University of Cagliari

PhD Degree in
Mathematics and Computer Science

XXXI cycle

Course in Computer Science

Cages and Skeletons in Digital Animation

A Novel Skeleton-based Approach for Cage Generation

Scientific Disciplinary Sector: INF/01

Ph.D. Student:
Sara Casti

Ph.D. Supervisor:
Prof. Riccardo Scateni

Ph.D. Coordinator:
Prof. Michele Marchesi

Final exam: Academic Year 2017 – 2018
Thesis defence: January – February 2019 Session

Abstract

Cages together with skeletons are the most widely used structures to pose a digital character, giving the illusion of its movement. While skeletons are generally used for primary motion, like limbs movements, cages instead, are usually used to enrich primary motion with secondary effects, like body jiggling, character breath, cloth movements etc. In the light of the importance of cages in the animation pipeline, we have investigated two main subproblems related to cage-based deformations techniques: build the cage; provide powerful tools to perform assist in the process of deforming digital models.

Cages are intrinsically more complicated to be created than skeletons. During their design, well-established properties have to be fulfilled: the cage must tightly envelop the original model without intersecting it; it must be coarse enough to be easily manipulated, and it must be shape-aware, i.e., its control nodes should be close to the parts of the model one would like to deform or bend. Due to these hard constraints, cages are often hand-made, and their creation may require hours of extensive work by skilled artists. For this reason, we focused on defining a novel skeleton based approach which allows the user to quickly design high-quality cages for animation; and developing a novel research-oriented software tool to perform cage-based deformations in a lightweight and easy to use environment.

In this thesis after seeing the properties of the cages and the cage-based deformations, we will see the state-of-the-art of the existing cage generation method and their drawbacks. In the core of the thesis, we will look at the details of our approach, the results obtained and a comparison with the cages on state of the art. In conclusion, we will see our novel research-oriented software tool for the visualization, editing and generation of cage-based animation whose goal is to support the growing interest of the research community.

Contents

List of Figures

Chapter 1

Introduction

Computer Animation is one of the fundamental branches of Computer Graphics which adopts geometry processing algorithms, to bring a purely static three-dimensional model into life. The entertainment industries, like cinematography and video games, need to design the virtual world reasonably; this obviously entails reproducing character's movements.

Animating a digital character is labor intensive. In the years, several methods have been developed to simplify the setup of the animation. Moreover artists, researchers, and digital animators require effective tools that allow them to create animations quickly.

To give the illusion of characters' movements the animator has to specify a set of poses for each model in the animated scene. Posing the models through handles, instead of manipulating them directly, facilitates the job of the animators and reduces the time required to accomplish their work. However, to use them properly high skills are required, therefore, such structures can not be easily manipulated from non-expert users.

Among the different handle-based techniques, skeletons and cages are certainly the most widely used structures to setup the animation of a digital character. Cages play a key role in animation, since they provide a smooth control over the volume in their interior, they allow to perform several kinds of deformations, therefore they are often used to enrich primary motions with secondary effects like body jiggling, character breath, cloth movements etc. Skeletons, instead, cannot control such motions, but they are the most appropriate structure to control primary motions, offering an intuitive paradigm of animation similar to the motion in the real world. Cages are intrinsically more complex to create than skeletons. Several properties have to be satisfied during the cage design, to make the consequent deformation work properly. In Chapter 2, in section 2.1 a detailed discussion of these properties can be found.

This thesis focuses on cages, and investigates the two main subproblems related to the cage-based animation setup:

1. designing high quality cages;
2. providing powerful tools with an intuitive user interface which assist in the process of deforming digital models.

In chapter 4, the first subproblem is analyzed. Although apparently it can appear as a trivial problem, it is still a challenging issue, which has received increasing interest in the research field over the last years. In cage-based deformation setup, the most tedious task is the cage construction: it is usually made by hand through modelling software such as Maya, Blender or 3DStudio Max. For this reason several approaches for the construction of cages has been developed. The majority of these works [XLG09, XLG12, XLX15, DLM11, SVJ15, CB17], build an envelope for the input model by using purely geometric approaches. In these works, they address the cage construction in a fully automatic manner, therefore such methods do not provide to the animators the adequate degrees of freedom to realize the animation. The main limitation of the geometric approaches is that they may discard the semantically important parts of the input model, as well as, refine too much some areas that do not require such level of refinement.

In contrast to these approaches, Le and Deng [LD17] propose an interactive approach for cages construction. The user sketches a set of planes, in correspondence of which the cage will be refined. The main limitations of this work are: (1) the location of the handles is restricted to stay on the planes sketched by the user, therefore it can cause some issues, for example impeding the separation of certain parts of the input model or creating self-intersections hard to be fixed by the user; (2) sketching planes to refine the cage can still be hard for non-expert users.

Relatively few works investigate the use of skeleton as a mean to build the cage. In [JZvdP*08, YCSZ13] the cage is built automatically starting from a set of predefined templates associated to each joint of the skeleton provided as input. Their goal is to use such cage to improve skinning, thus avoiding well-known artifacts. The main limitation is the limited number of existing templates, hence the number of joints it can handle; Another limitation is that such methods do not handle models with arbitrary complex geometry.

In the light of these limitations, considering the existing duality of cages and skeleton, and the fact that curve skeletons encode topological information of the model from which they are directly derived, we developed a method where the skeleton drives the construction of the cage. The main idea behind our approach is that control cages can be designed

through a powerful descriptor, the skeleton, which encodes the higher level of information, that geometric methods are not able to represent.

Building high quality cages is a very important and essential task in the cage-based animation pipeline, but it is also extremely important to provide an easy, lightweight and efficient way to manipulate the cage and perform deformations. Chapter 5 of this thesis presents CageLab, a novel tool to perform cage-based deformations.
This thesis is organized as follows:

In chapter 2 the background of handle based animation will be introduced, where in section 2.1 the cage qualities, and the required properties, will be discussed. In subsection 2.1.1 their paradigm of animation will be described. In section 2.2 the concept of skeleton will be provided, in particular focusing on curve skeletons in 2.2.1, and their properties that made this kind of structure the most suitable to guide the construction of the cage.

In chapter 3 the related works to the construction of cages and cage-based deformations will be discussed. In section 3.1 we will talk more about the related works of the generation of cages, where the consideration of existing methods and their limitations will be thoroughly showed.

In chapter 4 our novel skeleton-driven approach for cage generation is presented.

In chapter 5 our novel software tool for the visualization, editing and generation of cage-based animation will be described.

In the final part of this thesis, in chapter 6, the final considerations and the future works are presented and discussed.

Chapter 2

Background

Character animation is a fundamental task in Computer Graphics, which plays a crucial role in contemporary computer games, animated feature films and also virtual reality applications. Digital animation has received, even in the research field, an increasing interest in the last years, because there are still open problems and techniques which can be improved.

The main goals in computer animation are: (i) producing believable deformations, taking into account physical effects, being accurate in case of collisions, and representing secondary motions effects like fat jiggling or cloth movement (ii) simplifying the manual work, and (iii) keep the real-time interactivity during the animation. To realize high quality deformations these requirements, which are essentially in conflict, should be satisfied, therefore a trade-off between believability and performances is necessary.

An articulated 3D model is represented as a polyhedral mesh, generally composed by triangular or quadrilateral faces; the model geometry can be obtained through acquisition, for example by physical scanning, or it can be modeled by an artist with professional tools like Maya, Blender or 3DStudio Max etc.

The animation pipeline is labor intensive. To give the illusion of movement, the animator has to produce a sequence of scenes which will be displayed at certain interval rate. Therefore, the animator has to define, for each articulated model, a series of its poses to design each scene. More precisely, the animators manually specify the pose of the character at finite set of frames, and the remaining poses are automatically computed via interpolation [IMH05]. Several researches have focused their attention on trying to automatize this job . Each one of these poses, can be performed through the direct manipulation of the model geometry, or through simplified structures, called handles. The former process is time

consuming and requires a lot of manual work, the latter is the system mainly used in animation, called handle-based animation.
The handle-based animation pipeline is based on 3 different steps:

1. define a control structure (handles)
2. specify a set of weights for each handle and vertex of the character
3. manipulate the control structure to affect the character geometry, updating the position of each vertex.

In the following paragraphs we will go through the details of each step.

In the first step, the animator defines a control structure, which can be designed inside, around or on the model to deform. This control structure is composed by geometric primitives, called handles, which are manipulated by the user. The main types of control structures are points, skeletons and cages; their handles are respectively points, bones and vertices. They have increasing geometry complexity. Each one has its strength and limitations, thus the animator choose the most appropriate to achieve a particular kind of deformation. The control structure design is still a challenging problem: in section 3.1 we will go through the methods to design such structures.

Once this control structure is designed, a relation between the character and this structure has to be defined. The deformation technique establishes the way the character skin moves, according to the handle movements. The process of defining how the geometry of the articulated model is deformed in function of the deformation primitives is called skinning. The character's geometry may be arbitrary complex, but as long as the geometry of the handle is simple, the resulting deformation of the character is fast.

In the second step, for each handle of the control structure and each vertex of the character geometry a weight is defined. The weights can be automatically computed or manually painted by an artist. In chapter 5 we will see a novel research oriented tool to automatically compute and visualize weights to perform cage-based deformations.

The third step is the deformation: once the weights have been defined, the artist manipulate the geometry of the control structure, moving the position of the handles. The position of each vertex is updated according to the transformation applied to the handles, thus obtaining the desidered pose wanted by the animator.

Each vertex v of a shape's surface (or skin), is updated as a new position $v\prime$ as a weighted combination of affine transformations:

$$v\prime = \sum_j \omega_j(v) T_j v$$

where Tj is the transformations which is specified by the animator in the third step.

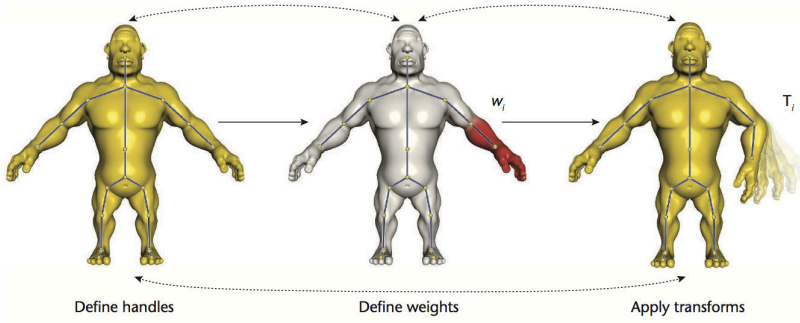
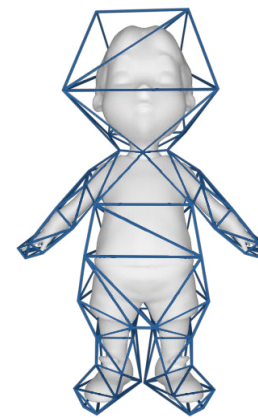


Figure 2.1: From the left to the right, the steps of the handle-based animation pipeline are shown. Image courtesy of [Jac15]

The research in this field has focused on finding which parts of the pipeline are the most time consuming, and based on this consideration, define methods to automatize these steps.

2.1 Cages

A cage is a type of control structure which is designed to be outside the model.
More precisely, a cage is a close polyhedron containing the character shape, which means the character is bounded by the cage. It can be considered as a low resolution version of the character. In [NS13] a detailed survey on cages is presented.



The deformation technique which involves this control structure is known as cage-based deformations. In the cage-based deformation setup, the cage vertices (handles) are used to create the deformation and propagate it to the character skin by using the barycentric coordinates as weights. We will see details of this technique in section 2.1.1.

Cage-based deformations are directly descendant of Free Form Deformations (FFD) where the manipulation of the

vertices of a control lattice provide an intuitive and smooth deformation over the character skin. In spite of its benefits, this structure is not flexible enough to realize more accurate deformations. On the contrary, cages are characterized by topological flexibility, whose means adaptability in terms of the representation of the shape embedded inside the cage. To better explain the concept of flexibility, we can imagine to deform the fingers, the legs or the tail of an articulated model. Roughly speaking if there is a part of cage geometry wrapping these portions of the model, the animator can deform this part, which would have been impossible with lattices.

Cages are in general more complicated than the other control structures (skeletons and points). Not only they have a more complicated structure, they also have some requirements to be strictly fulfill. A detailed discussion on these properties can be found in [SVJ15, LD17, NS13, JDKL14a]. These requirements are not just advised, but they are strictly mandatory. Some of them are necessary to guarantee the resulting deformation is properly working. The properties a cage should satisfy are:

- **be as coarse as possible**, meaning that the number of the cage vertices is the lowest possible;
- **be an envelop for the model without intersecting it**. This means there are not interpenetrations between the cage and the character. (strict constraint)
- **be shape-aware and deformations-aware**, which means that the cage should adhere to the shape and have the handles in correspondence of the part of the model the animator would bend
- **be homotopic to the shape**, which means they should reflect the shape topology
- **tight bound to the geometry of the model**, which means that the handles (cage vertices) should be as close as possible to the surface of the model.
- **be symmetric aware**, which means the cage should reflect the symmetries of the bounded model.

Satisfy all criterion together make the cage design quite complicated. Moreover, some of the above mentioned requirements are in conflict each other, thus is impossible to guarantee they are all perfectly satisfied (for example a coarser cage will not tightly adhere to the shape). Additionally to this, some properties are not well-defined, for example where the handles should be placed depends on the user will and it is not always well-known.

2.1.1 Cage-based deformations

Cage-based deformation techniques can be considered as an evolution of the lattice-based freeform deformation, where the regular control lattice, is substitute by a polyhedral mesh, the cage, which better approximate the character topology. Between the cage and the character there is a well defined relationship, which allows to control the volume inside the cage. When the deformation is applied to the cage, it is also propagated in its interior, thus affecting the volume, and therefore any object it contains (the character). This procedure allows to use control handles (cage verteces) to deform whatever complex model inside the cage. More precisely, in cage-based deformations, the position of the model vertices are expressed as affine sum of the cage vertices. Let us denote as $\mathcal{M}$ the model to be deformed, and $\mathcal{C}$ the cage. $\mathcal{M}$ is a polyhedral mesh and $\mathcal{C}$ is a coarse triangle mesh enveloping $\mathcal{M}$. The deformation formula is the following:

$$p_i = \sum_l \omega_l(p_i) c_l$$

where p_i is a point belonging to the $\mathcal{M}$, c_l is a point belonging to $\mathcal{C}$, and $\omega_l(p_i)$ is the weight function applied to the model vertex p_i and the cage vertex c_l.

The weights are calculated once in pre-processing, thus the model deformation can be performed in realtime. The weight values differ for the barycentric coordinates definition used to calculate them. As we will see in section 3.2 each barycentric coordinates definition has different properties. To guarantee the deformation will be natural and intuitive, some of these properties have to be satisfied:

Ensure continuity all over the domain The deformation domain is the space region influenced by the cage. Deformation needs to be well defined inside the mesh volume, for this reason is required that the cage totally envelope the character to be deformed. Even, the Barycentric Coordinates must be well defined (ensuring *continuity* all over the domain) and they should be *smooth*, which means continuity at first derivatives.

Affine invariance A curve is *affine invariant* if the coordinate system can change, without affecting the relative geometry of the curve. This mean that the geometry of the curve remains consistent when the curve is rotated, scaled, or translated. In mathematical term $\varphi_l(p_i) = 1$ that implies $\sum \varphi_l(p_i) = 1$ for all points p_i of domain.

Local deformation the vertices influence is restricted to their neighborhood. In this way, a control point (cage vertex) must interrupt the

influence of other control points over the part of the cage volume that should be over its influence.

Conformality is an important property that applies local transformation preserving shape and surface details, without lose shape semantic.

Positiveness The coordinates can assume positive values in the entire domain, or they can be negative. If this is achieved, deformation will be more intuitive.

Low computation time One of the main goals in animation is producing believable deformation but also being real-time interactive. In cage-based deformations setup the deformation can be performed in real-time: the more expensive part is the coordinates computation, but since it is performed once in preprocessing, the deformations are performed in realtime.

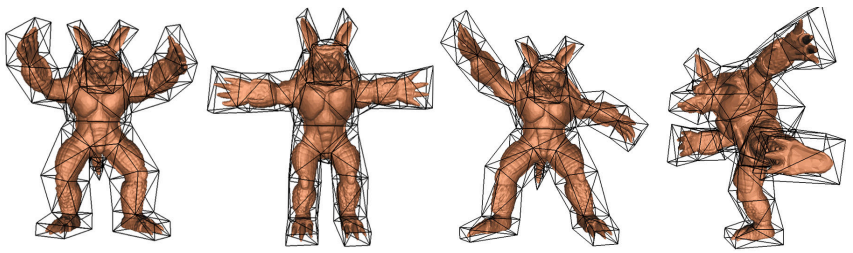


Figure 2.2: Example of cage-based deformations of the armadillo model. Image courtesy of [JSW05a].

Concerning these properties, the coordinates definitions differ in deformation domain, coordinates values (positive or not), and the capability to preserve the shape details. Among coordinates the ones which produce the most intuitive and believable deformation are the Green Coordinates. They overcame the major limitations of the other definitions: they are characterized by a local domain, their coordinates values are non-negative and they can preserve the model shape and its surface details. In fact, the Green Coordinates formula is slightly different from the general one. They add a weight function defined over cage faces which allow to preserve the surface details. The Green's formula is the following:

$$p_i = \sum_l \omega_l(p_i)c_l + \phi_k(p_i)s_k t_k$$

where $\phi_k(p_i)$ is the weight computed for the cage faces, s_k is the scaling factor representing the stretch of the face t_k during the deformation, and t_k is a face of the cage.

2.2 Skeletons

Skeletons are the other control structure which is broadly used in the animation pipeline. The concept of skeleton is relatively simple and intuitive: in nature, the invertebrates, like humans and animals, have a inner structure called skeleton, which drives and controls their motion. Analogously, in computer animation, the skeleton is a inner structure composed by bones and joints, through which the animator control the character movements, and properly defines a set of its poses. Traditionally, animators construct such skeleton as a 3D model defined as hierarchy of line-segments, which is a tree where nodes are called joints connected by one, two or more segments, called bones. Generally, skeletons are constructed as parent-child relationship, where each joint is defined as relative transformation from the father.

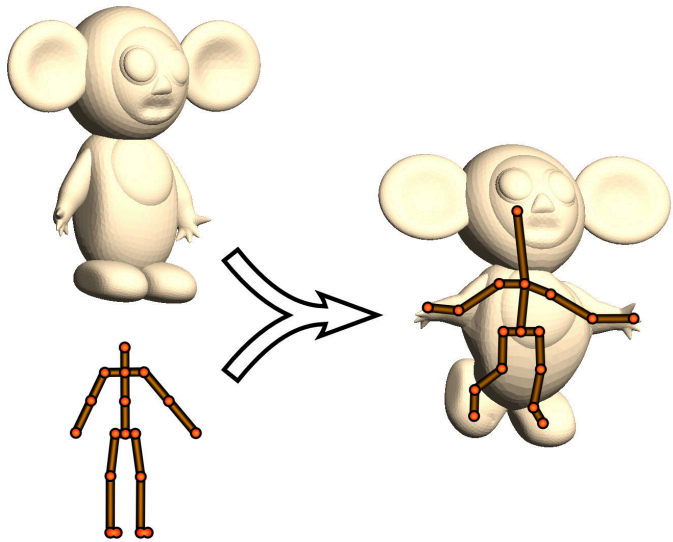


Figure 2.3: Example of animation skeleton. Image courtesy of [BP07a]

In the skeleton-based skinning setup, the bones and joints are the handles which are used to deform the character skin. The deformation is computed making use of the so called skinning weights, which can be

automatically generated or manually defined by the animator. Among the techniques that make use of animation skeletons in order to perform skin deformation, the most widespread ones are the Linear Blend Skinning (LBS) and the Dual Quaternion Skinning (DQS) [KCvO07]. There exists other techniques not only based on geometric approach like the ones aforementioned. Detailed surveys of skinning methods can be found in [RF17] and [JDKL14b]. Linear blend skinning is the standard De facto in the entertainment industry. In applications like video games, there should be a trade-off between visual realism and speed of user interactions, and since they require realtime interactivity, they strongly favour for the speed. In the LBS, the skeleton handles are represented as spacial deformation matrix T_j and the position of the character vertices is updated using this formula:

$$v_i\prime = \sum_j \omega_{i,j} T_j v_i$$

where $v_i\prime$ represents the deformed position of the i-vertex of the character, T_j is the transformation of the j-th skeleton handle, $\omega_{i,j}$ is the weight, namely the amount of influence, of the j-th skeleton handle over the i-th vertex of the character.

In DQS, instead of using a rigid transformation matrix to perform deformations, it makes use of the dual-quaternions theory. Both methods, respectively, present two well known artifacts: "candy-wrapper" and "joint-bulging". Researches in this field have focused on modifying the skinning to adjusting these artifacts [MZS*11]; proposing novel skinning algorithms to overcame the well-known issues of this animation stage [KCvO08], [VBG*13]; automatize traditional animation pipeline, but at the same time, keeping realism and realtime performances [JBK*12].

2.2.1 Curve Skeletons

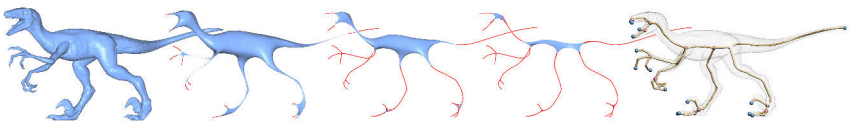


Figure 2.4: Example of curve skeleton extracted through mesh contraction of the input shape. Image courtesy of [ATC*08]

Curve Skeletons are a set of mono-dimensional curves meeting together at branch points, which are meant as a compact, abstract representation of the shape. They are very good shape descriptor because they essentially

capture the topology of the underlying shape. These properties make the skeleton beneficial to many applicative fields, including animation.

The concept of curve skeleton directly derives from the concept of Medial Axis Transform, which has been extended in 3D as the locus of center of the maximal spheres inscribed in the shape $\mathcal{M}$. Although a rigorous definition of such skeleton is still missing, it can be intuitively defined as the medial path of the medial axis of a 3D shape.

In the state of the art there are several different approaches to automatically extract curve skeletons from the shape [ATC*08, TDS*16, LS13, LGS12, TAOZ12]. Alternatively, it can also be manually designed through [BMU*16].

In Chapter 4, we exploit curve skeletons to guide the construction of cages for animation.

Chapter 3

Related works

Cage-based deformation is is a well-known animation technique, which is one of the most active research subjects in both geometry modelling and computer animation. It involves two main sub-problems: generating a well-shaped cage around the character to be animated, and defining a set of smooth barycentric coordinates that govern the deformation space [NS13]. Since this theses will mainly focus on cages and their paradigm of animation, we will see all works are related to cage generation methods and cage-based deformation techniques. In the following sections we will have an overview of the methods have been developed to simplify the design of the cages for animation and then the literature of barycentric coordinates definition.

3.1 Cage generation methods

Traditionally, a coarse cage is constructed either manually, or by subdividing a bounding box of the model [SP86a]. In production, such a process is a burden to the professional artists as it requires a significant amount of tedious manual labor, usually taking several hours. Furthermore, naive bounding box subdivision cannot capture the details of a complex model such as fingers, and garments. Many techniques have been proposed to generate cages, which can be classified into automatic and interactive cage generation techniques.

3.1.1 Automatic cage generation.

Cages can be considered as a simplified version of the original model [SGG*00]. For this reason, many of the fully automatic techniques, which have been proposed, focused on building a cage as a decimation of the

input model.

Based on mesh simplification

To this class of methods belong [SVJ15, DLM11, BCWG09]. In [BCWG09], Ben-Chen et al. employed simplification envelopes [CVM*96] to develop a heuristic method for automatic cage generation, which works as follow: (1) from the input shape it generates a set of points with normal directions which represent the shape. (2) then, it creates an envelope $\mathcal{E}$ of the input shape, by applying a reconstruction algorithm such as [BKBH09] to the points and normals computed in 1; (3) the envelope $\mathcal{E}$ is simplified, within a tolerance ϵ; the more the offsetting is iterated, the most is coarse the cage. They iteratively repeat these 3 steps (envelope-simplify-offsetting the simplified envelope) to get the cage. Their goal is to use such cage for deformation transfer. So they do not take into account some of the cage desired properties.

Edge collapse techniques have also been used to generate cages automatically. Deng et al. [DLM11] built a coarse cage by simplifying iteratively the input model with quadratic error metrics, then two scalar functions, about curvature and face normals, are used to maintain similarities with the input model; finally self-intersetions are removed from the coarse cage using Delaunay partitions. During the simplification, their method keeps checking whether the current coarse cage meets user's constraints (like the vertex amount, or the tolerable error, etc.), but the requirements specified by the user may not always be satisfied. They generate coarse cages enveloping the input model and without self-intersections. Unfortunately, the effectiveness of this approach significantly decreases when handling complex mesh models, therefore, their cages cannot reflect all joints of the input models exactly.

In [SVJ15] a set of nesting meshes are build, where each layer is coarser then the previous one and it fully encloses without intersection that layer. Their method rely on a sequence of decimation of the input model, a flow to shrink and inflate the meshes and a conctact-aware optimization steps. Their method produce bounding meshes, with different scales, but since they are not designed to be part of the animation pipeline, the handle position in their cages do not take into account any semantic information.

Based on mesh voxelization

Another way to automatically generate cages is based on mesh voxelization. Xian et al. [XLG09, XLX15] proposed a voxelization-cage from a dense mesh. In the first approach, they compute the bounding box of the dense mesh, then they voxelize it, identifying the feature voxels, and calculating

a tri-value distance field; they extract and triangulate the outer faces of the feature voxels; and, finally, they perform a smoothing of the cage, based on mean curvature flow. Their method creates effective cages in most cases, their bounding cages can keep the topological structure and major geometric features of the original mesh model. However, the topology of the resulting cage relies on voxel resolution, and the resulting cages are not compliant with the articulation system of the character.

An improved version of these works, was introduced in [XLG12], in which the cage was built for an input mesh by generating an oriented bounding box (OBB) for each mesh part and then registering the OBBs together. However, the OBB is too rigid to provide the desired control in the joint areas, where the deformation can be complex.

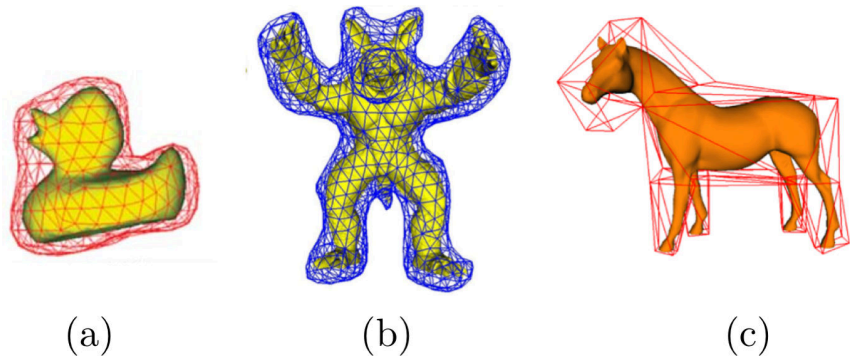


Figure 3.1: Cages obtained through geometric approaches: (a) [DLM11], (b) [XLG09] and (c) [XLG12].

Recently, Calderon and Boubekeur [CB17] presented a bounding shape approximation algorithm, which takes as input an arbitrary surface mesh and generates a bounding proxy which is tightened on the input. Despite their method is automatic, the user can adaptively control the geometry of the proxy scale through a scalar field and with a virtual brush, refining some areas of the bounding shape. Although the user can refine the cage using a brush, the main drawback of their method is that the cage miss the semantically important parts of the 3D model, which can be indispensable for a certain animation.

These methods are purely geometric, and tend to produce sub-optimal cages not always suitable for animation. In fact, control points may be badly positioned with respect to the bending points of the character, and may prevent to realize certain poses.

Skeleton based

In order to generate cages with more awareness of the skeletal structure, template-based techniques have been proposed [YCSZ13, JZvdP*08]. They reuse skinning templates, where the outside cage is constructed by piecing together the predefined templates. Although these methods can construct cages automatically for character animation, they are usually limited in the number of joints they can handle, and they do not scale on complex shapes with arbitrary topology. Chen and Feng [CF14], substituted template cages by planar cross-sections locally orthogonal to the skeleton. Restricting cage vertices to planar cross-sections may easily lead to self-intersections in cages for complex characters, or human-like characters not posed in the classical T-pose.

Various works addressed the problem of producing a sequence of cage motions to compactly encode an animation sequence [CFB16, CF14, TTB12, CHSB10] or video-based animation [Sav16]. This latter problem is orthogonal to the one we address in this work, with almost no algorithmic overlap.

3.1.2 Interactive cage generation

Interactive methods allow the user to interact with the system and choose the cage topology, thus generate cages that are dedicated to the needs of the artist/animator. The system we propose belongs to this category.

The state of the art of this class of methods is the work proposed by Le and Deng [LD17]. They propose an interactive tool based on user-defined planar cross-sections. This approach requires the user to position and orient planes over the model from which the cage is inferred. Even though it significantly reduces the time required to build the cage, it remains time consuming and it can still be hard for non-expert users to sketch the planes and build the proper cage. Moreover, the handle locations are restricted to be on the planes sketched by the user. This may limit them to properly adapt the cage to the underlying model, for example impeding the separation in certain parts or it can create self-intersecting cage, the user is not able to fix.

Skeleton-induced semantics

The relation between skeletons and the logical parts of an object have been exploited in a number of previous works, involving applications such as shape recognition [KCATCO*10, BMSF06], consistent mesh partitioning [ZYH*15, MPS06], and re-meshing [LAPS17, LMPS16, ULP*15]. Our work has been inspired from these latter works. We have investigated, the

possibility to use skeleton induced semantics in the generation of cages for animation. However, due to the different constraints and goals, none of these approaches are directly applicable to the cage generation context.

3.2 Cage-based deformation.

First researches in this field were done at the beginning of digital film animation. The first method developed by Sederberg and Parry [SP86b] is the Free Form Deformation (FFD), which uses three dimensional control lattices, the primordial cages. This technique allows to smoothly deform the character through the lattice, but it is not flexible enough to realize complicated deformations like legs or arms movements. An extension of this work was proposed in [MJ96], and consists in recursively subdivide of the control lattice to obtain the topological flexibility needed to effectively deform the character.

Since the control lattice can hardly fit an articulated model, even combining sever lattices, a cages and their deformation technique have been developed. Cage-based animation can be seen as an evolution of FFD methods, where the lattice is substituted by a polyhedral mesh, the cage. Cages allow for a better approximation of the digital character at a far lower complexity than lattices. A number of desirable properties for cages are listed in section 2.1, whose ensure the deformation will be optimal.

In order to obtain this deformation, there must exist a well defined relationship between the cage surface and its inside volume. This is done, defining a set of coordinates as weight functions. These coordinates should have some important properties, a detailed description of these qualities can be found in 2.1.1. Barycentric coordinates realize the connection between a character and its cage. They were first introduced by Möbius in 1827, and have been subsequentially generalized to 3D in many ways [Flo03, ZDL*14, HS08]. Mean Value Coordinates [FKR05, JSW05a, FHK06] were the firsts to be introduced. They are generally smooth and well defined, but suffer from two drawbacks: they are not very local, and they may be negative for concave cages [JMD*07], leading to non intuitive deformations. Harmonic coordinates [DM06, JMD*07] were proposed in alternative, and effectively address both issues, being more local and guaranteed positive inside any cage. For completeness, in [LKCOL07] Lipman et. al proposed an evolution the of Mean Value Coordinates method which solves the negativity issue using a GPU visibility test. A step forward in cage based animation was achieved one year later with Green coordinates [LLCO08], which offer better preservation of surface details under deformations, producing more realistic animations. They use not only vertices but also cage normals, and relax the constraint of having the character completely

inside the cage. In recent years, various biharmonic coordinates [JBPS11a, WPG12] and local barycentric coordinates [ZDL*14] were introduced, offering better locality than previous methods. In particular, bounded biharmonic coordinates [JBPS11a] received a lot of attention, because they offer for the first time a unified framework were cages, skeletons and point handles co-exist in the same deformation space.

Part I

A Novel Approach for Cage Generation

Skeletons and control cages are certainly the most widely used techniques to pose a digital character, enabling its animation [JBPS11b]. In complex animations, high-level motions (e.g. posing the limbs of a character) are often enriched with secondary effects driven by the movement itself, such as the swish of a cloak or the jiggle of a body part. While skeleton-based animation is perfectly suited to control primary motions – such as posing the limbs of a character – cage-based animation provides a more flexible tool, which is, overall, better suited for secondary effects driven by the movement itself. In cage-based animation, the artist manipulates a coarse control mesh containing the character: the space enclosed by the cage, hence the character itself, are deformed by moving the cage vertices.

This part of the theses will analyze, discuss and propose a solution for one of the two big sub-problems that cage-based deformation technique involves: the cage design. Although, apparently this can appear as a trivial problem, it is not so easy to address. Cages are intrinsically more complex to create than skeletons. Well established properties have to be fulfilled. A detailed discussion on these attributes can be found in the section 2.1. Summarizing, the properties which influence and in a certain way limit the construction of the cage are: (i) the cage must tightly envelop the original model without either intersecting it, or self-intersecting; (ii) the cage must be animation-aware, i.e. its control nodes should be close to the parts of the model one would like to deform or bend; (iii) the cage must be coarse enough to be easily manipulated, yet fine enough to capture the necessary details, thus it might need variable resolution; (iv) the cage must endow the symmetries present in the model.

The properties required to properly build a cage for animation purpose, make the design of that cage particularly hard. In the view of these requirements, building a polyhedral mesh which satisfies all constraints is not obvious.

Most often, cages are hand-made and their creation may require several hours of extensive work. Moreover, since it involves the use of professional modelling tools, like Maya, Blender, 3DStudio Max, it consequently require skilled artists as cage designer. It is thus important to provide automatic or semi-automatic methods to generate an initial coarse cage satisfying the construction properties, letting the animators' efforts concentrate only on the refinement stage. As seen in section 3.1 several works attempt to automatically build such structure. The automatic approaches are purely geometric: they rely on the character pose, and generate a cage to fit it, thus ignoring any possible semantics associated to its movement. For instance, when a human-like character is in the canonical T-pose, limbs are straight and purely geometric approaches may misplace or completely miss

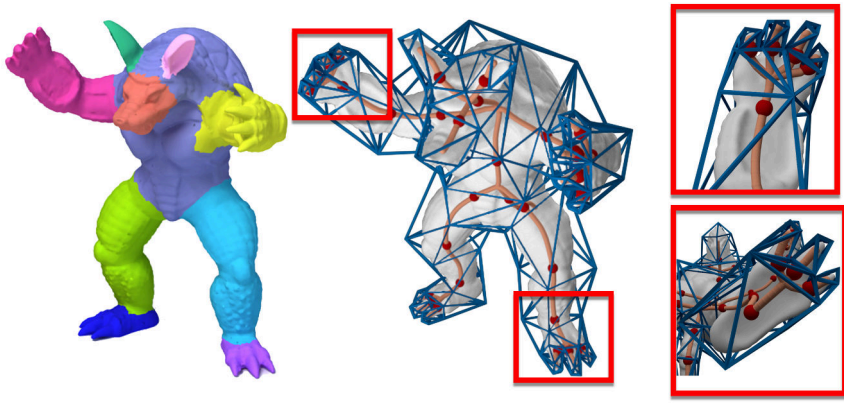


Figure 3.2: Left: a purely geometric segmentation obtained with a state-of-the-art method [ZYH*15] does not capture bending junctions between disjoint semantic parts (neck, elbows, knees, wrists, joints of fingers and toes). Right: in our interactive system the user manually selects bending points, inducing an animation-aware semantic decomposition of the character.

important bending points, such as knees and elbows, not to mention finer details like fingers, jaws, or eyelids (Figure 3.2). Likewise, for invertebrates such as the Octopus in Figure 4.21, limbs must be allowed fully flexible motion: in this case, the complexity of the cage depends on the poses the character will assume during the animation, and this is something that cannot be inferred by geometric analysis of a static pose.

Semi-automatic approaches let the user specify some constraints, enabling artistic touch and customization of the process. The effectiveness of a method in this class is thus assessed by the trade-off between the amount of necessary user effort and the level of control allowed during cage construction. Ideally, one should aim at maximum control with minimal effort. Notice that having the user in the loop is not a way to reduce the algorithmic complexity, but rather an additional value that cannot be achieved otherwise.

In this perspective, we propose a skeleton-driven method that lets the user address shape-awareness by controlling the topology of the coarse cage in an extremely simple way, while automatically fulfilling the requirements outlined above. The skeleton can be either an animation skeleton (i.e., a hierarchy of rigid bodies representing joints and bones), or a geometric

one, which can be either automatically extracted with a state of the art method [TDS*16, LS13, LGS12, TAOZ12] or manually crafted [BMU*16]. The user is just required to select a few *bending nodes* on the skeleton of the character, while a corresponding segmentation of the shape and a coarse cage coherent with it are automatically generated.

We address all the requirements outlined before. In order to fulfill requirement (i) our method relies on a harmonic field defined inside the shape volume. We use it to propagate cutting surfaces from the interior of the character to its skin, enabling a robust tracing of non-planar cross sections that adapt to the local shape of the character (Figure 4.4). This approach overcomes the burden and the limitations of previous user-assisted methods, based on cutting planes [LD17], while providing a robust placement of vertices of a tight cage, which is finally inflated just enough to eliminate intersections. The structure of the input skeleton, as well as the distribution of the bending nodes, totally determine the coarse topological structure of the final cage, thus fulfilling requirements (ii) and (iii). Finally, we address requirement (iv) by exploiting the work of Panozzo et al. [PLPZ12] to evaluate model symmetry. We rely on such information to find a consensus between symmetric cuts and build a symmetric cage.

The pipeline of our method is summarized in Figure 4.1: starting at the initial mesh with an underlying skeleton (a), the user selects bending nodes (b); cutting surfaces across bending nodes are computed, and cross polygons are extracted from them, which conform to the desired cross section of the cage, as well as to the symmetry of the object (c); an initial tight cage is obtained by connecting vertices of the cross polygons (d); that are used to define the topology of the cage (c), which is inflated in order to approximate the character shape (d), and finally the cage is inflated and adjusted to avoid intersections (e). Our main contributions are in steps (c) and (d) of the pipeline, which both rely on the harmonic field mentioned above; we also contribute for a pre-inflation in step (e), which is necessary to enable the *Nested cages* by Sacht et al. [SVJ15] to perform final inflation. Besides, we provide a complete working tool that requires a level of interaction as simple as the one in step (b). In Section 4.2, we also show that a fully automatic initial placement of bending nodes is possible, thus providing a fully automatic initial cage, which can be edited and completed by the user at will.

Our method enables the generation of high quality coarse cages with controlled topology, also for characters which are not given in the standard T-pose (see examples in Figure 4.21). Our bounding cages preserve all model features captured by the underlying skeleton, as well as all user selected cross sections, thus providing a desired/effective shape abstraction for animation purposes.

The following chapter (Chap. 4) will discuss and present the proposed method, with a detailed overview of each step of the pipeline. In section 4.1 every step of the method is presented; section 4.2 describes a simple heuristic to automatically detect the set of bending nodes which determine the resolution of the cage; finally section 4.3 presents the results obtained through our method, the comparison with the existing related methods, the times and the evaluation of the achievable deformation. The work presented is currently under submission to a peer-reviewed journal with relevant impact factor for the community.

Chapter 4

Skeleton-based Cage Generation

4.1 The method

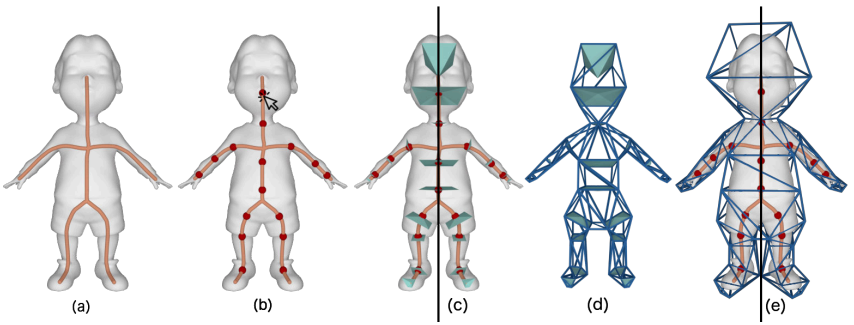


Figure 4.1: Our pipeline starts from a digital character and its curve skeleton (a). The user manually selects nodes where bending occurs (b, red dots). Symmetric cross sections at bending nodes are obtained from a harmonic field in the volume (c). An initial cage is created by connecting the cross sections (d). The cage is finally inflated to accommodate the character without intersections (e).

The pipeline of our method is described in Figure 4.1 and has been summarized in the Introduction. In this section, we will go through the details of each step.

Our input consists of a triangle mesh $\mathcal{M}$ representing the digital

character, together with its line skeleton $\mathcal{S}$. We just require the skeleton to be fully contained in the volume bounded by the mesh. Nodes of $\mathcal{S}$ that have more than two incident branches, or just one incident branch, will be called *branching nodes*, and *leaf nodes*, respectively. Additional nodes selected by the user along the branches of the curve skeleton will be called *bending nodes*.

Roughly speaking, the cage will be made of tubular structures, one for each branch of the skeleton, connected at polyhedral joints surrounding the branching nodes of $\mathcal{S}$. Each tubular structure will have a polygonal cross section, and it will be subdivided transversely at bending nodes. For simplicity, we always adopt quadrilateral sections, as in [LD17], but the method can support generic polygonal sections, possibly different for each branch. Note that, although the boundaries of sections may usually be nearly planar, their corresponding cutting surfaces are always forced to pass through the central node of the skeleton and may result strongly non-planar. Non-planarity will usually be emphasized about leaf nodes and where relatively thin limbs are attached to a larger body (e.g., armpit, groin). See Figures 4.1 and 4.3.

4.1.1 Pre-processing: guiding field

Our pipeline has a pre-processing step in which we compute the field which will drive the creation of the cage. Figure 4.2 explain this pre-processing step.

Given the surface mesh $\mathcal{M}$ and the skeleton $\mathcal{S}$, we define a volumetric field that is propagated radially from the skeleton to the outer surface. Such a field is inspired by the radius component of the cylindrical parameterization described in [LAPS17], and will be used later on to partition our cage.

In order to compute this field, we first generate a tetrahedral mesh whose outer surface is perfectly conforming to the surface of the character and embeds the skeleton, making the skeleton edges and points being part of the tetrahedralization, as chains of internal edges of the volumetric mesh. Then we compute a harmonic function f defined over the volume spanned by the tetrahedral mesh, by resolving the Laplace problem $\triangle f = 0$, subject to Dirichlet boundary conditions

$$\begin{aligned} f(p) &= 0 \quad \forall p \in \mathcal{S} \\ f(p) &= 1 \quad \forall p \in \mathcal{M}. \end{aligned}$$

The integral lines of f give us a robust way to project points from the

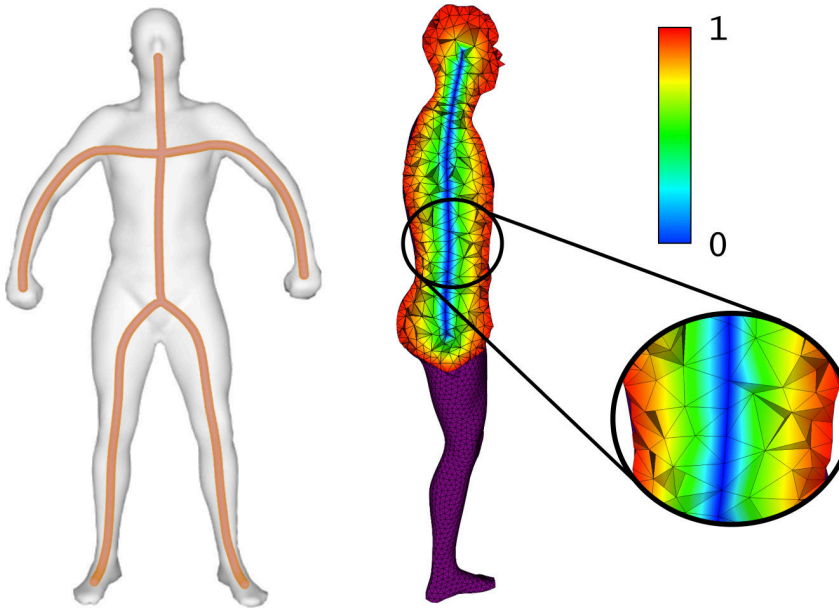


Figure 4.2: We generate a shape-aware harmonic field which evaluates 0 at the skeleton and 1 at the character's skin. The integral curves of such field radially emanate from the skeleton, traversing the interior volume along a short trajectory. We exploit this property to project points from the interior to the surface, generating cross-sections that locally adapt to the character's shape.

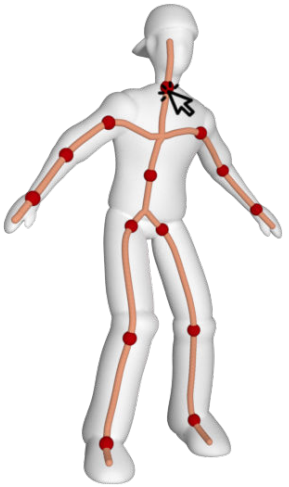
inner volume to the surface. We will exploit this property to generate the control polygons associated to each bending node.

Given any regular (i.e., non branching) point p on the skeleton, the integral lines of f starting at p span a surface that cuts the volume transversely with respect to $\mathcal{S}$. We call this surface the *cutting surface* at p. The cutting surface may bend according to the local shape of $\mathcal{M}$ in the proximity of p, and it will intersect $\mathcal{M}$ at a ring made of points that are "closer" to p than to any other point of $\mathcal{S}$. These cutting surfaces will provide a better alternative to the cutting planes used in [LD17].

This pre-processing step is computed off-line once, before any interaction occurs, and is used without modification in later stages of our method.

4.1.2 User interaction

The recognition of the features of a mesh is still a challenging issue, especially when the semantics underlying the feature definition is related to an intrinsically not formalized context, as can be in the animation environment. We seek a combined geometric and semantic approach in which the topology of the cage reflects the structure of the character that will be animated with it. Embedding semantics into the process allows us to keep the complexity of the cage low (i.e., small vertex count) and at the same time have all the necessary degrees of freedom to animate the model. As highlighted and explained in sections 2.2.1 and 3.1.2 the skeleton is a good shape descriptor, effectively capturing the topology of the shape of the underlying mesh. Similarly to [ULP*15, LMPS16, LAPS17] where the input skeleton has been exploited to get information on the high level structure, namely, how many limbs, and how they connect to each other, our approach takes advantage of this informations to build the cage.



The user refines this structure prescribing additional degrees of freedom wherever necessary. Interaction is intuitive and happens in real-time. The user is just asked to click on skeleton curves where bending may occur (e.g., adding knees, elbows, wrists). This selection induces a partition of the skeleton that will then be translated into the cage. The section 4.1.3 will explain how this is performed in our method.

User selected bending nodes will originate cutting surfaces, which constitute a better alternative to the cutting planes used in [LD17]. In fact, while cutting planes require accurate and tedious placement in 3D, our non-planar cutting surfaces are determined solely from the bending nodes and make our method quite tolerant against inaccurate user selection (Figure 4.3).

It is noteworthy that, if the skeleton provided as input, is the curve skeleton one, the initial partition (supplied by the skeleton topology) is still meaningful; while, if an animation skeleton is provided, this partitioning may or may not reflect the same structure. It's up to the user deciding whether the cage should be compliant with it.

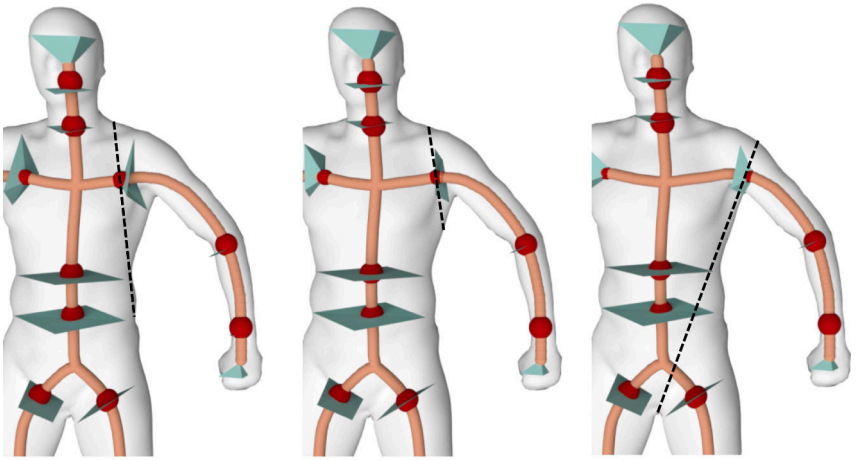


Figure 4.3: Example of cuts induced by different selections of bending nodes. The cyan pyramids approximate the cuts induced by the harmonic field, while the dashed black lines represent the corresponding cuts induced by a plane orthogonal to the skeleton and through the selected node. While our cuts are all valid, the corresponding planes may induce inconsistent cuts.

4.1.3 Cage generation

This section describes the core of our method. We start from the skeleton $\mathcal{S}$ and the bending nodes added by the user, and we generate a coarse cage that fits the input mesh $\mathcal{M}$. Intersections between cage and mesh will be removed in a subsequent step of the pipeline (Section 4.1.5). Let B be the set of bending nodes selected by the user, and L be the set of leaf nodes of $\mathcal{S}$ (i.e., terminal nodes of its branches). We jointly refer to $B \cup L$ as *control nodes*. For each control node p, we create a cross polygon which is roughly orthogonal to $\mathcal{S}$ and has its vertices on $\mathcal{M}$. The cross polygon gives the transversal section of the cage at p. We overcome previous approaches [LD17] by using non planar cross sections, thus enabling a more flexible caging process for complex geometries.

We first generate a sampling of the cutting surface at p by tracing a number of integral curves of the harmonic field f. Notice that f is encoded at the vertices of $\mathcal{M}$ and linearly interpolated inside each tetrahedron, thus its gradient field is piece-wise constant. This fact may result in poor integral lines that do not radially propagate in a uniform way, especially if they are traced from the immediate proximity of $\mathcal{S}$ [MLP18]. To avoid

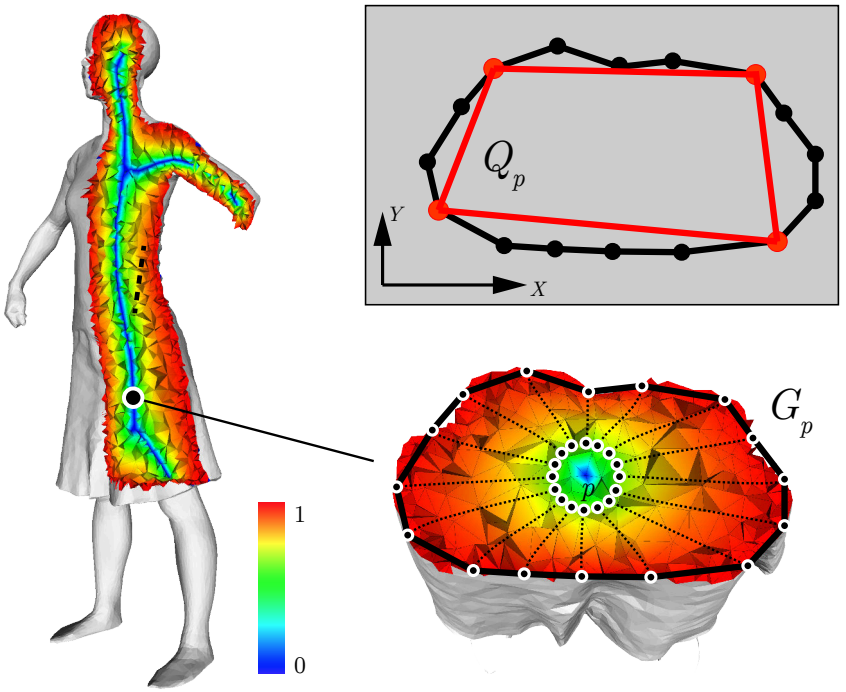


Figure 4.4: Curved cuts are superior to linear cuts [LD17] as they better adapt to the geometry of the character, avoiding intersections and ill-defined cross sections (top left). We exploit the harmonic field f (left) to define the non planar cross sections of our cages. For each control point p we sample a ring locally orthogonal to the skeleton, and project it on the surface along the integral lines of f (bottom right). We project the resulting polygon G_p on the plane defined by its PCA, and find the quadrilateral Q_p that better approximates it (top right).

this issue, we sample seeds on a circle centered at p and locally orthogonal to $\mathcal{S}$, and we trace the integral lines starting at such seeds (see Figure 4.4, bottom right).

The radius of the circle is a fraction of the radius of the maximal ball centered at p and enclosed in $\mathcal{M}$ (we use 20% in all our experiments); the number of sampled points is a non-critical parameter of the method (we sampled 20 points in all our experiments).

The integral lines we trace meet the surface $\mathcal{M}$ at a finite set of points G_p, which provide a discrete approximation of the intersection ring between the cutting surface at p and $\mathcal{M}$ itself. We further approximate this ring

with a polygon having its vertices at some of the points of G_p. We describe here a technique to generate a quadrilateral section $Q_p(q_0, q_1, q_2, q_3)$; it is straightforward to extend it to a generic n-gon for any given n. The figure 4.4 provide the graphical explanation of this procedure. We first compute the PCA of G_p and we take the first two principal directions to define a XY planar frame. We then project G_p to such plane and select the four projected points that define the quad Q_p that better aligns with the planar projection of G_p. Let λ_1, λ_2 be the two positive eigenvalues associated with the X and Y axes given by the PCA: we define q_0, q_1, q_2, q_3 as the four points that maximize the signed sums of per vertex coordinates, weighted by λ_1, λ_2:

$$\begin{aligned}
&\max_{p \in G_p} \; +\lambda_1 p_x + \lambda_2 p_y \\
&\max_{p \in G_p} \; -\lambda_1 p_x + \lambda_2 p_y \\
&\max_{p \in G_p} \; -\lambda_1 p_x - \lambda_2 p_y \\
&\max_{p \in G_p} \; +\lambda_1 p_x - \lambda_2 p_y.
\end{aligned}$$

Once the cross polygons have been generated for all points $B \cup L$ of $\mathcal{S}$, we build the tubular portions of the cage. We complete this tubular portions by connecting quad sections pairwise and computing the convex hull of the two cross polygons bounding it. Notice that the triangles generated between adjacent pairs of quad sections are independent from the others, thus avoiding the accumulation of torsion along the limbs of the character, which is never present in our cages.

We finalize the topology of the cage by welding together the tubular structures we have built so far about the branching nodes of $\mathcal{S}$. We follow a technique similar to [LD17], considering one branching node at a time and projecting its incident quad sections to a sphere centered at the center of mass of their vertices. The convex hull of the so projected point set defines the connectivity of the cage around each branching node (Figure 4.5).

Even if not reported in [LD17], we observe that in some pathological cases the convex hull algorithm may fail to define an edge for each of the sides of the quadrilateral cross-sections, resulting in an invalid cage connectivity. To fix this issue, in the cases it occurs, we add a Steiner vertex at the midpoint of each missed edge, and iterate this operation until all the edges of the involved quad sections are included in the convex hull. Additional vertices can be easily removed in post-processing by iteratively applying the edge collapse operator to the modified quad-sections, removing all the edges that contain at least one Steiner vertex. The need for insertion of Steiner points is very rare and we did not need it in any of the cages we show.

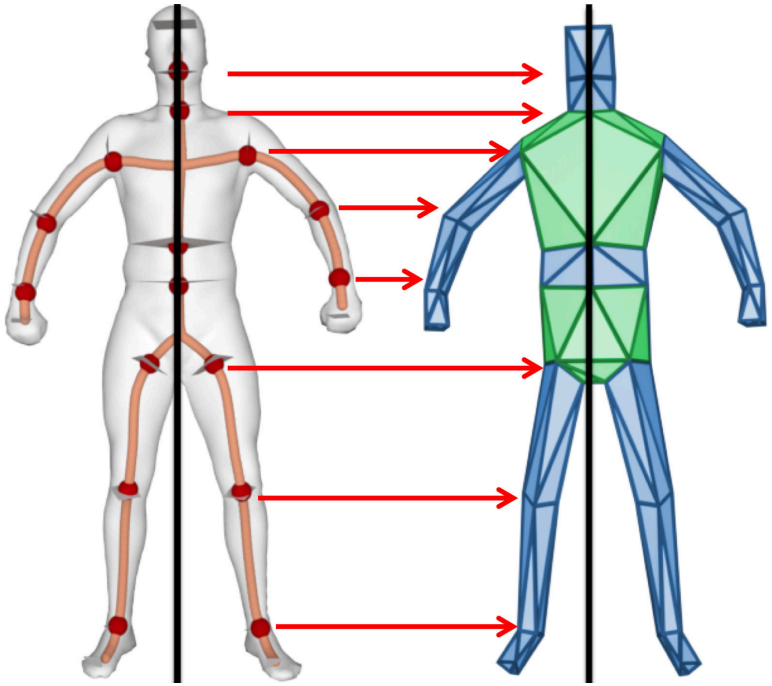


Figure 4.5: Correspondence between the bending nodes (red dots) and the cross polygons (grey polygons) of the cage. Polyhedra join the tubular structures (blue) at polyhedra (green) about branching nodes of the skeleton to form the tight cage (before inflation).

4.1.4 Symmetry

In the digital modeling pipeline, most of the times only half of a character is explicitly modeled, while the second half is obtained by reflecting the so generated mesh along a symmetry plane. In order to generate consistent animations for both sides of a symmetric character, it is therefore important to replicate such symmetries at cage level. Moreover, as explained in section 2.1, having a symmetric cage is a desidered properties because it improves the deformation quality. In order to automatically detect extrinsic symmetry of the input model, we exploit the method described in [PLPZ12], and we employ it, to get the symmetry plane and adjust the position of cage sections and handles according to it.

Given the symmetry plane Π, extracted with [PLPZ12], we adjust the cross-sections, previously computed, to be perfectly symmetric respect that plane. Our strategy to symmetrize a cage works as follows. We consider

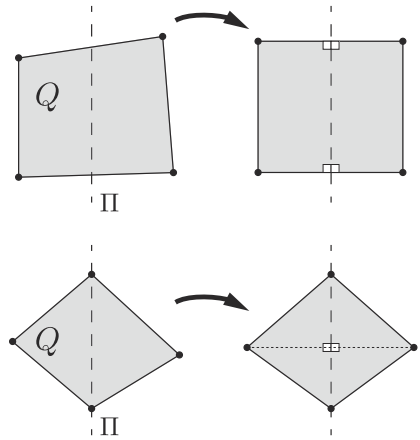


Figure 4.6: Symmetrizing a quad across the symmetry plane Π. Top: two vertices of the quad lie to the left of Π, and the other two lie to the right of Π – we impose symmetry of edges connecting vertices on the same side of Π. Bottom: two vertices lie on Π and the other two lie on opposite sides of Π – we impose symmetry between the vertices not lying on Π.

each quad section Q_i and reflect it through Π, generating a target section Q_i^{Π}. We then find the quad section Q_j that is closest to Q_i^{Π}. Distance between quad sections is computed point-wise; since the vertices of each quad section are ordered, we test the four possible vertex pairings and consider the one that minimizes the sum of pair-wise distances. If the two sections are not equivalent (i.e. if $i \neq j$) we impose a symmetry between them, by computing the average between Q_j and Q_i^{Π}, and we use this average section and its reflected counterpart to substitute Q_j and Q_i, respectively. If $i = j$, the quad section is traversed by the symmetry plane. There are two possible cases, summarized in Figure 4.6: (i) Q_i has two out of four vertices on Π, in which case we symmetrize the pair of vertices which are not on Π; (ii) none of the vertices of Q_i is on Π, in which case we symmetrize the two edges of the quad section, which do not cross Π. Figure 4.7 shows a visual example of caging with and without symmetry enforcement.

This approach can be easily extended to deal with any symmetric character given in general pose (which does not exhibit extrinsic symmetry), by using the method for intrinsic symmetry detection described in [PLPZ12].

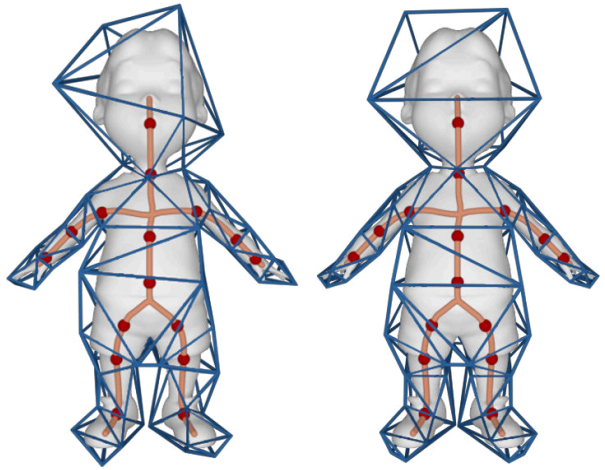
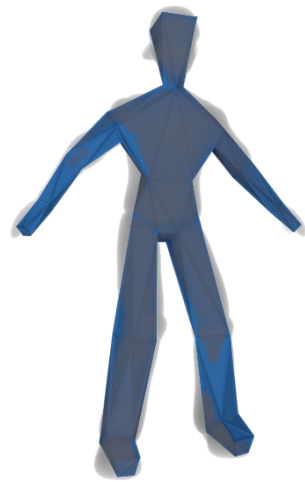


Figure 4.7: A visual comparison between our cage without (left) and with (right) symmetry enforcement on the Boy dataset.

4.1.5 Cage inflation

The tight cage we have built so far has its vertices on the skin $\mathcal{M}$, as shown in the image in the inset on the right. Although the vertices lie on the skin, its faces are still deep inside of $\mathcal{M}$. Obviously, such cage can not be used, as it is, for animation purpose since it does not envelope the input model and there are intersections between the cage and the model. In the final step of our method, we pull the vertices of the cage further out, ensuring that none of its faces intersects $\mathcal{M}$. We inflate the cage using the *Nested Cages* expansion mechanism [SVJ15].



This is all but straightforward, though. Such a method is efficient and guarantees the absence of collisions. However, as already acknowledged by the authors, it may fail if the tight cage and the skin are not sufficiently close to each other (see Figure 4.8). In our experiments, for the majority of the models, this failure case was occurring, thus we develop an additional step, which pre-inflate the tight cage before the final inflation with *Nested cages*. This pre-inflation step is described in the following paragraph. We pull

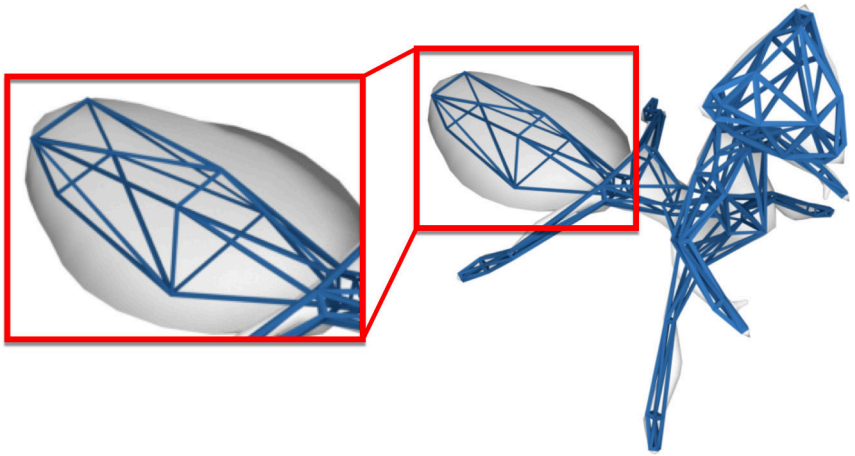


Figure 4.8: Nested Cages [SVJ15] will fail without our pre-inflation mechanism. Even though the vertices of the tight cage are on the surface, its faces are too far from the skin to support the mesh contraction, which is at the basis of the Nested Cages algorithm. Antcat dataset.

the cage faces, which may lay too much inside the model, towards the outer surface of $\mathcal{M}$ with a simple technique summarized in Figure 4.9. We sample the faces of the cage and we project each sample onto $\mathcal{M}$ by following the integral lines of the harmonic field f (Section 4.1.1). Samples that are already outside the skin are ignored. For each face f_i of the cage, we take the sample s that lies farthest from its projection s_p, and define a face displacement vector $\vec{f}_i = s_p - s$. Then, for each vertex v_j of the cage, we build a vertex displacement vector $\vec{v}_j$ as follows:

- the direction of $\vec{v}_j$ is the (normalized) average of the per face displacements vectors incident at v_j (Figure 4.9, left);
- the magnitude of $\vec{v}_j$ is the length of the largest projection of the same face displacement vectors on this direction (Figure 4.9, right).

Since every vertex of the cage is moving out from the surface of the model, the cage could self-intersect during pre-inflation . This may happen in the proximity of tiny features or small spacing between different parts of the model. (see, e.g., the example in Figure 4.10). This issue is fixed by adding a check of collision during the inflation (Figure 4.11). We check if a face of the cage intersects any other face after pulling them outwards; then, for each pair of intersecting faces, we move all their vertices a small step towards their previous positions, and we repeat until no intersection occurs:

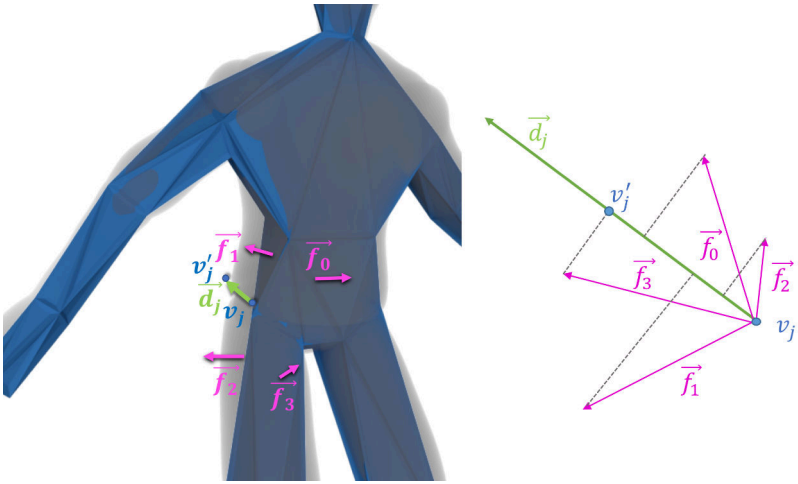


Figure 4.9: Pre-inflation of the cage. Left: each face is assigned a displacement vector that brings it closer to the skin. Right: such displacements are averaged at cage vertices to pull them outwards.

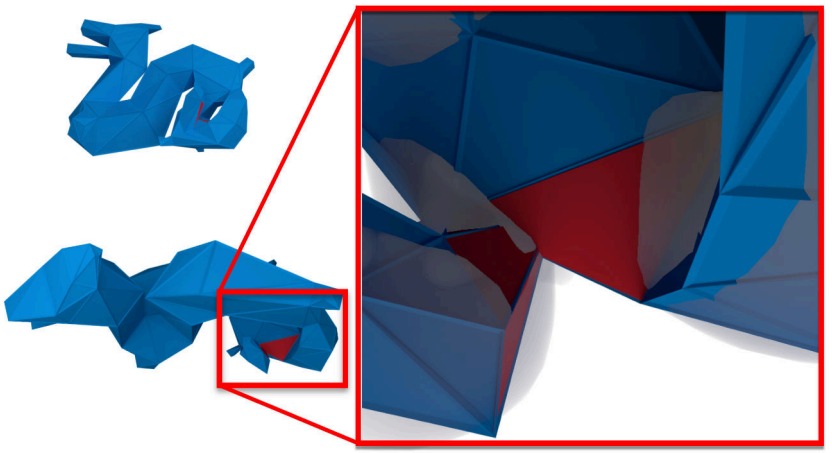


Figure 4.10: Faces overlap after the projection step. The faces belonging to the tail of the dragon overlap with the faces of the body.

the offset at each step is set as a small fraction of the displacement vector $\vec{v}_j$ defined above. In all our experiments, this procedure was sufficient to provide a clean input to *Nested cages*. In case final inflation still fails, in

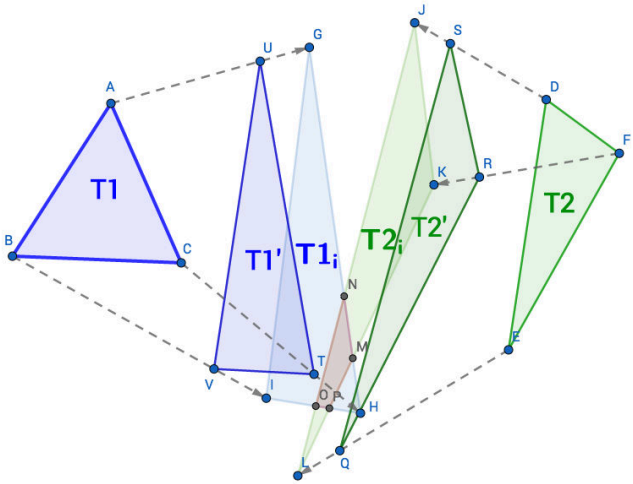


Figure 4.11: Example of faces intersection: the two faces T1(A,B,C) and T2(F,D,E) are projected. After projection, they became respectively $T1_i$(G,I,H) and $T2_i$(J,L,K). Their projections partially overlap (the intersection area is highlighted in red in the figure). Once the intersection is detected, we locally solve the intersection moving backward the vertices of the two faces (G,I,H and J,L,K) until there do not intersect any longer. The final faces are $T1'$ and $T2'$.

most of the cases the number of bending nodes selected by the user was not sufficient to provide a fine enough articulation of the cage. This is easily fixed with one more cycle of interaction. Note that although a symmetric energy is presented in [SVJ15], the available implementation of Nested Cages does not support symmetry. We therefore interleave cage inflation and symmetrization (Section 4.1.4) of the cage to obtain the desired result.

4.2 Automatic placement of bending nodes

Although the method we are proposing is designed to be user-assisted, we may also suggest an initial guess of bending nodes. The rationale consists in placing bending nodes in correspondence of abrupt changes in the local thickness of the shape (e.g where arms and torso meet). This is a classical criterion for mesh segmentation [Sha08], and it is sometimes sufficient to obtain a fully automatic construction of the cage. In most cases, though, it just provides some hint to the user, as bending points may

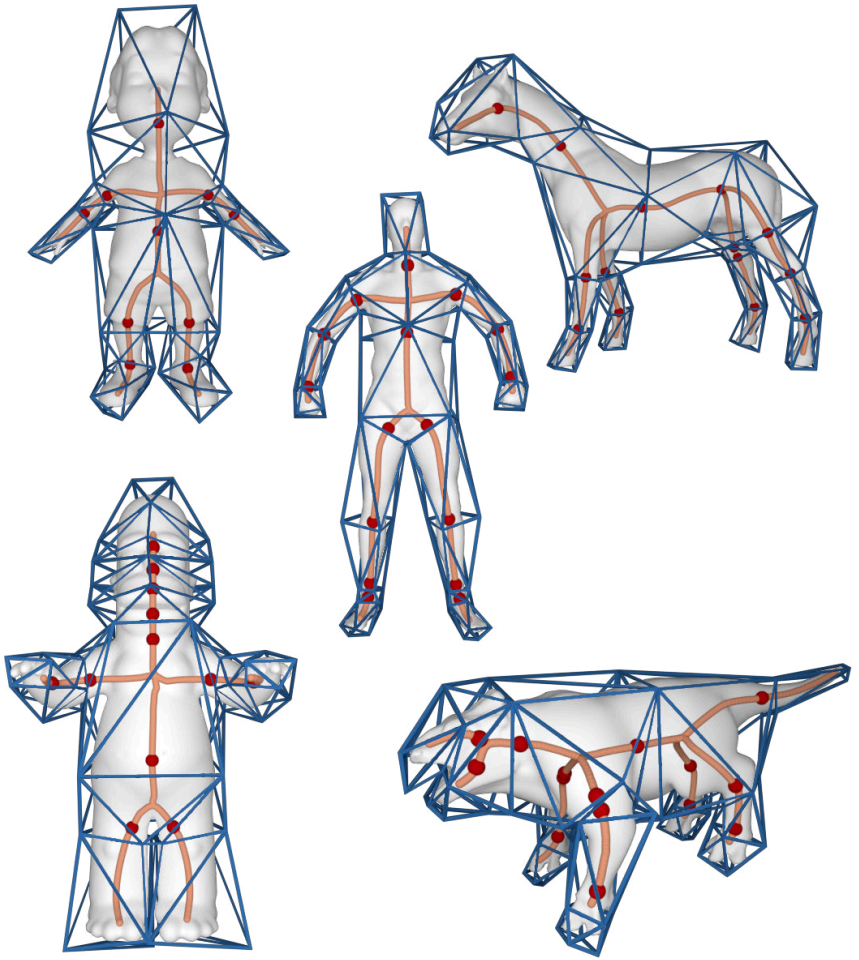


Figure 4.12: A collection of cages obtained by running our pipeline in fully automatic mode. Boy, Scape, Horse, Homer and Animal datasets. Some bending points may be either missing (e.g., ankles and knees for Homer), or redundant (ankles for the humanoid, elbows for the animal, head for Homer).

not match geometrically relevant features of the mesh. We propose here a simple heuristic which exploits the curve-skeleton. Alternative strategies to segment a shape based on its local thickness exist in literature and may accommodate better results. We define a function over the curve skeleton

$\mathcal{S}$

$$r(p) = Rad(p) \quad \forall p \in \mathcal{S},$$

where $Rad(p)$ is the radius of the maximal sphere centered at p, which is fully inscribed inside the model. Function r is provided as a byproduct of skeleton computation by several methods [LS13, LGS12]. In case it is not available from the input, it can be easily computed by finding the point on the character that is closest to p. We place bending nodes at points of the skeleton where there is a large variation of function r. We compute the first derivative r' ad we find its critical points. To alleviate sensitivity to the noise, we smooth function r' before using it. From a practical point of view, this method allows us to further speed up the cage construction. Starting at the initial guess, interaction is limited to adding/removing/displacing bending nodes. Despite its simplicity, this strategy already allowed us to produce some quality cages without any user interaction. See Figure 4.12.

4.3 Results and discussion

We implemented our cage generation tool as a single threaded C++ application and run our experiments on a MacBook Pro equipped with a 2,7 GHz Intel Core i5 and 8GB of RAM. In the context of the pipeline, we have used a few well established techniques implemented in a variety of publicly available libraries, specifically: Tetgen [Si15] for volumetric mesh generation; CGAL [FP09] to compute convex hulls; Eigen [GJ*10] to solve linear systems; and CinoLib [Liv17] for mesh and scalar field processing.

We applied our caging tool to a variety of 3D models listed in Table 4.1, producing cages that fulfill all the requirements listed in [LD17, NS13] and reflect the extrinsic symmetries of models. Skeletons were computed using the implementation of [TAOZ12] available in CGAL, as well as other automatic [LS13, LGS12] and interactive [BMU*16] techniques. Galleries of cages obtained with our method are shown in Figures 4.20 and 4.21, for models with and without extrinsic bilateral symmetry, respectively. Notice that we effectively deal with rather complex objects and poses, like the Joker, the Octopus and the Dragon. In addition to that, we observed that despite its natural ability to cage tubular shapes, also digital characters with features that are not well described by a skeleton, such as the skirt in Figure 4.18, are robustly handled.

Tubular shapes are ubiquitous in computer animation. Our skeleton-driven approach makes the method naturally suitable to process such shapes, even when semantic features at different scales are present (Figure 3.2 and 4.13).

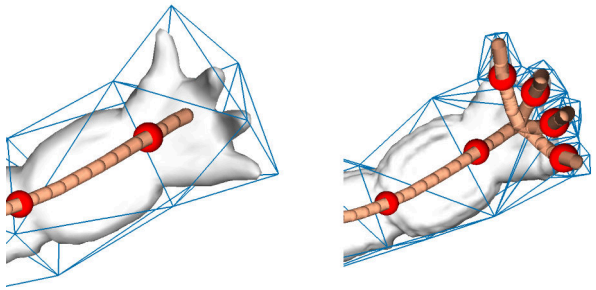


Figure 4.13: Detail of the Armadillo hand. Since it is the skeleton which drives the construction of the cage, it is possible to add or remove details just adding or removing limbs in the skeleton provided as input. Two possible cages are shown.

Despite the ability of the user to control the caging process by prescribing bending nodes, an additional source of control is granted by the

skeleton itself, which sets the overall structure of the cage. Coarse skeletons that do not catch all the tiny protuberances of a shape induce a simplified cage (Figure 4.14), while finer skeletons are able to catch all the tiny details of the character and enable an explicit control of such structures for animation (Figure 4.15).

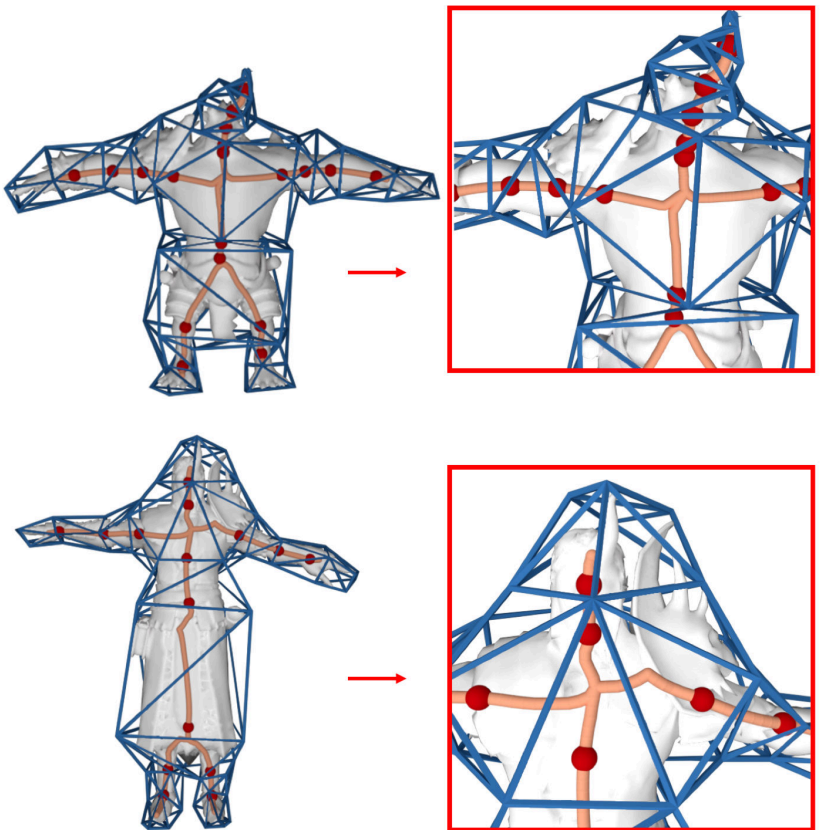


Figure 4.14: Models Warrok (top) and Ganfaul (bottom) contain spiky elements or protrusions that are not captured by the skeleton. The close-up (right) of the spiky elements in these models is shown. Our method is tolerant to these features and builds a cage that correctly contains them.

Nevertheless, also digital characters with features that are not well described by a skeleton, such as the skirt in Figure 4.18 and the hat in Figure 4.1, are robustly handled. We also validated our cages by posing digital characters using the CageLab tool [CCLS18]; some of the key poses we generated are depicted in Figure 4.19.

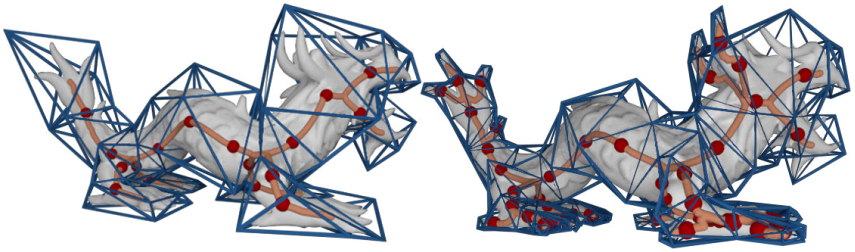


Figure 4.15: The Asian dragon model with two different skeletons: on the left, a coarser cage is generated embedding all the spikes which are not represented by the skeleton; on the right, a high-resolution cage captures the finer protrusions (fingers, tail, horns) of the input model.

4.3.1 Comparisons

In Figure 4.16 we compare our results against state-of-the-art methods for caging. Examples in the left column are obtained by combining aggressive mesh decimation [GH97], followed by inflation with the Nested Cages [SVJ15]. As expected, being based on purely geometric criteria, this fully automatic method fails to preserve symmetry, it misses important bending points, and it produces a highly irregular cage, yet with a larger number of vertices than ours. In the middle column, we show results obtained with the recent interactive method by [LD17]. This method requires the user to define cutting planes that progressively partition the model and define the cage cross-sections. Planes are designed with the mouse, sketching segments on a 3D canvas. This operation takes minutes, let alone further adjustments that may be necessary to solve intricate configurations. Notice that the use of cutting planes limits the cage to have flat sections and, as acknowledged by the authors, may produce self-intersections that cannot be easily removed. We argue that our interaction is faster, as well as more intuitive and robust. We only require the user to click on the skeleton to locate bending points, an operation that can be done in a few seconds (Table 4.1, T_{ui}). Geometric complexity is automatically addressed by our volumetric approach based on a harmonic field, which naturally follows the shape of the character, introducing non-planar cross sections wherever necessary (see, e.g., Figure 4.4).

Figure 4.17 shows a comparison with respect to the bounding shapes produced by the recent automatic method presented in [CB17]. Although such method is not meant to produce cages for animation, we notice that they nicely enclose the characters in rather tight cages. However, as for the other automatic method we compare with, the distribution of vertices in such cages totally misses the semantics needed for animation. In [CB17],

coarser cages are also shown, which were not released in the public domain for comparison. So we argue that they may possibly produce cages as coarse as ours, but without considering a proper placement of control points. Moreover, aggressive simplification of the envelope may change the global topology of the cage, e.g., by joining two feet/legs if they rest close to each other, as shown in several examples in [CB17]. This is indeed a benefit for application in collision detection, while it makes the cage no longer suitable for animation. Finally, in order to obtain variable resolution, e.g., to build a cage that may control every single finger of a hand, they need interaction with a virtual brush, which is probably heavier than the placement of bending nodes in our method.

4.3.2 Timing

Processing times for the various models used in our experiments are reported in Table 4.1. Pre-processing is performed once as the model is loaded and times depend on the complexity of the input models, varying between less than one second to about one minute in our experiments. Roughly speaking, user interaction requires about one second per bending node selected on the skeleton, making this phase about one order of magnitude faster than user-assisted techniques based on cutting planes. The construction of the base complex and its pre-inflation phases together report times from less than one second to about three seconds for all models, except the Tyra at high resolution, which requires almost 20 seconds. Overall, these phases are compatible with an interactive usage: the topology of the cage is immediately visible on the base complex, and the user is allowed to cycle on them to edit the bending nodes and correct the cage after running the final inflation.

The final inflation with Nested Cages is computation intensive and requires between 45 seconds for the simplest model to about 15 minutes for the hi-res Tyra. Although this last phase is typically one-shot, long processing times may be not compatible with practical usage. In order to bring processing times to reasonable bound, even with high resolution characters – such as the ones created with 3D sculpting tools like ZBrush [Pix07] – a relevant speedup can be achieved by substituting the original character with a proxy shape obtained via aggressive mesh decimation [GH97]. A cage built upon a low-res proxy containing just a few thousands faces may be inflated in less than a minute, yet giving a result quite close to the final one. The full resolution character is eventually re-introduced for a further step of inflation with Nested Cages, which takes in input the cage inflated about the proxy. Thanks to this warm start in the final inflation, the total time required with this techniques is much less than that the time spent by the method running directly on the hi-res model.

In Table 4.1 we report two examples of this approach. To give concrete numbers, building a cage directly for the high resolution version of the *Gecko* (∼75K tris) required 870 seconds, while the whole pipeline required just 40 seconds on the low resolution proxy (1000 tris); with an additional 130 seconds for the final inflation, the cage was adapted to the hi-res model, yielding a 80% speedup. With the more complex *Tyra* (200K tris),and a relatively larger proxy (25K tris), we still get a 50% speedup.

The table 4.1 reports the times of each step of the pipeline in details. Here are reported the informations presented in the table: M_v and M_f are the vertices and faces of the input model; BN are the bending nodes selected in the interactive stage; C_v and C_f are the vertices and faces of the cage; T_{pp} is the pre-processing time; T_{ui} is the user interaction time; T_{bc} is the time required to build the base complex; T_{inf} is the time required to inflate the base complex; T_{nc} is the time for the execution of the *Nested Cages* algorithm; T_{tot} is the sum of the times in the previous columns rounded to integer. $T_{\text{nc}*}$ is the time for the final *Nested Cages* when the model is used as a proxy for the hi-res model in the previous row. All times are in seconds.

4.3.3 Deformations

To evaluate the effective quality of the cages generated with our method, we tried their usability for animating digital characters. To this purpose we used our tool CageLab [CCLS18] for cage-based deformations. The tool has the implementation of two different barycentric coordinates: Mean Value [JSW05b] and Green Coordinates [LLCO08]. We tried to use our cages to reproduce the deformations obtained with manually generated cages. In our experiments we derived several poses to simulate a possible animation sequence. An example of the produced deformations is in Figure 4.19. Even if a usability test is well beyond the scope of this paper, based on these experiments we believe that the animations produced with our cages are of good quality, and that our tool may be useful in a professional animation setup.

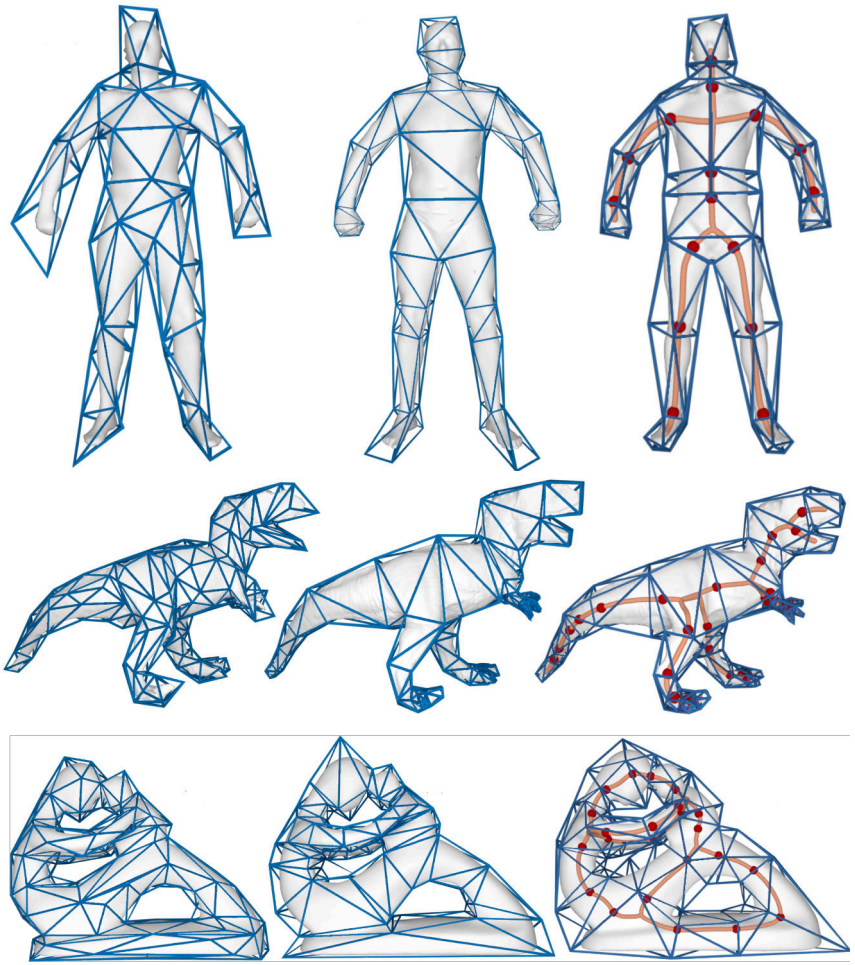


Figure 4.16: Comparison between cages obtained with: aggressive mesh decimation followed by inflation via Nested Cages [SVJ15] (left column); interactive cutting planes [LD17] (middle column); our method (right column). For each cage, we report vertex count.

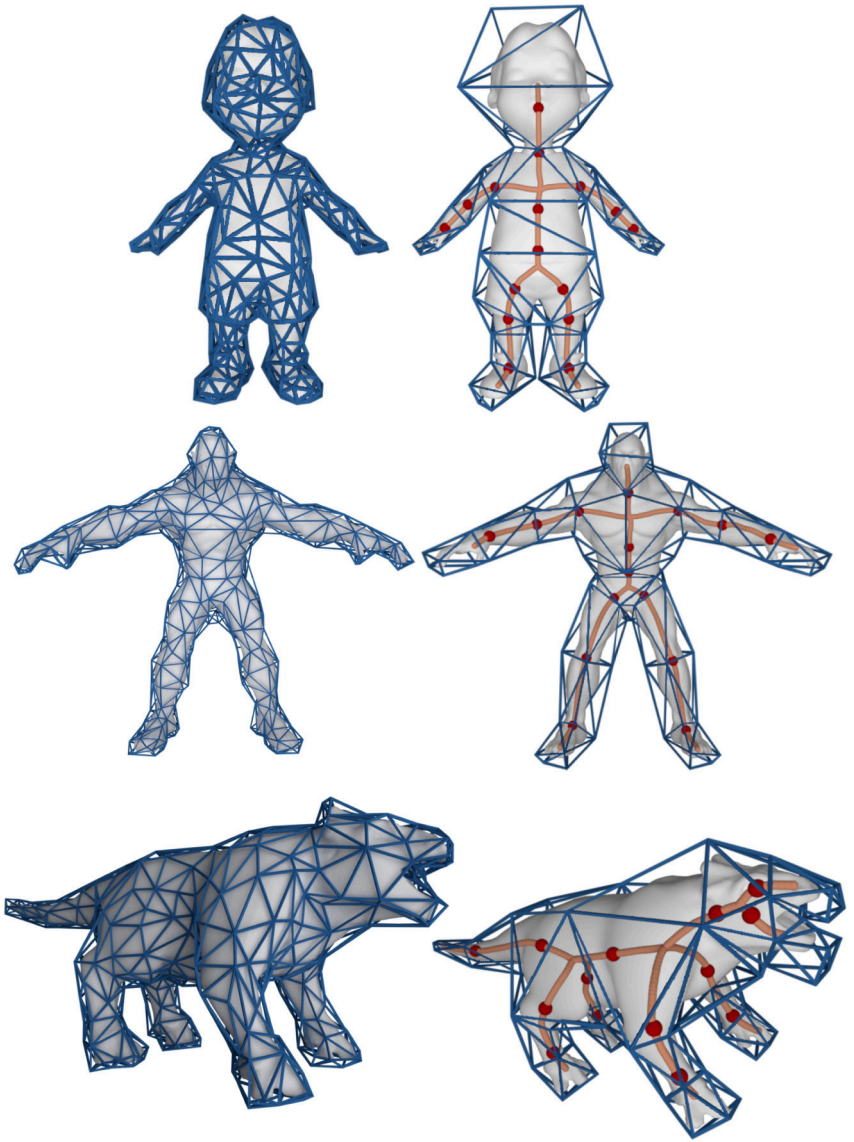


Figure 4.17: Comparison with [CB17] on the Boy, Beast and Animal datasets. Our cages to the right.

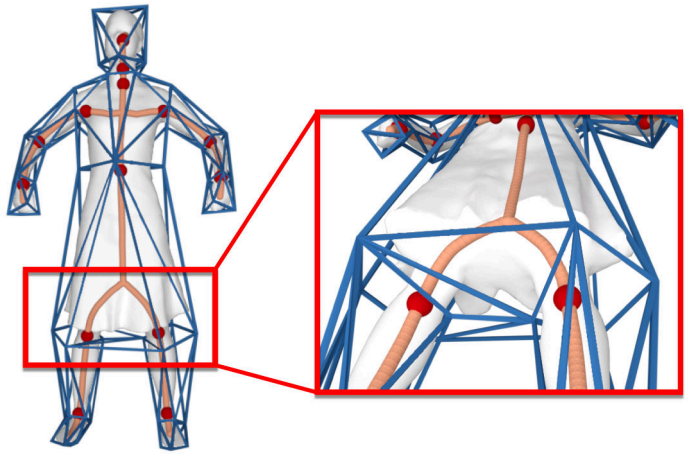


Figure 4.18: Cage of a woman with long skirt. Even if the skirt is not properly caught by the skeleton, our algorithm is able to produce a quality cage that tightly encloses it (see closeup).

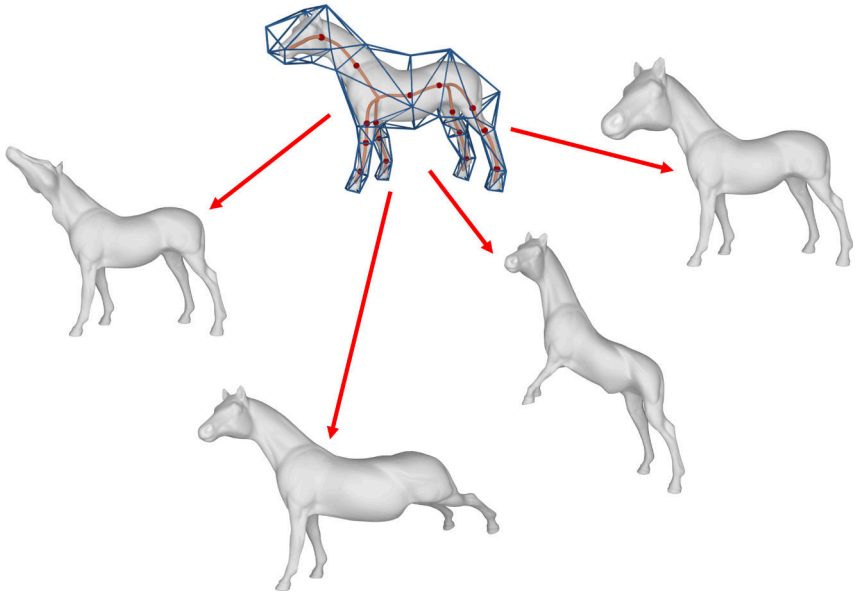


Figure 4.19: Different poses of a horse obtained editing a cage produced with our technique. For cage editing we used CageLab [CCLS18].

Model	Figure	M_v	M_f	BN	C_v	C_f	T_{pp}	T_{ui}	T_{bc}	T_{inf}	T_{nc}	T_{tot}	T_{nc*}
Animal	4.12	19552	39040	16	84	164	6.74	15	1.02	0.29	813.26	836	
Antcat	4.8;4.20	1550	3096	29	156	308	0.66	17.75	0.16	0.26	45.37	64	
Armadillo	3.2	10002	20000	17	100	196	2.92	13.2	0.75	0.33	135.21	152	
Asian Dragon	4.15	10000	19996	17	96	188	3.11	15.89	0.61	2.46	108.54	131	
Asian Dragon	4.15	10000	19996	51	348	692	3.45	50.32	1.43	1.17	169.21	226	
Beast	4.17	15122	30264	15	76	148	5.26	10	0.6	0.22	887.38	903	
BigBunny	4.20	21763	43522	14	108	212	10.2	12.22	2.43	1.1	541.5	567	
Boy	4.7;4.12;4.17	7502	15000	16	84	164	2.34	12.53	0.5	0.32	67.88	84	
Cat	4.21	27246	54488	19	108	212	11.73	13.11	1.55	0.5	297.76	325	
Dance	4.4;4.18	9971	19938	13	72	140	3.52	11.99	0.33	0.14	102.28	118	
Dinopet	4.20	4500	8996	27	156	308	1	20.1	0.33	0.34	70.6	92	
Dragon	4.21	5000	10000	22	124	284	1.47	14.58	0.35	0.2	109.7	126	
Elk	4.20	23114	46228	15	88	176	7.14	11.2	1.29	0.14	170.12	190	
Fertility	4.16	9994	20000	23	92	196	33.01	16.59	0.26	0.34	147	197	
Ganfaul	4.14	14590	29192	15	76	148	4.12	12.54	0.58	0.29	491.32	509	
Gecko (high)	4.21	37352	74700	24	120	236	12.55	14.54	1.96	0.68	840	870	
Gecko (low)	4.21	500	1000	24	120	236	0.15	16.45	0.06	0.16	23.37	40	130
Hand	4.21	14347	28690	15	84	164	5.2	13.5	0.7	0.2	197.27	217	
Homer	4.12	12997	25990	16	84	164	4.79	13.7	0.62	0.32	80.45	100	
Horse	4.12;4.19	19850	39696	15	76	148	8.67	9.55	0.8	0.29	175.49	195	
Joker	4.21	13328	26652	22	132	260	4.26	13.25	1.52	0.39	191.89	211	
Lion	4.21	27899	55794	17	92	180	10.18	15.2	1.22	0.4	152.06	179	
ManTpose	4.20	13356	26708	14	76	148	4.02	10.2	0.51	0.11	86.05	101	
Octopus	4.21	10002	20000	47	224	444	3.09	30.13	0.76	0.51	62.61	97	
Scape	4.5;4.12;4.16	6318	12632	15	80	156	1.19	12.6	0.2	0.11	62.94	77	
Skater	4.20	13332	26660	14	76	148	4.59	10.26	0.68	0.18	112.23	128	
Tyra (high)	4.16	100002	200000	33	184	364	54.99	22.1	17.51	1.65	886	982	
Tyra (low)	4.16	12501	24998	33	184	364	4.56	22.45	1.41	0.67	386.48	416	90
Warrior	4.20	9474	18944	17	88	172	3.02	14.21	0.33	0.15	72.12	90	
Warrok	4.14	23528	11746	19	96	188	3.11	18.25	0.43	0.26	666.74	689	

Table 4.1: Size of meshes and timing for the whole pipeline (user-assisted cages only)

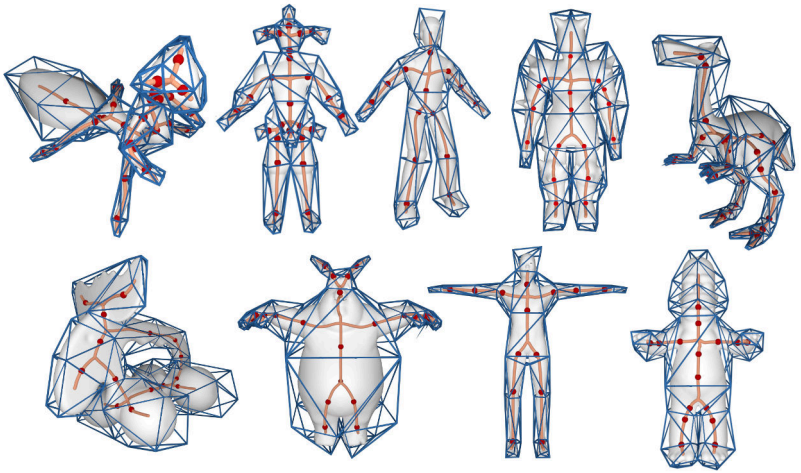


Figure 4.20: A collection of cages for models with extrinsic symmetry: Antcat, Jocker, Skater, Warrior, Dinopet, Elk, BigBunny, ManTpose and Homer.

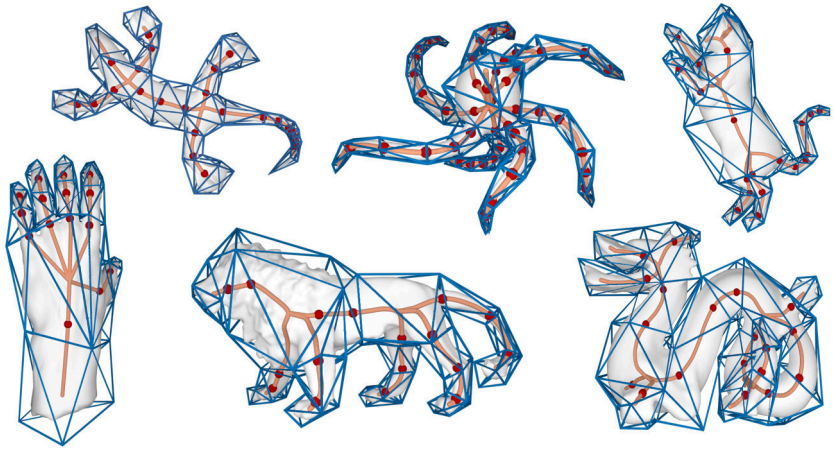


Figure 4.21: A collection of models, which are not extrinsically symmetric: Gecko, Octopus, Cat, Hand, Lion and Dragon.

Part II

Cage-based deformations

Animating a digital character is a fundamental task in computer graphics, with huge impact in the film and game industries. To realize the animation, several frames has to be defined and displayed at certain interval rate. As explained in chapter 2, the animator has to define the key-frames of the animated scene he would like to represent. The design of these key-frames is tedious and time-consuming, thus several method to simplify the animation pipeline have been proposed.

In the first part of this thesis the concept of handles has been introduced, focusing in particular on cages (2.1), and their paradigm of animation (2.1.1). Posing a digital character by manipulating the vertices of a coarse control cage is, after skeleton-based, probably the most widely used technique for digital animation. While skeleton-based techniques have been deeply researched and a variety of industrial and academic tools are available for it, cage-based techniques have historically received less attention.

For these reasons, we introduce CageLab, a novel software tool for the visualization, editing and generation of cage-based animation. The main purpose of CageLab is to support this growing interest for cage-based animation, providing an open source and easy to install shared platform where researchers and practitioners can take advantage of a system that allows them to:

- **Animate a digital character**, setting a number of its poses, building the key-frames and generating the remaining ones, interpolating between them. CageLab 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, verifying 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). CageLab 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; The user can also adjust the cage, by acting on its vertices, to make it better suited for the animation purposes.

- **Evaluate barycentric coordinates**, plotting the relation between each cage vertex and the underlying digital character. CageLab uses the widespread color map, (from red to white, which is the mainly used to display weights in animation) to visualize the influence of a subset of cage nodes directly on the surface of the character (Figure 5.3). As for cages, direct comparisons between alternative barycentric coordinates can be created by fixing a point of view and plotting cage-character attraction with respect to a specific node. These visuals are very popular in literature [ZDL*14], 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, CageLab can be used for **educational** purposes, both for preparing educational material, but also as support tool to teach animation at universities and high schools.

The following chapter will present Cagelab [CCLS18], a novel research oriented tool for cage-based deformations. This work has been presented at the conference STAG (Smart Tools and Application in Graphics) 2018, held in Brescia (Italy).

Chapter 5

CageLab

The tool has been developed allow to the user to perform cage based deformation, visualize the weights influence comparing the different barycentric coordinates implementation, and save each pose building a sequence of keyframes. Similarly to tools that were released by the community and sustained the growth of skeleton-based techniques [BP07b, JP*17], CageLab is born to 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. This chapter presents the main features of our tool.

5.1 Motivation

Cage-based deformations have played an important role in research field in the last years. Several of these research works focused on developing a novel coordinates definition or improving the existing cage-based deformation system. We have noticed, that having a tool to test the existing coordinates definition, to have a graphical representation of the weight functions over the model, to perform and additionally to save and export a set of animations can be extremely useful. Moreover, we observe that having the chance to compare in practice the different kind of coordinates is valuable. For these reasons we have been motivated to develop an easy and intuitive tool to provide this functionality.

Plenty of modern and well-known professional software, like Maya, Blender and 3DStudio Max, are frequently used by skilled digital artists. These software offer a variety of 3D computer graphics techniques for modelling, rendering and animating a character. Regarding animation functionalities, they provide a variety of articulation methods such as en-

veloping [LCF00], blend shapes [JTDP06], cage-based animation [JMD*07] and a plethora of other deformation methods. Although several animation functionalities are already furnished by these professional software, they are not easy and intuitive to be efficiently used by newbies and they have a long learning curve, thus it requires several hours of training to get ready to use them, therefore it is difficult to be used inside an academic research pipeline. 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.

CageLab provides a keyframe 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.

5.2 Basic Functionalities

In its current version, CageLab can internally compute a selection of the barycentric coordinates, namely the Mean Value Coordinates and the Green Coordinates. Their main differences are the deformation domain, the coordinates negativity, and the shape and details preservation. The Green Coordinates are characterized by a local domain, their coordinates values are non-negative and they can preserve the model shape and its surface details. These differences can be easily appreciated through our color map functionality and by the direct deformation of the character.

We plan to add more options in future versions. Alternative coordinates can be pre-computed outside CageLab for a specific cage and then imported into our tool with a text file.

CageLab provide a key-framing system, through an intuitive interface and an easy tool to set up a complete animation pipeline. In this way, the animation can be exported and easily imported in another external software for different final purposes.

5.3 User-Interface

CageLab has a user-friendly as well as lightweight User Interface. The figure (fig. 5.1) is a screenshot taken from our tool. The following sections will discuss the basic opearation available in the tool as well as a description of the UI and all its elements.
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 displayed, and it is the mean the user manipulate to interact with them.

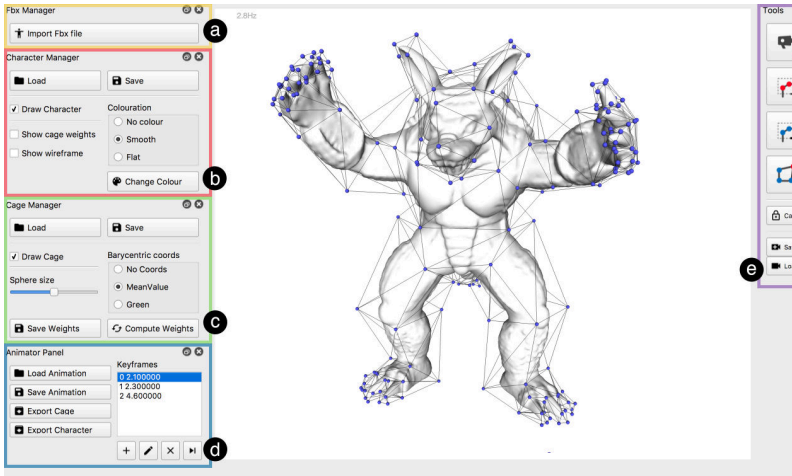


Figure 5.1: The CageLab User Interface. On the left side the FBX Importer is highlighted in yellow (a), the Character Manager panel in red (b), the Cage Manager panel in green (c), and the Animator Panel in blue (d). On the right side there is the Tools sidebar (e). The central part of the UI includes the canvas.

- 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).

5.3.1 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. Once 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 5.3.2.

For the three-dimensional character mesh, different graphical rendering settings are available: the settings are specified by the user in the *Character Manager* panel. The cage, instead, is rendered as a wireframe mesh, with each vertex (called also handle) rendered as coloured sphere, red in the case the current handle is 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.

5.3.2 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). To distinguish the active Interactive mode to the inactive ones, the relative button is displayed with 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). This functionality is useful to create images for educational or academic purposes with the same prospective.

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

Through the activation of the Camera Mode (see icon aside), the scene camera and the point of view can be manipulated by the user. 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

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.

5.3.3 Character Manager panel

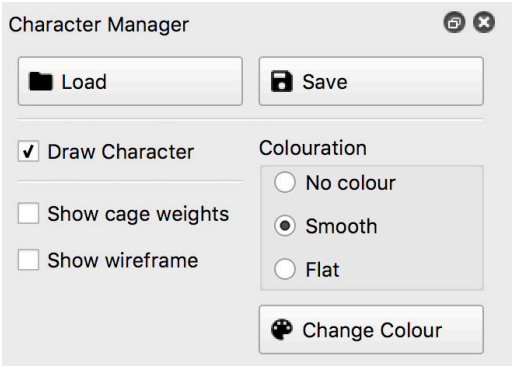


Figure 5.2: A screenshot of the Character Manager panel.

The Character Manager panel (fig. 5.2) provides all the functionalities and personalization settings related to the character mesh that is rendered into the canvas and is deformable using the cage.

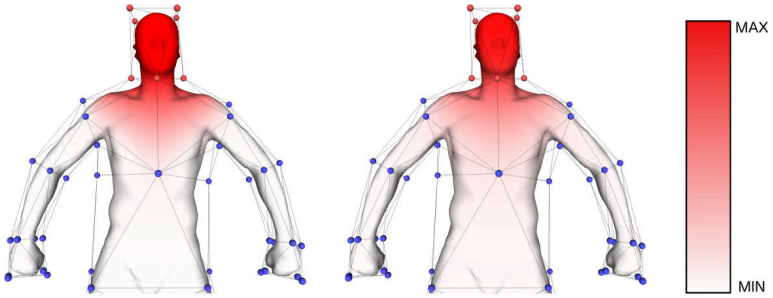
Load and **Save** buttons allow the user to import (or export) a triangle mesh file. The format available for these operations are: the *.obj*, *.off* or *.ply* format.

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.



(a) MVC (b) GC

Figure 5.3: In order to compare the smoothness and locality of alternative barycentric coordinates, CageLab allows to plot them with respect to a selection of cage node (see red spheres). This selection can be composed by a single node or by a set of the cages handles. In this example Mean Value (left) and Green (right) coordinates are shown. As can be noticed, Green are a bit less local.

The **Show cage weights** checkbox (fig. 5.3) 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.

5.3.4 Cage Manager panel

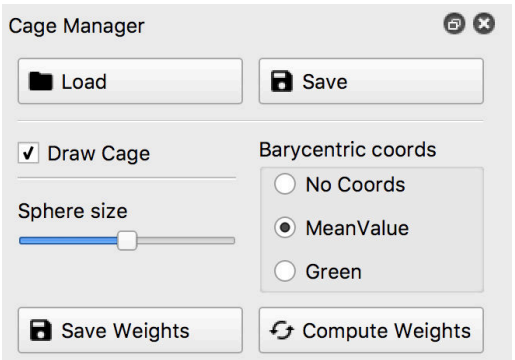


Figure 5.4: A screenshot of the Cage Manager panel.

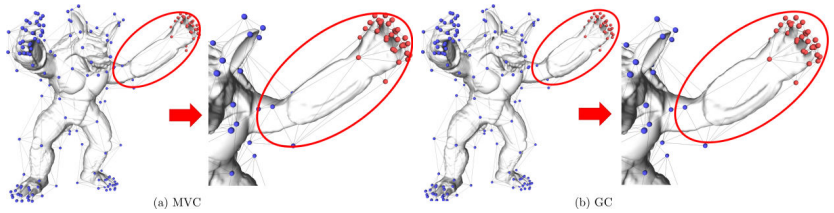


Figure 5.5: Stretching Armadillo's arm with Mean Value (left) and Green (right) coordinates. Green coordinates better preserve surface details (see closeup). CageLab allows to switch between them in real time, so that the use can spot the differences and change barycentric coordinates depending on the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

The 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.

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.

5.3.5 Animator panel

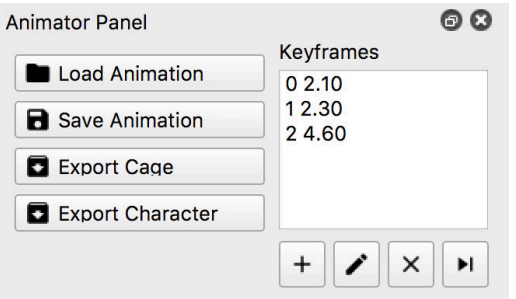


Figure 5.6: A screenshot of the Animator panel.

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.

5.4 Technical information

We have implemented our tool 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 and Sierra) and Linux (Ubuntu and Elementary) platform.

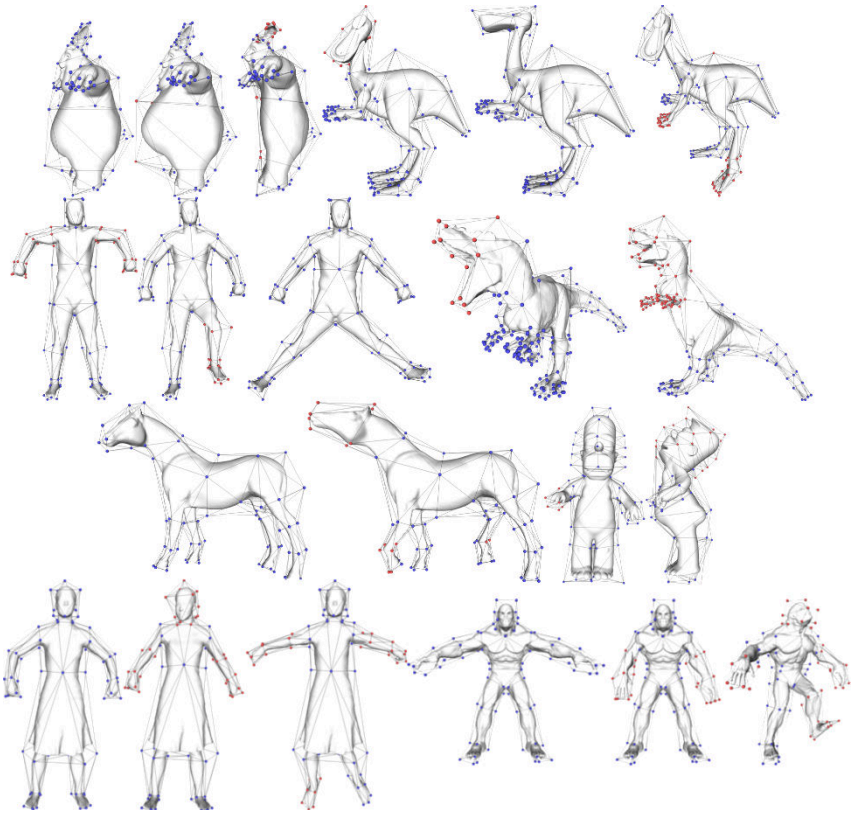


Figure 5.7: An overview of the deformations performed through CageLab.

Chapter 6

Conclusions

The deformation of a digital model is an extremely important task in Computer Animation, which plays a key role in industry jobs such as the film and game design. The animation industry demand influenced the rise of research effort in this field. Over the years, several techniques have been proposed to facilitate this task and simplify the animators work. After a brief introduction to the background (chapter 2) of handle based animation, the cage properties and the basis of cage-based deformation are presented.

In the last research papers, cages and their paradigm of animation have received an increasing interest by the scientific community. In 3.1 the related works to the cage generation are presented. As discussed in this chapter, the majority of existing methods are purely geometric approaches. Since they are fully automatic and they only rely on geometric methods, they may not fit the animators' goals. In fact, in animation context, the users need to automatize some parts of their work pipeline, but at the same time they require adequate degrees of freedom to realize the desired animation. For this reason user assisted methods have been explored. In [LD17] an interactive tool for cage generation is proposed. This method requires the user to draw planes to guide the generation of the bounding cage. Although it can effectively simplify the manual cage construction, it can still be hard for newbies to design the cage and some issues can limit the construction of the cage in certain scenarios.

In 3.2 the related works to cage-based deformations are presented. Several of these research works focused on developing a novel coordinates definition or improving the existing cage-based deformation system. We have noticed that having a tool to test the existing coordinates definition, to have a graphical representation of the weight functions over the model, to perform and additionally to save and export a set of animations can be

extremely useful.

The two main contributions of this thesis are: (1) a novel user assisted method to construct cages for animation, guided by the skeleton; (2) a novel research oriented tool to perform cage based deformation, visualize the weights influence comparing the different barycentric coordinates implementation, and save each pose building a sequence of keyframes.

In the chapter 4 we have presented our method for building animation cages. Cage generation is still considered a tedious task, even for expert users: several properties has to be fulfill, which can be in contradiction with each other. Moreover, the placement of the cage handles, requires to mix semantic requirements derived by the user intent, together with geometric constraints caused by cages properties. Our approach combines semantic and geometric informations to build cages which are more suitable for the animation pipeline. The curve skeleton drives the construction of the cage, whose provides high-level informations of the input model, therefore it allows to nicely reflect it on the cage. The novelty of our method is the definition of a harmonic field, which has been used to trace non-planar cross sections avoiding the well know problems related to planar ones. The cuts are decided by the user, who defines a set of bending nodes by clicking over the skeleton. The cross-section are built maintaining the model symmetry, then they are merged together to build the cage, which is finally inflated to envelope the input shape. As demonstrated by a variety of results, our algorithm scales well to complex shapes, which can be either provided in the canonical T-pose, or in arbitrary pose. Compared to similar approaches [LD17], we offer a faster and more intuitive user interaction, requiring just the selection of bending points on the skeleton. Placement of cage nodes is based on a flexible and reliable criterion, which allows for curved cross-sections extracted from a harmonic field in the volume, thus overcoming the popular (but tedious and limiting) cutting planes.

In the chapter 5 we have presented CageLab, a novel tool for interactive cage-based deformations of digital characters. Similarly to tools that were released by the community and sustained the growth of skeleton-based techniques [BP07b, JP*17], CageLab is born to sustain researchers and practitioners who want to get acquainted with and improve on the cage-based animation pipeline, as well as compare their ideas with the state of the art in the field. It allows users to perform cage-based deformations using two of the most popular barycentric coordinates (Mean Value and Green coordinates); it allows to compare alternative 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. We publicly release the tool to the community, with the hope

to support the recent cage-based animation growth that we observed in our community, and possibly foster even more research in the field.

6.1 Limitations

Even though our method can produce high-quality cages for a variety of characters of any topology, we are limited to the class of shapes that admit a skeletal representation. Although in the world of digital characters this is by far the dominant class of shapes, more complex shapes, such as those containing large and thin surfaces, may also occur. Such shapes may be addressed with the use of mixed line-sheet skeletons [MCM*12, TAOZ12]. Our approach can be extended to deal with such skeletons by just allowing the user to select *bending lines* on sheets, beside bending points on the line skeleton; the method for extracting cross sections based on the harmonic field nicely extends to this case, too. We plan to tackle these issues in our future work.

A second limitation is that, in some pathological cases, our method may fail to produce a valid cage if the user selects an insufficient number of bending points. This relates with the fact that keeping the topology of the cage fixed, i.e., totally determined by the skeleton and its bending points, the only mechanism to resolve intersections is inflation. For models having insufficient bending points and narrow tubular features with high curvature this may therefore result in excessive inflation, possibly resulting in collisions with distant parts of the character, and intersections that cannot be removed without refining the topology of the cage further. In all these cases, Nested Cages [SVJ15] fails. The user can easily address this issue by just adding one or more bending points in the pathological areas and running the automatic part of the method again.

6.2 Future works

Even if we provide multiples advances and insights within this thesis, cages and their paradigm of animation still remain a challenging topic in computer graphics. We will discuss here some possibilities for future work.

Concerning the cage generation, we have planned to improve our caging algorithm to support additional useful configurations: (i) n-points cross sections which may help to adapt the cage to the given input shape; (ii) quad cages, which can provide, as highlighted in [TMB], better control to animation in particular for twisting deformations. Such extensions are straightforward, since all cage elements along limbs and across the symmetry plane are already defined as quads, but they can also be easily

generalized to build n-points cross sections. As further improvement, we would investigate the possibility to introduce a machine learning step to enhance the initial placement of the bending nodes. Since, machine learning is having a substantial effect on many areas of computer science, we are convinced it may help to address also problems like cage generation and deformations issues. In particular, regarding our caging algorithm we would consider the possibility to make use of machine learning algorithms to understand semantical features, thus reducing the user intervention in the pipeline.

As future works, we plan to extend CageLab with additional barycentric coordinates, as well as new features for deformations and animation. Similarly to other research oriented tools [BTP*18], alongside the source code we will release a database of publicly available characters and cages produced with state of the art methods. With this, we hope to create the basis for a benchmark on cage generation, where new methods can be applied to a set of known digital characters for which cages produced with alternative methods are known and comparisons can be made. We plan also to perform a user test to improve the usability of the developed tool and furthermore make it available as a WebGL application.

There are still unexplored topics and challenging problems to investigate regarding cages and their paradigm of animation. An interesting discussion is about the expressiveness of the cage and the possibility to support secondary motions. The cage may be used as proxy to generate such motions (eg. oscillating chest handles to emulate the breath), but it requires extensive work and it may be just interleaved with the primary motion. An interesting idea is combining secondary motions with the standard cage-based pipeline to enrich the animations. Future research could examine the possibility to add physics constraints in the standard cage-based deformations pipeline. A naive idea could be making use of the information of model interior, to add motion constraints in the deformation (eg. neglecting stretch in the case of hard tissue or giving elastic freedom in the case of the soft body). This idea may constitute the object of future studies.

Bibliography

[ATC*08] Au O. K.-C., Tai C.-L., Chu H.-K., Cohen-Or D., Lee T.-Y.: Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 44:1–44:10. URL: http://doi.acm.org/10.1145/1360612.1360643, doi:10.1145/1360612.1360643.

[BCWG09] Ben-Chen M., Weber O., Gotsman C.: Spatial Deformation sfer. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, ACM, pp. 67–74. doi:10.1145/1599470.1599479.

[BKBH09] Bolitho M., Kazhdan M., Burns R., Hoppe H.: Parallel poisson surface reconstruction. In *Advances in Visual Computing* (Berlin, Heidelberg, 2009), Bebis G., Boyle R., Parvin B., Koracin D., Kuno Y., Wang J., Wang J.-X., Wang J., Pajarola R., Lindstrom P., Hinkenjann A., Encarnação M. L., Silva C. T., Coming D., (Eds.), Springer Berlin Heidelberg, pp. 678–689.

[BMSF06] Biasotti S., Marini S., Spagnuolo M., Falcidieno B.: Sub-part correspondence by structural descriptors of 3D shapes. *Computer-Aided Design 38*, 9 (2006), 1002–1019. doi:10.1016/j.cad.2006.07.003.

[BMU*16] Barbieri S., Meloni P., Usai F., Spano L. D., Scateni R.: An Interactive Editor for Curve-Skeletons: SkeletonLab. *Computer & Graphics 60* (2016), 23–33. doi:10.1016/j.cag.2016.08.002.

[BP07a] Baran I., Popović J.: Automatic rigging and animation of 3d characters. *ACM Trans. Graph. 26*, 3 (July 2007). URL: http://doi.acm.org/10.1145/1276377.1276467, doi:10.1145/1276377.1276467.

[BP07b] Baran I., Popović J.: Automatic rigging and animation of 3d characters. *ACM Transactions on graphics (TOG) 26*, 3 (2007), 72.

[BTP*18] Bracci M., Tarini M., Pietroni N., Livesu M., Cignoni P.: Hexalab. net: an online viewer for hexahedral meshes. *arXiv preprint arXiv:1806.06639* (2018).

[CB17] Calderon S., Boubekeur T.: Bounding Proxies for Shape Approximation. *ACM Trans. Graph. 36*, 5 (2017), 57:1–57:13. doi:10.1145/3072959.3073714.

[CCLS18] Casti S., Corda F., Livesu M., Scateni R.: CageLab: an Interactive Tool for Cage-Based Deformations. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference* (2018), The Eurographics Association. doi:10.2312/stag.20181299.

[CF14] Chen X., Feng J.: Adaptive skeleton-driven cages for mesh sequences. *Computer Animation and Virtual Worlds 25*, 3-4 (2014), 445–453. doi:10.1002/cav.1577.

[CFB16] Chen X., Feng J., Bechmann D.: Mesh Sequence Morphing. *Computer Graphics Forum 35*, 1 (2016), 179–190. doi:10.1111/cgf.12718.

[CHSB10] Chen L., Huang J., Sun H., Bao H.: "Cage-based deformation transfer". *Computers & Graphics 34*, 2 (2010), 107 – 118. doi:https://doi.org/10.1016/j.cag.2010.01.003.

[CVM*96] Cohen J., Varshney A., Manocha D., Turk G., Weber H., Agarwal P., Brooks F., Wright W.: Simplification Envelopes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 119–128. doi:10.1145/237170.237220.

[DLM11] Deng Z.-J., Luo X.-N., Miao X.-P.: Automatic Cage Building with Quadric Error Metrics. *Journal of Computer Science and Technology 26*, 3 (2011), 538–547. doi:10.1007/s11390-011-1153-4.

[DM06] DeRose T., Meyer M.: Harmonic coordinates. In *Pixar Technical Memo 06-02, Pixar Animation Studios* (2006). doi:10.1.1.90.1208.

[FHK06] Floater M. S., Hormann K., Kós G.: A general construction of barycentric coordinates over convex polygons. *advances in computational mathematics 24*, 1-4 (2006), 311–331.

[FKR05] Floater M. S., Kós G., Reimers M.: Mean value coordinates in 3d. *Comput. Aided Geom. Des. 22*, 7 (Oct. 2005), 623–631. URL: `http://dx.doi.org/10.1016/j.cagd.2005.06.004`, `doi:10.1016/j.cagd.2005.06.004`.

[Flo03] Floater M. S.: Mean value coordinates. *Computer aided geometric design 20*, 1 (2003), 19–27. `doi:10.1016/S0167-8396(03)00002-5`.

[FP09] Fabri A., Pion S.: CGAL: The Computational Geometry Algorithms Library. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (New York, NY, USA, 2009), GIS '09, ACM, pp. 538–539. `doi:10.1145/1653771.1653865`.

[GH97] Garland M., Heckbert P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 209–216.

[GJ*10] Guennebaud G., Jacob B., et al.: Eigen v3. http://eigen.tuxfamily.org, 2010.

[HS08] Hormann K., Sukumar N.: Maximum entropy coordinates for arbitrary polytopes. In *Computer Graphics Forum* (2008), vol. 27, Wiley Online Library, pp. 1513–1520.

[IMH05] Igarashi T., Moscovich T., Hughes J. F.: Spatial keyframing for performance-driven animation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2005), SCA '05, ACM, pp. 107–115. URL: `http://doi.acm.org/10.1145/1073368.1073383`, `doi:10.1145/1073368.1073383`.

[Jac15] Jacobson A.: Breathing Life into Shapes. *IEEE Computer Graphics and Applications 35* (2015), 92–100.

[JBK*12] Jacobson A., Baran I., Kavan L., Popović J., Sorkine O.: Fast automatic skinning transformations. *ACM Trans. Graph. 31*, 4 (July 2012), 77:1–77:10. URL: http://doi.acm.org/10.1145/2185520.2185573, doi:10.1145/2185520.2185573.

[JBPS11a] Jacobson A., Baran I., Popović J., Sorkine O.: Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4 (July 2011), 78:1–78:8. URL: http://doi.acm.org/10.1145/2010324.1964973, doi:10.1145/2010324.1964973.

[JBPS11b] Jacobson A., Baran I., Popovic J., Sorkine O.: Bounded Biharmonic Weights for Real-time Deformation. *ACM Trans. Graph. 30*, 4 (2011), 78:1–78:8. doi:10.1145/2010324.1964973.

[JDKL14a] Jacobson A., Deng Z., Kavan L., Lewis J.: Skinning: Real-time shape deformation. In *ACM SIGGRAPH* (2014), vol. 22.

[JDKL14b] Jacobson A., Deng Z., Kavan L., Lewis J.: Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH 2014 Courses* (2014).

[JMD*07] Joshi P., Meyer M., DeRose T., Green B., Sanocki T.: Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3 (July 2007). URL: http://doi.acm.org/10.1145/1276377.1276466, doi:10.1145/1276377.1276466.

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

[JSW05a] Ju T., Schaefer S., Warren J.: Mean value coordinates for closed triangular meshes. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 561–566.

[JSW05b] Ju T., Schaefer S., Warren J.: Mean Value Coordinates for Closed Triangular Meshes. *ACM Trans. Graph. 24*, 3 (2005), 561–566. doi:10.1145/1073204.1073229.

[JTDP06] Joshi P., Tien W. C., Desbrun M., Pighin F.: Learning controls for blend shape based realistic facial animation. In *ACM Siggraph 2006 Courses* (2006), ACM, p. 17.

[JZvdP*08] Ju T., Zhou Q.-Y., van de Panne M., Cohen-Or D., Neumann U.: Reusable skinning templates using cage-based deformations. *ACM Trans. Graph. 27*, 5 (2008), 122:1–122:10. doi:10.1145/1409060.1409075.

[KCATCO*10] Kin-Chung Au O., Tai C.-L., Cohen-Or D., Zheng Y., Fu H.: Electors voting for fast automatic shape correspondence. *Computer Graphics Forum 29*, 2 (2010), 645–654. doi:10.1111/j.1467-8659.2009.01634.x.

[KCvO07] Kavan L., Collins S., Žára J., O'Sullivan C.: Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 39–46. URL: http://doi.acm.org/10.1145/1230100.1230107, doi:10.1145/1230100.1230107.

[KCvO08] Kavan L., Collins S., Žára J., O'Sullivan C.: Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27*, 4 (Nov. 2008), 105:1–105:23. URL: http://doi.acm.org/10.1145/1409625.1409627, doi:10.1145/1409625.1409627.

[LAPS17] Livesu M., Attene M., PatanÃ¨ G., Spagnuolo M.: Explicit Cylindrical Maps for General Tubular Shapes. *Computer-Aided Design 90* (2017), 27 – 36. SI:SPM2017. doi:10.1016/j.cad.2017.05.002.

[LCF00] Lewis J. P., Cordner M., Fong N.: 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* (New York, NY, USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., pp. 165–172. URL: http://dx.doi.org/10.1145/344779.344862, doi:10.1145/344779.344862.

[LD17] Le B. H., Deng Z.: Interactive Cage Generation for Mesh Deformation. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2017), I3D '17, ACM, pp. 3:1–3:9. doi:10.1145/3023368.3023369.

[LGS12] Livesu M., Guggeri F., Scateni R.: Reconstructing the Curve-Skeletons of 3D Shapes Using the Visual Hull.

IEEE Transactions on Visualization and Computer Graphics 18, 11 (2012), 1891–1901. doi:10.1109/TVCG.2012.71.

[Liv17] Livesu M.: cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes., 2017. https://github.com/mlivesu/cinolib/.

[LKCOL07] Lipman Y., Kopf J., Cohen-Or D., Levin D.: GPU-assisted Positive Mean Value Coordinates for Mesh Deformations. In *Geometry Processing* (2007), The Eurographics Association. doi:10.2312/SGP/SGP07/117-123.

[LLCO08] Lipman Y., Levin D., Cohen-Or D.: Green Coordinates. *ACM Trans. Graph. 27*, 3 (2008), 78:1–78:10. doi:10.1145/1360612.1360677.

[LMPS16] Livesu M., Muntoni A., Puppo E., Scateni R.: Skeleton-driven Adaptive Hexahedral Meshing of Tubular Shapes. *Computer Graphics Forum 35*, 7 (2016), 237–246. doi:10.1111/cgf.13021.

[LS13] Livesu M., Scateni R.: Extracting Curve-Skeletons from Digital Shapes Using Occluding Contours. *The Visual Computer 29*, 9 (2013), 907–916. doi:10.1007/s00371-013-0855-8.

[MCM*12] Martin T., Chen G., Musuvathy S., Cohen E., Hansen C.: Generalized Swept Mid-structure for Polygonal Models. *Comput. Graph. Forum 31*, 2pt4 (May 2012), 805–814. URL: http://dx.doi.org/10.1111/j.1467-8659.2012.03061.x, doi:10.1111/j.1467-8659.2012.03061.x.

[MJ96] MacCracken R., Joy K. I.: Free-form deformations with lattices of arbitrary topology. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 181–188. URL: http://doi.acm.org/10.1145/237170.237247, doi:10.1145/237170.237247.

[MLP18] Mancinelli C., Livesu M., Puppo E.: Gradient Field Estimation on Triangle Meshes. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*

(2018), The Eurographics Association. doi:10.2312/stag.20181301.

[MPS06] Mortara M., Patané G., Spagnuolo M.: From geometric to semantic human body models. *Computers & Graphics 30*, 2 (2006), 185–196. doi:10.1016/j.cag.2006.01.024.

[MZS*11] McAdams A., Zhu Y., Selle A., Empey M., Tamstorf R., Teran J., Sifakis E.: Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph. 30*, 4 (July 2011), 37:1–37:12. URL: http://doi.acm.org/10.1145/2010324.1964932, doi:10.1145/2010324.1964932.

[NS13] Nieto J. R., Susín A.: "Cage Based Deformations: A Survey". In *Deformation Models: Tracking, Animation and Applications*, González Hidalgo M., Mir Torres A., Varona Gómez J., (Eds.). Springer Netherlands, Dordrecht, 2013, pp. 75–99. doi:10.1007/978-94-007-5446-1_3.

[Pix07] Pixologic: ZBrush, 2007. http://pixologic.com.

[PLPZ12] Panozzo D., Lipman Y., Puppo E., Zorin D.: Fields on Symmetric Surfaces. *ACM Trans. Graph. 31*, 4 (July 2012), 111:1–111:12. URL: http://doi.acm.org/10.1145/2185520.2185607, doi:10.1145/2185520.2185607.

[RF17] Rumman N. A., Fratarcangeli M.: Skin deformation methods for interactive character animation. In *Computer Vision, Imaging and Computer Graphics Theory and Applications* (Cham, 2017), Braz J., Magnenat-Thalmann N., Richard P., Linsen L., Telea A., Battiato S., Imai F., (Eds.), Springer International Publishing, pp. 153–174.

[Sav16] Savoye Y.: Cage-based Performance Capture. In *SIGGRAPH ASIA 2016 Courses* (New York, NY, USA, 2016), SA '16, ACM, pp. 12:1–12:53. doi:10.1145/2988458.2988459.

[SGG*00] Sander P. V., Gu X., Gortler S. J., Hoppe H., Snyder J.: Silhouette Clipping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2000), SIGGRAPH '00,

ACM Press/Addison-Wesley Publishing Co., pp. 327–334. doi:10.1145/344779.344935.

[Sha08] Shamir A.: A survey on mesh segmentation techniques. In *Computer graphics forum* (2008), vol. 27, Wiley Online Library, pp. 1539–1556.

[Si15] Si H.: TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM s. Math. Softw. 41*, 2 (2015), 11:1–11:36. doi:10.1145/2629697.

[SP86a] Sederberg T. W., Parry S. R.: Free-form Deformation of Solid Geometric Models. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 151–160. doi:10.1145/15922.15903.

[SP86b] Sederberg T. W., Parry S. R.: Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 4 (Aug. 1986), 151–160. URL: http://doi.acm.org/10.1145/15886.15903, doi:10.1145/15886.15903.

[SVJ15] Sacht L., Vouga E., Jacobson A.: Nested cages. *ACM Trans. Graph. 34*, 6 (2015), 170:1–170:14. doi:10.1145/2816795.2818093.

[TAOZ12] Tagliasacchi A., Alhashim I., Olson M., Zhang H.: Mean curvature skeletons. *Computer Graphics Forum 31*, 5 (2012), 1735–1744.

[TDS*16] Tagliasacchi A., Delame T., Spagnuolo M., Amenta N., Telea A.: 3D Skeletons: A State-of-the-Art Report. *Computer Graphics Forum 35*, 2 (2016), 573–597. doi:10.1111/cgf.12865.

[TMB] Thiery J.-M., Memari P., Boubekeur T.: Mean value coordinates for quad cages in 3D. *ACM Trans. Graph. - Proc. SIGGRAPH Asia 2018*. To appear.

[TTB12] Thiery J.-M., Tierny J., Boubekeur T.: CageR: Cage-Based Reverse Engineering of Animated 3D Shapes. *Computer Graphics Forum 31*, 8 (2012), 2303–2316. doi:10.1111/j.1467-8659.2012.03159.x.

[ULP*15] Usai F., Livesu M., Puppo E., Tarini M., Scateni R.: Extraction of the Quad Layout of a Triangle Mesh Guided by Its Curve Skeleton. *ACM Trans. Graph. 35*, 1 (2015), 6:1–6:13. doi:10.1145/2809785.

[VBG*13] Vaillant R., Barthe L., Guennebaud G., Cani M.-P., Rohmer D., Wyvill B., Gourmel O., Paulin M.: Implicit skinning: Real-time skin deformation with contact modeling. *ACM Trans. Graph. 32*, 4 (July 2013), 125:1–125:12. URL: http://doi.acm.org/10.1145/2461912.2461960, doi:10.1145/2461912.2461960.

[WPG12] Weber O., Poranne R., Gotsman C.: Biharmonic coordinates. *Comput. Graph. Forum 31*, 8 (Dec. 2012), 2409–2422. URL: http://dx.doi.org/10.1111/j.1467-8659.2012.03130.x, doi:10.1111/j.1467-8659.2012.03130.x.

[XLG09] Xian C., Lin H., Gao S.: Automatic generation of coarse bounding cages from dense meshes. In *Shape Modeling and Applications, 2009. SMI 2009. IEEE International Conference on* (2009), IEEE, pp. 21–27. doi:10.1109/SMI.2009.5170159.

[XLG12] Xian C., Lin H., Gao S.: Automatic cage generation by improved OBBs for mesh deformation. *The Visual Computer 28*, 1 (2012), 21–33. doi:10.1007/s00371-011-0595-6.

[XLX15] Xian C., Li G., Xiong Y.: Efficient and effective cage generation by region decomposition. *Computer Animation and Virtual Worlds 26*, 2 (2015), 173–184. doi:10.1002/cav.1571.

[YCSZ13] Yang X., Chang J., Southern R., Zhang J. J.: Automatic cage construction for retargeted muscle fitting. *The Visual Computer 29*, 5 (2013), 369–380. doi:10.1007/s00371-012-0739-3.

[ZDL*14] Zhang J., Deng B., Liu Z., Patanè G., Bouaziz S., Hormann K., Liu L.: Local barycentric coordinates. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 188:1–188:12. URL: http://doi.acm.org/10.1145/2661229.2661255, doi:10.1145/2661229.2661255.

[ZYH*15] Zhou Y., Yin K., Huang H., Zhang H., Gong M., Cohen-Or D.: Generalized Cylinder Decomposition. *ACM Trans. Graph. 34*, 6 (2015), 171:1–171:14. doi:10.1145/2816795.2818074.