Computers & Graphics (2019)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Skeleton Based Cage Generation Guided by Harmonic Fields

Sara Casti[a,b], Marco Livesu[c], Nicolas Mellado[d], Nadine Abu Rumman[e], Riccardo Scateni[b], Loïc Barthe[d], Enrico Puppo[a]

[a] *Department of Informatics, Bioengineering, Robotics and System Engineering, University of Genoa, Genoa, Italy*
[b] *Department of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy*
[c] *IMATI - CNR, Genoa, Italy*
[d] *IRIT-CNRS - Université Paul Sabatier, Toulouse, France*
[e] *University College London, London, United Kingdom*

ARTICLE INFO

Article history:
Received April 24, 2019

Keywords: Computational Geometry, Object Modeling, Curve Surface Solid and Object Representations

ABSTRACT

We propose a novel user-assisted cage generation tool. We start from a digital character and its skeleton, and create a coarse control cage for its animation. Our method requires minimal interaction to select bending points on the skeleton, and computes the corresponding cage automatically. The key contribution is a volumetric field defined in the interior of the character and embedding the skeleton. The integral lines of such field are used to propagate cutting surfaces from the interior of the character to its skin, and allow us to robustly trace non-planar cross sections that adapt to the local shape of the character. Our method overcomes previous approaches that rely on the popular (but tedious and limiting) cutting planes. We validated our software on a variety of digital characters. Our final cages are coarse yet entirely compliant with the structure induced by the underlying skeleton, enriched with the semantics provided by the bending points selected by the user. Automatic placement of bending nodes for a fully automatic caging pipeline is also supported.

1. Introduction

Skeletons and control cages are possibly the two most widely used techniques to pose a digital character, enabling its deformation and animation [1]. 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 – such as the swish of a cloak or the jiggle of a body part. 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.

Cages are intrinsically more complex to create than skeletons. Well established properties have to be fulfilled [2, 3]: (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.

Most often, cages are hand-made and their creation may require hours of extensive work by skilled artists. 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.

Automatic approaches are purely geometric: they rely on the character in a given 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 important bending points, such as

knees and elbows, not to mention finer details like fingers, jaws, or eyelids (Figure 1). Likewise, for invertebrates such as the Octopus in Figure 17, 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.

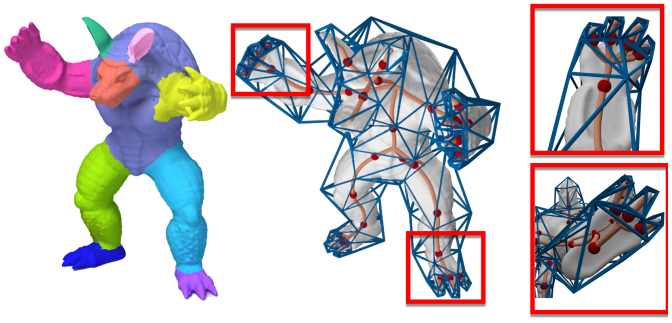

Fig. 1. Left: a purely geometric segmentation obtained with a state-of-the-art method [4] does not capture bending junctions between disjoint semantic parts (neck, elbows, knees, wrists, joints of fingers and toes). Right: in our interactive system the user manually selects bending points, inducing an animation-aware semantic decomposition of the character.

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 [5, 6, 7, 8] or manually crafted [9]. 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). This approach overcomes the burden and the limitations of previous user-assisted methods, based on cutting planes [3], 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. [10] 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 2: 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); the cage is finally 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. [2] 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). Thanks to the flexibility of our non-planar cuts, user interaction is greatly simplified if compared to previous approaches [3], and imprecise inputs from the user are robustly handled, without affecting the quality of the final cage (Figure 3). In Section 4, 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 (Figure 17). Our bounding cages preserve all model features captured by the underlying skeleton, as well as all user selected cross sections; at the same time, they are robust against details that are not represented by the skeleton and nicely envelope them too (Figure 13 and 12); thus providing a desired/effective shape abstraction for animation purposes.

2. Related works

Early methods for cage generation based on the subdivision of bounding boxes date back to late 80's [11]. Despite the robustness, approaches of this kind lack the proper flexibility, and are not suitable to generate cages that fulfill the requirements animation imposes. In a recent survey, Nieto and Susín [12] list such requirements, also offering an overview of modern techniques for automatic cage generation. The survey is still an excellent resource for practitioners, but a few important techniques have been released after its publication, both automatic and user-assisted.

Several automatic methods define cages as a simplified version of the character they contain [2, 13, 14, 15, 16, 17, 18], and obtain them either by applying a decimation strategy (e.g. [19]) or a remeshing technique at a coarser scale (e.g. [20]), often followed by vertex relocation to resolve intersections. Being purely geometric, these methods fail to be animation-aware, meaning that cage handles may be distant from the parts the animator wants to deform or bend. Furthermore, these methods may disrupt the symmetry of the cage. For all these reasons, this class of algorithms is usually not desirable for animation –

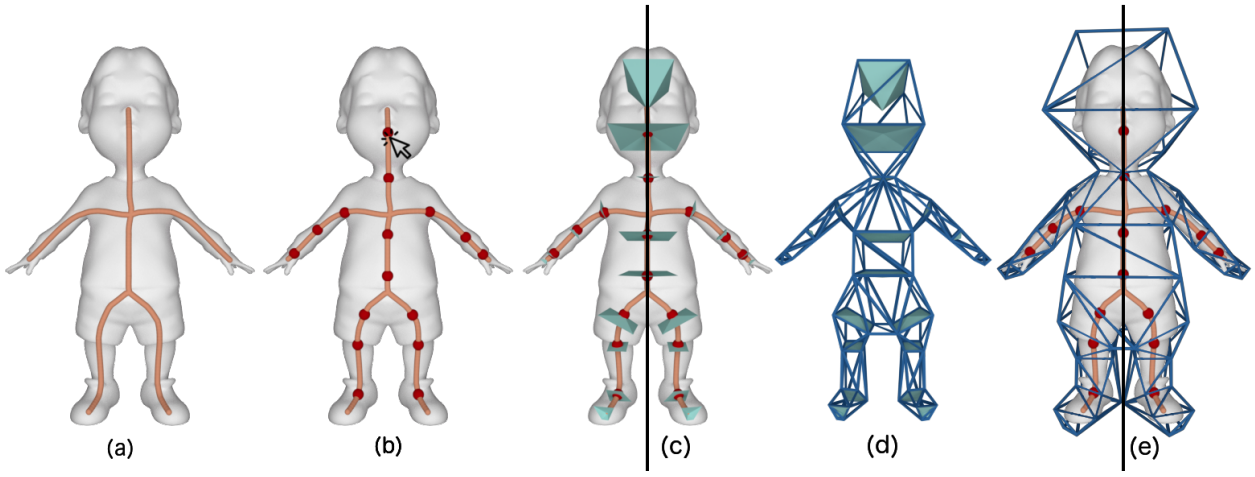

Fig. 2. Our pipeline starts from a digital character and its curve skeleton (a). The user manually selects nodes where bending occurs (b, red dots). Symmetric cross sections at bending nodes are obtained from a harmonic field in the volume (c). An initial cage is created by connecting the cross sections (d). The cage is finally inflated to accommodate the character without intersections (e).

or, at least, it requires substantial post-processing. Recently, Calderon and Boubekeur [21] presented a shape approximation algorithm which takes as input an arbitrary mesh and generates a bounding proxy that tightly encloses it. Their method is designed to support collision detection, thus it does not account for semantically important structures. If automatically computed, bounding proxies tend to be either too refined to be used as coarse control cages, or they may affect the global topology of the object, closing handles and filling holes. In order to overcome such limitations, the local scale can be corrected with an interactive brush. Our method automatically determines the local scale of the cage that is necessary to accommodate all the semantically relevant features of the character.

Similarly to us, template-based techniques [22, 23] use a guiding skeleton to infer the logical components of the character, and cage it by using predefined templates chosen according to type of joints of the underlying skeleton. Although these methods can automatically construct cages for animation, they are usually limited in the number of joints they can handle, and do not scale on complex shapes with arbitrary topology. In contrast, our approach is able to handle any line skeleton, regardless of its topology or the valence of its junction nodes. Chen and Feng [24] substituted templates with planar cross-sections locally orthogonal to the skeleton. Depending on the complexity of both the character and the pose, their approach may lead to self-intersections.

The work of Le and Deng [3] is, to date, the most advanced tool for interactive cage generation. The user constructs a cage by sketching planar cross-sections on a 3D canvas. As for [24], achieving a proper shape segmentation with planar cross-sections may require substantial manual effort, and designing a good cage around deep concavities remains challenging. The authors of [3] already acknowledge these limitations in their article, and suggest the introduction of non-planar cross sections to ameliorate their tool. Despite geometric limits, cutting planes are also difficult to handle in the user interface. Creating a plane by sketching a segment on a 2D GUI requires a deep 3D insight, and is extremely sensitive to the precision of the user. Our non-planar cuts nicely complement the user interface, making it easy to use and more robust against imprecise inputs from the user (Figure 3).

Various works addressed the problem of producing a sequence of cage motions to compactly encode an animation sequence [25, 24, 26, 27] or video-based animation [28]. This latter problem is orthogonal to the one we address in this work, with almost no algorithmic overlap.

Skeletons and semantics. Besides its application in cage generation and animation, curve-skeletons have been extensively used as a mean to encode the structure of a 3D shape, and have been exploited in a number of applications, ranging from shape recognition [29, 30], consistent mesh partitioning [4, 31], and re-meshing [32, 33, 34]. Our tool is inspired from these latter works; however, the generation of animation cages poses different constraints and goals, making none of these approaches directly applicable to our needs.

Barycentric coordinates. Typically, the cage deformers are augmented with coordinates, which impact the quality of the final deformation. A plethora of different types of barycentric coordinates have been proposed, including Mean Value Coordinates [35, 36], Green Coordinates [37, 38], Harmonic Coordinates [39, 40], Biharmonic Coordinates [1], and complex barycentric coordinates and their variants [41, 42, 43]. The main differences between the different types of coordinates regard their locality and smoothness. The cages generated with our method support any type of barycentric coordinates, which can be conveniently chosen according to the animator's needs. An interesting discussion of cage-based deformation techniques can be found in Jacobson et al. [44].

3. Method

The pipeline of our method is described in Figure 2 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 [3], 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 (see e.g. the head section in Figure 2). This feature is widely exploited by our user interface, making it robust against imprecise inputs from the user (Figure 3). Non-planarity will usually be emphasized about leaf nodes and where relatively thin limbs are attached to a larger body (e.g., armpit, groin).

3.1. Pre-processing: guiding field

Given the surface mesh $\mathcal{M}$ and the skeleton $\mathcal{S}$, we define a volumetric field that emanates radially from the skeleton to the outer surface. Such a field is inspired by the radius component of the cylindrical parameterization described in [32], and will be used later on to partition our cage.

In order to compute this field, we first generate a tetrahedral mesh that conforms to the outer surface of the character and embeds the skeleton as chains of internal edges. We then compute a harmonic function f defined over the volume spanned by the tetrahedral mesh, by resolving the Laplace problem $\Delta f = 0$, subject to Dirichlet boundary conditions

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

The integral lines of f give us a robust way to project points from the inner volume to the surface. 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 $\mathcal{S}$ (Figure 4). 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 $\mathcal{S}$.

Our guiding field has a consistent behaviour on meshes of various density (Section 5), and therefore the tracing system does not depend on a specific tetrahedralization. In our implementation we only take care of a few corner cases that may flatten the field inside the volume and block the tracing of the integral lines. Specifically, we split each inner edge having its two endpoints exposed on the surface (or on the skeleton), and also split each inner triangular face having all its vertices on the surface (or on the skeleton). This guarantees that extending the function by linear interpolation within each element, f will be flat only at $\mathcal{S}$ and $\mathcal{M}$. Note that the pre-processing is computed off-line once, before any interaction occurs, and is used without modification in later stages of our method.

3.2. User interaction

Similarly to [34, 33, 32] the input skeleton provides information on the high level structure. Namely, how many limbs, and how they connect to each other. The user may refine this structure by 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.

User selected bending nodes will originate cutting surfaces, which constitute a better alternative to the cutting planes used in [3]. 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 3).

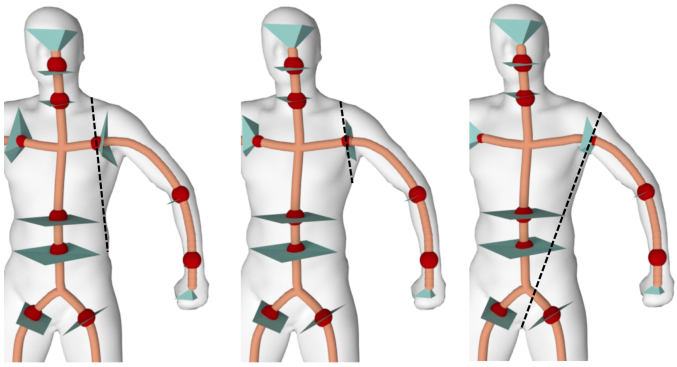

Fig. 3. Example of cuts induced by different selections of bending nodes. The cyan pyramids approximate the cuts induced by the harmonic field, while the dashed black lines represent the corresponding cuts induced by a plane orthogonal to the skeleton and through the selected node. While our cuts are all valid, the corresponding planes may induce inconsistent cuts.

Notice that, if an animation skeleton is provided as input, the cage partition will not necessarily reflect the same structure. The user may freely decide whether the cage should be compliant with it or not.

3.3. Cage generation

This is 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 3.5).

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

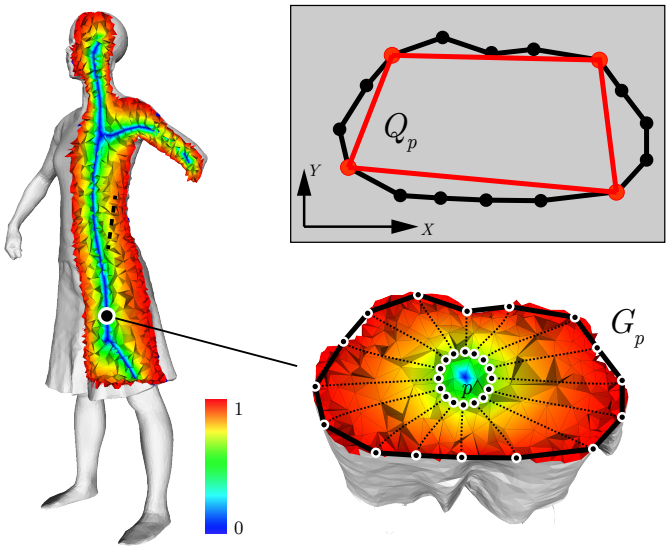

Fig. 4. We exploit the harmonic field f (left) to define the non planar cross sections of our cages. For each control point p we sample a ring locally orthogonal to the skeleton, and project it on the surface along the integral lines of f (bottom right). We project the resulting polygon G_p on the plane defined by its PCA, and find the quadrilateral Q_p that better approximates it (top right).

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 is (a simplified version of) the boundary of a cutting surface and will give a transversal section of the cage.

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 piecewise 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 $\mathcal{S}$ [45]. To avoid 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, 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 between the cutting surface at p and $\mathcal{M}$ itself. Note that, as long as the skeleton is a deformation retract of $\mathcal{M}$ and p is away from a branching node, such intersection should be a ring; however, our method is tolerant to more complicated situations, like having saddles in the harmonic field or even intersecting the surface at multiple rings, as long as the main shape is captured from a polygon spanning the points of G_p. Rings here are just meant to initialize the cross sections, which will be inflated as described in Section 3.5 to fully incorporate the object.

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. Refer to Figure 4 for a 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:

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

In our figures, the cutting surface at p is approximated with the pyramid obtained by connecting p to the vertices of Q_p (Figures 2 and 3).

We complete the tubular portions of the cage by connecting quad sections pairwise and computing their convex hull. 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.

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 [3], 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 5).

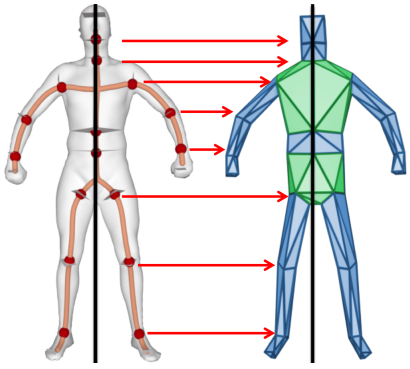

Fig. 5. The tubular structures, corresponding to portions between consecutive bending nodes along a branch of skeleton (blue) are joined with polyhedra (green) centered about branching nodes of the skeleton to form the tight cage (before inflation).

Even if not reported in [3], 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. If this is the case, 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

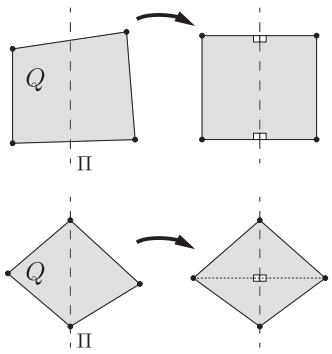

Fig. 6. Symmetrizing a quad across the symmetry plane Π. Top: two vertices of the quad lie to the left of Π, and the other two lie to the right of Π – we impose symmetry of edges connecting vertices on the same side of Π. Bottom: two vertices lie on Π and the other two lie on opposite sides of Π – we impose symmetry between the vertices not lying on Π.

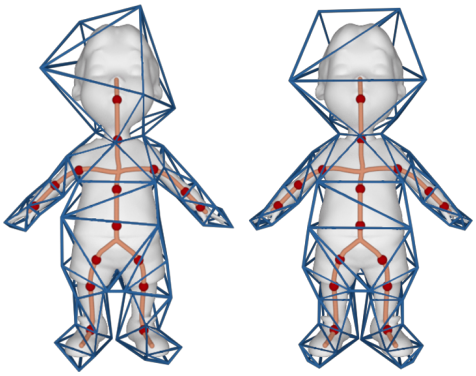

Fig. 7. A visual comparison between a cage without (left) and with (right) symmetry enforcement on the Boy dataset.

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.

3.4. Symmetry

In the digital modeling pipeline, most of the times only half of a character is explicitly modelled, 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. We automatically detect extrinsic symmetry planes with the method described in [10], and use them to adjust the position of cage sections and handles.

Let us consider a symmetry plane Π. Our strategy to symmetrize a cage works as follows. We consider 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 using it 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 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 7 shows a visual example of cages obtained with and without symmetry enforcement.

This approach can be easily extended to deal with symmetric characters given in general pose (which does not exhibit extrinsic symmetry), by using the method for intrinsic symmetry detection described in [10].

3.5. Cage inflation

The tight cage we have built so far has its vertices on the skin $\mathcal{M}$. 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 [2]. 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 8). We address this issue by pre-inflating the tight cage before the final inflation with *Nested cages*.

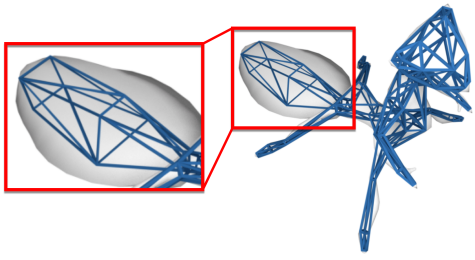

Fig. 8. Nested Cages [2] may fail without our pre-inflation mechanism. Even though all vertices of the tight cage are on the surface, on the abdomen of the ant its faces are too far from the skin to support skin contraction, which is at the basis of the Nested Cages algorithm. Antcat dataset.

We pull cage faces towards the outer surface of $\mathcal{M}$ with a simple technique summarized in Figure 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 3.1). 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 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 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. This is easily fixed as follows: we check if a face of the cage intersects any other face

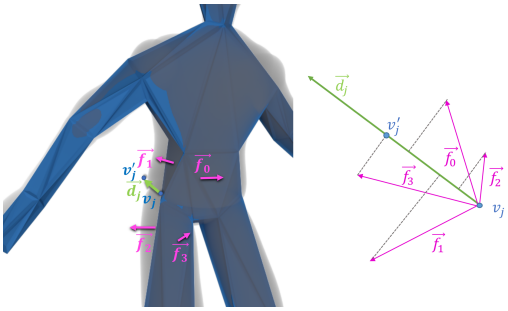

Fig. 9. Pre-inflation of the cage. Left: each face is assigned a displacement vector that brings it closer to the skin. Right: such displacements are averaged at cage vertices to pull them outwards.

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: 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 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 [2], the available implementation of Nested Cages does not support symmetry. We therefore interleave cage inflation and symmetrization (Section 3.4) of the cage to obtain the desired result.

4. 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 [46], 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 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 [6, 7]. 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' and we find its critical points. To alleviate sensitivity to the noise, we smooth function r' before using it. From

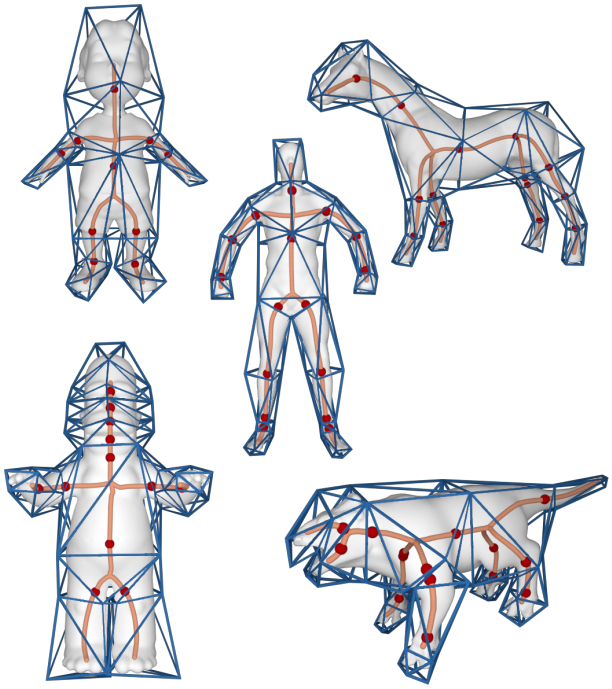

Fig. 10. A collection of cages obtained by running our pipeline in fully automatic mode. Boy, Scape, Horse, Homer and Animal datasets. Some bending points may be either missing (e.g., ankles and knees for Homer), or redundant (ankles for Scape, elbows for Animal, head for Homer).

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 10.

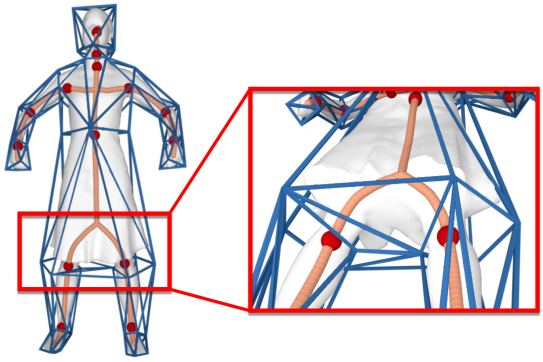

Fig. 11. Cage of the Dance dataset. Even if the skirt is not properly captured by the skeleton, our algorithm is able to produce a quality cage that tightly encloses it (see closeup).

5. 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

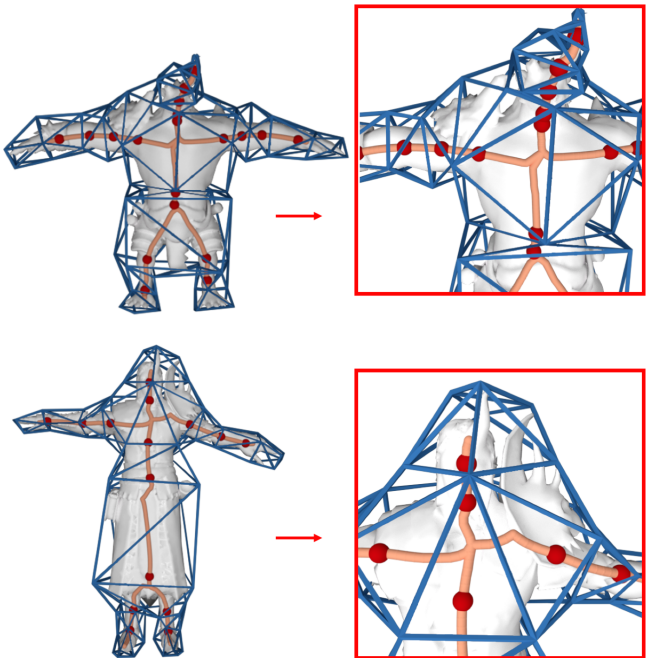

Fig. 12. Models Warrok (top) and Ganfaul (bottom) contain spiky elements or protrusions that are not captured by the skeleton. The close-up (right) of the spiky elements in these models is shown. Our method is tolerant to these features and builds a cage that correctly contains them.

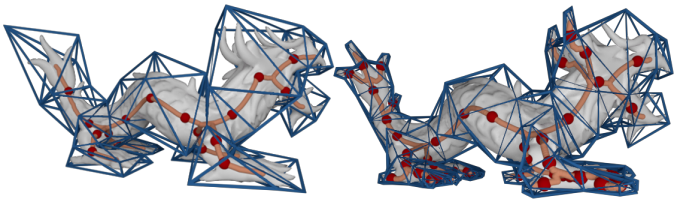

Fig. 13. The Asian dragon model with two different skeletons: on the left, a coarser cage is generated embedding all the spikes which are not represented by the skeleton; on the right, a high-resolution cage captures the finer protrusions (fingers, tail, horns) of the input model.

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 [47] for volumetric mesh generation; CGAL [48] to compute convex hulls; Eigen [49] to solve linear systems; and cinolib [50] for mesh and scalar field processing. A reference implementation of our tool can be found on GitHub at the following address: `https://github.com/SaraCasti/Skeleton-Based-Cage-Generation`.

We have applied our caging tool to a variety of 3D models listed in Table 1, producing cages that fulfill all the requirements listed in [3, 12] and reflect the extrinsic symmetries of models. Skeletons were computed using the implementation of [8] available in CGAL, as well as other automatic [6, 7] and interactive [9] techniques. Galleries of cages obtained with our method are shown in Figures 16 and 17, 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 two dragons. 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 11, are robustly handled.

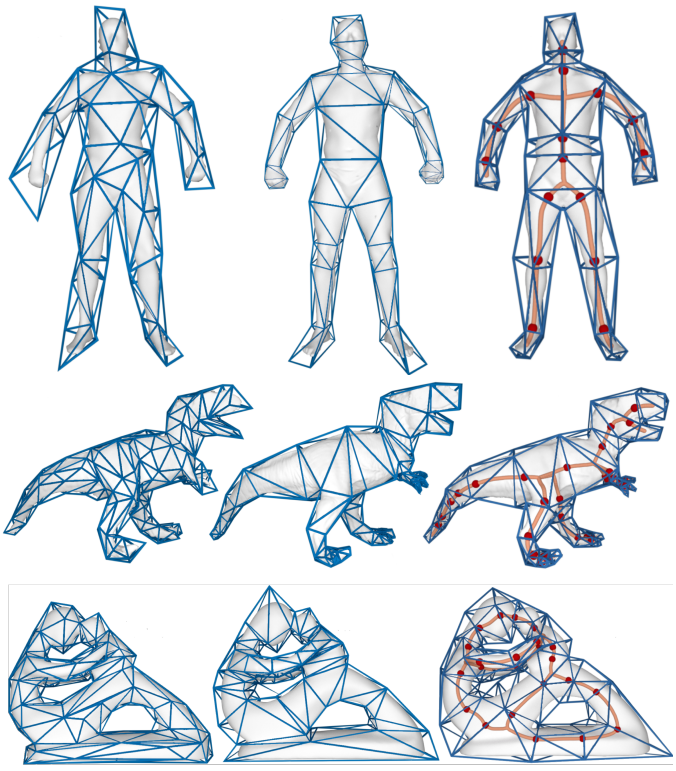

Fig. 14. Comparison between cages obtained with: aggressive mesh decimation followed by inflation via Nested Cages [2] (left column); interactive cutting planes [3] (middle column); our method (right column). Scape, Tyra and Fertility datasets.

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 12), 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 13).

We also validated our cages by posing digital characters using the CageLab tool [51]; some of the key poses we generated are depicted in Figure 19. See also the accompanying video.

Comparisons. In Figure 14 we compare our results with two state-of-the-art methods for caging, one automatic and one interactive. For automatic caging, we combined aggressive mesh decimation [19] with inflation with Nested Cages [2]. For interactive caging, we considered the recent method of Le and Deng [3]. Since we started from the cage meshes provided by [3], we generated our results and the results of the automatic approach so as to produce cages with similar complexity (i.e. same amount of vertices). To the best of our knowledge, there exists no benchmark to compare two alternative cages of the same character, or to evaluate a given character for specific animation tasks. Here we therefore compare the cages based on the principles devised by previous literature, and listed in Sec-

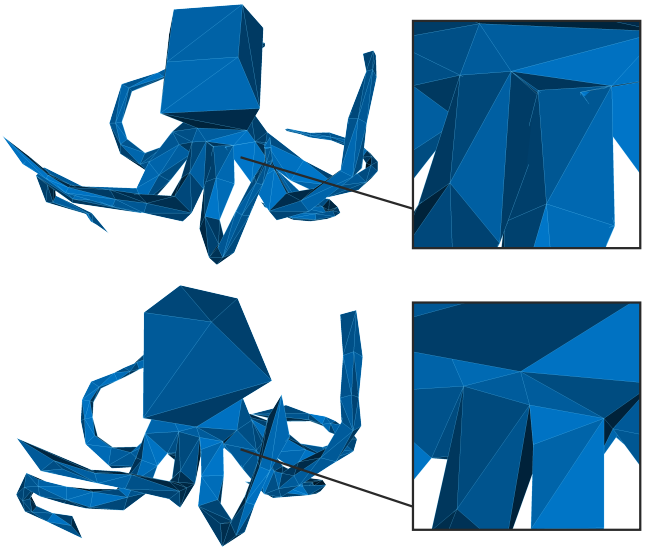

Fig. 15. Top: The caging method of [3] fails to generate a valid cage (see the self-intersection in the close-up). As discussed in their paper (Section 4), this problem stems from the use of unbounded planar cuts. Bottom: Our method produces a valid cage that contains the character and is free from self-intersections.

tion 1. As expected, being based on purely geometric criteria, fully automatic caging fails to preserve symmetry, it misses important bending points, and it produces a highly irregular cage. On the other hand, both interactive methods produced similar cages, with no perceivable defects.

Interaction-wise, we observe that our cutting system is able to robustly and reliably split the cage around a bending area, even if the user is not very precise in selecting the bending node (Figure 3). Conversely, the method of Le and Deng [3] is based on sketching a cutting plane by drawing a segment on a 2D GUI. Such an operation that is much more complex, requires a deeper 3D insight, and is extremely sensitive to the precision of the user. This type of interaction operation takes minutes, let alone further adjustments that may be necessary to solve intricate configurations. We argue that our interaction is faster: We only require the user to click on the skeleton to locate bending points, which can be done in a few seconds (Table 1, T_{ui}). Finally, as acknowledged in [3], the use of unbounded planar cuts may produce invalid cages containing self-intersections that are not easy to remove. We also observe that their heuristic for cage inflation is purely based on a distance function with respect to the character skin, and does not guarantee (and does not even promote) the absence of self-intersections in the cage. Therefore, these pathological configurations cannot be recovered. In Figure 15, we show a self-intersecting cage produced with [3], together with our (valid) result.

Figure 20 shows a comparison with respect to the bounding shapes produced by the recent automatic method presented in [21]. 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 [21], coarser cages are also shown, which were not released to 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 [21]. 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.

Resolution. Our method exposes a good independence from mesh resolution, both at a surface and volumetric level. Surface-wise, a good caging algorithm should focus on the high level appearance of a digital character, and not on low-level tessellation details. Ideally, the influence of the latter should be negligible. We demonstrate this property in Figure 21, where we consider two very different triangulations of the Tyra, obtaining two cages which are similar to one another. Volume-wise, we studied how much the volumetric discretization of the character influences the final result. As it can be noticed in Figure 22, the behaviour of the level sets and integral lines of the field remains consistent at different resolutions, making the algorithm substantially independent from the volumetric discretization. This is very important, because it means that the generation of the tetrahedral mesh is not a *hidden parameter* difficult to deal with, and that the algorithm can be reproduced by other developers.

Timing. Processing times for the various models used in our experiments are reported in Table 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 [52] – a relevant speedup can be achieved by substituting the original character with a proxy shape obtained via aggressive mesh decimation [19]. A cage built upon a low-res proxy containing just

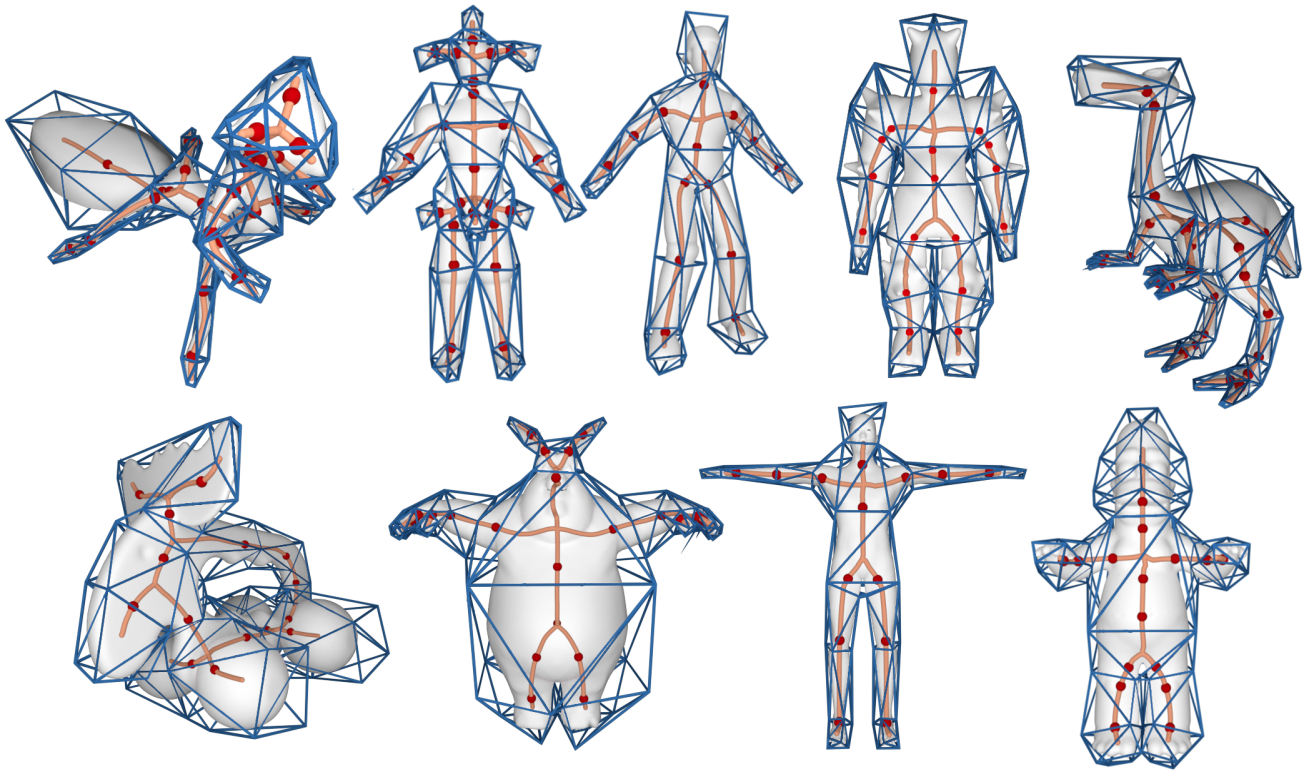

Fig. 16. A collection of cages for models with extrinsic symmetry: Antcat, Jocker, Skater, Warrior, Dinopet, Elk, BigBunny, ManTpose and Homer.

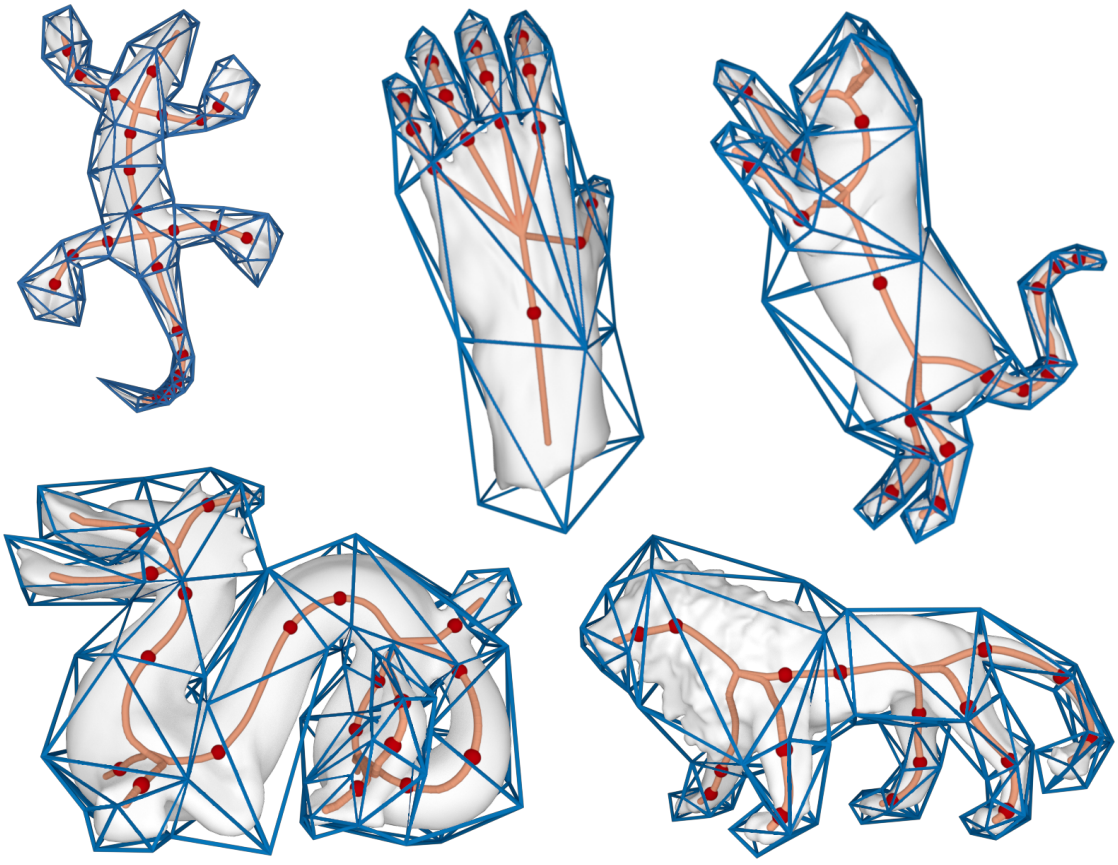

Fig. 17. A collection of models, which are not extrinsically symmetric: Gecko, Hand, Cat, Dragon, Lion

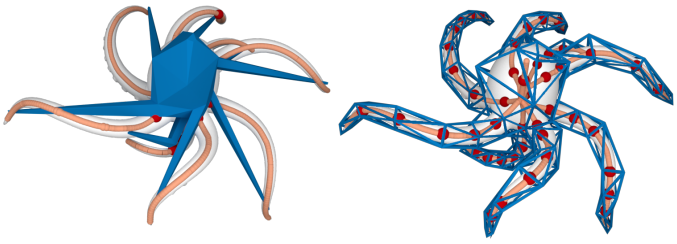

Fig. 18. Left: Our method fails to produce a valid cage if the number of selected bending nodes is not sufficient to accommodate the geometry of the character. Inflating such an unrefined cage to contain the whole octopus is in fact impossible, as cage self-intersections will arise. Right: The user can easily recover from such pathological cases by prescribing additional bending nodes, which in turn produce a more refined cage that tightly encloses the whole octopus.

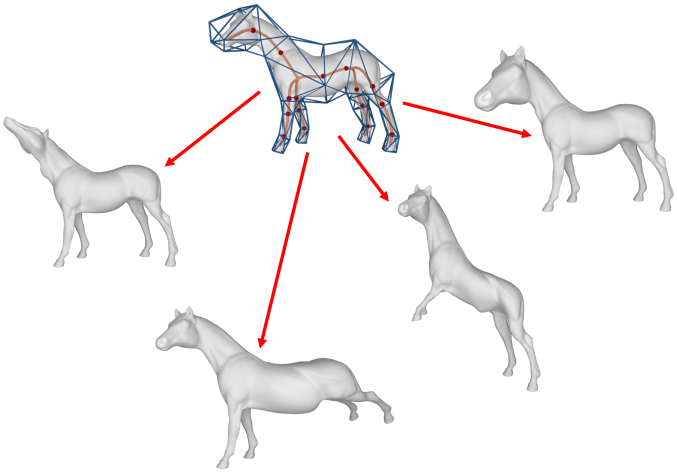

Fig. 19. Different poses of the Horse dataset obtained by editing a cage produced with our technique. For cage editing we used CageLab [51].

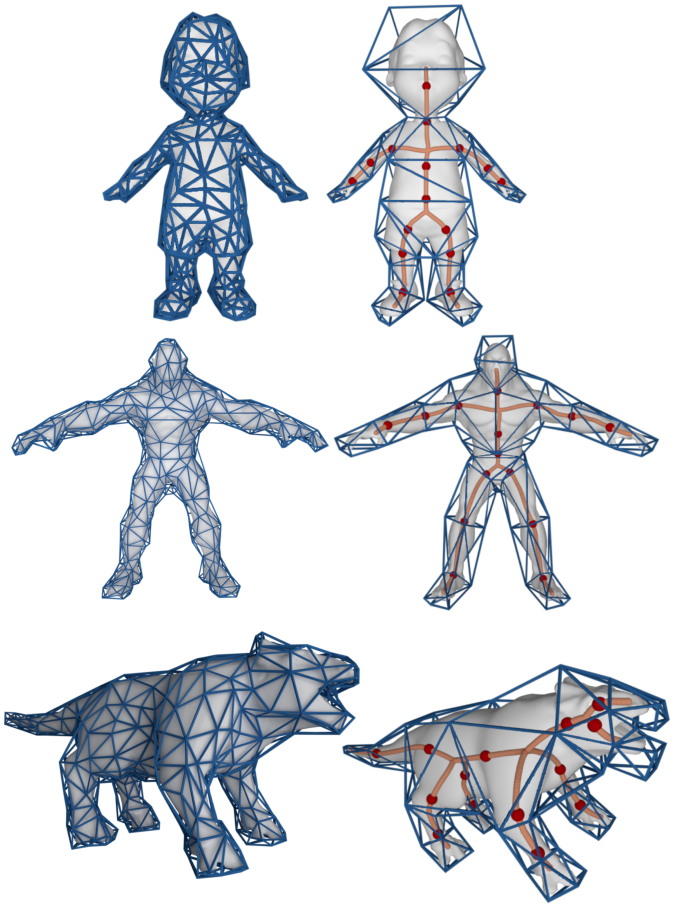

Fig. 20. Visual comparison between the state of the art caging method for collision detection [21] (left), and our method (right). General purpose cages are not suitable for animation, as they introduce unnecessary vertices, and do not align with the semantic features of the character.

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 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.

6. Limitations

Even though our method can produce 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 [53, 8]. 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 (Figure 18). 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 [2] 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. A

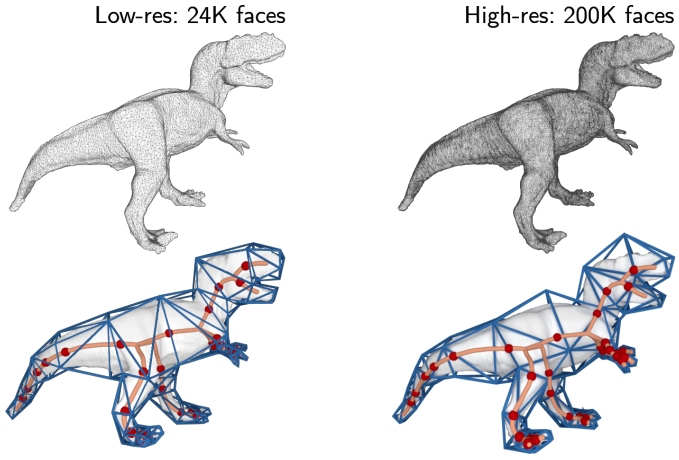

Fig. 21. Our method is substantially independent from the resolution of the input mesh. Here we show two cages, obtained by processing two alternative discretizations of the same character. Despite that the input meshes are very different in size (24K triangles vs 200K triangles), the output cages are quite similar.

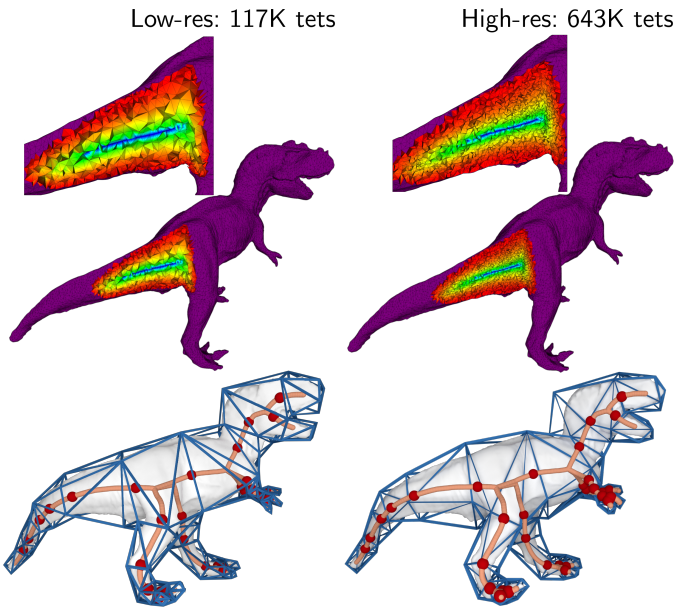

Fig. 22. The behaviour of the level sets (hence the integral lines) of the guiding harmonic field we use to propagate the cutting surfaces remains consistent at all resolutions. This makes our method substantially independent from the volumetric discretization we use to define the field. Here we show two alternative cages of the same character, obtained by processing the same input mesh, but using two different volumetric tessellations (117K tets vs 643K tets). The two cages are quite similar.

minor technical issue regards the generation of the guiding harmonic field, which requires a volumetric discretization of the interior of the character. While this may seem to suggest that surface meshes containing topological defects or open boundaries cannot be processed with our method, we point out that scientific literature offers a variety of methods for mesh repairing (e.g., MeshFix [54]) and robust tetmesh extraction (e.g., TetWild [55]). Therefore, having a watertight 2-manifold is not a strict requirement.

7. Conclusion and future work

We have presented a novel user-assisted method for building animation cages. 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 [3], we offer a faster, more intuitive, and more robust 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 a very recent paper [56], evidence is presented that mixed tri-quad cages can provide better control to animation and deformation. The extension of our method to support such cages is straightforward, as all cage elements along limbs and across the symmetry plane are naturally defined as quads.

References

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

[2] Sacht, L, Vouga, E, Jacobson, A. Nested cages. ACM Trans Graph 2015;34(6):170:1–170:14.

[3] Le, BH, Deng, Z. Interactive Cage Generation for Mesh Deformation. In: Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. I3D '17; New York, NY, USA: ACM; 2017, p. 3:1–3:9.

[4] Zhou, Y, Yin, K, Huang, H, Zhang, H, Gong, M, Cohen-Or, D. Generalized Cylinder Decomposition. ACM Trans Graph 2015;34(6):171:1–171:14.

[5] Tagliasacchi, A, Delame, T, Spagnuolo, M, Amenta, N, Telea, A. 3D Skeletons: A State-of-the-Art Report. Computer Graphics Forum 2016;35(2):573–597.

[6] Livesu, M, Scateni, R. Extracting Curve-Skeletons from Digital Shapes Using Occluding Contours. The Visual Computer 2013;29(9):907–916.

[7] Livesu, M, Guggeri, F, Scateni, R. Reconstructing the Curve-Skeletons of 3D Shapes Using the Visual Hull. IEEE Transactions on Visualization and Computer Graphics 2012;18(11):1891–1901.

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

[9] Barbieri, S, Meloni, P, Usai, F, Spano, LD, Scateni, R. An Interactive Editor for Curve-Skeletons: SkeletonLab. Computer & Graphics 2016;60:23–33.

[10] Panozzo, D, Lipman, Y, Puppo, E, Zorin, D. Fields on Symmetric Surfaces. ACM Trans Graph 2012;31(4):111:1–111:12.

[11] Sederberg, TW, Parry, SR. Free-form Deformation of Solid Geometric Models. SIGGRAPH Comput Graph 1986;20(4):151–160.

[12] Nieto, JR, Susín, A. "Cage Based Deformations: A Survey". In: González Hidalgo, M, Mir Torres, A, Varona Gómez, J, editors. Deformation Models: Tracking, Animation and Applications. Dordrecht: Springer Netherlands. ISBN 978-94-007-5446-1; 2013, p. 75–99.

[13] Xian, C, Li, G, Xiong, Y. Efficient and effective cage generation by region decomposition. Computer Animation and Virtual Worlds 2015;26(2):173–184.

[14] Xian, C, Lin, H, Gao, S. Automatic cage generation by improved OBBs for mesh deformation. The Visual Computer 2012;28(1):21–33.

[15] Deng, ZJ, Luo, XN, Miao, XP. Automatic Cage Building with Quadric Error Metrics. Journal of Computer Science and Technology 2011;26(3):538–547.

[16] Ben-Chen, M, Weber, O, Gotsman, C. Spatial Deformation sfer. In: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on

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	10	19552	39040	16	84	164	6.74	15	1.02	0.29	813.26	836	
Antcat	8;16	1550	3096	29	156	308	0.66	17.75	0.16	0.26	45.37	64	
Armadillo	1	10002	20000	17	100	196	2.92	13.2	0.75	0.33	135.21	152	
Asian Dragon	13	10000	19996	17	96	188	3.11	15.89	0.61	2.46	108.54	131	
Asian Dragon	13	10000	19996	51	348	692	3.45	50.32	1.43	1.17	169.21	226	
Beast	20	15122	30264	15	76	148	5.26	10	0.6	0.22	887.38	903	
BigBunny	16	21763	43522	14	108	212	10.2	12.22	2.43	1.1	541.5	567	
Boy	7;10;20	7502	15000	16	84	164	2.34	12.53	0.5	0.32	67.88	84	
Cat	17	27246	54488	19	108	212	11.73	13.11	1.55	0.5	297.76	325	
Dance	4;11	9971	19938	13	72	140	3.52	11.99	0.33	0.14	102.28	118	
Dinopet	16	4500	8996	27	156	308	1	20.1	0.33	0.34	70.6	92	
Dragon	17	5000	10000	22	124	284	1.47	14.58	0.35	0.2	109.7	126	
Elk	16	23114	46228	15	88	176	7.14	11.2	1.29	0.14	170.12	190	
Fertility	14	9994	20000	23	92	196	33.01	16.59	0.26	0.34	147	197	
Ganfaul	12	14590	29192	15	76	148	4.12	12.54	0.58	0.29	491.32	509	
Gecko (high)	17	37352	74700	24	120	236	12.55	14.54	1.96	0.68	840	870	
Gecko (low)	17	500	1000	24	120	236	0.15	16.45	0.06	0.16	23.37	40	130
Hand	17	14347	28690	15	84	164	5.2	13.5	0.7	0.2	197.27	217	
Homer	10	12997	25990	16	84	164	4.79	13.7	0.62	0.32	80.45	100	
Horse	10;19	19850	39696	15	76	148	8.67	9.55	0.8	0.29	175.49	195	
Joker	17	13328	26652	22	132	260	4.26	13.25	1.52	0.39	191.89	211	
Lion	17	27899	55794	17	92	180	10.18	15.2	1.22	0.4	152.06	179	
ManTpose	16	13356	26708	14	76	148	4.02	10.2	0.51	0.11	86.05	101	
Octopus	17	10002	20000	47	224	444	3.09	30.13	0.76	0.51	62.61	97	
Scape	5;10;14	6318	12632	15	80	156	1.19	12.6	0.2	0.11	62.94	77	
Skater	16	13332	26660	14	76	148	4.59	10.26	0.68	0.18	112.23	128	
Tyra (high)	14	100002	200000	33	184	364	54.99	22.1	17.51	1.65	886	982	
Tyra (low)	14	12501	24998	33	184	364	4.56	22.45	1.41	0.67	386.48	416	90
Warrior	16	9474	18944	17	88	172	3.02	14.21	0.33	0.15	72.12	90	
Warrok	12	23528	11746	19	96	188	3.11	18.25	0.43	0.26	666.74	689	

Table 1. Size of meshes and timing for the whole pipeline (user-assisted cages only): 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.

Computer Animation. SCA '09; New York, NY, USA: ACM; 2009, p. 67–74.

[17] 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. IEEE; 2009, p. 21–27.

[18] Sander, PV, Gu, X, Gortler, SJ, Hoppe, H, Snyder, J. Silhouette Clipping. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '00; New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.; 2000, p. 327–334.

[19] Garland, M, Heckbert, PS. Surface simplification using quadric error metrics. In: Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co.; 1997, p. 209–216.

[20] Kazhdan, M, Bolitho, M, Hoppe, H. Poisson Surface Reconstruction. In: Sheffer, A, Polthier, K, editors. Symposium on Geometry Processing. The Eurographics Association. ISBN 3-905673-24-X; 2006,.

[21] Calderon, S, Boubekeur, T. Bounding Proxies for Shape Approximation. ACM Trans Graph 2017;36(5):57:1–57:13.

[22] Yang, X, Chang, J, Southern, R, Zhang, JJ. Automatic cage construction for retargeted muscle fitting. The Visual Computer 2013;29(5):369–380.

[23] Ju, T, Zhou, QY, van de Panne, M, Cohen-Or, D, Neumann, U. Reusable skinning templates using cage-based deformations. ACM Trans Graph 2008;27(5):122:1–122:10.

[24] Chen, X, Feng, J. Adaptive skeleton-driven cages for mesh sequences. Computer Animation and Virtual Worlds 2014;25(3-4):445–453.

[25] Chen, X, Feng, J, Bechmann, D. Mesh Sequence Morphing. Computer Graphics Forum 2016;35(1):179–190.

[26] Thiery, JM, Tierny, J, Boubekeur, T. CageR: Cage-Based Reverse Engineering of Animated 3D Shapes. Computer Graphics Forum 2012;31(8):2303–2316.

[27] Chen, L, Huang, J, Sun, H, Bao, H. "Cage-based deformation transfer". Computers & Graphics 2010;34(2):107 – 118.

[28] Savoye, Y. Cage-based Performance Capture. In: SIGGRAPH ASIA 2016 Courses. SA '16; New York, NY, USA: ACM; 2016, p. 12:1–12:53.

[29] Kin-Chung Au, O, Tai, CL, Cohen-Or, D, Zheng, Y, Fu, H. Electors voting for fast automatic shape correspondence. Computer Graphics Forum 2010;29(2):645–654.

[30] Biasotti, S, Marini, S, Spagnuolo, M, Falcidieno, B. Sub-part correspondence by structural descriptors of 3D shapes. Computer-Aided Design 2006;38(9):1002–1019.

[31] Mortara, M, Patanè, G, Spagnuolo, M. From geometric to semantic human body models. Computers & Graphics 2006;30(2):185–196.

[32] Livesu, M, Attene, M, Patanè, G, Spagnuolo, M. Explicit Cylindrical Maps for General Tubular Shapes. Computer-Aided Design 2017;90:27 – 36. SI:SPM2017.

[33] Livesu, M, Muntoni, A, Puppo, E, Scateni, R. Skeleton-driven Adap-

tive Hexahedral Meshing of Tubular Shapes. Computer Graphics Forum 2016;35(7):237–246.
[34] 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 2015;35(1):6:1–6:13.
[35] Floater, MS. Mean value coordinates. Computer aided geometric design 2003;20(1):19–27.
[36] Ju, T, Schaefer, S, Warren, J. Mean Value Coordinates for Closed Triangular Meshes. ACM Trans Graph 2005;24(3):561–566.
[37] Lipman, Y, Levin, D, Cohen-Or, D. Green Coordinates. ACM Trans Graph 2008;27(3):78:1–78:10.
[38] Lipman, Y, Kopf, J, Cohen-Or, D, Levin, D. GPU-assisted Positive Mean Value Coordinates for Mesh Deformations. In: Geometry Processing. The Eurographics Association; 2007,.
[39] Joshi, P, Meyer, M, DeRose, T, Green, B, Sanocki, T. Harmonic coordinates for character articulation. ACM Trans Graph 2007;26(3):71:1–71:9.
[40] DeRose, T, Meyer, M. Harmonic coordinates. In: Pixar Technical Memo 06-02, Pixar Animation Studios. 2006,.
[41] García, FG, Paradinas, T, Coll, N, Patow, G. *Cages:: A Multilevel, Multi-cage-based System for Mesh Deformation. ACM Trans Graph 2013;32(3):24:1–24:13.
[42] Ben-Chen, M, Weber, O, Gotsman, C. Variational Harmonic Maps for Space Deformation. ACM Trans Graph 2009;28(3):34:1–34:11.
[43] Zhang, J, Deng, B, Liu, Z, Patanè, G, Bouaziz, S, Hormann, K, et al. Local Barycentric Coordinates. ACM Trans Graph 2014;33(6):188:1–188:12.
[44] Jacobson, A, Deng, Z, Kavan, L, Lewis, J. Skinning: Real-time Shape Deformation. In: ACM SIGGRAPH 2014 Courses. 2014,.
[45] Mancinelli, C, Livesu, M, Puppo, E. A Comparison of Methods for Gradient Field Estimation on Simplicial Meshes. Computers & Graphics 2019;.
[46] Shamir, A. A survey on mesh segmentation techniques. In: Computer graphics forum; vol. 27. Wiley Online Library; 2008, p. 1539–1556.
[47] Si, H. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. ACM s Math Softw 2015;41(2):11:1–11:36.
[48] 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. GIS '09; New York, NY, USA: ACM; 2009, p. 538–539.
[49] Guennebaud, G, Jacob, B, et al. Eigen v3. http://eigen.tuxfamily.org; 2010.
[50] Livesu, M. cinolib: a generic programming header only c++ library for processing polygonal and polyhedral meshes. Transactions on Computational Science 2019;Https://github.com/mlivesu/cinolib/.
[51] 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. The Eurographics Association. ISBN 978-3-03868-075-8; 2018,.
[52] Pixologic, . ZBrush. 2007. `http://pixologic.com`.
[53] Martin, T, Chen, G, Musuvathy, S, Cohen, E, Hansen, C. Generalized swept mid-structure for polygonal models. Comput Graph Forum 2012;31(2pt4):805–814.
[54] Attene, M. A lightweight approach to repairing digitized polygon meshes. The visual computer 2010;26(11):1393–1406.
[55] Hu, Y, Zhou, Q, Gao, X, Jacobson, A, Zorin, D, Panozzo, D. Tetrahedral meshing in the wild. ACM Transactions on Graphics (TOG) 2018;37(4):60.
[56] Thiery, JM, Memari, P, Boubekeur, T. Mean value coordinates for quad cages in 3D. ACM Trans Graph - Proc SIGGRAPH Asia 2018 ????;To appear.