Università degli Studi di Cagliari

PHD DEGREE

MATHEMATICS AND COMPUTER SCIENCE

Cycle XXXII

TITLE OF THE PHD THESIS

Real-Time Deformation with coupled Cages and Skeletons

Scientific Disciplinary Sector(s)

INF/01 INFORMATICA

PhD Student: Fabrizio Corda

Coordinator of the PhD Programme Prof. Michele Marchesi

Supervisor Prof. Riccardo Scateni

Final exam. Academic Year 2018 – 2019

Thesis defence: January-February 2020 Session

Università degli Studi di Cagliari

Ph.D. Program in Mathematics
and Computer Science

XXXII Cycle

Real-time Deformation with Coupled Cages and Skeletons

Author:
Fabrizio Corda

Ph.D. Supervisor:
Prof. Riccardo Scateni

Final examination academic year 2018 - 2019

–Will I dream?
–Of course you will.
All intelligent beings dream. Nobody knows why

Arthur C. Clarke

Alla mia famiglia

Sommario

La deformazione in tempo reale di personaggi digitali è un compito molto importante nell' animazione computerizzata. Tali deformazioni possono essere ottenute tramite svariate tecniche, ma le più popolari sono quelle legate agli scheletri. Attraverso gli scheletri, è possibile deformare un personaggio muovendo le sue ossa. Altre tecniche, come quelle basate su gabbie, fanno fatica ad essere inserite nei flussi di lavoro degli artisti poiché richiedono un radicale cambiamento nelle linee di produzione delle animazioni.

Questa tesi descrive una tecnica che consente agli animatori di includere deformazioni basate su gabbie nelle line di produzione basate su scheletri. Infatti qui viene formalizzato un ibrido scheletro-gabbia, chiamato **SuperCages**, che permette agli artisti di arricchire il potere espressivo degli scheletri con i gradi di libertà offerti dalle gabbie.

Inoltre, questa tesi descrive anche due interfacce utente dedicate a deformazioni e animazioni. La prima interfaccia è **CageLab** e consente agli artisti di realizzare deformazioni basate su gabbie e modifiche alle gabbie stesse. La seconda interfaccia è **SuperCages GUI** e permette di definire animazioni e deformazioni basate sull'ibrido scheletro-gabbia presentato nella prima parte della tesi.

Abstract

Real-time character deformation is an essential topic in Computer Animation. Deformations can be achieved by using several techniques, but the skeleton-based ones are the most popular. Skeletons allow artists to deform articulated parts of the digital characters by moving their bones. Other techniques, like cage-based ones, are gaining popularity but struggle to be included in animation workflows because they require to change the animation pipeline substantially.

This thesis formalizes a technique that allows animators to embed cage-based deformations in standard skeleton-based pipelines. The described **skeleton/cage hybrid** allows artists to enrich the expressive powers of the skeletons with the degrees of freedom offered by cages.

Furthermore, this thesis describes two Graphical User Interfaces dedicated to deformations and animations. The first one, **CageLab**, allows artists to define cage-based deformations and perform cage editing. The second one, **SuperCages GUI**, allows artists to author animations and deformations by using the skeleton/cage hybrid described earlier.

Contents

Chapter 1

Introduction

Computer Graphics is widely used by all kinds of industries: cinema, videogames, engineering, and so on. Computer Animation is one of the essential branches of Computer Graphics, and it makes it possible to give the idea of movement of three-dimensional characters or objects.

To create an animation, a character and the scene objects must be deformed and moved in the scene. For this reason, interactive shape deformation is a fundamental building block in 3D real-time animation.

To perform deformations, artists and digital animators require powerful tools and techniques. In fact, the choice of a simple and efficient technique can dramatically improve the final graphical result, reducing the artist workload and making this process more intuitive.

There are several kinds of techniques used in the animation pipeline, one of the most common ones is **Skinning**. It is composed of a family of algorithms that allow users to perform deformations on characters by using the so-called **rigging structures** as a deformation driver. By using a rigging structure the user can specify a deformation, and the skinning algorithm transfers that deformation to the character.

Two popular rigging structures are cages and skeletons. A **cage** is an external structure that envelopes the character, and its deformation is obtained by manipulating the cage vertices. Then, the skinning algorithm deforms the character according to the barycentric coordinates defined between its skin and the cage itself. A **skeleton**, instead, is an internal structure composed of bones and joints. In this case, geometric transformations are applied to these, allowing the skinning algorithm to calculate

the character deformation.

This thesis explores several aspects of the deformation process and analyzes possible new techniques and tools for character deformations. The thesis is divided in four parts.

In the **first part**, the **background concepts** related to the animation pipeline are discussed. In fact, in Chapter 2, skinning algorithms, skeletons and cages are described.

In the **second part** of this thesis, starting from Chapter 3, a **novel approach to combine skeleton and cages** that allows users to use both of them at the same time is described.

Cages and skeletons have different expressive powers and distinct positive and negative aspects. Skeletons are more suited to deform rigid parts and pose articulated bodies. Cages, instead, are better for smooth volumetric deformations.

The described approach aims to create a pipeline that combines the strengths of both techniques. 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, obtaining good graphical results, and finding consensus between the deformations they induce is still open. Having this motivation in mind, a hybrid structure that merges the expressive powers of skeletons with the ones of cages is defined in this thesis. The hybrid structure is called **SuperCages**, and it is a framework composed of three operators and a pipeline that allows bridging skeleton deformations with cage-based ones.

The final goal is to obtain a new kind of structure that improves the expressiveness of the classical rigging structures, by proposing an "enriched" deformation pipeline that guarantees flexibility and back-compatibility with pre-existing rig structures. The new enriched structure unifies the expressive powers of skeletons and cages, allowing artists to use the preferred skinning primitive based on the kind of pose or deformation they want to achieve.

In the **third part** of this thesis, I present two **graphical tools that allow artists to author deformations**. The first one, discussed in Chapter 5, is **CageLab**, a tool that allows users to perform cage-based deformations and animations, to edit cage geometry, and to compute

barycentric coordinates. The second one, discussed in Chapter 6, is **SuperCages GUI**, and is a Graphical User Interface dedicated to the SuperCages framework presented in the second part of this thesis. By using this tool it is possible to perform deformations and author animations with both skeletons and cages.

The **fourth part** of this thesis, in Chapter 7, is focused on the **conclusions** and the future works.

Part I

A small recap on modeling and animation

Chapter 2

Background

Computer Graphics is one of the most important fields in modern computer science. It is used by a large amount of industries: from medicine to engineering; from architecture to digital entertainment, and many more.

An important task in Computer Graphics is modeling. It can be branched in sculpting and deformation. The former is used to model or sculpt a three-dimensional shape like a digital character or a virtual object, while the latter is useful to deform and edit the shape of a pre-existing 3D model.

A 3D model is usually represented as a mesh, a collection of polygons or polyhedra that approximate the surface of the represented shape. The polygons or polyhedra can be expressed as a collection of vertices, faces and edges, so they can be generalized as a graph.

Computer Animation is another branch of the Computer Graphics field. It aims to study and create techniques and processes needed to bring a virtual character "to life". It allows, in fact, to generate motions and movements for digital objects and characters.

Deformation is a crucial task in the computer animation field, because a shape must be moved continuously to perform the movement we want to represent. In fact, to create every "fragment" of the animation, each one should slightly differ from the previous one to create the illusion of movement. The fragments are called frames, and each frame represents a still scene. By reproducing a sequence of frames at a fast enough rate we are able to trick the human vision, creating the illusion of the movement.

The creation of a keyframe requires the editing of the objects' position in the scene, usually also deforming them. The deformation of a 3D model involves the movement of its vertices. Usually, a mesh is composed of thousands of vertices, and moving them one by one is a hard, if not impossible task to perform by hand. For this reason, **skinning** algorithms and **rigging** structures have been created. By using these structures, we are able to associate groups of vertices to so-called **handles** (or deformation primitives), while the algorithms allow us to compute the new vertices positions with respect to the movement of the associated handles. In this way we are able to simplify the deformation process, reducing the complexity of this task.
Comprehensive surveys about skinning techniques can be found in [JDKL14] and [RF16].

The most popular rigging structure is the **skeleton**. It is composed of a set of joints and bones, like a real human skeleton, and they are associated, respectively, to the articulated and the rigid portions of a model. In this kind of rigging structures, the joints and bones can be used as handles. The association of the model vertices to the skeleton is defined by specifying the **weights of influence** of each handle over the character vertices (also called **skin**). To compute the skin deformation after a handle edit, there are several skeleton skinning algorithms available. More information about skeletons is detailed in Section 2.3.

A rigging structure that is gaining popularity among researchers and artists in the last years is the **cage**. A cage is a coarse version of a character it represents, enveloping it. Cage vertices represent the deformation primitives. By moving them we are able to deform the character according to the defined weights of influence, that are usually barycentric coordinates. More information about cages are detailed in Section 2.4.

Another possible rigging structure is defined by **point handles**. Each handle allows to specify local deformations like translations, rotations and scaling. This deformation will then be smoothly propagated onto nearby areas of the handle itself.

Skeletons and cages have different expressive powers: Skeletons are very useful and intuitive for primary motions, deforming characters articulations, while cages are suitable for smooth volume deformations and secondary motions (like jiggles and so on). For example, if an artist needs to rotate a humanoid arm, he can simply apply a geometric rotation to the relative bone, propagating the consequent deformation to the skin. Vice versa,

to perform the same kind of deformation using the cage, he must apply geometric translations to all the handles that enclose the desired arm. Intuitively, this task is much harder to perform because of the higher number of handles involved. On the other hand, if we want to inflate a digital character (like the chest of a breathing humanoid, or a balloon), the cage can be a natural choice because we can simply translate the specific handles that influence the area we want to inflate. To perform this kind of deformation with a skeleton can be a very hard if not impossible task, because we need to define extra bones and joints (with the correspondent weights) that lead us to the loss of semantic meaning of the skeleton and add technical complexity to it. So, skeletons are naturally apt to perform primary motions, while cages are perfectly suitable to enrich the deformation with secondary effects.

2.1 Animation Pipeline

As discussed in [Par12], several professional figures are involved in the Computer Animation creation process. One of them is the Digital Animator, a professional who makes use of the tools and algorithms from an artistic perspective to define the animation.

In order to create the illusion of motion through the animation, its frames need to be reproduced at a certain rate, which can be about 24 frames per second (for some kind of animated movies), 60 fps (for certain videogames) or even over 100 fps (for virtual reality devices).

In Computer Animation, frames are often produced using the keyframing technique. Using this, artists and animators can represent just a few key frames. Then, the remaining ones, called inbetweens, are obtained using interpolation techniques. In this way it is possible to produce smooth animations.

When an animation is produced, the animator must keep in mind what kind of target the animation is intended for and what kind of computational resources are available. If we need extreme realism or very detailed scenes, it is very difficult to perform the rendering in real time, since each frame requires a large amount of time to be rendered and sufficient computational resources are necessary in order to perform the computations. For example, in a full feature 3D animation movie or in a realistic CG special effect, real-time rendering is not necessary because the movie is not interactive and it will be played at a second time. Thus the animator is free to set every kind of illumination, deforming and physics algorithms to

render the scene. Moreover, if we want to obtain all the rendered scenes within an acceptable amount of time, no particular requirements must be satisfied except for the use of a powerful rendering system. However, if the animation is intended for an interactive application like videogames, the real time rendering is required. Therefore, computationally expensive algorithms (for rendering) cannot be used. On the contrary, the use of efficient illumination as well as deformation and physics techniques which do not exceed the processing power of the computer is required.

Modeling tools also play a fundamental role in the artistic process. There are several tools available on the market: two of them are Blender [Ble19] and Maya [may19]. They are general purpose tools, offering a wide range of functionalities: from modeling, to animation authoring. From texturing to simulation. Blender is a free, community-driven, open source tool, while Maya is focused on industry needs. General purpose tools are very powerful, but they have a steep learning curve, so artists need to master the User Interface, and understand all the functionalities to achieve the tasks they need to complete. There are also 3D graphics tools that are specialized in certain areas. Two of them are ZBrush [zbr19], focused on digital sculpting, and Substance [sub19] for texturing.

To sum up, digital animators constantly compromise between photo-realistic results and real-time rendered animations, depending on which target the animation is intended for and how many resources are available. Furthermore, they need powerful and simple tools that are focused on the tasks they need to accomplish.

2.2 History of CG and animations

Computer Animation is derived directly from the traditional hand-drawn animation. With the traditional animation, indeed, the animator must necessarily draw or represent manually almost every single frame. Walt Disney developed several techniques that reduce and simplify the work of the “classic” animators [Par12], one of those techniques is the Multiplane Camera, i.e. a system that makes use of a camera mounted over several planes and each plane holds an animation cell. Every plane can be moved in six directions in order to obtain an animation (using the so-called parallax effect) without drawing every frame, but simply moving the components of the represented scene. This enables animators to reduce their workload.

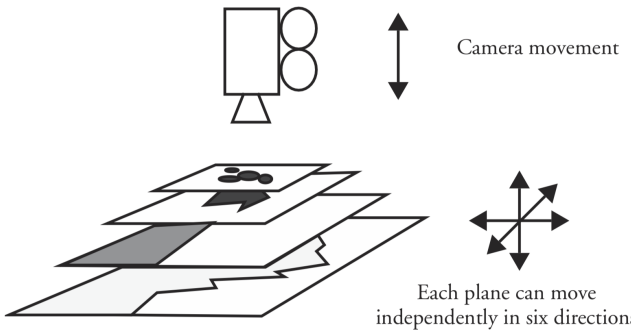


Figure 2.1: An example of Multiplane Camera. (Image taken from [Par12])

The birth of the Computer Animation allowed professionals to further simplify this process, making use of mathematical and geometrical notions. In fact, using the keyframing technique, it is no longer necessary for the animator to draw or rearrange every single component, but only a few key frames with the fundamental scenes need to be manually represented. The remaining frames will be automatically computed simplifying and reducing the animators' work as well as the time and the economical resources needed.

The first steps towards modern computer animation were made between 1960 and 1980. Pioneers in this discipline were the Massachusetts Institute of Technology and the University of Utah.

Typical computer-generated animations consisted of animated abstract line drawings, simple movements or deformation of meshes such as "Hummingbird" by Chuck Csuri (1967), "A computer animated hand"/"Faces" by Ed Catmull (1972), "Dream Flight" by Bergeron and Magnenat Thalmann (1982) and "Tony de Peltrie" by Bergeron (1985). In that time, the study of Computer Animation became more and more widespread among research institutes and started to be used in the form of very simple CG effects in movies such as "Future World" (1976), "Star Wars" (1977) and "Alien" (1979).

In the 80s, Computer Animation began to be supported by several commercial companies, especially with the diffusion of personal computers like VAX or IBM, and the introduction of CG-oriented dedicated hardware and software technologies like z-buffering, texture mapping, antialiased ray tracing etc.

The birth of the SIGGRAPH conference (Special Interest Group on

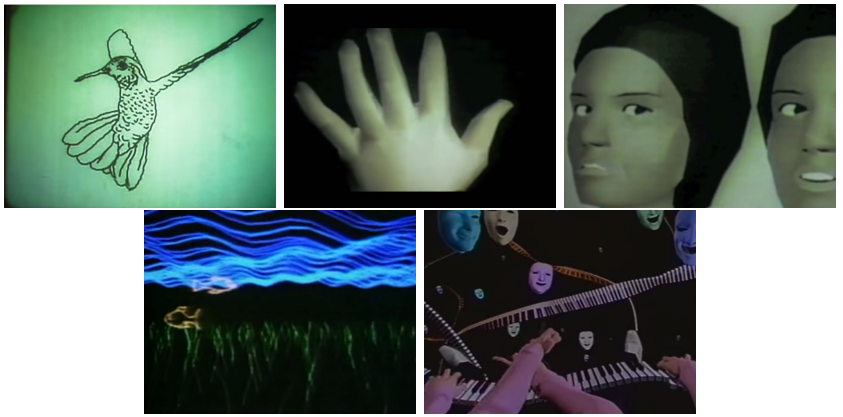


Figure 2.2: Frames from the first CG animations. In order, “Hummingbird”,“A computer animated hand”, “Faces”, “Dream Flight” and “Tony de Peltrie”.

GRAPHics and Interactive Techniques) helped to further spread Computer Animation among research institutes and industries.

In the mid-80s, Pixar, a computer animation-focused company founded by George Lucas (and then acquired by Apple and Disney), started to create true short movies to demonstrate their own technological achievement in the research field of advanced computer animation techniques. Pixar shorts gained visibility over the years. This led to the production, in collaboration with Disney, of the first full-length, fully computer-generated three-dimensional animated feature movie in 1995.

A fundamental role in the development of computer animation is that of Sony and its PlayStation home videogame console, that makes massive use of applications of three-dimensional Computer Graphics and Computer Animation. The PlayStation was a huge commercial success and was capable of drawing more attention to Computer Animation.

It soon became clear that computer animation would have been the future of entertainment.

Figure 2.3: A frame from the Pixar movie Toy Story.

Figure 2.4: A screenshot from Tomb Raider, one of the first popoular 3D games available on consoles and personal computers.

2.3 Skeletons

Skeletons are the most popular rigging structure. A skeleton is contained inside the character skin, and is composed of bones and joints: the first ones represent the rigid parts of a character and the second ones are usually placed in the character articulation. In a skeleton-based skinning setup, bones and joints can be used as handles to define the deformation, that is propagated to the character skin vertices using skinning weights. The skinning weights can be automatically generated or manually defined by artists and riggers.

2.3.1 Skinning algorithms

As is customary, instead of representing the skeleton S explicitly, we consider the set of rigid transformations $\mathcal{T} = \{\mathbf{T}_1, \ldots, \mathbf{T}_s\}$ that determine a given pose. Each transformation $\mathbf{T}_j$, is associated to the j-th bone of S and represents a rotation around one of its endpoints, 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 a $m \times s$ (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.

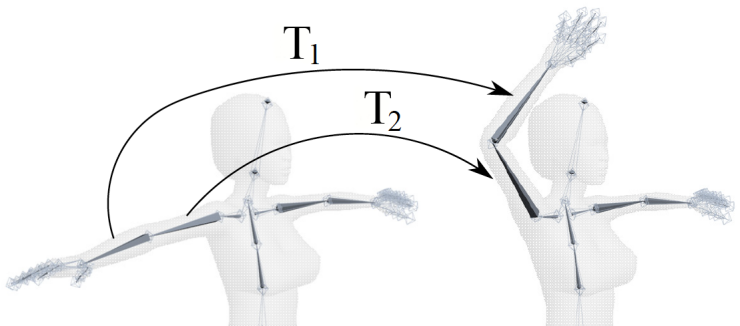


Figure 2.5: Example of bone transformation achieved through the manipulation of its transformation matrices T_1 and T_2. (Source [JDKL14])

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

$$M' = F(\mathcal{T}, \Omega, M) \tag{2.1}$$

where M' is the deformed (current pose) mesh.

Linear Blend Skinning

The most popular skeleton skinning algorithm is the Linear Blend Skinning (LBS), also known as skeleton-subspace deformation. This technique appeared a long time ago in the literature ([MTLT89], [LCF00a]). It computes deformations defined by linearly blending transformation matrices associated to the bones of a control skeleton.

LBS is encoded vertex-wise by

$$\mathbf{v}'_i = \sum_{j=1}^{s} \omega_{i,j} \mathbf{T}_j \mathbf{v}_i \tag{2.2}$$

which is often presented in the following form where the linear part applied to the vertex is separated from the translation part

$$\mathbf{v}'_i = (\sum_{j=1}^{s} \omega_{i,j} \mathbf{T}^r_j) \cdot \mathbf{v}_i + (\sum_{j=1}^{s} \omega_{i,j} \mathbf{T}^t_j) \tag{2.3}$$

where $\mathbf{v}'_i$ is the (deformed) position of the vertex $\mathbf{v}_i$, $\omega_{i,j}$ is the skinning weight of the vertex i associated to the j-th bone, and $\mathbf{T}_j$ is the j-th bone transformation ($\mathbf{T}^r_j$ is the rotational part, and $\mathbf{T}^t_j$ is the translational one).

Unfortunately, Linear Blend Skinning presents visual artifacts like volume loss, and self-intersection, known also as *candy wrap* artifact. It is caused by singular transformation matrices generated by the linear interpolation of rotations, making the shape locally collapse to a single point.

Dual Quaternion Skinning

Another popular skeleton-based skinning technique is called Dual Quaternion Skinning (or DQS) (as defined in [KCŽO08], and [KCvO07]). Rather than using a rigid transformation matrix to compute deformations, DQS makes use of the dual-quaternions theory [Ken12] to blend rotations and translation, that has a more robust behavior with interpolation. For this reason, DQS avoid the generation of candy wrap artifacts at the cost of a minimal overhead in the real-time deformation pipeline.

Dual Quaternion Skinning is encoded by

$$\mathbf{v}'_i = \text{DQ}_{\text{BLEND}}(\mathcal{T}_i, \Omega_i)\ \mathbf{v}_i \tag{2.4}$$

where DQBLEND is the linear combination function used to blend transformations represented via dual quaternions (called DLB by Kavan et al.

in [KCvO07]), while $\mathcal{T}_i$ and Ω_i denote the i-th rows of $\mathcal{T}$ and Ω, respectively.

Dual Quaternion Skinning, though, creates the so-called *joint bulging* artifacts.

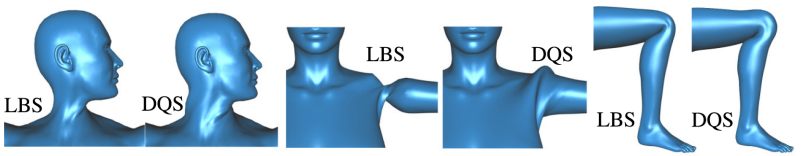


Figure 2.6: A comparison between Linear Blend Skinning and Dual Quaternion Skinning. We can observe the "candy-wrapper" artifact for the LBS, and the "joint bulging" artifact for the DQS. Source [KS12])

CoR Skinning

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

$$p_i := \int_{x \in M} \delta(\omega_{i,.}, \omega_{x,.})\, x\, dx \;/\; \int_{x \in M} \delta(\omega_{i,.}, \omega_{x,.}) dx \tag{2.5}$$

$\delta(\omega_{i,.}, \omega_{x,.})$ 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 CoRs method

can be summarized as

$$\begin{cases} \mathbf{v}'_i = R(i) \cdot \mathbf{v}_i + t(i) & \text{(deformation)} \\ R(i) = \text{DQBlendRot}(\{\omega_{i,j}, \mathbf{T}^r_j\}_j) & \text{(linear part)} \\ t(i) = \tilde{p}_i - R(i) \cdot p_i & \text{(translation part)} \\ \tilde{p}_i = \sum_{j=1}^{s} \omega_{i,j}(\mathbf{T}^r_j \cdot p_i + \mathbf{T}^t_j) & \text{(transformed CoR)} \end{cases} \tag{2.6}$$

where DQBlendRot returns the linear (or rotational) component of the matrix DQBlend defined above, obtained as Dual Quaternion linear blending.

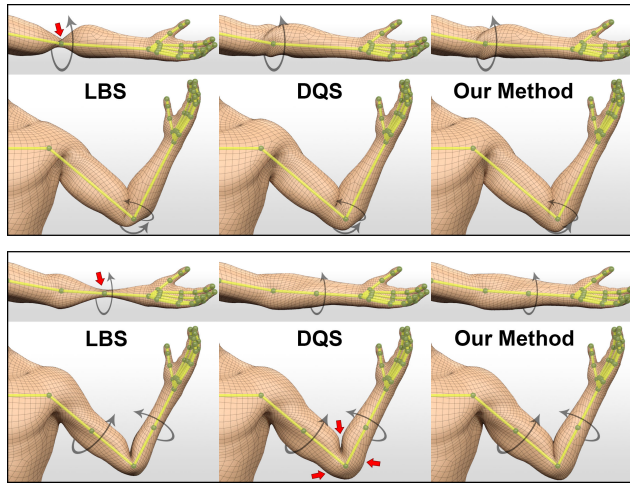


Figure 2.7: A comparison between Linear Blend Skinning, Dual Quaternion Skinning and CoRs. Top and bottom images represents two different sets of skeleton handles. (Source [LH16])

Other Methods

More recent non-linear skinning methods, such as Stretchable and Twistable bones [JS11], Differential Blending [OBP+13] and Elasticity Inspired Deformers [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.

2.4 Cages

A cage is a simplified (coarse) version of the mesh it represents, so it can be seen as a low-resolution approximation of the character. It is completely external to its mesh and envelops it, and it represents the mesh morphology. In a cage-based skinning setup, the cage vertices can be used as handles to create the deformation and propagate it to the character skin vertices by using barycentric coordinates as weights. A cage should preferably have a small number of vertices.

Each point of the character within the cage volume is defined as a weighted sum of cage vertices. Consequently every character vertex position can be efficiently updated each time the cage is deformed by the user.

The cage-based skinning deformation derives directly from the Free Form Deformation [SP86] (FFD) that offers an intuitive and smooth control over the character skin by only using lattice control points, but this technique does not take into account the character morphology. In fact, for a complex mesh such as a character articulated with several limbs, FFD becomes difficult or impossible to use in order to obtain a significant and meaningful deformation. Cage-based deformation, instead, is based on the concept of generalized barycentric coordinates first introduced by Möbius in 1827 [Möb27], through which we are able to interpolate the position of every character vertex in relation to its cage vertices. Some definitions of them are discussed in 2.4.1.

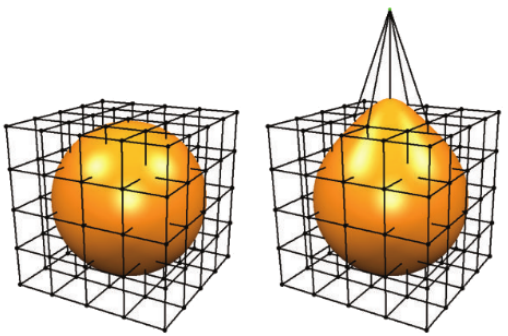


Figure 2.8: Example of Free Form Deformation (Source [BKP+10])

In a cage-based skinning setup, the deformation is given vertex-wise

by the standard linear equation

$$\mathbf{v}'_i = \sum_{j=1}^{c} \phi_{i,j} \mathbf{c}'_j \tag{2.7}$$

where $\mathbf{v}'_i$ represents the (deformed) position of vertex $\mathbf{v}_i$ of the character mesh, $\mathbf{c}'_j$ is the deformed cage vertex j, and $\phi_{i,j}$ is the barycentric coordinate of mesh vertex i with respect to cage vertex k. Barycentric coordinates are stored in a $m \times c$ (sparse) matrix called Φ.

By convention, we assume that we have a rest pose for the cage C when

$$\mathbf{v}_i = \sum_{j=1}^{c} \phi_{i,j} \mathbf{c}_j \tag{2.8}$$

so the equality $M = \Phi C$ is satisfied, or, in other words, when the pose of the cage induces the rest pose of the character skin. For this reason, generalized barycentric coordinates are computed using 2.8 as constraint.

Note that Equation 2.7 does not refer to any rest position for either the cage or the skin, compared to what happens with skeletons. For this reason cage skinning uses an absolute approach to compute deformations, while skeletons use a relative one.

Similarly to skeletons, cages can be either automatically computed or manually crafted, thus ensuring a seamless integration with most available implementations.

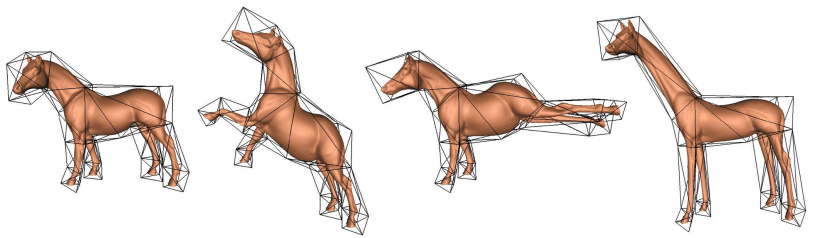


Figure 2.9: Deformation of a horse character using the cage-based skinning with Mean Value Coordinates. Resting pose on the left, deformed pose in the other pictures. (Source [JSW05])

2.4.1 Barycentric coordinates definitions

Barycentric coordinates allow us to represent a vertex placed inside a polyhedral shape as function of the shape vertices. There are several possible definitions of them, and they differ to each other mainly for several qualities.

As explained in [NS13, GPCP13, JBPS11], ideally, the qualities that barycentric coordinates should respect to obtain good looking results, avoiding graphical artifacts are:

- **Smoothness** The coordinates must be well-defined on the inner volume of the Cage, avoiding the presence of discontinuities.
- **Partition of unity** For each vertex of the mesh, the sum of its weights must be equal to 1 (this leads to affine invariance). This means that if a certain transformation T is applied on all the cage handles, the whole charater skin will be deformed by T in an uniform way.
- **Locality** The mesh vertices must be influenced by the nearest cage handles only. At the moment, the majority of barycentric coordinates definitions lacks of locality, so each cage handle tends to influence all the mesh vertices.
- **Conformality** The weights must preserve shape and surface details without losing shape semantic, after a local deformation (especially rotations).
- **Positiveness** The weights values must be greater or equal to 0. Negative weights, in fact, can lead to counterintuitive results because they deform the skin in the opposite direction respect to a given transformation T.
- **Computational efficiency** The weights precomputation and runtime deformation must be computed in an acceptable amount of time. The memory footprint of cage weights should be as low as possible.

Note that the equation 2.7 is compliant with all barycentric coordinates described here, except the Geeen coordinates [LLCO08] and the Variational Harmonic Maps [BCWG09], which require also face normals that set a non-linear relation between the cage and the skin.

Here we describe some definitions of barycentric coordinates, but a more comprehensive description is available in [JDKL14], [NS13], and [CLM+19].

Mean Value Coordinates is, probably, the most famous barycentric coordinates definition. Described in [Flo03], [FKR05], [HF06], and [JSW05], can be observed in figure 2.9. They offer a good interpolation method, and they are efficient from a computation point of view. Their main cons is the lack of positiveness and locality.

To prevent negative weights and improve locality of MVC, Lipman proposes **Positive Mean Value Coordinates** [LKCOL07], but this method can generate distortion problems and singularities in concave polygons.

Be aware that, usually, barycentric coordinates are defined for triangle-based cages, and this fact can lead to artifacts and distortions in cages defined with symmetric sections. To improve the symmetry of the deformation, **Quad Mean Value Coordinates** has been defined in [TMB18], making use of quad-based cages rather than triangle-based ones.

Green Coordinates [LLCO08] are another popular barycentric coordinate definition. Its main advantage is that it preserve the shape and its details in a good way, but the mathematical definition is slightly complex than the one presented in 2.7. In fact, it is

$$\mathbf{v}'_i = \sum_{k=1}^{c} \phi_{i,k}\mathbf{c}'_j + \omega_{i,l}\mathbf{s}_l\mathbf{t}_l \tag{2.9}$$

where $\omega_{i,l}$ is the Green barycentric coordinate of the vertex i with respect to the face l, $\mathbf{s}_l$ is a scaling factor that represent the stretch of the face l, and $\mathbf{n}_l$ is the normal of the face l.

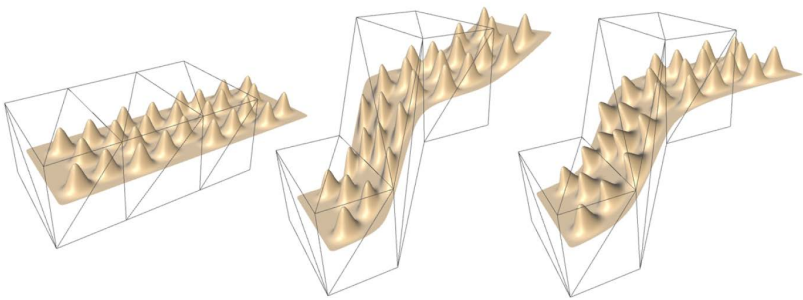


Figure 2.10: Mean Value Coordinates vs Green Coordinates. (Image source [LLCO08])

Another definition of barycentric coordinates is the **Maximum Entropy Coordinates (MEC)** described by Hormann and Sukumar in

[HS08]. They explain how to derive proper cage weights given a set of priors which do not respect properties like smoothness or partition of unity. Imagine to create a group of prior weights that does not constitute a valid set of cage coordinates, associating values to the skin vertices with respect to the cage handles. Using MEC we are able to compute a set of valid barycentric coordinates that are as close as possible to the invalid prior weights, using Information Theory to maximize their cross-entropy.

Additional definitions proposed in the literature are Harmonic [JMD+07], and [TTB13, ZDL+14a, BCWG09]. All methods, with the exception of [LLCO08] and [BCWG09], compute skin coordinates as a linear combination of cage vertices.

2.5 Hybrid Methods

Some methods depart from the classical skeleton-based and cage-based deformation paradigms, trying to improve on them on some aspect.

For example, Ju et al. [JZvdP+08] (figure 2.11) combined the use of cages and skeletons to avoid the candy wrap artifacts of LBS. In fact, it uses the skeleton to pose the cage, and the cage to pose the skin. Similar systems are currently supported by commercial software (e.g. Blender [Ble19]), but in this case they propose an automatic method to produce templated cages.

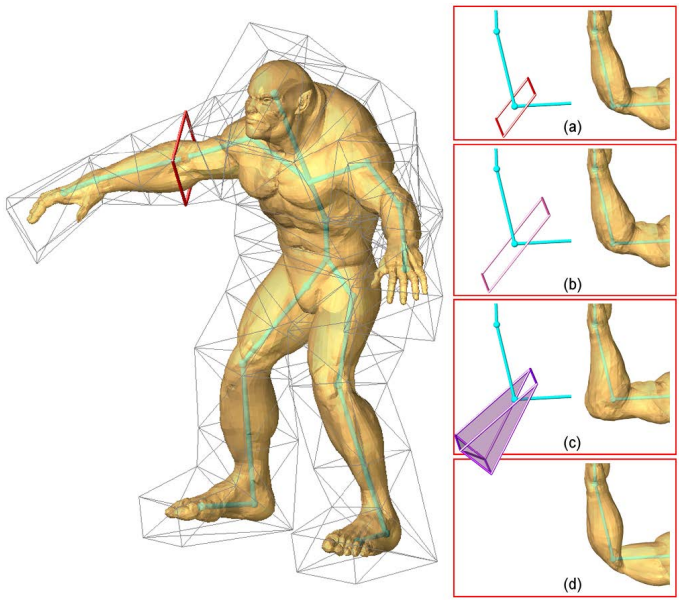


Figure 2.11: Ju et al. propose a deformation method that makes use of cages constructed from templates. Different deformations with alternative templates are shown in (a), (b), (c), while (d) shows the same deformation performed with the Linear Blend Skinning directly from the skeleton. In the last picture the "candy-wrapper" effect is noticeable. (Source [JZvdP+08])

Gonzales 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. They aim to use multiple barycentric coordinates definitions to improve locality of deformation, and achieve better results.

Mukai and Kuriyama [MK16] propose the use of automatically gen-

erated bone helpers to enrich the space of deformations of linear blend skinning with secondary motions, enabling the animation of muscles and soft tissues.

In [WJBK15] Wang et al. explore the combination of skeletons and point handles. This idea is further explored in [JBPS11] by Jacobson et al. (figure 2.12). They propose a system where skeleton, cage and point handles are all integrated into the same framework. Their system require 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.

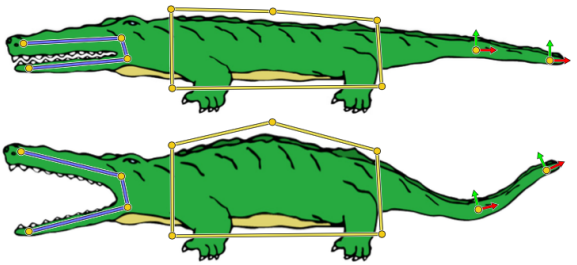


Figure 2.12: Example of simultaneous deformation using skeleton, cage and generic handles, using bounded biharmonic as skinning weights on a two-dimensional character. (Source [JBPS11])

Part II

A Skeleton/Cage hybrid paradigm

Chapter 3

SuperCages

In this Chapter we are going to formalize a collection of operators organized in a pipeline that makes it possible to deform three-dimensional meshes by using skeletons and cages seamlessly.

As explained in Section 2 and in [Cor17], Skeleton and Cages have different expressive powers. By merging them, we are able to unify their expressive powers, enriching skeletal deformations with the degrees of freedom offered by cages. This deformation pipeline and the relative operators are combined together inside a framework that from now on we will call **SuperCages**.

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

Our goal

Using SuperCages we are able to unify the expressive powers of skeletons and cages. In this way, artists, modelers, animators and graphic programmers have the freedom to use the preferred skinning primitive based on the kind of pose or deformation they want to achieve: the manipulation of the cage handles will result in the deformation of the character and the skeleton; vice-versa, the manipulation of the skeleton handles will

result in the deformation of the character and the cage. For example, if the animator wants to animate a running man while also representing also the character's breath, he can use the skeleton handles to define the movements of arms and legs, while using at the same time the cage handles to represent the breathing animation.

Related Works

At the moment, there are two main contributions to the skeleton/cages merging for skinning purposes, but both of them are focused on other aspects of this merging.

The first one is the method proposed by Ju et al. with the cage-based skinning templates [JZvdP+08], shown in figure 2.11. This work aims to create a rig that allows to perform deformations that are replicable on multiple characters using a skeleton/cage hybrid. In this method the skeleton is rigged to a cage and drives the motion of the cage vertices. In details, given the character mesh and a skeleton, the algorithm creates a full cage from "templates", partial cages associated to a particular skeleton part. Then, manipulating the skeleton handles, the algorithm will deform the associated cage using a particular deformation function similar (but different) to Linear Blend Skinning. Then the deformation will be propagated from the cage to the character skin using Mean Value Coordinates as barycentric weights.

The main purpose of the aforementioned work is not to merge together the expressive power of skeleton and cages but to create volume preserving deformations using skeleton handles, overcoming the "candy-wrapper" artifact typical of the Linear Blend Skinning algorithm. So, in this case, we do not have a true skeleton/cage hybrid because it is not possible to interact directly with the cage handles to perform the deformation. In fact, to create a pose, the artist must interact only with the skeleton handles and the cage is used only the define the skin deformation. Also, the cage must be generated ad-hoc from the associated skeleton, so previously modelled and rigged cages can not be used. Furthermore, all the previously computed skeleton skinning weights must be dropped because the skeleton will be rigged directly to the cage vertex using a specific function rather than directly to the character skin.

The second method is the one proposed by Jacobson et al. with the Bounded Biharmonic Weights [JBPS11], shown in figure 2.12. This work aims to make the digital animator free to use every kind of skinning primitive it prefers, like skeleton handles, cage handles or simple points

to perform two-dimensional or three-dimensional character deformation. They give a definition for automatically computed skinning weights that can be used simultaneously and seamlessly with the three types of skinning structures listed earlier.

Despite the very good graphical result for the deformation, the digital artist must necessarily use the bounded biharmonic as skinning weights, so previously computed or manually generated skinning weights can not be used, and all the pre-existing skeleton and cages must be rigged again with the new automatically generated weights. Moreover, the skeleton, the cage and the handles are part of the same meta-structure and must be jointly animated: manipulating the skeleton (resp. the cage) will not induce a deformation of the cage (resp. the skeleton), contrary to what we aim at.

Moreover, Blender [Ble19] allows to link a control cage (called *Mesh Deformer*) to a skeleton (called *Armature*) and move the cage through it. However, 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 centers of rotation of each bone. Furthermore, only the cage skinning is involved, using Harmonic Coordinates barycentric coordinates [JMD+07], consequently dropping the skeleton weights.

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.

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 in a collection of synchronization operators that maintain the pose set up-to-date during the editing session in real-time.

In details, the four aspects we fulfill are:

- Understand how the manipulation of skeleton handles drives the motion of the cage handles.
- Understand how to refit the skeleton handles based on the cage deformation.

- Define how the global skinning algorithm should work.
- Understand how to simplify the user interaction with the seamless use of skeletons and cages.

Furthermore, the SuperCages hybrid paradigm is independent to skinning weights definitions. In fact, in order to simplify the rigging phase, we want to enable users to use pre-existing rigged skeletons with pre-existing rigged cages, using the already defined weights for each structure. By using this approach, in a production studio, there is no need to create ad-hoc skeletons, cages or weights from scratch, but we can simply use the pre-existing ones, adapting the skinning algorithm, implementing the appropriate operators in the deformation pipeline.

Rigging structures

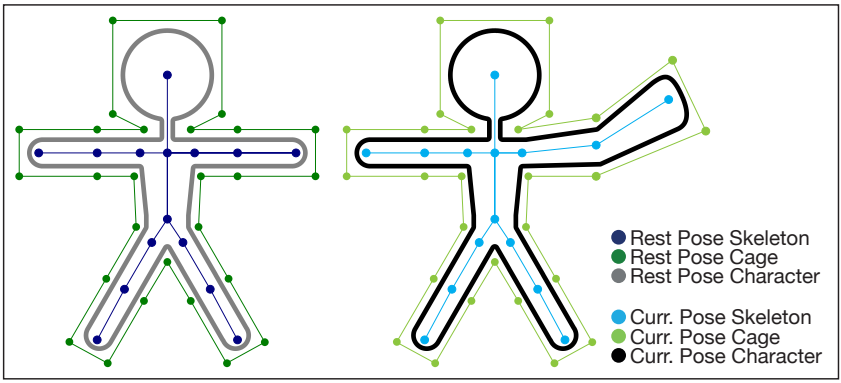


Figure 3.1: A schematic representation of how the character rig is organized in our framework. The Rest Pose rig is on the left, and the Current Pose rig one the right. Note that the current pose differs from the rest pose because of the arm cage inflation and the skeleton arm rotation.

As previously introduced in Section 2, a character rig is composed of several entities. Usually those entities are the mesh skin M in its rest pose, the rest-pose rigging skeleton S, and the rest-pose cage C. Their deformed counterparts are also present, like the M' model skin, associated to the "current pose" S' skeleton, and C' cage. To bind the skeleton to the model skin, we use the skeleton skinning weights Ω, in both rest and current (or deformed) pose. In the same way, to bind the cage to the mesh skin, we use the barycentric coordinates Φ. To define which transformations define

the skeleton movements between the skeleton rest and deformed pose, we use a set of transformations $\mathcal{T}$.

How to perform a deformation

In our system pipeline three of those entities can be edited to obtain deformations:

- the S' current pose skeleton
- the C' current pose cage
- the C rest pose cage

Those edits that lead to skin deformations can be performed by the user, by an animation algorithm, or by a physical engine. Our pipeline automatically update all the other structures in a coherent e meaningful way using three operators (or components):

- Cage Updater: updates the current pose Cage C' when the Skeleton S' is edited. It is described in Section 3.2.
- Skeleton Updater: refit the S rest pose skeleton joints when the rest pose cage C is deformed. It is described in 3.3.
- Reverse Cage Deformer: transfers the edits performed on the current pose cage C' to the rest pose cage C. It is described in 3.4.

So, when an editable entity is deformed, the pipeline described in details in Section 3.1 is invoked. The pipeline keeps in sync all the rig entities, avoiding visual artifacts in the deformation, and maintaining the handle positions coherent.

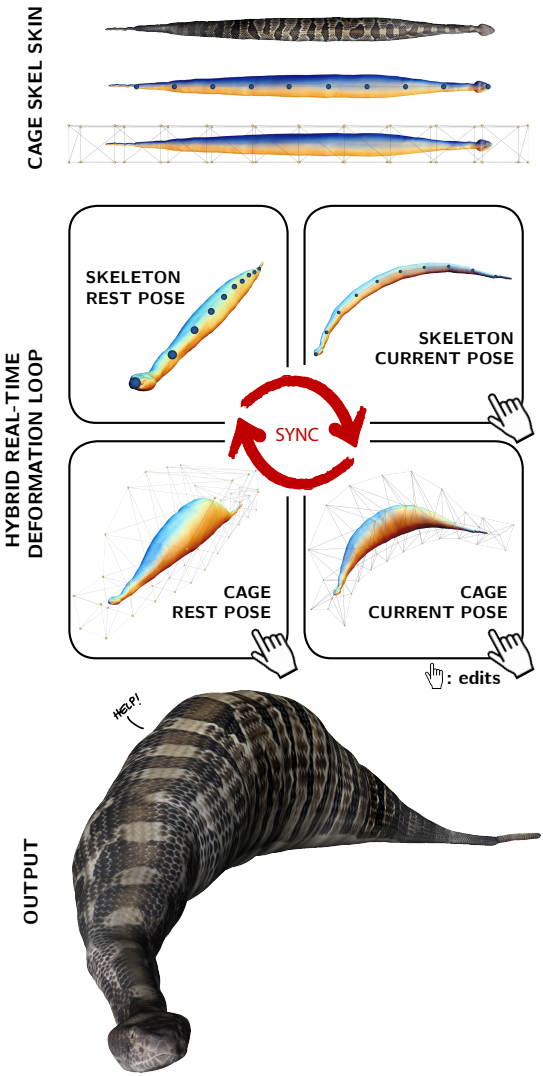


Figure 3.2: Given an input 3D shape equipped with its deformation skeleton and cage (top), our framework seamlessly combines both structures, merging their associated deformation spaces. We achieve this result by using rest and current pose both for the skeleton and the cage (middle). The user can deform 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 the rest pose. Our operators automatically maintain all these entities in sync in real-time.

3.1 The pipeline

To couple skeletons and cages, obtaining seamless deformations, the operators described in the next sections must be organized and structured together. Here we present the SuperCages pipeline, describing how the operators must be used in a closed loop to obtain the final deformation.

The main idea for the whole pipeline is that the cage-skinning algorithm must deform explicitly and only the M rest mesh. These edits will then be propagated and kept in sync with the other structures by the pipeline itself. We achieve this synchronization among the six structures described in Section 3 composing in the appropriate way the SuperCages operators and the skinning algorithms described in Section 2. The pipeline can be described introducing a set of *transitions*, which reflect editing among the operators and are summarized in Figure 3.3.

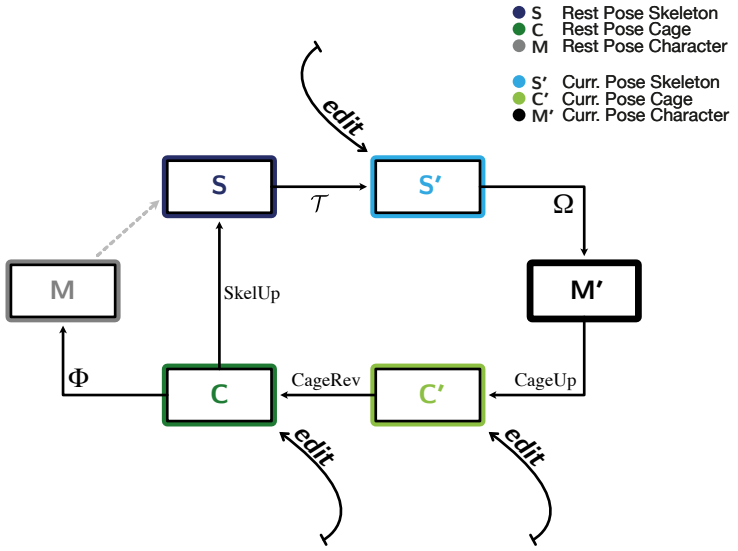


Figure 3.3: A diagram of our method showing transitions among the different structures; symbols attached to arrows denote the parameters or algorithms used to effect such transitions. Hooked arrows marked EDIT denote possible entry point in the pipeline performed as edits on the relative structure. The dotted arrow is a symbolic link that represents a step needed for the general skeleton skinning equation.

The possible entry points of the pipeline are S', C, and C'. In fact, by manipulating their handles, we are able to trigger the relative operators, invoking the relative skinning algorithm.

More in depth:

- $M \dashrightarrow S \rightarrow S' \rightarrow M'$ is the standard skeleton skinning pipeline derived from Equation 2.1; In fact, applying to S the $\mathcal{T}$ transformations we are able to compute S'. Then, using the Ω skeleton weights, M, and one of the skeleton skinning algorithms (described in Section 2.3) we are able to compute M'.

- $C \rightarrow M$ is the standard application of barycentric coordinates Φ during the cage skinning algorithm described in Equation 2.7; As mentioned before, in our framework the cage skinning is applied on the M rest pose mesh.

- $C \rightarrow S$ is the transition that performs the skeleton joint refitting. It is needed to preserve the relationship between the skeletons and the Ω weights after a cage edit. We update the centers of rotations of S on a modified skin M at rest pose, invoking the *Skeleton Updater* (SkelUp) described in Section 3.3.

- $M' \rightarrow C'$ finds a consensus between the current poses of the mesh and the cage, invoking the *Cage Updater* (CageUp) described in Section 3.2. Its job is to obtain a new C' cage that corresponds to the new M' deformed mesh.

- $C' \rightarrow C$ is the transition that is invoked when an edit is performed on C'. It invokes the *Reverse cage deformer* (CageRev) described in Section 3.4, reflecting the performed edits to the cage at rest pose C. We must guarantee that the current cage C' determined from editing by the user coincides with the current cage resulting from the transition $C \rightarrow S \rightarrow S' \rightarrow M' \rightarrow C'$, as induced by the rest cage C modified by C'. In order to achieve this result, we must set this transition in a proper way.

Note that after the CageUp transition, the system reaches a convergence state because all structures are in sync. A new loop of the pipeline will start after a new edit.

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 $C \rightarrow S \rightarrow S' \rightarrow M' \rightarrow C' \rightarrow C$ in Figure 3.3: no matter where the interaction starts, we propagate its effects

through this cycle to maintain all four structures synchronized.

The four structures in the central cycle of the diagram are synchronized after each editing operation so that the central cycle remains at a steady state. Note that depending on the needs, some transitions can be ignored during the implementation as described in Section 3.5.2.

In Figure 3.4 we can observe a scheme representing which actions are performed at each pipeline entry point.

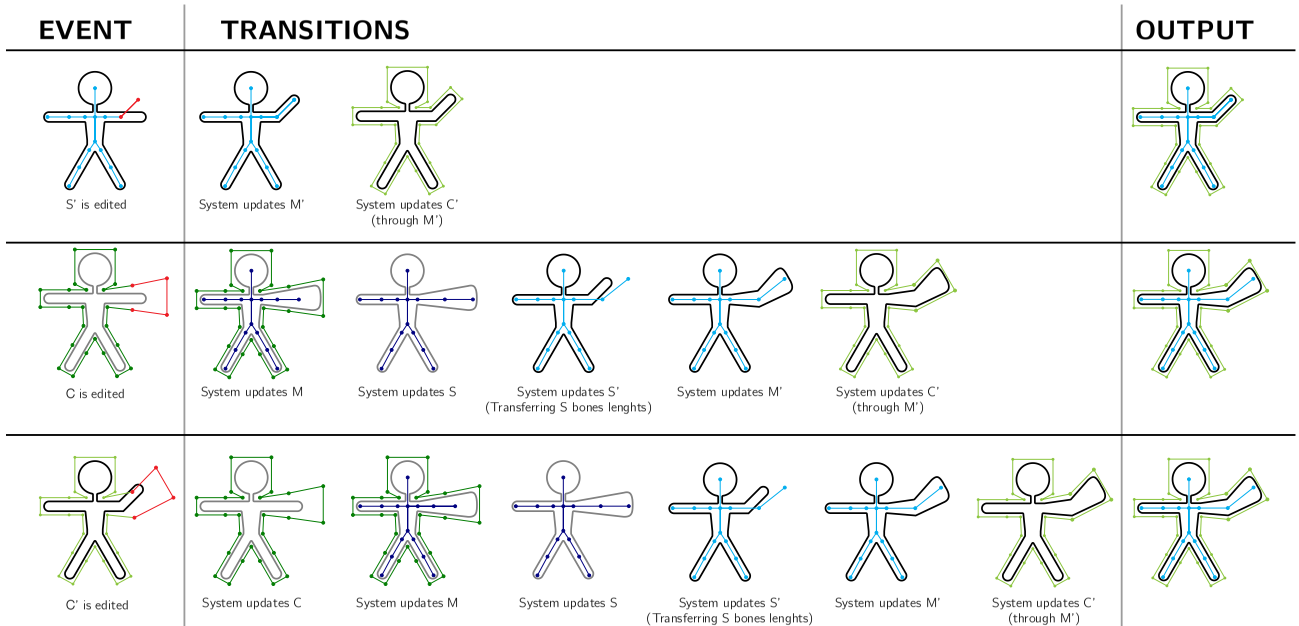


Figure 3.4: Interactions with our hybrid deformation system: the user can edit the skeleton in the current pose (S') or the cage, both in the rest (C) and the current (C') pose. The diagram illustrates the chain of reactions that automatically update the system, maintaining the sync among the various entities involved in the deformation. All interactions occur in real-time.

3.2 Cage Updater

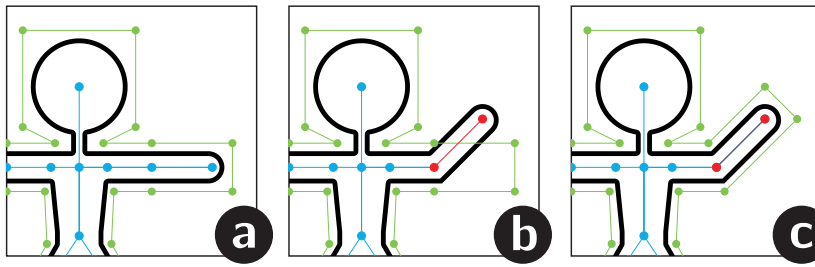


Figure 3.5: The Cage Updater role is to fit the cage with respect to a skeleton deformation. In this scheme we can observe the initial pose for the character (a) that is deformed rotating the skeleton arm (b). This edit triggers the Cage Updater operator that refits the current pose cage accordingly to the new deformed skin.

The Cage Updater is the operator that updated the C' current pose cage according to the skin deformations defined by the S' current skeleton on the M'. In figure 3.5 we can observe that after a M' skin deformation, C' must be updated to reflect those edits, maintaining the consistency of the global rig. In fact, in the initial pose the cage envelopes the arm, and if that arm is deformed by the skeleton, the cage updater fit the positions of the handles to the new position. Without the cage updater, instead, C' handles would be located in the wrong place. In this way we are able to keep in sync the cage according to the skeleton modifications.

3.2.1 Mathematical formulation

The main idea is to seek a cage C' that generates a skin as close as possible to M', according to the (static) barycentric coordinates Φ. This algorithm can be performed in two different ways: using a least square solver, or using the *MaxVol* relaxation proposed in [TTB12].

Using the least square version, we can update the cage by solving

$$C' = \Phi^{+} \cdot M', \tag{3.1}$$

where Φ^{+} denotes the pseudo-inverse of matrix Φ. The system remains fixed and can be pre-factorized once and for all, after what updates are obtained in real-time.

This approach is simple to implement and doesn't require particular pre-processing steps.

The main consideration about this method is that it generates an approximation in the least square sense, so the problem may (and in general does not) admit an exact solution.

To avoid solving a least squares problem of the size of M', generating an approximation, we can apply the *MaxVol* relaxation. For the MaxVol alternative, 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 (and consequently, with maximal determinant). Then, we consider the corresponding set $\tilde{M}'$ in the current mesh M', and we solve the linear system

$$\tilde{\Phi} C' = \tilde{M}', \tag{3.2}$$

where $\tilde{\Phi}$ is the sub-matrix 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 the 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, even though it's more complex to implement. Also, using the MaxVol relaxation, the computation time of the solver is lower than the one in the least-square method.

3.3 Skeleton Updater

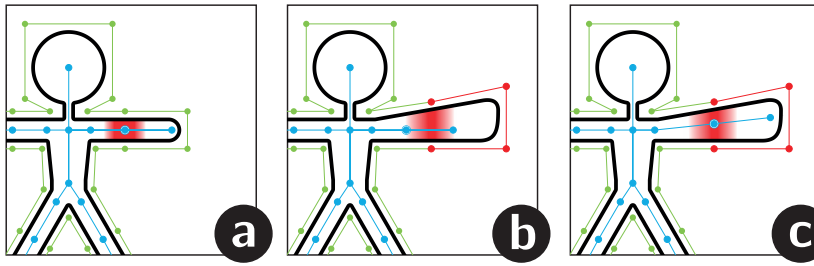


Figure 3.6: The Skeleton Updater role is to fit the skeleton with respect to a cage deformation. In this scheme we can observe the localization function of the highlighted skeleton joint (a). Editing the character cage, the highlighted joint lose its initial semantic meaning (b). Triggering the Skeleton Updater operator, we are able to refit the skeleton joints in the position defined by the localization function (Skeleton Updater Weights).

When an edit on C rest pose cage is performed, it deforms the M skin. As a side-effect, the Ω skeleton weights lose their semantic meaning because the skeleton handles, remaining still, represent incoherent centers of rotations. To overcome this problem we implement the Skeleton Updater operator, that updates the skeleton handles positions according to the new deformed M, preserving the Ω semantic as much as possible, and updating consequently the skeleton centers of rotations. If this semantic is not preserved, several artifacts will occur, generating many self intersections, as we can observe in figure 3.10 and 3.7.

Through the Skeleton Updater, we introduce a new relation between the cage vertices and the position of skeleton joints. This relation is defined 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). This relation is expressed in the form of linear combination weights. The weights can be manually defined/painted, or automatically computed, and in the next section we describe a possible approach to automatically compute those weights.

In synthesis, the general idea for the skeleton updater is to fit every S skeleton joint in the center of its influenced M vertices. So when a

Figure 3.7: A comparison between the original skeleton deformation (left), skeleton deformation with cage edits plus skeleton updater disabled (middle), skeleton deformation with cage edits and skeleton updater enabled. (right).

cage modification is performed, we must locate where we have to move the joints positions so that they can maintain as much as possible their previous semantic meaning. When the S is updated, the joints positions are propagated to the S' current pose skeleton handles keeping in mind that the length of every bone is equal between S and S'. In this way, we are able to keep in sync skeleton and cage modifications.

3.3.1 Mathematical formulation

Skeleton Updater weights

To create the relation between C and S, our idea is to create a localization function that can be defined manually or automatically.

In this section we describe a possible approach to automatically compute the Skeleton Updater weights to define this relation.
All computations in the following are performed once in pre-processing for the initial pose, when $C \equiv C_0$ and $S \equiv S_0$. C_0 and S_0 represent the unedited rest pose cage and rest pose skeleton. Be aware also that with "once in pre-processing" we mean that these weights can be saved and reused later for future use (without recomputing everything again) on the

same character rig.

To compute the Skeleton Updater weights we first identify to which mesh vertices a joint j corresponds to, by defining the *joint localization function* $L^{\Omega}_{j,\cdot}$ for each joint of S. These functions depend only on the skinning weights Ω and are defined vertex-wise on M as follows

$$L^{\Omega}_{j,i} = -1 + {\omega_{i,j}}^{s} + (\sum_{k \neq j} \omega_{i,k})^{s}, \tag{3.3}$$

with $s << 1$ (we use $s = 0.1$ in our implementation). Function $L^{\Omega}_{j,\cdot}$ takes value 0 in rigid regions (i.e., for $\omega_{i,j} = 1$ and $\omega_{i,j} = 0$) and larger values as $\omega_{i,j}$ approaches 0.5, i.e., near the joint, where the skinning weights blend the most (see Figure 3.8).

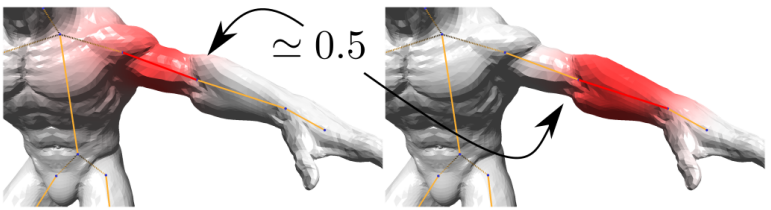


Figure 3.8: Skeleton weights are typically close to 1 around the middle of the bones, resulting in highly rigid transformations in these parts of the mesh, while they blend the most with the other weights near the articulations. With this motivation in mind, we defined the joint localization function described in 3.3.

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 $\{\text{mvc}_{j,i}\}_i$ 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 $\{\mathbf{L}^{\Omega}_{j,i}\}_i := \{\text{mvc}_{j,i} * L^{\Omega}_{j,i}\}_i$ are not valid barycentric coordinates at this step.

The joints barycentric coordinates are then defined as

$$\psi_{j,\cdot} = \text{MEC}\left(\sum_i \mathbf{L}^{\Omega}_{j,i}\phi_{i,\cdot}\right), \tag{3.4}$$

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], described also in Section 2.4.1

$$s.t. \quad \begin{cases} \sum_k b_k.c_k = a_j \\ \sum_k b_k = 1 \end{cases}$$

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

$$A = MEC(\mathbf{L}^{\Omega} \cdot \Phi) \cdot C := \Psi \cdot C \tag{3.5}$$

where $MEC(\cdot)$ 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 $L^{\Omega}_{i,j}$ 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

Using the Skeleton Updater weights we are able to keep in sync the S joints when C is deformed. In fact, we update the joints of S at positions $\{a_j\}$ as a linear combination of cage vertices with

$$a_j = \sum_{k=1}^{c} \psi_{j,k} \mathbf{c}_k, \tag{3.6}$$

where the $\psi_{j,k}$ are the weights 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'. In order to trigger these changes, we update each skinning transformation $\mathbf{T}_j$ of $\mathcal{T}$ by keeping its rotational component $\mathbf{T}^r_j$ unchanged and by recomputing its translational component $\mathbf{T}^t_j$ 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 CoRs method makes use of per-vertex centers of rotations p_i for vertices i, which are pre-computed following Equation 2.5, and our framework is compatible with this skinning algorithm.

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 2.5 in matrix form as $CoRs = \Phi_{CoRs} \cdot M$, and using the fact that the rest pose mesh M is expressed as $M = \Phi \cdot C$, we can pre-compute the matrix $\Lambda := \Phi_{CoRs} \cdot \Phi$ when computing the centers of rotation, and reposition them at run time using

$$CoRs = \Lambda \cdot C. \tag{3.7}$$

Doing so, we assume that the area terms in the surface averaging remain similar (see Equation 2.5). 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 the quality of the resulting deformation negatively (see Figure 3.9). In particular, vertices with similar input skinning weights are still transformed by the same rigid transformation. Lastly, our joint repositioning method is highly compatible in spirit with the CoRs 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.

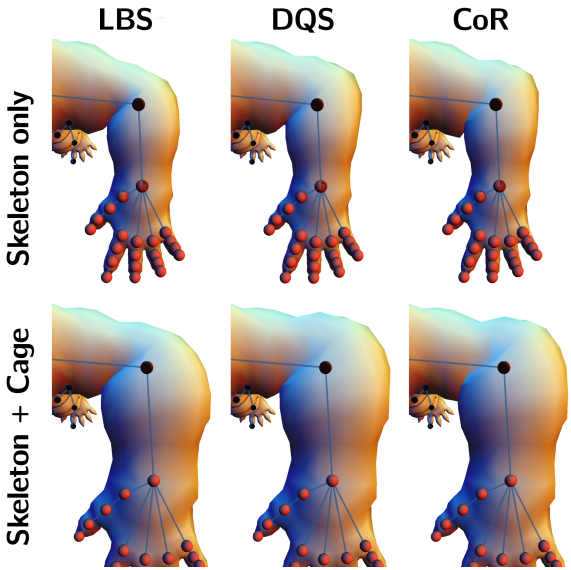
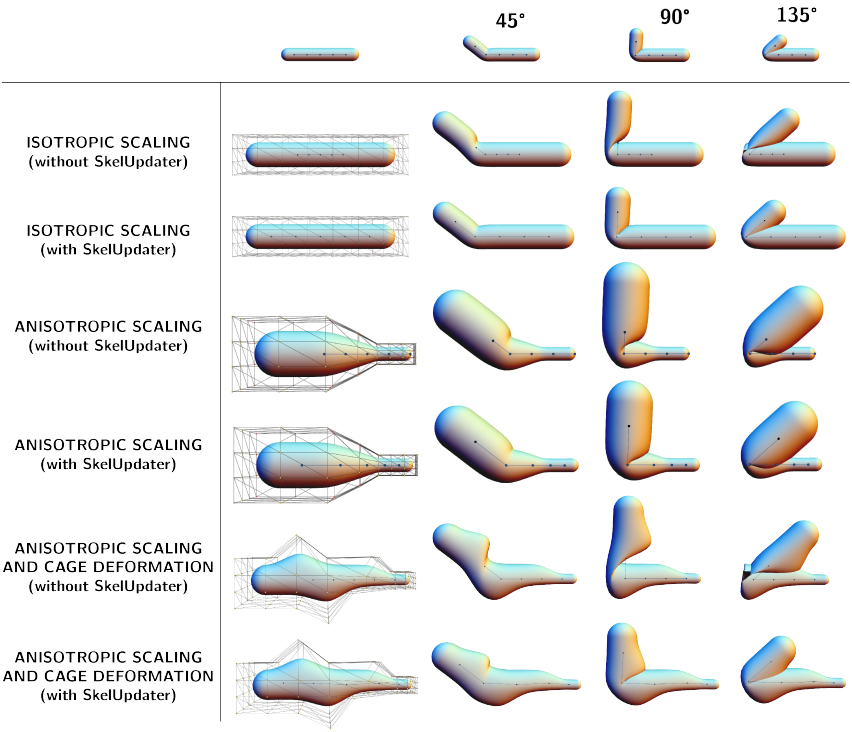


Figure 3.9: Results obtained with various alternative skinning methods implemented in our framework. The top row shows deformations obtained with a skeleton edit only (a 90 degrees rotation of the elbow). The bottom row shows results with an additional cage edit (a uniform stretch of the arm). Our joints and CoRs dynamic repositioning method handles both transformations in a natural manner.


45°
90°
135°
ISOTROPIC SCALING
(without SkelUpdater)
ISOTROPIC SCALING
(with SkelUpdater)
ANISOTROPIC SCALING
(without SkelUpdater)
ANISOTROPIC SCALING
(with SkelUpdater)
ANISOTROPIC SCALING
AND CAGE DEFORMATION
(without SkelUpdater)
ANISOTROPIC SCALING
AND CAGE DEFORMATION
(with SkelUpdater)

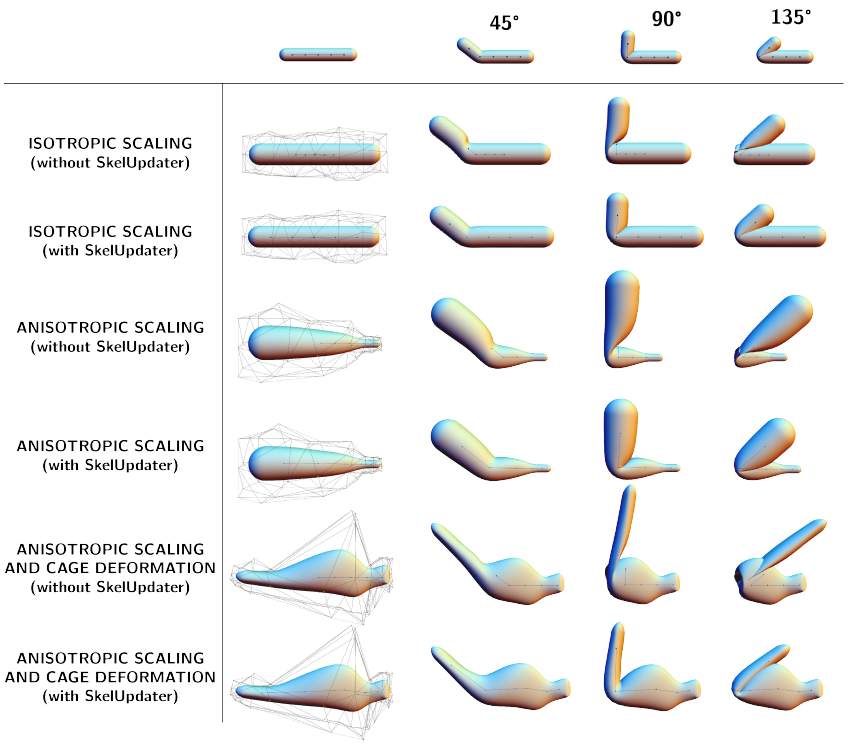


Figure 3.10: A straight bar bent at 45, 90 and 135 degrees using LBS, on top of which we applied various skeleton edits (isotropic and anisotropic scaling, single handle displacement). When the skeleton updater is disabled, cage edits move the skeleton away from the correct centers of rotation, generating various visual artifacts. 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 (first table) and irregular cage (second table).

3.4 Reverse Cage Deformer

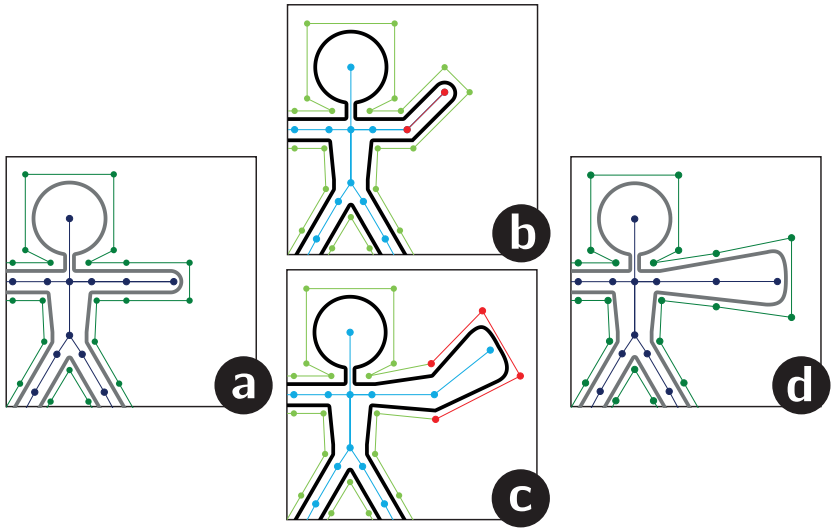


Figure 3.11: The Reverse Cage Deformer operator allows us to transfer the edits performed on the current pose cage to the rest pose cage. Given the initial rest pose (a), editing the current pose skeleton (b) and the current pose cage (c), using the Reverse Cage Deformer operator we are able to invert our framework, translating the current pose edits to the rest pose (d).

One of the ideas of our pipeline is that the cage skinning algorithm deforms the character in its rest pose, but often, during the posing and deformation phase, users need to specify those deformations on the current pose character. So, often, the C' is used as an interaction cage to specify the deformations we want to apply on the character. For this reason we introduce the Reverse Cage Deformer component that translates the edits performed on C to C'. In this way we can apply the cage deformations to the rest pose character. We need to do this because if we apply the cage edits directly to the current pose character this edits are deleted by the skeleton skinning. The skeleton skinning, in fact, use a relative approach to compute the skeleton deformation at each frame starting from the rest pose, and discarding all the edits performed directly on M'. Instead, translating the C' modifications to the rest pose we are able to preserve them.

The main advantage of the Reverse Cage Deformer is that it allows us to simplify the user interaction. For example, if the user wants to inflate the arm of a character in its current pose, the Reverse Cage Deformer transfer this modification by inflating the arm in the rest pose.

To do this, we need to understand how to inverse our framework, so that the edits in C' are transferred to C. The idea is to express the generic modification $\partial C'$ as a function of ∂C through the sequence $C \to S \to S' \to M' \to C'$ as described in Section 3.1, and we finally reverse this function. In order 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.

3.4.1 Mathematical formulation

In order 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 the current cage, respectively. 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 $\mathbf{T}_j^t$, where, as previously, $\mathbf{T}_j^r$ and $\mathbf{T}_j^t$ 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

$$\begin{cases} A = \Psi \cdot C \\ \tilde{M} = \tilde{\Phi} \cdot C \\ \tilde{M'} = \tilde{\mathcal{R}} \cdot \tilde{M} + \tilde{\Omega} \cdot T \\ C' = \tilde{\Phi}^{-1} \cdot \tilde{M'} \ , \end{cases} \tag{3.8}$$

where, as in section 3.2, $\tilde{\cdot}$ (like $\tilde{M'}$, $\tilde{M}$, $\tilde{\Omega}$, and so on) is used to denote all quantities that require only the subset of mesh vertices selected by MaxVol (the cage updater can require the transformed position of those

vertices only). We can observe that the third equation is nothing but a matrix expression of Equation 2.1, where the rotational and translational components of the transformation at each vertex have been separated, following Equation 2.3.

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 3.8 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 $\{\mathbf{T}_k^t\}$ in the following manner

$$\begin{cases} \mathbf{T}_j^r \cdot (a_j + \partial a_j) + \mathbf{T}_j^t + \partial \mathbf{T}_j^t = a_j + \partial a_j & \text{if } j \text{ is a root} \\ \mathbf{T}_j^r \cdot \partial a_j + \partial \mathbf{T}_j^t = \mathbf{T}_f^r \cdot \partial a_j + \partial \mathbf{T}_f^t & \text{if } j \text{ has father } f, \end{cases} \tag{3.9}$$

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 $\{\mathbf{T}_k^r\}$ of the skeletal deformation $\mathcal{T}$. This system can be rewritten as

$$\begin{cases} (I_3 - \mathbf{T}_j^r) \cdot \partial a_j = \partial \mathbf{T}_j^t & \text{if } j \text{ is a root} \\ (\mathbf{T}_j^r - \mathbf{T}_f^r) \cdot \partial a_j = \partial \mathbf{T}_f^t - \partial \mathbf{T}_j^t & \text{if } j \text{ has father } f, \end{cases} \tag{3.10}$$

or, in matrix expression

$$\mathcal{A}_R \cdot \partial A = \mathcal{B}_{\text{topo}} \cdot \partial T. \tag{3.11}$$

We note that, while $\mathcal{A}_R$ depends on the current skeletal deformation parameters (the linear part $\{\mathbf{T}_k^r\}$), $\mathcal{B}_{\text{topo}}$ 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 cages offsets.

Finally, we can gather the previously derived equations to obtain the $c \times c$ linear system

$$\left(\tilde{\mathcal{R}} \cdot \tilde{\Phi} + \tilde{\Omega} \cdot \mathcal{B}_{\text{topo}}^{-1} \cdot \mathcal{A}_R \cdot \Psi\right) \cdot \partial C = \tilde{\Phi} \partial C' \tag{3.12}$$

that can be resolved efficiently.
Note that when no skeletal deformation is performed i.e., the linear part $\{\mathbf{T}_j^r\}$ is composed of Identity matrices only and all translations $\{\mathbf{T}_j^r\}$ 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: first, all matrices in Equation 3.12 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 the mesh used for inversion, the system in Equation 3.12 is exactly invertible, and the loop $C' \rightarrow C \rightarrow S \rightarrow S' \rightarrow M' \rightarrow C'$ is **exact**.

If we use the the least square version rather than the MaxVol (as described in the Cage Updater section) for the inversion process, this will produce an approximate loop. While the approximation is extremely subtle and unnoticeable, the biggest issue in this approach is that it result in reduced performance: we can not obtain results that are fast enough for a modeling session on large models, as the user have 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 3.12 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 use of a ghost mesh that is deformed with LBS and use this ghost mesh to drive the fitting of C' 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 $C' \to C \to S \to S' \to M' \to C'$ 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.

3.5 Additional Considerations

3.5.1 Keyframing

Animation authoring requires artists to create keyframes that represents the evolution of the movement of a digital character. Consequently in each keyframe, the character must be deformed. In our framework, we allow animators to define keyframes for skeleton and/or cage based deformations.

From a technical point of view, there are two ways to define keyframes using our framework, bridging cages and skeletons deformations: asynchronous and synchronous keyframing.

Asynchronous

The asynchronous keyframing allows the animator to maintain skeleton and cage animations independent, making possible to "swap" animations (or blend different animations together) at runtime. In fact, if a walking animation is playing back for the skeleton, and a chest inflating one in playing for the cage, we are able to change the walking animation for the skeleton with a different one easily at any time. The same is valid for the cage one.

In this case, skeleton and cages have their own independent set of keyframes (rotations for the S', and translations for C), and during the computation of the in-betweens the Skeleton Updater operator is invoked to compute the S translations. If it's needed, we can also invoke the Cage Updater to compute the updated C'. Furthermore, if the cage keyframes represents C' translations, we can invoke the Cage Reverse operator too.

The main disadvantage of this approach is that it require the implementation of the Skeleton Updater operator (and the Cage Updater and Cage Rev if used), so a little computational overhead is added to the animation pipeline (even though, as shown in Chapter 4 the computation time remain perfectly suitable for real-time applications).

The main advantage, instead, is that this approach is very versatile because allows us to swap dynamically skeleton and cage animations independently at run time.

Synchronous

The synchronous keyframing allows the animator to precompute (bake) the animation, making it usable without the need of reimplement the whole pipeline. In fact, using the C translations, S Skeleton Updater translations

and S' rotations as key parameter for each keyframe, we are able to play the previously authored cage+skel animation, without the need of the SuperCages operators.

The main drawback of this approach is that the cage and the skeleton animation are merged together, and a post-process or a post-bake edit of the animation is a extremely difficult task.

The main advantage is that it allows to reproduce the animation just computing slerp and linear interpolation, without the need to use the framework operators.

This approach is useful if the animation is not going to change during the playback, for example, in cutscenes.

3.5.2 Implementation and re-implementation

As described in the previous sections, the framework is composed of different operators. Because of this modular design, based on the necessities, there is no need to implement all the operators and the whole pipeline. In fact, there are several cases where some operators can be ignored and bypassed from re-implementation.

The simpler case is the one already described in 3.5.1, where the animation is fixed and is not edited after its authoring. In this case, the S, C and S' are already computed and keyed, so no operator is used, and we do not need to reimplement any of them. We can use directly the classic Skeleton and Cage Skinning algorithms.

Another case is the one where the user interaction is not needed for the S' edits. So, the reimplementation of the Reverse Cage Deformer described in 3.4 can be bypassed.

Last, but not least, if there is no need to recompute the C' at the end of the whole pipeline to obtain a current pose cage through which we are able to interact with it, also the Cage Updater described in 3.2 can be bypassed.

Chapter 4

Results

We have implemented the SuperCages framework as a single-threaded C++ program. Models have been either manually 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 some of them. For the others, and in case the skeleton rig was missing, we manually created them using Maya [may19] or Blender [Ble19]. In terms of performances, our hybrid modeling 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 4.1).

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 pratictioners depending on their taste and needs. For the skeleton part we implemented the two most popular skeleton-based deformation methods, namely Linear Blend Skinning [MTLT89], Dual Quaternions [KCŽO08], 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, bulging for DQS). A side by side comparison between these three alternatives is shown in Figure 3.9. For the skinning weights, although various automatic methods exist in 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

Model	Verts	Skel joints	Cage handles	SkelUp preprocess *ms*	CageUp preprocess *ms*	CoR update *ms*	CageRev update *ms*	LBS frame *ms*	DQS frame *ms*	CoR frame *ms*
Arm (coarse cage)	2089	24	28	401	5	$\leq$1	3	0.88	1.62	2.26
Arm (medium cage)	2089	24	58	417	29	$\leq$1	20	1.61	1.53	1.63
Arm (fine cage)	2089	24	79	439	45	$\leq$1	40	2.22	1.84	2.17
Warrok	6557	64	104	2672	106	$\leq$1	83	7.81	7.66	9.51
Ely	7512	64	88	3018	75	$\leq$1	56	9.24	8.73	9.99
Airplane	41425	19	69	3738	44	5	30	23.70	21.68	26.40
Timber Rattlesnake	120066	98	44	46771	32	9	16	50.03	39.43	52.35

Table 4.1: Performances of our modeling system, measured on a MacBook pro early-2015 equipped with an Intel i5 processor, 8GB of RAM and and Intel Iris 6100 GPU. Columns labeled **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 centers of rotation each time SkelUp is executed (this update does not occur when LBS or DQS are 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 modeling 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.

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 2.7 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 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 3.10 we evaluate our skeleton updater with a synthetic shape, consisting on a straight bar bent at 45/90/135 degrees using LBS. Editing the bar with the cage moves the skeleton away from the correct centers of rotation, and without the skeleton re-fitting procedure described in Section 3.3 extremely evident artifacts 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 dfferent 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 4.1 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 on a cage.

Hybrid modeling

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

deformation paradigms. 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 modeling is given in Figure 4.2, where several variations of an airplane are produced from a single item. Skeleton bones are used to control the rigid parts the plane (e.g. to bend the core and the 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 in using the various shapes it produces as keyframes, to guide a computer generated animation sequence. In Figure 4.3 we show a few interpolated frames of an animation, obtained keyframing some of the airplanes shown in Figure 4.2. 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 4.4 we show a skeleton driven punch sequence downloaded from Adobe Mixamo [mix19], which we 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 new interesting effects. In this specific case only one manually edited keyframe was used, 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 detail. Similar results are also shown in Figure 4.5.

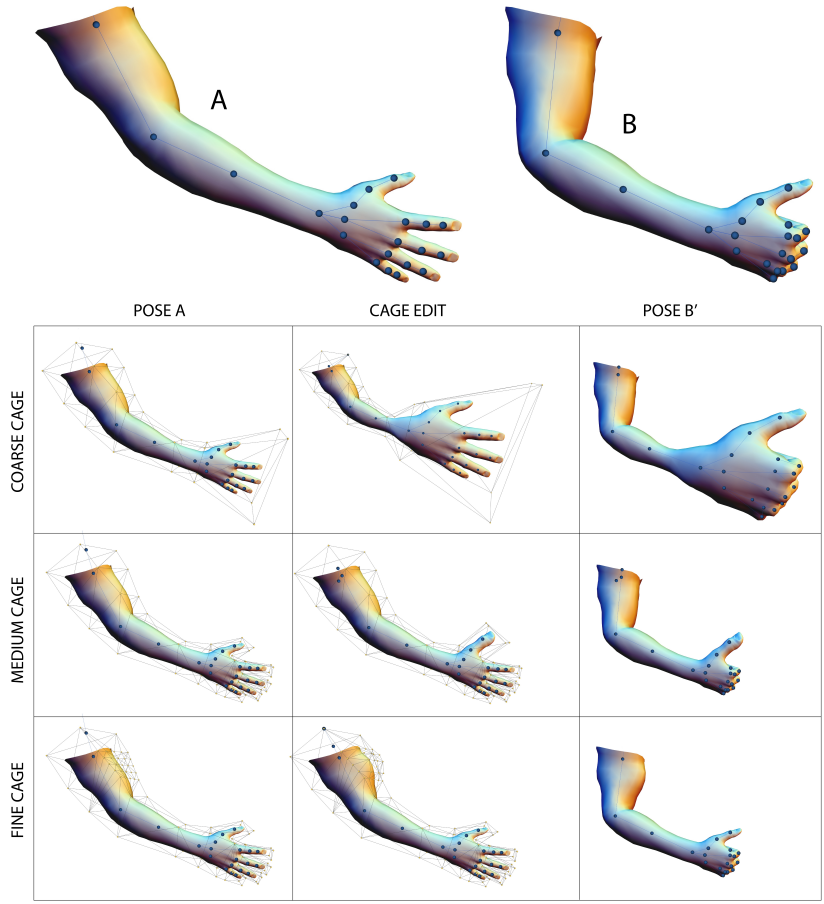


Figure 4.1: Our method can seamlessly combine edits defined on skeletons and cages that operate on features at different scales. Here a simple arm bent with a skeleton (left) is enriched with additional edits with three alternative cages that operate at different levels of details (right). The coarse 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.

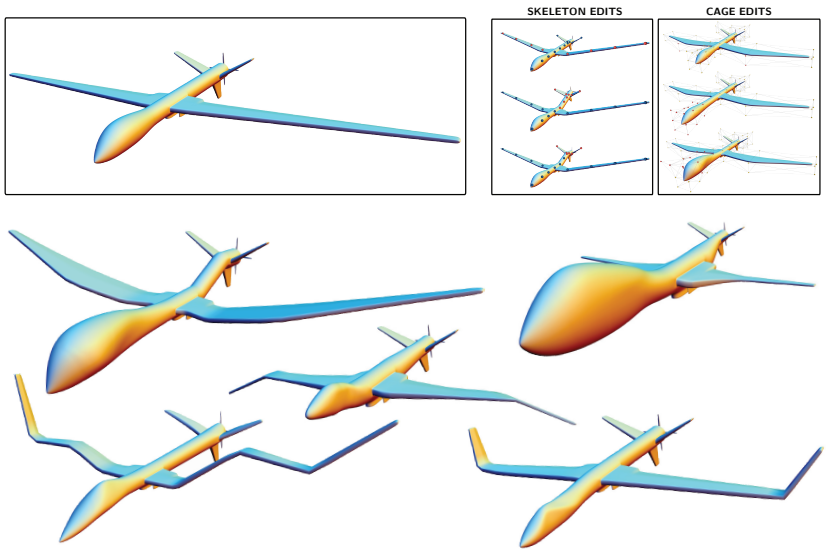


Figure 4.2: Jointly acting on skeletons and cages allows to easily control complementary aspects of the modeling and explore the space of shapes starting from a simple example (top left). Skeletons are best to bend tubular parts and, more in general, deform the rigid components of a shape. Cages are more appropriate to control locally smooth deformations, such as changes of the local thickness of the airplane or the profile of its wings.

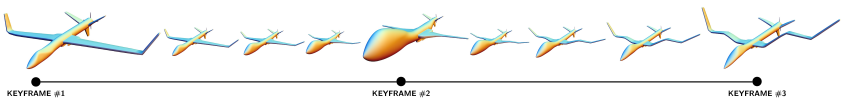


Figure 4.3: Using deformed models as keyframes we can create computer animations. Bone rotations are interpolated using Slerp [DKL98], cage edits are linearly interpolated.

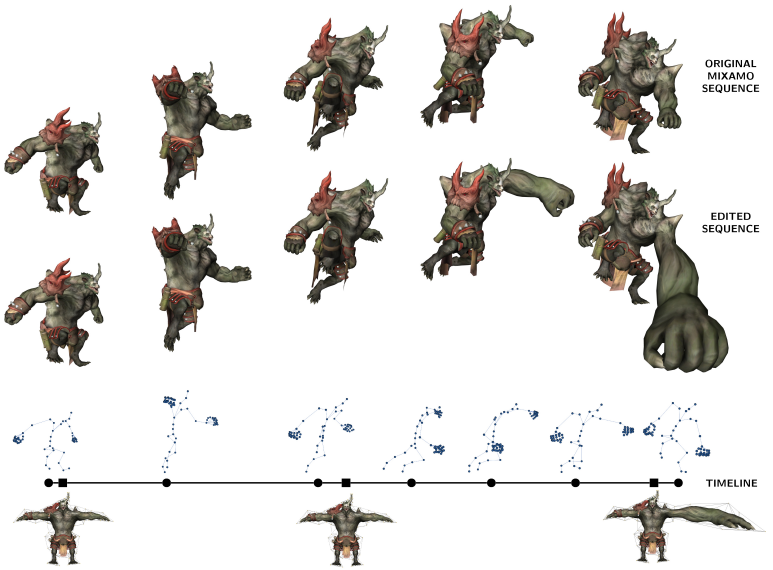


Figure 4.4: Top: a legacy skeleton-based animation downloaded from Adobe Mixamo [mix19]. Middle: an edited sequence obtained by inflating the punch with a control cage. Bottom: the animation timeline, with both skeleton keyframes (circles, from Mixamo), and cage keyframes (squares). The first two cage 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 that 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.

Figure 4.5: Our framework allows animating and deforming characters simultaneously. In the top row we show the Mixamo's Ely character [mix19], 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.

Figure 4.6: Another example with the Mremireh character from Mixamo [mix19]. On the bottom row we deformed the hands while it moves.

Figure 4.7: Another example with the Warrok character from Mixamo [mix19]. In this case, the Warrok mesh has been deformed with our framework, and then rendered in Maya [may19]

Part III

Tools for deformations

Chapter 5

CageLab

In this chapter we introduce **CageLab** [CCLS18], a software tool for the visualization, editing and assessment of animation cages. The main purpose of is to support this growing interest for cage-based animation, providing an open source and easy to install shared platform were researchers and practitioners can interact with a system that allows them to:

- **Animate a digital character**, setting a number of key-frames and interpolating between them. accepts data in the most common file formats used in our community (e.g. OFF, OBJ), and can internally compute barycentric coordinates of various types. Resulting deformations can be closely inspected in a 3D canvas, controlling that the impact of any action on the nodes of the cage is sufficiently smooth and local;

- **Evaluate a cage**, visualizing it on top of the digital character, and checking whether its nodes are well-positioned with respect to it (and the poses one wants to realize). can also be used to directly compare two given cages, posing the character with both of them and comparing the obtained deformations. To this end, users can exploit a convenient copy paste system for camera parameters, which allows to observe different versions of the same pose always from a fixed point of view, with same perspective and amount of zoom;

- **Evaluate barycentric coordinates**, plotting the relation between each cage vertex and the underlying digital character. (Figure 5.1). As for cages, direct comparisons between alternative barycentric coordinates can be created by fixing a point of view and plotting

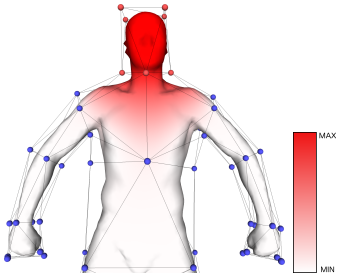


Figure 5.1: In order to compare the smoothness and locality of alternative barycentric coordinates, allows to plot them with respect to a selection of cage node (see red spheres). This selection can be composed of a single node or by a set of the cages handles.

cage-character attraction with respect to a specific node. These visuals are very popular in literature [ZDL+14b], and allow to easily compare locality and smoothness of each tested coordinate. New barycentric coordinates can be loaded into the system in the form of ASCII files;

- **Take snapshots or videos** of an animation, obtained interpolating a sequence of key-frames. This is a useful feature for researchers, to create images for their papers and content for the accompanying videos;

- Last but not least, can be used for **educational** purposes, both for preparing educational material, but also as support tool to teach animation at universities and high schools.

We present here the main features of our tool. Cage-based deformations have received growing attention in recent years. Similarly to tools that were released by the community and sustained the growth of skeleton-based techniques [BP07, JP+17], we believe that CageLab can sustain researchers and practitioners who want to improve the cage-based animation pipeline, as well as compare their ideas with the state of the art in the field.

CageLab is essentially a key-framing system to create a complete animation pipeline. In this way, the animation can be exported and easily imported in another external software for different final purposes. Note that professional software such as Maya, Blender or 3D Studio Max in general offer a wider set of functionalities, such as enveloping [LCF00b],

blend shapes [JTDP06] and a variety of other deformation methods. These tools are intended for professionals (i.e. animators), therefore they are difficult to master and may be overly complex or intimidating for a young researcher.

5.1 User-Interface

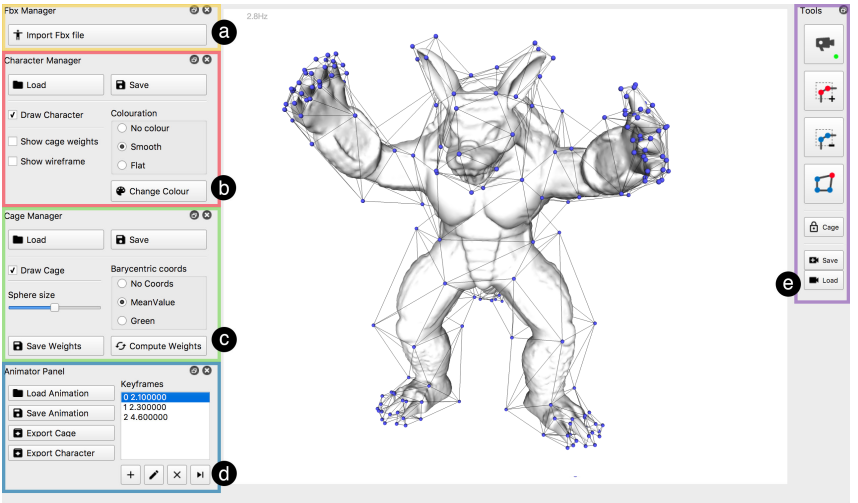


Figure 5.2: The CageLab User Interface. On the left side the FBX Importer is highlighted in yellow (a), the Character Manager panel in red (b), the Cage Manager panel in green (c), and the Animator Panel in blue (d). On the right side there is the Tools sidebar (e). The central part of the UI includes the canvas.

During the development of CageLab (fig. 5.2), we have been extremely motivated to develop a user-friendly as well as lightweight User Interface.

The main window of CageLab is composed of three sections:

- The central section includes the Canvas, where the three-dimensional character mesh and the cage are represented, and where the user can interact with them.
- On the right side, the Tools sidebar enables the user to select the canvas interaction modes, and activate other features described in the next sections.
- On the left side, there are four panels:

- The first one, on the top, is related to the **FBX Importer**. It is useful to open a compatible fbx file (containing the character mesh and its cage).
- The second one, the **Character Manager panel**, enables the user to configure the settings related to the character mesh rendered in the canvas.
- In the **Cage Manager panel**, the user can configure the settings related to the cage rendered in the canvas and to the cage weights.
- The last one, the **Animator Panel**, enables the user to import and export the cage animation, and manage all the keyframes that compose the animation.

In the next paragraphs we will discuss in details every single function provided by the User Interface (UI).

5.1.1 The Canvas

The Canvas is the UI element used to render the cage and the relative character mesh, and to directly interact with the user. After the interaction mode is selected through the sidebar or by using the keyboard shortcuts, the user may use the mouse click or the mouse wheel to perform different tasks, such as camera movements, cage vertex selection or deselection and cage deformation. These actions are described in section 5.1.2.

The three-dimensional character mesh will be rendered with the graphical settings specified by the user in the *Character Manager* panel. The cage will be rendered as a wireframe mesh, with each vertex (called also handle) rendered as coloured sphere, red if selected, blue otherwise. Selected vertices are the ones involved in the deformation process.

In the lower side of the canvas small snippets of text with graphical hints and feedback are shown. They help the final user to understand which action is being performed.

5.1.2 Tools sidebar

The Tools sidebar on the right side of the User Interface allows to perform several actions, like the activation of different interaction modes.

The first four buttons from the top represent the available interaction modes (Camera Mode, Selection Mode, Deselect Mode, Deformation Mode). The active Interaction mode will be characterized by a green dot. The fifth button allows to lock the cage, not allowing the user to perform deformation on it. The sixth button (Camera Save) allows to Save the current camera

point of view, allowing to restore it later after a modification, using the last button (Camera Load).

In order to make it easier for the user to understand which action is performed by each command button, we designed every action button trying to use clear and intuitive icons.

Camera Mode

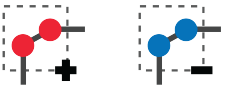
When this mode is active, the interaction with the camera is made possible by moving the mouse over the canvas while the following buttons are pressed (as a typical 3D modeling software):

- The **left mouse button** rotates the camera
- The **right mouse button** translates the camera
- The **mouse wheel** scrolling enables the user to zoom in/zoom out the scene

Through the activation of the Camera Mode (see icon aside), the scene camera and the point of view can be manipulated by the user. This Interaction Mode can also be activated using the **C** keyboard key and is the default interaction mode.

Select/Deselect Cage Vertex Mode

These interaction modes (see icons aside), allows the user to select or deselect one or multiple handles of a cage. The selected handles will, then, be involved in the deformation process. To select/deselect a single handle, the user must simply click on it. To choose multiple handles, the user needs to press the left mouse button on the canvas, move the cursor over the desired handles, and then release the left button.



This interaction mode can be activated by pressing the **S** keyboard button for selection or **R** for deselection. Besides, if another interaction mode has been already activated, it is also possible to select the cage vertices preserving the active interaction mode, by pressing the **SHIFT** keyboard button for selection or **ALT** for deselection while clicking and dragging the mouse on the desired area. Once the SHIFT or ALT key are released, the previous interaction mode is restored.

Cage Deformation Mode

The Cage Deformation Mode (see icon aside), enables the user to deform the selected cage handles by moving them in space and consequently deforming the associated mesh.

It is possible to rotate the handles along their barycenter by clicking the left mouse button and dragging the cursor on the canvas. By clicking the right mouse button and dragging the cursor, it is possible to translate the selected cage vertices. The user can also scroll the mouse wheel to expand or contract the handles around their centroid, in order to inflate or deflate the mesh.

Every time a cage vertex deformation is performed, this deformation will be propagated to its mesh accordingly to the selected barycentric coordinate.

This interaction mode can be also activated by pressing the **D** button, or temporarily pressing the **CTRL** key during the mouse manipulation. Using the **x**, **y** or **z** key it is possible to constrain the cage vertices rotation and translation along the x, y and z axis of the camera point of view.

5.1.3 Character Manager panel

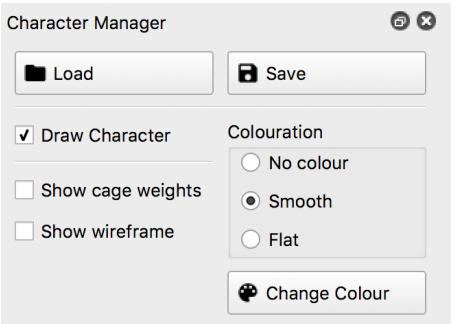


Figure 5.3: A screenshot of the Character Manager panel.

The Character Manager panel (fig. 5.3) provides all the functionalities and personalization settings related to the character mesh that is rendered into the canvas and is deformable using the cage.

Load and **Save** buttons allow the user to import (or export) a triangle mesh file. The format used for these operations is the *.obj*, *.off* or *.ply* format.

The **Colouration** radio buttons allow the user to choose the rendering options of the character mesh, using a smooth or a flat triangle shading.

The **Draw Character** checkbox can activate or deactivate the rendering of the character mesh on the canvas.

The **Show Wireframe** checkbox enables or disables the rendering of the character mesh wireframe. It is possible to render only the wireframe, without showing the mesh surface (flat or smooth), activating the wireframe checkbox and using the *No Colour* colouration setting.

The **Change Colour** button allows the user to choose the character mesh colour.

The **Show cage weights** checkbox (fig. 5.1) allows the user to observe the influence of the selected cage vertices over the character mesh, based on the current cage weights. The red parts of the character are the areas more involved by the selected cage vertices (or handles) during the cage deformation process. The blue parts, instead, are not influenced by those handles.

5.1.4 Cage Manager panel

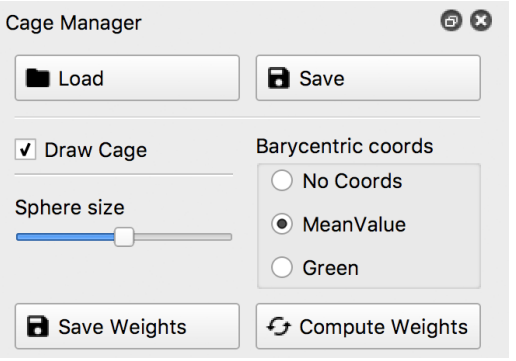


Figure 5.4: A screenshot of the Cage Manager panel.

The Cage Manager panel (fig. 5.4) provides all the functionalities and settings relative to the cage that is rendered into the canvas.

Load and **Save** buttons allow to import (or export) the cage mesh from (or in) a file on the hard drive. The file will be saved in *.obj*, *.off* or *.ply* format, which represents the cage as a triangle mesh.

The **Draw** checkbox allows the user to activate/deactivate the cage rendering on the canvas.

The size of the cage spheres is set through the **Sphere size** slider. By default, this value is set to 0.5% of the diagonal of the cage bounding box. By moving the slider to the left or to the right, the sphere size may be decreased or increased.

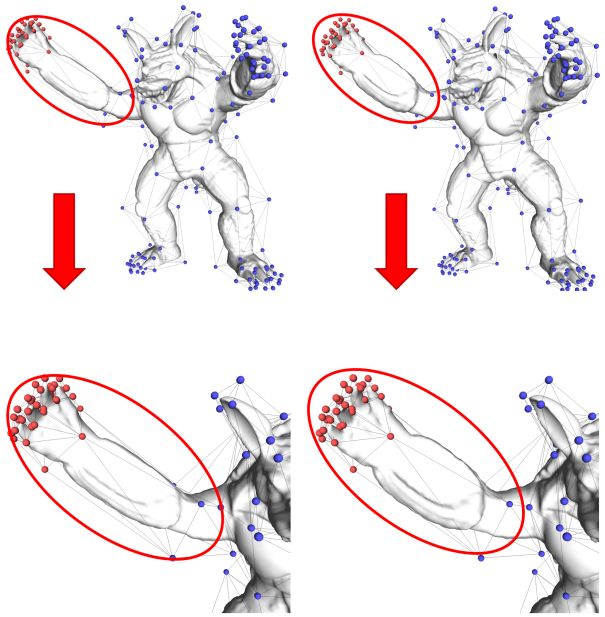


Figure 5.5: Stretching Armadillo's arm with Mean Value (left) and Green (right) coordinates. Green coordinates better preserve surface details (see closeup). CageLab allows to switch between them in real time, so that the use can spot the differences and change barycentric coordinates depending on the intended deformation.

The **Compute Weights** button allows the computation of the *Mean Value Coordinates (MVC)* and *Green Coordinates*, which, in this way, can be used in the deformation process. Subsequently, this button activates the *Barycentric Coords* selection radio-buttons.

The **Barycentric Coords** radio-buttons allow the user to choose what kind of barycentric coordinates must be used in order to generate the deformation of the character mesh using the cage (MVC or Green).

With the **No Coords** setting, the deformation of the character will be disabled. This is particularly useful if we want to edit the cage easily, moving its vertices to better envelop the mesh but without generating a character deformation.

The **Save Weights** button, allows the user to save on a text file the active barycentric coordinates of the current character.

5.1.5 Animator panel

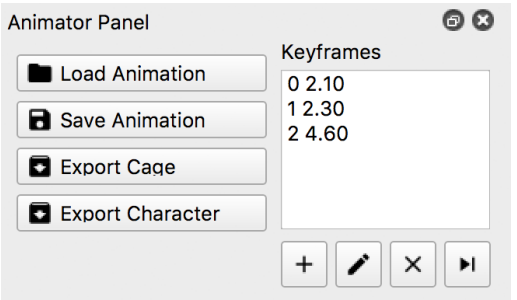


Figure 5.6: A screenshot of the Animator panel.

The Animator panel (fig. 5.6) provides all the functionalities needed to define the keyframe for the character animation.

On the right side of the panel, a list of all the animation keyframes is available. Each keyframe is defined with its sequence number and its timing (in seconds). Clicking on a keyframe on this list, it will be shown in the canvas.

The user can add, edit and erase a keyframe. Each operation can be performed through the dedicated buttons placed below the keyframes list.

When all the keyframes are defined, the user is able to save the animation sequence on a txt file using the **Save Animation** button. The saved animation can be reloaded in another session using the **Load Animation** button.

Using the **Export Cage** or **Export Character** buttons the user is able to export the sequence of all the deformed cage keyframes or character keyframes. Every keyframe is saved as a single obj or ply. The name of each file starts with a user defined string and the timing of the keyframe.

5.2 Implementation details

We have implemented CageLab as a single threaded C++ application on a MacBook Pro equipped with a 2,7GHz Intel Core i5 and 8 GB of RAM. Our tool relies on the **Qt Framework** and it makes use of **Eigen** [GJ+10] to perform mathematical operations and the library **libQGLViewer** for the creation of the user interface. The UI icons come from the Material Design icon library, but some graphical elements are designed by us. We use FBX SDK to import the fbx files. The developed tool has been successfully tested under both MacOs (Yosemite, Sierra, and Mojave) and Linux (Ubuntu and Elementary) platform.

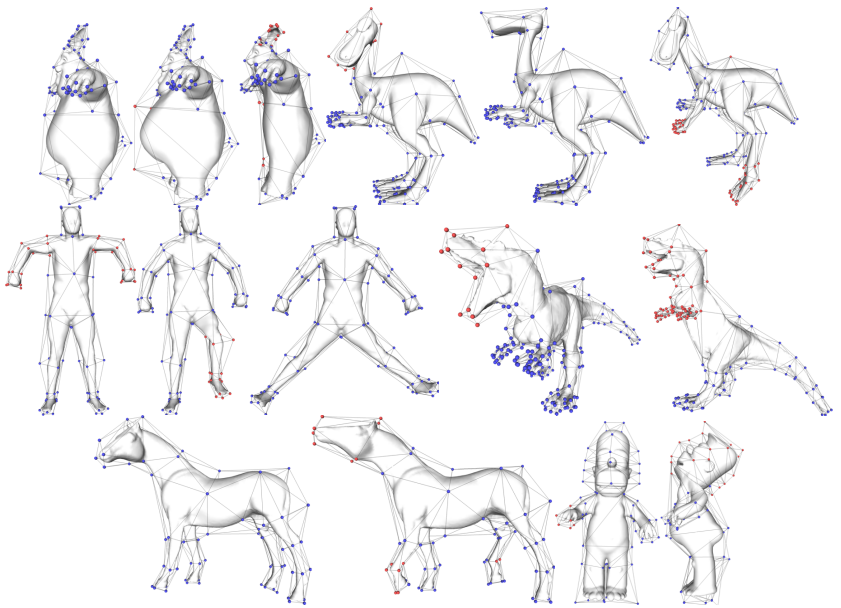


Figure 5.7: An overview of the deformations performed.

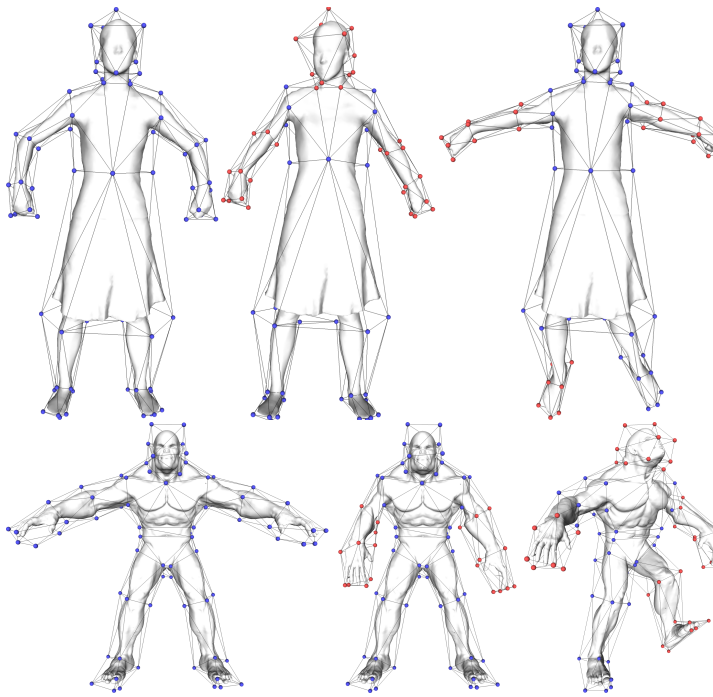


Figure 5.8: Some deformations created using CageLab.

Chapter 6

SuperCages User Interface

In this chapter, we introduce the Graphical User Interface for the SuperCages framework. This GUI is a direct evolution of the CageLab app described in Chapter 5.

While CageLab is focused to cage-based deformation only, here we add the possibility to perform also skeleton-based ones, implementing all the necessary functionalities to manage the relative animations, At the same time, in this new tool we removed some component of Cagelab to avoid to overload the UI with SuperCages unrelated functionalities.

Here we implement some graphical components to create an easier interaction with the SuperCages framework operators. An example is the availability of a second canvas that renders the rest pose character, allowing users to perform edits to the rest-pose character.

We developed the User Interface of our tool trying to keep it as user-friendly and lightweight as possible.

The main window of SuperCages is composed of three sections:

- The central section includes the canvas that is splitted in two parts: the first one on the left side shows the character rig (character mesh, skeleton and cage) in its current pose; the second one on the right side, instead, shows the character rig in its rest pose. The user is able to interact with all the rig structures in order to perform the relative edits.

- On the right side, the Tools sidebar allows the user to activate or deactivate the skeleton updater and the cage updater operators, lock the skeleton and cage deformation, and save or load the camera position.
- On the left side, there are five panels:
  - The first one, on the top, is the **Rig Manager**. It allows to import or export all the rig-related files, like the character mesh, its skeleton, the cage, the skinning weights, and all the support files useful for the various framework operators.
  - The second one, the **Character Manager panel**, enables the user to configure the settings related to the character mesh rendered in the canvas.
  - In the **Cage Manager panel**, the user can tweak the settings related to the cage rendered in the canvas and to the cage weights.
  - The **Skeleton Manager panel**, allows the user to chose which Skinning Algorithm he wants to use for the Skeleton (Linear Blend, Dual Quaternion, or CoR). Here it is possible to activate the root motion movement for the skeleton animation (fig. 6.3).
  - The **Animator Panel**, enables the user to import and export the cage and/or skeleton animation, manage their keyframes, and tweak other settings relative to the animation.

In the next paragraphs we will discuss in details some functions provided by the User Interface (UI).

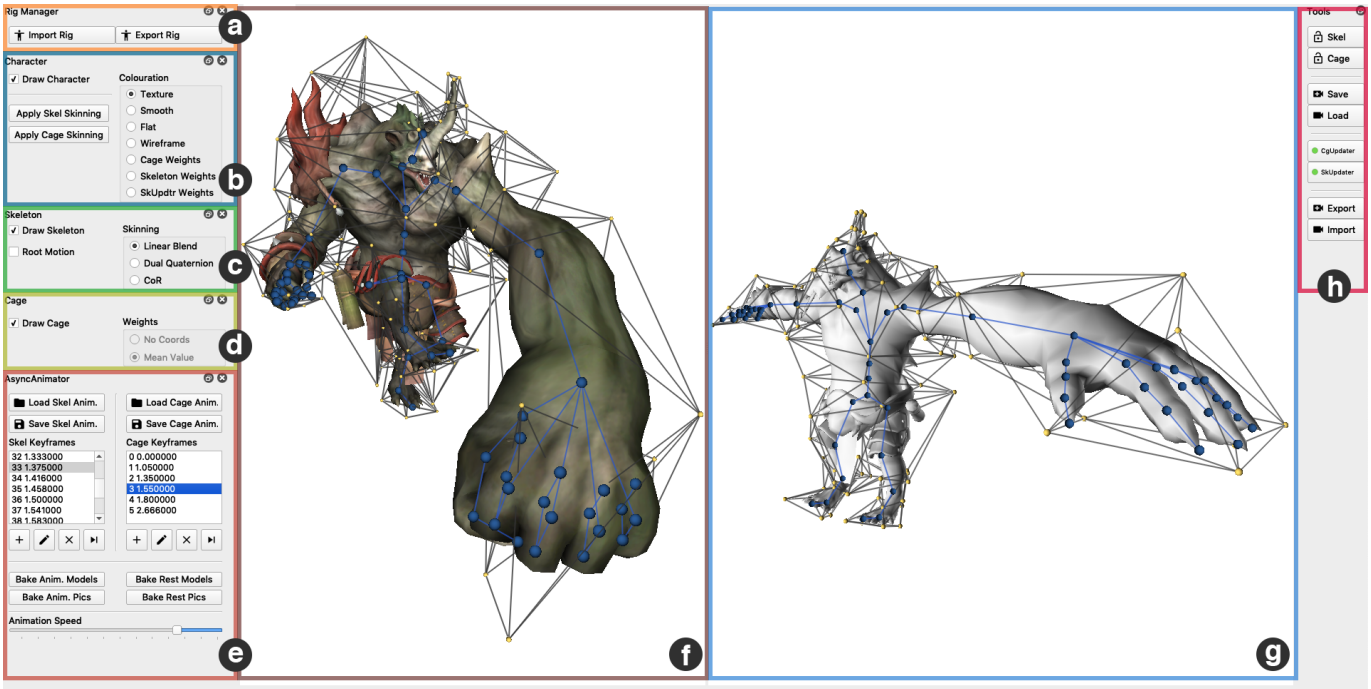


Figure 6.1: The SuperCage GUI. On the left side we can observe the Rig Manager (a), the Character Manager panel (b), the Skeleton Manager panel (c), the Cage Manager panel (d), and the Animator Panel (e). On the right side there is the Tools sidebar (h). The central part of the UI includes the canvas for the current pose rig (f), and rest pose rig (g).

6.1 The Canvas

The Canvas is the UI element used to render the cage, the skeleton and the relative character mesh. It is used to directly interact with the user. It is splitted in two parts: the left side of the canvas is reserved to the M', C', S' current pose character, while the right side is for the M, C, S rest pose character. Similar to CageLab, using keyboard shortcuts the user may use the mouse click or the mouse wheel to perform different tasks such as camera movements, cage/skeleton handle selection/deselection and deformations. The user can, indeed, rotate the S' handles and translate or inflate the C or the C' handles. Note that if the S' cage handles are deformed, the Reverse Cage deformer operator will be invoked as described in Section 3.4 to compute the correspondent C deformation.

The three-dimensional character mesh will be rendered with the graphical settings specified by the user in the *Character Manager* panel. The cage will be rendered as a wireframe mesh, with each vertex (or handle) rendered as coloured sphere, red if selected, blue otherwise. Selected vertices are the ones involved in the deformation process. The skeleton will be rendered as a collection of spheres (one per joint) connected by lines (the bones). Also in this case, the red joints are the ones involved in the deformation, and the blue ones are deselected. Note that, selecting a joint in the skeleton, all its children in the skeleton tree will be selected as well, following the forward kinematic principle.

6.1.1 Managers

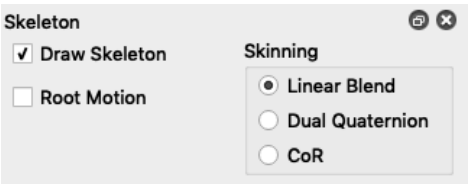


Figure 6.2: A screenshot of the Skeleton Manager panel in SuperCages GUI.

Character The Character Manager panel (fig. 6.3) provides all the customization settings related to the character mesh that is rendered into the canvas.

The main addition compared to CageLab is the presence of the **Apply Skel Skinning** or **Apply Cage Skinning** that force the rendering of the

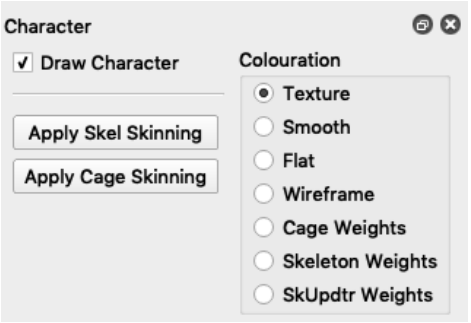


Figure 6.3: A screenshot of the Character Manager panel in SuperCages GUI.

active Skeleton skinning algorithm result or the Cage skinning algorithm result over the Character skin.

The **Colouration** radio buttons allow the user to choose the rendering options of the character mesh. In fact, it is possible to switch between the classical smooth, flat, wireframe triangle shading, and character texture rendering (fig. 4.5, 4.6). It is also possible to display the selected skeleton handle weights influence, or the Skeleton Updater Weights described in 3.3.

6.1.2 Animator

The Animator panel (fig. 6.4) provides all the functionalities needed to define the skeleton and cage keyframes for character animation.

On the top side of the panel, a list of all the skeleton (on the left) and cage (on the right) animation keyframes is available. Each keyframe is defined with its sequence number and its timing (in seconds). Clicking on a keyframe, it will be shown in the canvas.

The user can add, edit and erase a keyframe. Each operation can be performed through the dedicated buttons placed below the keyframes lists.

When all the keyframes are defined, the user is able to save the relative animation sequence on a txt file using the **Save Skel Animation** or **Save Cage Animation** buttons. The saved animation can be loaded in another session using the **Load Skel Animation** or **Load Cage Animation** button.

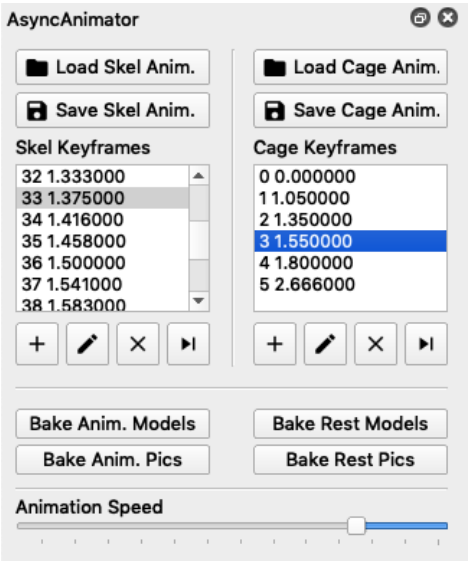


Figure 6.4: A screenshot of the Animator panel in SuperCages GUI.

Using the **Export Current Pose** or **Export Rest Pose** buttons the user is able to export the sequence of all the deformed character keyframes in its current or rest pose. Each keyframe is saved as a single obj or ply file. The name of each file starts with a user defined string followed by the timing of the keyframe.
Lastly, the **Animation Speed** slider, allows the user to speed up or slow down the animation speed.

Part IV

Final considerations

Chapter 7

Conclusions

Different aspects of cage and skeleton-based deformations have been presented in this thesis.

The first topic of this thesis was related to the SuperCage framework. I started from the observation that skeleton-based and cage-based deformations control different aspects of shape modeling, and are to a large extent complementary to one another. We therefore proposed a real-time modeling 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. Three operators has been created: the Cage Updater that keeps in sync the current-pose cage with the skin modifications induced by the skeleton; the Skeleton Updater that refits the skeleton joints according to the modifications performed on the rest-pose cage; the Reverse Cage Deformer that translates the deformations performed on the current-pose cage to the rest pose cage. All the operators are organized in a closed-loop pipeline.

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.

We believe that our contribution, which aims at accessing a much larger deformation space, could support advanced deformation control. A SuperCages paper has been accepted to the CGF journal, and a preprint version is available at [CTL+19]. Furthermore, the source code of this project is publicly available on GitHub (`https://github.com/cordafab/SuperCages`).

Our framework 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 2.7).

The second topic of this thesis has been reserved for Graphical User Interfaces for character deformations.

The first tool presented, CageLab, is intended for researchers and practitioners who want to get acquainted with the digital animation pipeline. It allows users to perform cage-based deformations by using two of the most popular barycentric coordinates (Mean Value and Green coordinates); it allows to compare different cages for the same character; and to compare different differential coordinates (and the deformations they produce). It is also possible to define, export and import animation key-frames. This tool has been publicly released with the aim of supporting the recent cage-based animation growth that we observed in our community.

The second tool, SuperCages GUI, is intended to create deformations with the SuperCages framework by using Skeletons and Cages seamlessly, allowing animators and artists to author animations based on both structures. Compared to CageLab, it has several additional functionalities such as skeleton skinning deformation and the possibility to interact with the rest-pose character rig. Note that all the images related to the SuperCages framework have been produced with the tool itself, with a partial exception for figure 4.7. Instead, it was produced by exporting the deformed obj files from SuperCages GUI, and rendering them in Maya [may19].

Even though these topics are presented in this thesis as separate chapters, they are tightly related. In fact, SuperCages GUI implements the homonym framework presented as the first thesis topic, and it is based on the CageLab code.

7.1 Future Works

In our future work, we plan to extend the SuperCages framework with more deformation controllers (e.g. point handles) which we can incorporate with the same approach to synchronization while remaining agnostic on the specific technique used for their implementation.

Furthermore, nonlinear cage-based deformation techniques such as [LLCO08] could potentially be incorporated in SuperCages, 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 might affect the frame rate and the real-time experience.

We are also planning to extend the framework with a GPU implementation of its algorithms.

Regarding CageLab and SuperCages GUI, we plan to extend them with new features, trying to fix possible issues with the user interaction. In fact, one issue of the graphical representation is that the visualization of controllers for both the skeleton and the cage may sometimes clutter the screen, especially for complex characters requiring numerous skeleton bones and complex control cages. We can see a potential improvement by adopting a dynamic rendering of the controllers, that fades away from the mouse position.

Moreover, we plan to include new definitions of barycentric coordinates and some techniques for automatic rig definition, as well as other skinning algorithms.

In conclusion, the idea is to merge CageLab with SuperCages GUI, creating a tool that allows users to perform all the operations related to cages and skeletons at once.

Acknowledgements

When I began this adventure, I thought that three years was a long time, but I quickly realised this was not the case. However, here I am, having fininshed this challenging journey that in the end has presented me with so many great experiences and has been very rewarding.

Writing these acknowledgements is very difficult because there are so many people I am grateful to, and it is hard to find the right words for you all.

First of all, I must thank my supervisor, Riccardo Scateni, who, over these ten years, has been a fantastic mentor, always managing to push me further while putting so much trust in me.

Thanks also to Marco Livesu, Enrico Puppo, Jean-Marc Thiery and Tamy Boubekeur, who allowed me to reach this goal, guiding me and teaching me so much (without them this thesis would not exist). Thanks also to the reviewers for their precious advice, and, last but not least, I want to say thanks to the BatCave colleagues, to my friends, and, most of all, to my family.

THANKS!

Ringraziamenti

Quando ho iniziato questa avventura pensavo che tre anni fossero tanti, mentre invece mi sono dovuto ricredere piuttosto in fretta. Infatti eccomi arrivato alla fine di questo difficile percorso che, dopo tutto, mi ha regalato tantissime belle esperienze e dato altrettante soddisfazioni.

Scrivere questi ringraziamenti, però, è comunque estremamente complicato perché sono tante le persone da ringraziare, ed è difficile trovare le parole adatte per tutti.

Per primo devo dire grazie al mio supervisor, Riccardo Scateni, che in questi dieci anni è stato un mentore straordinario, riuscendo a trovare il modo di spronarmi a cercare di fare sempre meglio e riponendo in me una enorme fiducia.

Grazie anche a Marco Livesu, Enrico Puppo, Jean-Marc Thiery e Tamy Boubekeur chi mi hanno permesso di raggiungere questo traguardo, guidandomi e insegnandomi tantissimo (e senza cui questa tesi non esisterebbe). Grazie ai revisori per i preziosi consigli, e per ultimo, ma non per importanza, un enorme ringraziamento ai colleghi della BatCaverna, agli amici, ma soprattutto alla mia famiglia.

GRAZIE!

Bibliography

[BCWG09] Mirela Ben-Chen, Ofir Weber, and Craig Gotsman. Variational harmonic maps for space deformation. *ACM Transactions on Graphics (TOG)*, 28(3):34, 2009.

[BKP+10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. AK Peters/CRC Press, 2010.

[Ble19] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2019.

[BP07] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Transactions on graphics (TOG)*, 26(3):72, 2007.

[CCLS18] Sara Casti, Fabrizio Corda, Marco Livesu, and Riccardo Scateni. CageLab: an Interactive Tool for Cage-Based Deformations. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2018.

[CLM+19] Sara Casti, Marco Livesu, Nicolas Mellado, Nadine Abu Rumman, Riccardo Scateni, Loïc Barthe, and Enrico Puppo. Skeleton based cage generation guided by harmonic fields. *Computers & Graphics*, 2019.

[Cor17] Fabrizio Corda. A skeleton/cage hybrid paradigm for digital animation. In *DCPD @ CHItaly*, pages 1–12, 2017.

[CTL+19] Fabrizio Corda, Jean-Marc Thiery, Marco Livesu, Enrico Puppo, Tamy Boubekeur, and Riccardo Scateni. Real-time deformation with coupled cages and skeletons. *arXiv preprint arXiv:1909.02807*, 2019.

[DKL98] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation*, volume 2. Citeseer, 1998.

[FKR05] "Michael S. Floater, Géza Kós, and Martin Reimers". "mean value coordinates in 3d". *"Computer Aided Geometric Design"*, "22"("7"):"623 – 631", "2005".

[Flo03] "Michael S. Floater". "mean value coordinates". *"Computer Aided Geometric Design"*, "20"("1"):"19 – 27", "2003".

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[GPCP13] Francisco González García, Teresa Paradinas, Narcís Coll, and Gustavo Patow. *cages:: A multilevel, multi-cage-based system for mesh deformation. *ACM Trans. Graph.*, 32(3):24:1–24:13, 2013.

[HF06] Kai Hormann and Michael S. Floater. Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph.*, 25(4):1424–1441, October 2006.

[HS08] Kai Hormann and Natarajan Sukumar. Maximum entropy coordinates for arbitrary polytopes. In *Computer Graphics Forum*, volume 27, pages 1513–1520. Wiley Online Library, 2008.

[JBK+12] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. Fast automatic skinning transformations. *ACM Transactions on Graphics (TOG)*, 31(4):77, 2012.

[JBPS11] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. In *ACM Transactions on Graphics (TOG)*, volume 30, page 78, 2011.

[JDKL14] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*, 2014.

[JMD+07] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Transactions on Graphics (TOG)*, 26(3):71, 2007.

[JP+17] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2017. http://libigl.github.io/libigl/.

[JS11] Alec Jacobson and Olga Sorkine. Stretchable and twistable bones for skeletal shape deformation. *ACM Transactions on Graphics (TOG)*, 30(6):165, 2011.

[JSW05] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 561–566. ACM, 2005.

[JTDP06] Pushkar Joshi, Wen C Tien, Mathieu Desbrun, and Frédéric Pighin. Learning controls for blend shape based realistic facial animation. In *ACM Siggraph 2006 Courses*, page 17. ACM, 2006.

[JZvdP+08] Tao Ju, Qian-Yi Zhou, Michiel van de Panne, Daniel Cohen-Or, and Ulrich Neumann. Reusable skinning templates using cage-based deformations. *ACM Trans. Graph.*, 27(5):122:1–122:10, 2008.

[KCvO07] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 39–46, New York, NY, USA, 2007. ACM.

[KCŽO08] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics (TOG)*, 27(4):105, 2008.

[Ken12] Ben Kenwright. A beginners guide to dual-quaternions. *WSCG 2012 Communication proceedings*, pages 1–10, 2012.

[KS12] Ladislav Kavan and Olga Sorkine. Elasticity-inspired deformers for character articulation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA)*, 31(6):196:1–196:8, 2012.

[LCF00a] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 165–172. ACM Press/Addison-Wesley Publishing Co., 2000.

[LCF00b] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and

skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[LH16] Binh Huy Le and Jessica K Hodgins. Real-time skeletal skinning with optimized centers of rotation. *ACM Transactions on Graphics (TOG)*, 35(4):37, 2016.

[LKCOL07] Yaron Lipman, Johannes Kopf, Daniel Cohen-Or, and David Levin. Gpu-assisted positive mean value coordinates for mesh deformations. In *Symposium on geometry processing*, 2007.

[LLCO08] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Transactions on Graphics (TOG)*, 27(3):78, 2008.

[may19] Autodesk maya. `https://www.autodesk.com/products/maya/overview`, 2019.

[mix19] Adobe mixamo dataset. `https://www.mixamo.com`, 2019.

[MK16] Tomohiko Mukai and Shigeru Kuriyama. Efficient dynamic skinning with low-rank helper bone controllers. *ACM Transactions on Graphics (TOG)*, 35(4):36, 2016.

[Möb27] A.F. Möbius. *Der barycentrische Calcul*. J.A. Barth, 1827.

[MTLT89] N Magnenat-Thalmann, R Laperrière, and D Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface'88*, pages 26–33. Canadian Information Processing Society, 1989.

[NS13] Jesús R Nieto and Antonio Susín. Cage based deformations: a survey. In *Deformation models*, pages 75–99. Springer, 2013.

[OBP+13] A. Cengiz Öztireli, Ilya Baran, Tiberiu Popa, Boris Dalstein, Robert W. Sumner, and Markus Gross. Differential blending for expressive sketch-based posing. In *Proceedings of the 2013 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, New York, NY, USA, 2013. ACM.

[Par12] Rick Parent. *Computer animation: algorithms and techniques*. Newnes, 2012.

[RF16] Nadine Abu Rumman and Marco Fratarcangeli. State of the art in skinning techniques for articulated deformable characters. In *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications: Volume 1: GRAPP*, pages 200–212. SCITEPRESS-Science and Technology Publications, Lda, 2016.

[sf19] Sketchfab. `https://sketchfab.com`, 2019.

[SP86] Thomas W Sederberg and Scott R Parry. Free-form deformation of solid geometric models. *ACM SIGGRAPH computer graphics*, 20(4):151–160, 1986.

[sub19] Adobe substance. `https://www.substance3d.com/`, 2019.

[TE18] J-M Thiery and Elmar Eisemann. Araplbs: Robust and efficient elasticity-based optimization of weights and skeleton joints for linear blend skinning with parametrized bones. *Computer Graphics Forum*, 37(1):32–44, 2018.

[TMB18] Jean-Marc Thiery, Pooran Memari, and Tamy Boubekeur. Mean value coordinates for quad cages in 3d. *ACM Transactions on Graphics (Proc. SIGGRAPH 2018)*, 2018.

[TTB12] Jean-Marc Thiery, Julien Tierny, and Tamy Boubekeur. Cager: Cage-based reverse engineering of animated 3d shapes. *Comput. Graph. Forum*, 31(8):2303–2316, 2012.

[TTB13] Jean-Marc Thiery, Julien Tierny, and Tamy Boubekeur. Jacobians and hessians of mean value coordinates for closed triangular meshe. *The Visual Computer*, pages 1–15, 2013.

[WJBK15] Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. Linear subspace design for real-time shape deformation. *ACM Transactions on Graphics (TOG)*, 34(4):57, 2015.

[zbr19] Pixologic zbrush. `https://pixologic.com/`, 2019.

[ZDL+14a] Juyong Zhang, Bailin Deng, Zishun Liu, Giuseppe Patanè, Sofien Bouaziz, Kai Hormann, and Ligang Liu. Local barycentric coordinates. *ACM Trans. Graph.*, 33(6):188:1–188:12, 2014.

[ZDL+14b] Juyong Zhang, Bailin Deng, Zishun Liu, Giuseppe Patanè, Sofien Bouaziz, Kai Hormann, and Ligang Liu. Local barycentric coordinates. *ACM Trans. Graph.*, 33(6):188:1–188:12, November 2014.