \mathbf{c}'_k, \quad (7)$$

where \mathbf{v}'_i represents the (deformed) position of vertex \mathbf{v}_i of M when the cage vertices are set at (edited) positions \mathbf{c}'_j . This equation is compliant with all barycentric coordinates reviewed in Section 2, except the green coordinates [LLCO08] and the variational harmonic maps [BCWG09], which also require face normals that set a non-linear relation between the cage and skin. Indeed, our method is not compliant with the latter two techniques. Note that Equation (7) does not refer to any rest position for either the cage or skin. By convention, we assume that we have a rest pose for the cage C , when its vertices are at positions where equality $M = \Phi C$ holds. Namely, the rest pose of the cage induces the rest pose of the skin. In fact, this is the usual setting from which the barycentric coordinates are obtained.

4. Method

In the following section, we will refer to six structures, namely: the skin M , the skeleton S and the cage C at rest pose, together with their deformed counterparts M' , S' and C' at current pose.

Our method aims at reaching consensus among these structures upon any interleaved sequence of skeleton-based and cage-based deformations, obtaining synchronization between them. Performing a cage-based deformation, the S' and S skeletons must preserve the relationship with the Ω skinning weights. On the other hand, an S' edit must update the position of the C' cage accordingly to the new M' .

Our system makes it possible to edit any structure, assuming the sets of skinning weights Ω and barycentric coordinates Φ remain constant. Note that M is induced by S via skinning when \mathcal{T} contains just identity transformations; and M is induced by C via barycentric coordinates: these invariants set the consensus at rest pose and will be maintained throughout.

Unfortunately, if a mesh M' is induced via skinning by a skeleton S' , there may not exist a cage C' that induces M' , and vice-versa. We address this issue by synchronizing S' and C' so that they produce similar skins: M' is obtained through skinning, with a process that incorporates C' also.

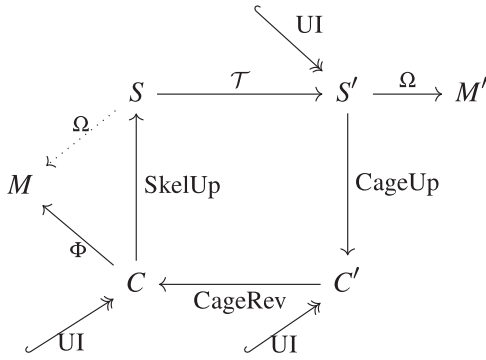


Figure 4: A diagram of our method showing transitions among the different structures; symbols attached to arrows denote the parameters or algorithms used to effect such transitions; hooked arrows marked UI denote user interaction. The dotted arrow represents rigging at rest pose and is just symbolic.

User interaction. The user can operate on three of the six structures defined above, namely the user can edit:

- the current pose S' of the skeleton;
- the cage C at rest pose;
- the cage C' at current pose.

Direct editing of either the skin (respectively, skeleton) at rest pose is not considered, as this operation is inherent to modelling (respectively, rigging) the input shape; it would be straightforward to include them in our framework, though. Note that the roles of the skeleton and cage are not fully symmetric: while the former can modify only the current pose of the skin, the latter can modify both current and rest poses. From the user point of view, editing the rest pose means adapting a different skin to the rigging, which remains untouched.

When executed, the three user interactions outlined above trigger a set of sync operations, automatically handled by the system. Specifically:

- if S' is edited, then both M' and C' will follow, while no change will occur to any structure in the rest pose (Figure 3, top);
- if C is edited, then M will follow, S will be adjusted to the new skin and the current set of transformations T will be adjusted accordingly, inducing a new current pose S' and a new skin M' – the current cage C' will be also updated (Figure 3, middle);
- if C' is edited, we have the most complicated situation: changes to C' will be reflected to C ; while all other modifications will occur as in the previous case up to C' itself, in a closed loop (Figure 3, bottom).

To get a consistent result in all cases, we must synchronize the different structures, as explained in the following.

Synchronization. We achieve synchronization among the six structures in the two poses by introducing a set of *transitions*, which reflect editing among them and are summarized in Figure 4:

- $S \rightarrow S' \rightarrow M'$ is the standard skinning from Equation (1);

- $C \rightarrow M$ is the standard application of barycentric coordinates from Equation (7);
- $C \rightarrow S$ adjusts the CoR of S on a modified skin M at rest pose – we call it the *skeleton updater* (SkelUp);
- $S' \rightarrow C'$ finds a consensus between the current poses of the skeleton and the cage – we call it the *cage updater* (CageUp);
- $C' \rightarrow C$ reflects the direct editing on the current pose to the cage at rest pose – we call it the *Reverse cage deformer* (CageRev). We must guarantee that the current cage C' determined from editing by the user coincides with the current cage resulting from the transition $C \rightarrow S \rightarrow S' \rightarrow C'$, as induced by the rest cage C modified by C' . To achieve this result, we must set this transition in a proper way.

No other direct transitions are considered. For instance, there is no direct transition from C to C' . This is a specific design choice, which determines the clockwise central cycle $C \rightarrow S \rightarrow S' \rightarrow C' \rightarrow C$ in Figure 4: no matter where interaction starts, we propagate its effects through this cycle to maintain all four structures synchronized using the operators described in the following subsections. Note that edits do not recursively propagate along the aforementioned cycle. Once C is synchronized, the system has reached its new convergence state and is ready for the subsequent input from the user.

4.1. Cage updater

The transition $S' \rightarrow C'$ corresponds to the algorithm denoted by CageUp in Figure 4. When the skeleton S' is modified, we skin M to obtain M' and update C' accordingly. More precisely, we seek a cage C' that generates a skin as close as possible to M' , according to the (static) barycentric coordinates Φ .

As already observed in Section 4, this problem may (and in general does not) admit an exact solution. To avoid solving a least squares problem of the size of M' , we apply the *MaxVol* relaxation proposed in [TTB12]. During pre-processing, we extract a subset \tilde{M} of vertices of M with the same cardinality of the vertices of C . Vertices are selected so as to result in a matrix of coordinates with the highest volume. Then, we consider the corresponding set \tilde{M}' in the current mesh M' , and we solve the linear system:

$$\tilde{\Phi}C' = \tilde{M}', \quad (8)$$

where $\tilde{\Phi}$ is the submatrix of Φ corresponding to the vertices of \tilde{M} . Note that this is a square system having same size of C' (which is assumed to be much smaller than M'). The system is invertible and remains fixed throughout; we factorize it once and efficiently solve it with back-substitution in real-time. Besides performances, as shown in [TTB12], the resulting fitted cages are also more stable than the cages fitted to the full geometry using a least squares approach. Also consider that the purpose of the CageUp is not to best reconstruct the mesh as a function of the fitted cage, but rather to provide the user with a stable cage that nicely envelopes it and aids interaction, therefore the use of the relaxation is appropriate.

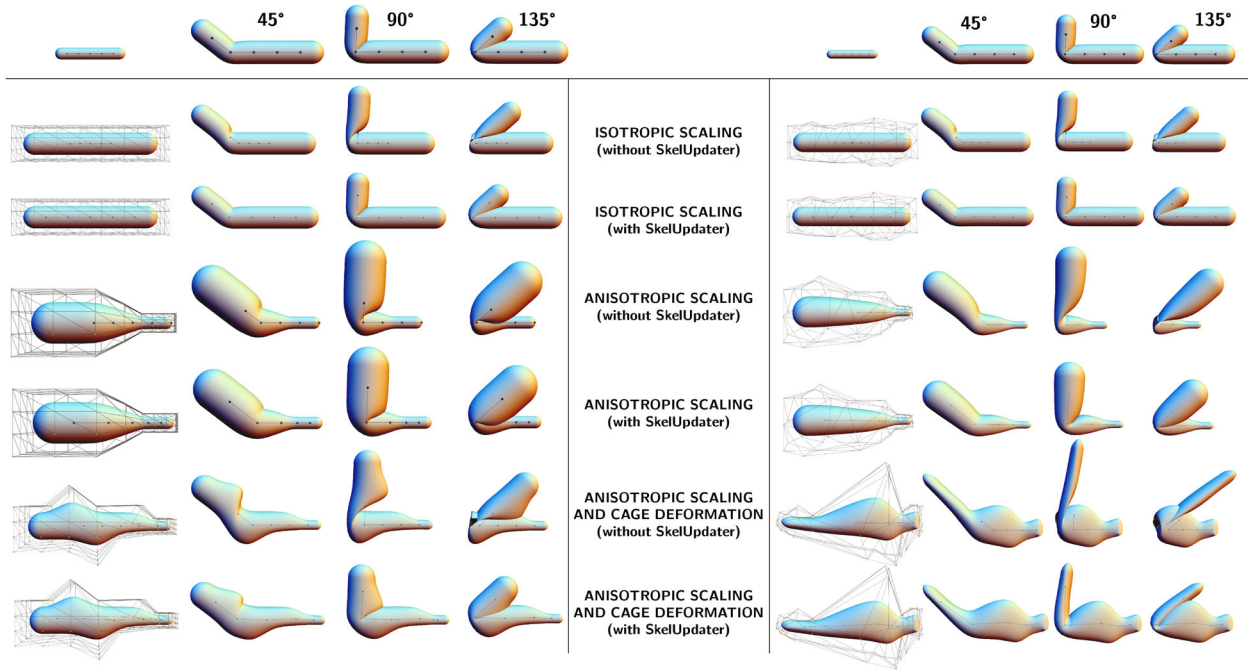


Figure 5: A straight bar bent at 45, 90 and 135 degrees using LBS, on top of which we applied various skeleton edits (isotropic and anisotropic scaling, single handle displacement). When the skeleton updater is disabled, cage edits move the skeleton away from the correct CoR, generating various visual artefacts. Our bone positioning system correctly recovers from all configurations, producing visually plausible deformations. Note that the system is not sensitive to the specific cage being used, and produces valid results both with a regular (left) and irregular (right) cage.

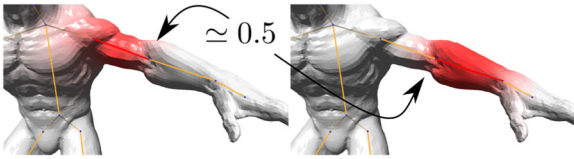


Figure 6: Weights are typically close to 1 around the middle of the bones, resulting in highly rigid transformations in these parts of the mesh, while they blend most with the other weights near the articulations.

4.2. Skeleton updater

The transition $C \rightarrow S$ corresponds to the algorithm denoted by SkelUp in Figure 4. When a deformation of C is performed – either directly or as the result of deforming cage C' – we update the skeleton at rest pose and propagate it down the skinning pipeline. Indeed, when the cage stretches limbs or creates a bulge on the skin, the position of the skeleton joints, which act as CoR during skinning, must be repositioned to avoid artefacts (Figure 5). In other words, we need this to preserve the semantic relation between the skeleton joints and their position relative to the skin.

We address this issue by introducing a new relation between the cage vertices and the position of skeleton joints. This relation is computed once and for all in pre-processing, and allows to express the position of skeleton joints at rest pose as a linear combination

of the cage vertices, giving us the ability to readily update/refit the skeleton in real-time. Note that this update is not limited to a simple global registration, but rather can change the local geometry of the skeleton (e.g. stretching/shrinking its bones).

Skeleton updater weights. All computations in the following are performed only once in the initial pose and involve $C \equiv C_0$ and $S \equiv S_0$. We first identify to which mesh vertices a joint j corresponds to, by defining a (discrete) joint localization function $L_{j,\cdot}^\Omega$ for each joint of S . These functions depend only on the skinning weights Ω and are defined vertex-wise on M as follows:

$$L_{j,i}^\Omega = -1 + \omega_{i,j}^s + \left(\sum_{k \neq j} \omega_{i,k} \right)^s, \quad (9)$$

with $s \ll 1$ (we use $s = 0.1$ in our implementation). Function $L_{j,\cdot}^\Omega$ takes value 0 in rigid regions (i.e. for $\omega_{i,j} = 1$ and $\omega_{i,k} = 0$) and larger values as $\omega_{i,j}$ approaches 0.5, i.e. near the joint, where the skinning weights blend the most (Figure 6).

Next, we use our joint localization functions to define barycentric coordinates for the joints positions $\{a_j\}$ w.r.t. the cage, and exploit them to transform the joints along with the skin with cage deformation. Specifically, we first compute mean value coordinates $\{mvc_{j,i}\}_i$ for the joints rest pose locations a_j w.r.t. the input *mesh*, which we localize around the articulation using the localization

function. Note that the resulting weights $\{\mathbf{L}_{j,i}^\Omega\}_i := \{\text{mvc}_{j,i} * \mathbf{L}_{j,i}^\Omega\}_i$ are not valid barycentric coordinates at this step. The joints barycentric coordinates are then defined as:

$$\psi_{j,\cdot} = \text{MEC}\left(\sum_i \mathbf{L}_{j,i}^\Omega \phi_{i,\cdot}\right), \quad (10)$$

where the free index \cdot is varying over the vertices of C and MEC denotes the projection of input masses in \mathbb{R}^c to the set of valid barycentric coordinates for a_j , i.e. verifying linear precision: $a_j = \sum_k \psi_{j,k} c_k$ and partition of unity: $\sum_k \psi_{j,k} = 1 \forall j$ – and closest to the input masses as the output barycentric coordinates maximize the cross entropy, following the strategy introduced by Hormann and Sukumar [HS08]:

$$\begin{aligned} \text{MEC}(\{m_k\}) &:= \arg\max_{\{b_k\}} \sum_k -b_k \ln(b_k / m_k), \\ \text{s.t. } &\begin{cases} \sum_k b_k \cdot c_k = a_j \\ \sum_k b_k = 1. \end{cases} \end{aligned} \quad (11)$$

One may see this construction as deriving barycentric coordinates for the input articulations A w.r.t. the input cage C_0 through the combination of (i) the input cage coordinates and (ii) the localization function derived from the input skinning weights, which allows us expressing (once fixed) the articulations as a linear combination of the cage vertices, since

$$A = \text{MEC}(\mathbf{L}^\Omega \cdot \Phi) \cdot C := \Psi \cdot C, \quad (12)$$

where $\text{MEC}(\cdot)$ is computed here per line j for each joint j with constrained rest-pose a_j independently. This construction presents several advantages. In particular, we make no assumption over the set of input coordinates Φ , input skinning weights Ω or quality of the input mesh. Additionally, the construction is intuitive, since one can simply edit the localization function $L_{i,j}^\Omega$ as an ad-hoc set of weights allowing for the reconstruction of the joint position as a combination of the mesh vertices. Moreover, it is highly efficient and parallelizable.

Skeleton joints refitting. When C is deformed, we update the joints of S at positions $\{a_j\}$ as a linear combination of cage vertices with

$$a_j = \sum_{k=1}^c \psi_{j,k} \mathbf{c}_k, \quad (13)$$

where the $\psi_{j,k}$ are the barycentric coordinates described earlier. Note that, after the joints have been relocated, the length and orientation of the bones in the skeleton have been changed; these changes must be reflected on the current skeleton S' and, consequently to the C' and M' . To trigger these changes, we update each skinning transformation \mathbf{T}_j of \mathcal{T} by keeping its rotational component \mathbf{T}_j^r unchanged and by recomputing its translational component \mathbf{T}_j^t according to the new joints rest-pose locations. We do so by simply following the hierarchical structure of the skeleton, updating first all roots, and then processing children in an iterative manner, so as to preserve the

iteratively deformed skeleton articulations. The effect of updating \mathcal{T} , hence S' , propagates down through standard skinning, and the cage updater described previously.

CoRs repositioning. As already discussed, the CoR method makes the use of per-vertex CoR p_i for vertices i , which are precomputed following Equation (5). Since a manipulation of the cage deforms the rest pose state for the skeletal deformation, we have to reposition the CoRs as well. Fortunately, those are defined as a linear combination of the mesh rest pose vertices M . Rewriting Equation (5) in matrix form as $\text{CoRs} = \Phi_{\text{CoRs}} \cdot M$, and using the fact that the rest pose mesh M is expressed as $M = \Phi \cdot C$, we can precompute the matrix $\Lambda := \Phi_{\text{CoRs}} \cdot \Phi$ when computing the CoR, and reposition them at run time using

$$\text{CoRs} = \Lambda \cdot C. \quad (14)$$

Doing so, we assume that the area terms in the surface averaging remain similar (see Equation 5). In fact, this introduces slight differences between the CoRs we obtain after a cage deformation of the rest pose mesh, and the ones one can obtain when recomputing them from scratch, every time the rest pose mesh is changed. However, these differences are minor, and do not impact negatively the quality of the resulting deformation (Figure 2). In particular, the vertices with similar input skinning weights are still transformed by the same rigid transformation. Lastly, our joints repositioning method is highly compatible in spirit with the CoR method, as both motivate the use of the cross-analysis of the mesh geometry and the skinning weights in the derivation of optimal pivot positions.

4.3. Reverse cage deformer

The transition $C' \rightarrow C$ corresponds to the algorithm denoted by CageRev in Figure 4. While the user interacts with C' , we must reflect any modification $\partial C'$ with a corresponding modification ∂C of the rest cage C , and such modification must maintain the framework consistent. To do so, we express the generic modification $\partial C'$ as a function of ∂C through the sequence $C \rightarrow S \rightarrow S' \rightarrow C'$, and we finally reverse this function. To obtain a linear problem and achieve efficiency, we compute the above function by assuming LBS throughout. We will discuss at the end of the section how to handle other types of skinning methods.

To exploit matrix computation, we linearize all our structures. For the sake of clarity and with abuse of notation for this section only, we use the same symbols as before to denote the linearized structures. We denote $C, C' \in \mathbb{R}^{3c}$ the vectors stacking the vertices of the rest and current cage, respectively; and we denote Ω, Φ, Ψ the matrices computed as the Kronecker products between their respective matrices as defined previously and the Identity matrix I_3 (note that the linearized matrices have sizes $3m \times 3s, 3m \times 3c$ and $3s \times 3c$, respectively). Moreover, we explicitly represent the skinning components as follows. We denote $A \in \mathbb{R}^{3s}$ the vector stacking the current articulations of the skeleton. And we split the skinning rotational and translational component as follows: we denote \mathcal{R} a $3m \times 3m$ matrix composed of m 3×3 matrices on the diagonal (block i is the linear part applied to vertex i – for LBS, $R(i) = \sum_j \omega_{i,j} \mathbf{T}_j^r$), and T the $3s$ vector stacking all translation

parameters \mathbf{T}'_j , where, as previously, \mathbf{T}^r_j and \mathbf{T}^t_j are the translation and rotation components of \mathbf{T}_j , respectively. Therefore, the dynamic rest pose skin M , the current skin M' and the current cage C' are obtained by the following formulas:

$$\begin{cases} A = \Psi \cdot C \\ \tilde{M} = \tilde{\Phi} \cdot C \\ \tilde{M}' = \tilde{\mathcal{R}} \cdot \tilde{M} + \tilde{\Omega} \cdot T \\ C' = \tilde{\Phi}^{-1} \cdot \tilde{M}', \end{cases} \quad (15)$$

where, as in Section 4.1, $\tilde{\cdot}$ is used to denote all quantities that require only the subset of mesh vertices selected by MaxVol (the cage updater requires the transformed position of those vertices only). We can observe that the third equation is nothing but a matrix expression of Equation (1), where the rotational and translational components of the transformation at each vertex have been separated, following Equation (3).

We aim at computing offsets ∂C to apply to the rest cage so as to obtain a resulting offset $\partial C'$, which the user wishes to apply to the current cage. Before pursuing the derivation, we stress that the set of Equations (15) is not sufficient for that purpose as, in fact, applying offsets to the skeleton articulations A results in changes in the skeletal deformation parameters T , since the translation parameters are affected to preserve the skeleton connectivity.

Relating joint offsets and skeletal deformations. Following [TE18], we note that applying an offset to an articulation a_j results in offsets applied to the translations $\{\mathbf{T}'_k\}$ in the following manner:

$$\begin{cases} \mathbf{T}'_j \cdot (a_j + \partial a_j) + \mathbf{T}'_j + \partial \mathbf{T}'_j = a_j + \partial a_j & \text{if } j \text{ is a root} \\ \mathbf{T}'_j \cdot \partial a_j + \partial \mathbf{T}'_j = \mathbf{T}'_f \cdot \partial a_j + \partial \mathbf{T}'_f & \text{if } j \text{ has father } f, \end{cases} \quad (16)$$

the first equation simply means that the pivot point is updated accordingly, and the second equation simply means that the joint j has to be preserved by the transformations of handle j and its father f both, under preservation of the linear parts $\{\mathbf{T}'_k\}$ of the skeletal deformation T . This system can be rewritten as:

$$\begin{cases} (I_3 - \mathbf{T}'_j) \cdot \partial a_j = \partial \mathbf{T}'_j & \text{if } j \text{ is a root} \\ (\mathbf{T}'_j - \mathbf{T}'_f) \cdot \partial a_j = \partial \mathbf{T}'_f - \partial \mathbf{T}'_j & \text{if } j \text{ has father } f, \end{cases} \quad (17)$$

or, in matrix expression

$$\mathcal{A}_R \cdot \partial A = \mathcal{B}_{\text{topo}} \cdot \partial T. \quad (18)$$

We note that, while \mathcal{A}_R depends on the current skeletal deformation parameters (the linear part $\{\mathbf{T}'_k\}$), $\mathcal{B}_{\text{topo}}$ depends on the topology of the skeleton only, and can safely be inverted once and for all, independently of the current skeletal deformation parameter set.

Relating interaction and deformation cage offsets. Finally, we can gather the previously derived equations to obtain the $c \times c$ linear system:

$$(\tilde{\mathcal{R}} \cdot \tilde{\Phi} + \tilde{\Omega} \cdot \mathcal{B}_{\text{topo}}^{-1} \cdot \mathcal{A}_R \cdot \Psi) \cdot \partial C = \tilde{\Phi} \partial C' \quad (19)$$

that can be resolved efficiently. Note that when no skeletal deformation is performed, i.e. the linear part $\{\mathbf{T}'_j\}$ is composed of Identity matrices only and all translations $\{\mathbf{T}'_j\}$ are null, the current cage C' and rest cage C match (as they should), since \mathcal{R} is then the identity matrix and \mathcal{A}_R is null.

Impact of the MaxVol relaxation. Note that using a subset of the mesh vertices in the cage fitting has several important consequences: firstly, all matrices in Equation (19) have dimensions bounded by $\max(3c, 3s)$, which results in updates that can be performed in real time on the examples we used, *these timings being in this case entirely independent from the mesh size*. Secondly, by matching the dimensionality of the cage and mesh used for inversion, the system in Equation (19) is exactly invertible, and the loop $C' \rightarrow C \rightarrow S \rightarrow S' \rightarrow C'$ is *exact*.

We originally tried using all vertices in the inversion process, resulting in an approximate loop. While the approximation was extremely subtle and unnoticeable, the biggest issue was that it resulted in reduced performance: we could not obtain results that were fast enough for a modelling session on large models, as the user had to wait a few seconds when switching from skeleton manipulation to cage manipulation. Note that, if desired, the user could still rely on more vertices than just the ones selected by MaxVol (a good strategy could be to use farthest sampling in the space of cage coordinates of the vertices, as done in [JBK*12] – Section 3.3); the construction described in this section would remain valid, but the inverse of the matrices would have to be replaced by pseudo-inverses and the loop would be only approximate (rigorously, the manipulation cage C' is then obtained by least-squares fitting as $(\tilde{\Phi}^T \cdot \tilde{\Phi}) \cdot C' = \tilde{\Phi}^T \cdot M'$, which leads to a modified Equation (19) where both terms are multiplied by $\tilde{\Phi}^T$ on their left).

Handling skeletal deformation methods other than LBS. As already emphasized earlier, the reverse cage deformer operator assumes LBS as the skinning deformation method. This will give us indeed an exact result if LBS is the current skinning method, and an approximated result with the other skinning methods. However, since the reverse cage deformer is always applied to small incremental modifications $\partial C'$, the approximation error is negligible and hardly noticeable during user interaction. Another possibility is to make the use of a ghost mesh that is deformed with LBS and use this ghost mesh to derive the fitting of the interaction cage in the cage updater, regardless of the actual final deformation method. The cage updater CageUp is then not optimal, in the sense that it best fits an LBS deformation of the mesh instead of the actual deformed mesh, but the loop $C' \rightarrow C \rightarrow S \rightarrow S' \rightarrow C'$ is always exact. Note that only the vertices selected for inversion need to be deformed in the ghost mesh, so this step is negligible in practice. Both options are satisfactory and trivial to implement, and will work well for all skinning methods producing deformations that resemble LBS at large scale (as illustrated in our results featuring DQS and CoRs) since the cage fitting performed by CageUp is a *global* operation as a result of using global cage weights.

5. Results

We have implemented our modelling framework as a single-threaded C++ program. Models have been either manually

Table 1: Performances of our modelling system, measured on a MacBook pro early-2015 equipped with an Intel i5 processor, 8GB of RAM and Intel Iris 6100 GPU.

Model	Verts	Skel joints	Cage handles	SkelUp preprocess <i>ms</i>	CageUp preprocess <i>ms</i>	CoR update <i>ms</i>	CageRev update <i>ms</i>	LBS frame <i>ms</i>	DQS frame <i>ms</i>	CoR frame <i>ms</i>
Arm (coarse cage)	2089	24	28	401	5	≤ 1	3	0.88	1.62	2.26
Arm (medium cage)	2089	24	58	417	29	≤ 1	20	1.61	1.53	1.63
Arm (fine cage)	2089	24	79	439	45	≤ 1	40	2.22	1.84	2.17
Warrok	6557	64	104	2672	106	≤ 1	83	7.81	7.66	9.51
Ely	7512	64	88	3018	75	≤ 1	56	9.24	8.73	9.99
Airplane	41 425	19	69	3738	44	5	30	23.70	21.68	26.40
Timber Rattlesnake	120 066	98	44	46 771	32	9	16	50.03	39.43	52.35

Columns labelled **SkelUp preprocess** and **CageUp preprocess** refer to the pre-processing time of the SkelUp and CageUp operators, respectively. The **CoR update** column reports the time necessary to update the centres of rotation each time SkelUp is executed (this update does not occur when LBS or DQS is used). **CageRev update** reports the execution time of the CageRev operator, when the user switches from skeleton manipulation to cage (in current pose) manipulation during a modelling session. This few-milliseconds latency is observed only when the user grabs the cage and not during cage manipulation after that, as all necessary matrix factorizations do not need being updated as long as the skeleton is untouched. Finally, the last three columns report the cost of updating the current pose with the various skinning methods implemented in the framework (the cost of rendering is not taken into account here). Note that all the timings we report refer to a CPU implementation. Moving to GPU should dramatically improve our performances. Also note that in our examples DQS seems to outperform LBS. While this is not true in the general case, in our codebase we used a carefully optimized implementation of DQS, as opposed to a naive implementation of LBS. Therefore, these numbers are strictly dependent on our specific software prototype.

crafted or downloaded from online repositories, such as Adobe Mixamo [mix19] and SketchFab [sf19]. Whenever a control cage was not provided in the original dataset, we used the method proposed in [CLM*19] to produce one. In case the skinning was missing, we manually created one using Maya [may19]. In terms of performances, our hybrid modelling system introduces only negligible overhead with respect to the classical skeleton- and cage-based pipelines, and for moderately complex characters runs in real time with high frame rate even on commodity hardware (Table 1).

Deformation options. A key feature of our hybrid deformation system is its ability to scale across multiple methods for skeleton- and cage-based deformation, which can therefore be chosen from practitioners depending on their taste and needs. For the skeleton part, we implemented the two most popular skeleton-based deformation methods, namely LBS [MTLT89], DQS [KCZO08], and the recently introduced CoR [LH16], which combines the positive aspects of the previous two and at the same time avoids their weak points (volume loss for LBS and bulging for DQS). A side-by-side comparison between these three alternatives is shown in Figure 2. For the skinning weights, although various automatic methods exist in the literature (Section 2), industry-level deformations often involve carefully designed weights that are manually painted on the surface by skilled artists. Our system is agnostic on the specific weights of choice, and transparently supports both automatic and manual approaches. Rigs are imported into the system using standard formats (i.e. FBX), securing an easy interface with commercial software and publicly available repositories. For the cage part, we used the mean value coordinates [JSW05] in all our tests, which are internally computed by our framework. Similarly to skeleton weights, alternative barycentric coordinates that obey the linear blend of Equation (7) can be loaded into the system and used in a transparent way. To the best of our knowledge, this includes the vast majority of the known barycentric coordinates that appeared in the literature, including

the recently proposed coordinates for quad cages [TMB18]. Two notable exceptions are the green coordinates [LLCO08] and the variational harmonic maps [BCWG09], which both use a blend equation that involves mesh vertices and face normals, and are therefore not directly applicable to our linear deformation paradigm.

Skeleton updater. In Figure 5, we evaluate our skeleton updater with a synthetic shape, consisting of a straight bar bent at 45/90/135 degrees using LBS. Editing the bar with the cage moves the skeleton away from the correct CoR, and without the skeleton re-fitting procedure described in Section 4.2, extremely evident artefacts arise. Our bone positioning system correctly recovers from all configurations, producing visually plausible deformations. Note that the system is not sensitive to the specific cage being used and produces valid results both with a regular (left) and irregular (right) cage.

Scale adaptivity. Skeletons and cages may be very different to one another, and are indeed able to control features of the same object at different scales. Our system is able to seamlessly combine skeletons and cages that operate at different levels of detail. In Figure 7, a simple bent arm controlled by a skeleton is further edited with three alternative cages. The coarse cage controls the whole hand and is used to enlarge it; the medium cage allows to selectively edit each finger and is used to thicken the thumb; the dense cage contains various control points around the bicep and is used to inflate it when the arm is bent. All three hybrid deformations are visually plausible and difficult to replicate by acting solely on a skeleton or a cage.

Hybrid modelling. The principal intent when designing our deformation framework was to offer artists a unique system where they could seamlessly combine multiple deformation paradigms.

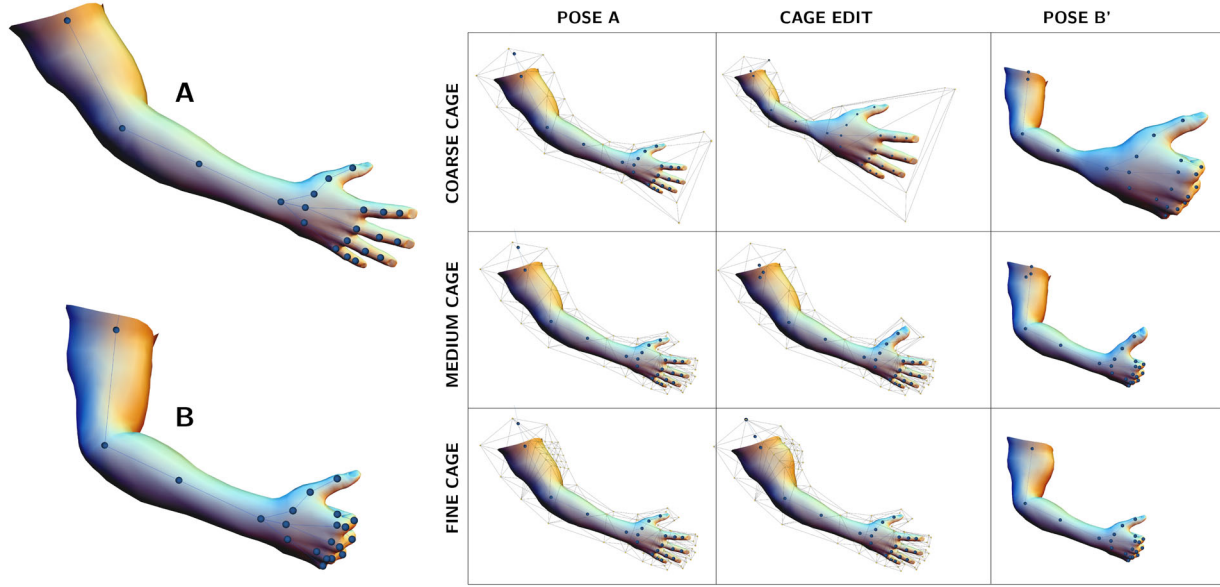


Figure 7: Our method can seamlessly combine edits defined on skeletons and cages that operate on features at different scales. Here, a simple arm bent with a skeleton (left) is enriched with additional edits with three alternative cages that operate at different levels of details (right). The coarse cage controls the whole hand, and is used to enlarge it; the medium cage allows to selectively edit each finger, and is used to thicken the thumb; the dense cage has many control points around the bicep, and is used to inflate it. All the three cages produce visually plausible deformations that are difficult to replicate by acting solely on the skeleton or on the cage.

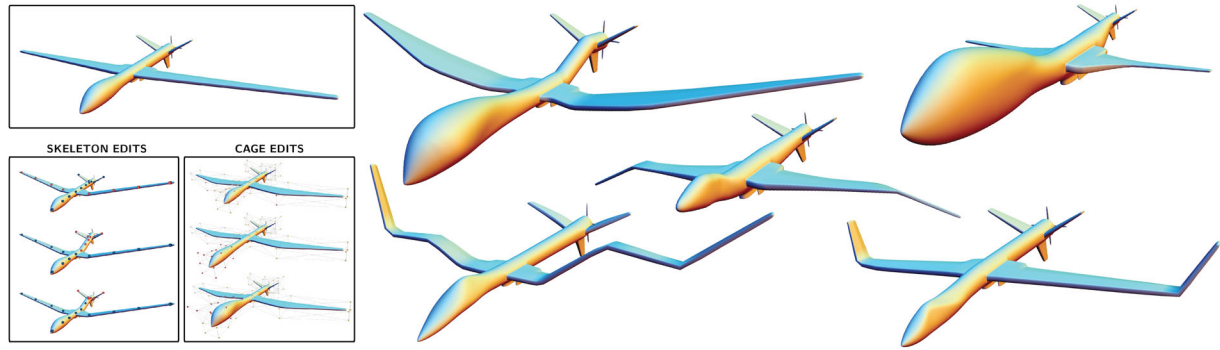


Figure 8: Jointly acting on skeletons and cages allows to easily control complementary aspects of the modelling and explore the space of shapes starting from a simple example (top left). Skeletons are best to bend tubular parts and, more in general, deform the rigid components of a shape. Cages are more appropriate to control locally smooth deformations, such as changes of the local thickness of the airplane or the profile of its wings.

Starting from an input shape linked to a skeleton and a cage, artists can explore the space of deformations to create a family of similar objects, using the more appropriate tool for each edit. An example of hybrid modelling is given in Figure 8, where several variations of an airplane are produced from a single item. Skeleton bones are used to control the rigid parts of the plane (e.g. to bend the core and wings). Cage handles are used to apply local volumetric deformations, for example to locally inflate parts of the core or to edit the profile of the wings.

Hybrid animation. Another interesting application of our framework consists of using the various shapes it produces as keyframes,

to guide a computer-generated animation sequence. In Figure 9, we show a few interpolated frames of an animation, obtained by keyframing some of the airplanes shown in Figure 8. In between frames are generated interpolating bone rotations with Slerp [DKL98] and cage vertices linearly. Note that the *deformation* cage C is keyframed, not the manipulation cage C' . The skeleton updater is, therefore, required at each reconstruction step (but the update is extremely fast as it is linear in the number of skeleton joints only), but the cage reverse updater is not involved in the process. Following a similar approach, legacy animations can be enriched with new effects. In Figure 10, we show a skeleton-driven punch sequence downloaded from Adobe Mixamo [mix19], which we

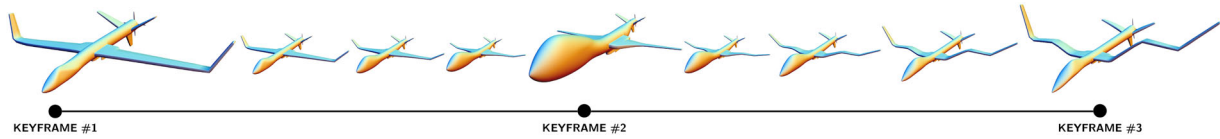


Figure 9: Using deformed models as keyframes, we can create computer animations. Bone rotations are interpolated using Slerp [DKL98], and cage edits are linearly interpolated.

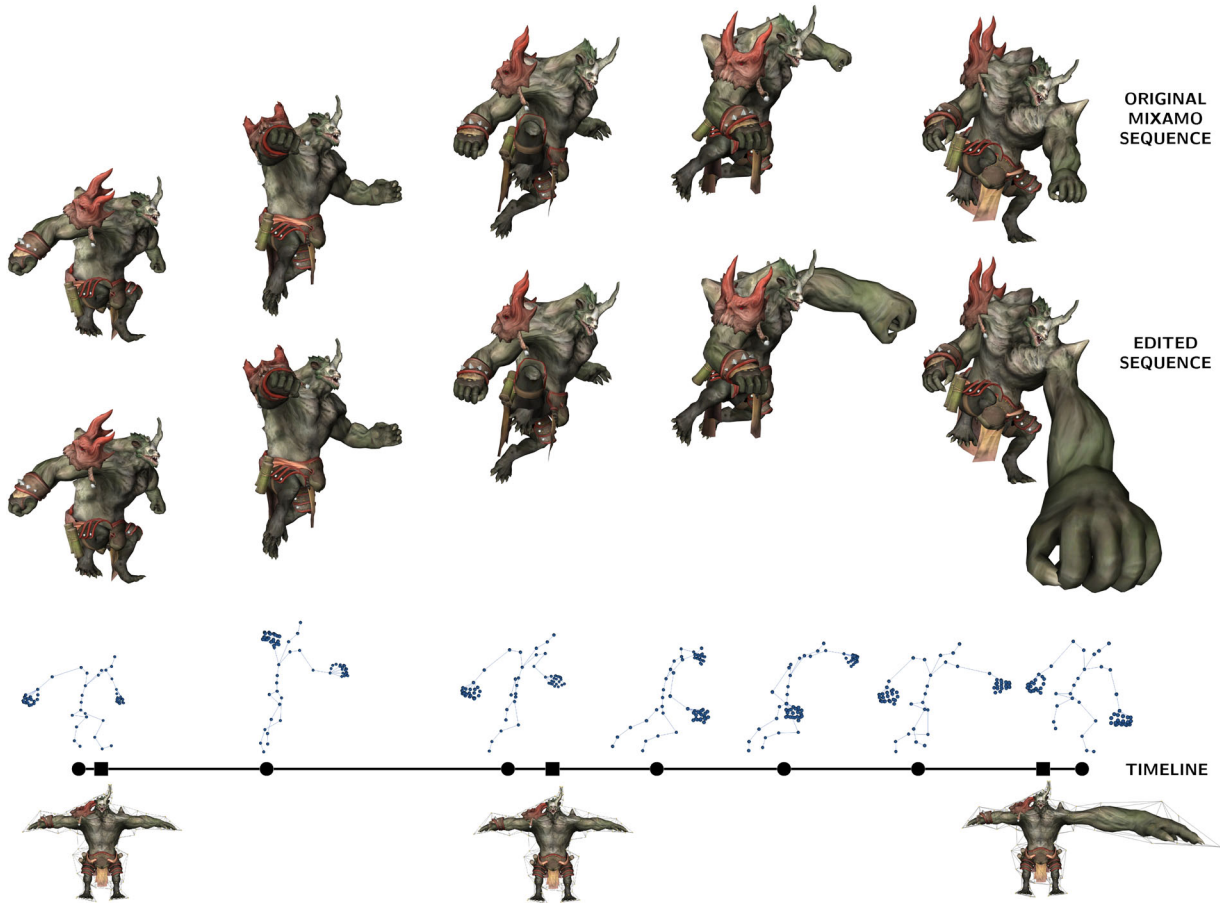


Figure 10: Top: A legacy skeleton-based animation downloaded from Adobe Mixamo [mix19]. Middle: An edited sequence obtained by inflating the punch with a control cage. Bottom: The animation timeline, with both skeleton keyframes (circles, from Mixamo) and cage keyframes (squares). The first two cage keyframes were automatically generated with [CLM*19], and simply enclose the rest pose; the third one was manually edited to inflate the punch. Editing a single keyframe we produced a new sequence containing a non-trivial twist. Note also that this example exhibits regions where the skeleton is much more detailed than the cage and vice-versa: while the hands embed a highly detailed bone structure allowing animating all the fingers and the cage around them resemble essentially paws, the belly contains a few bones only to mimick a simple spine behaviour, while the cage around it is finely detailed to allow for precise anisotropic volume editing.

enriched with three cage keyframes that inflate the punch at the proper time. Note that a minimal workload is already enough to incorporate in the animation with new interesting effects. In this specific case, only one manually edited keyframe was used, and the other two are simple envelopes of the rest pose, computed

with [CLM*19]. Also note that skeletons and cage keyframes are interpolated asynchronously, hence can work on the same character independently and at different levels of details. Similar results are also shown in Figure 11. We point the readers to the accompanying video to see the actual animations we obtained.



Figure 11: Our framework allows animating and deforming characters simultaneously. In the top row, we show the Mixamo's Ely character, with a cage we added on the left, and three frames of the walking sequence. In the bottom row, we deformed the same character fattening him (notice the changed cage on the left), and performed the same walking animation. Combining skeleton and cage controls, we can fatten the character while it walks as we show in the accompanying video.

6. Conclusion and Future Work

We started from the observation that skeleton-based and cage-based deformations control different aspects of shape modelling, and are to a large extent complementary to one another. We, therefore, proposed a real-time modelling framework based on a novel paradigm that seamlessly combines these structures. We obtained the desired effect by adopting the concept of rest pose and current pose for both skeletons and cages, introducing novel update operators that realize the sync between all these structures. As a result, we operate in a larger deformation space, containing poses that are impossible to obtain by acting solely on a skeleton or a cage.

To grant reproducibility of our results, we publicly release the source code of our framework on GitHub (<https://github.com/cordafab/SuperCages>). It is back-compatible with most existing techniques for skeleton-based and cage-based deformation. The only limitation in this sense comes from our assumption of a linear equation for cage-based shape editing (Equation 7). From a technical standpoint, non-linear cage-based deformation techniques, such as [LLCO08], could potentially be incorporated in the system, though at the cost of having more complex algorithms to maintain the sync. Similar considerations can be done for partial cages, such as the ones proposed in [GPCP13]. We did not perform tests in these directions yet, but it would be interesting to check how this will affect the frame rate and the real-time experience.

Regarding user interaction, the visualization of controllers for both the skeleton and cage may sometimes clutter the screen, especially for complex characters requiring numerous skeleton bones and complex control cages. We envisage a potential improvement by adopting a dynamic rendering of the controllers that fades away from the mouse position.

We believe that our contribution could support advanced deformation control. For future works, we plan to provide a GPU implementation, realize a user evaluation with skilled artists and extend the framework with more controllers (e.g. point handles), which we can incorporate with the same approach to synchronization, while remaining oblivious on the specific technique used for their implementation.

References

- [BCBiR*15] BANG S., CHOI B., BLANCO I RIBERA R., KIM M., LEE S.-H., NOH J.: Interactive rigging with intuitive tools. *Computer Graphics Forum* 34 (2015), 123–132.
- [BCWG09] BEN-CHEN M., WEBER O., GOTSMAN C.: Variational harmonic maps for space deformation. *ACM Transactions on Graphics (TOG)* 28, 3 (2009), 34.
- [BL18] BANG S., LEE S.-H.: Spline interface for intuitive skinning weight editing. *ACM Transactions on Graphics (TOG)* 37, 5 (2018), 174.
- [Ble18] BLENDER ONLINE COMMUNITY: *Blender — A 3D Modelling and Rendering Package*. Blender Foundation, Blender Institute, Amsterdam, 2018.
- [BP07] BARAN I., POPOVIĆ J.: Automatic rigging and animation of 3D characters. *ACM Transactions on Graphics* 26, 3 (2007), 72.
- [CLM*19] CASTI S., LIVESU M., MELLADO N., ABU RUMMAN N., SCATENI R., BARTHE L., PUPPO E.: Skeleton based cage generation guided by harmonic fields. *Computers & Graphics* (2019). <https://doi.org/10.1016/j.cag.2019.04.004>.
- [DdL13] DIONNE O., DE LASA M.: Geodesic voxel binding for production character meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2013), ACM, pp. 173–180.
- [DKL98] DAM E. B., KOCH M., LILLHOLM M.: *Quaternions, Interpolation and Animation*, vol. 2. Citeseer, 1998.
- [GPCP13] GARCÍA F. G., PARADINAS T., COLL N., PATOW G.: *Cages:: A multilevel, multi-cage-based system for mesh deformation. *ACM Transactions on Graphics* 32, 3 (2013), 24:1–24:13.
- [GSMCO09] GAL R., SORKINE O., MITRA N. J., COHEN-OR D.: iWIRES: An analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics* 28 (2009), 33.
- [HS08] HORMANN K., SUKUMAR N.: Maximum entropy coordinates for arbitrary polytopes. *Computer Graphics Forum* 27 (2008), 1513–1520.

- [JBK*12] JACOBSON A., BARAN I., KAVAN L., POPOVIĆ J., SORKINE O.: Fast automatic skinning transformations. *ACM Transactions on Graphics* 31, 4 (2012), 77.
- [JBPS11] JACOBSON A., BARAN I., POPOVIĆ J., SORKINE O.: Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics* 30 (2011), 78.
- [JDKL14] JACOBSON A., DENG Z., KAVAN L., LEWIS J.: Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses* (2014).
- [JMD*07] JOSHI P., MEYER M., DEROSE T., GREEN B., SANOCKI T.: Harmonic coordinates for character articulation. *ACM Transactions on Graphics* 26 (2007), 71.
- [JS11] JACOBSON A., SORKINE O.: Stretchable and twistable bones for skeletal shape deformation. *ACM Transactions on Graphics* 30, 6 (2011), 165.
- [JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics* 24 (2005), 561–566.
- [JWS12] JACOBSON A., WEINKAUF T., SORKINE O.: Smooth shape-aware functions with controlled extrema. *Computer Graphics Forum* 31 (2012), 1577–1586.
- [JZvdP*08] JU T., ZHOU Q.-Y., VAN DE PANNE M., COHEN-OR D., NEUMANN U.: Reusable skinning templates using cage-based deformations. *ACM Transactions on Graphics* 27, 5 (2008), 122:1–122:10.
- [KČŽO08] KAVAN L., COLLINS S., ŽÁRA J., O’SULLIVAN C.: Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics* 27, 4 (2008), 105.
- [KS12] KAVAN L., SORKINE O.: Elasticity-inspired deformers for character articulation. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH ASIA)* 31, 6 (2012), 196:1–196:8.
- [LH16] LE B. H., HODGINS J. K.: Real-time skeletal skinning with optimized centers of rotation. *ACM Transactions on Graphics* 35, 4 (2016), 37.
- [LKCOL07] LIPMAN Y., KOPF J., COHEN-OR D., LEVIN D.: GPU-assisted positive mean value coordinates for mesh deformations. In *Symposium on Geometry Processing* (2007).
- [LLCO08] LIPMAN Y., LEVIN D., COHEN-OR D.: Green coordinates. *ACM Transactions on Graphics* 27, 3 (2008), 78.
- [may19] Autodesk Maya: <https://www.autodesk.com/products/maya/overview> (2019). Last accessed on 27 January 2020.
- [mix19] Adobe Mixamo dataset: <https://www.mixamo.com> (2019).
- [MK16] MUKAI T., KURIYAMA S.: Efficient dynamic skinning with low-rank helper bone controllers. *ACM Transactions on Graphics* 35, 4 (2016), 36.
- [MTLT89] MAGNENAT-THALMANN N., LAPERRIÈRE R., THALMANN D.: Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface’88* (1989), Canadian Information Processing Society, pp. 26–33.
- [NS13] NIETO J. R., SUSÍN A.: Cage based deformations: A survey. In *Deformation Models. Lecture Notes in Computational Vision and Biomechanics*. M. González Hidalgo, A. Mir Torres and J. Varona Gómez (Eds.). Springer, Dordrecht (2013), vol. 7, pp. 75–99.
- [OBP*13] ÖZTIRELI A. C., BARAN I., POPA T., DALSTEIN B., SUMNER R. W., GROSS M.: Differential blending for expressive sketch-based posing. In *Proceedings of the 2013 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2013), SCA ’13, ACM, pp. 155–164.
- [sf19] Sketchfab: <https://sketchfab.com> (2019).
- [SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. *ACM SIGGRAPH Computer Graphics* 20, 4 (1986), 151–160.
- [TE18] THIERY J.-M., EISEMANN E.: ARAPLBS: Robust and efficient elasticity-based optimization of weights and skeleton joints for linear blend skinning with parametrized bones. *Computer Graphics Forum* 37 (2018), 32–44.
- [TMB18] THIERY J.-M., MEMARI P., BOUBEKEUR T.: Mean value coordinates for quad cages in 3D. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2018)* 37, 6 (2018), 229:1–229:14.
- [TTB12] THIERY J.-M., TIERNY J., BOUBEKEUR T.: Cager: Cage-based reverse engineering of animated 3D shapes. *Computer Graphics Forum* 31, 8 (2012), 2303–2316.
- [TTB13] THIERY J.-M., TIERNY J., BOUBEKEUR T.: Jacobians and Hessians of mean value coordinates for closed triangular meshes. *Visual Computer* 30, 9 (2013), 981–995.
- [WJBK15] WANG Y., JACOBSON A., BARBIČ J., KAVAN L.: Linear subspace design for real-time shape deformation. *ACM Transactions on Graphics* 34, 4 (2015), 57.
- [WL08] WAREHAM R., LASENBY J.: Bone glow: An improved method for the assignment of weights for mesh deformation. In *International Conference on Articulated Motion and Deformable Objects* (2008), Springer, pp. 63–71.
- [YCHK15] YUMER M. E., CHAUDHURI S., HODGINS J. K., KARA L. B.: Semantic shape editing using deformation handles. *ACM Transactions on Graphics* 34, 4 (2015), 86.
- [ZDL*14] ZHANG J., DENG B., LIU Z., PATANÈ G., BOUAZIS S., HORMANN K., LIU L.: Local barycentric coordinates. *ACM Transactions on Graphics* 33, 6 (2014), 188:1–188:12.
- [ZFCO*11] ZHENG Y., FU H., COHEN-OR D., AU O. K.-C., TAI C.-L.: Component-wise controllers for structure-preserving shape manipulation. *Computer Graphics Forum* 30 (2011), 563–572.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Data Movie S1