Interactive Cage Generation for Mesh Deformation

Binh Huy Le *
Disney Research at Pittsburgh

Zhigang Deng †
University of Houston

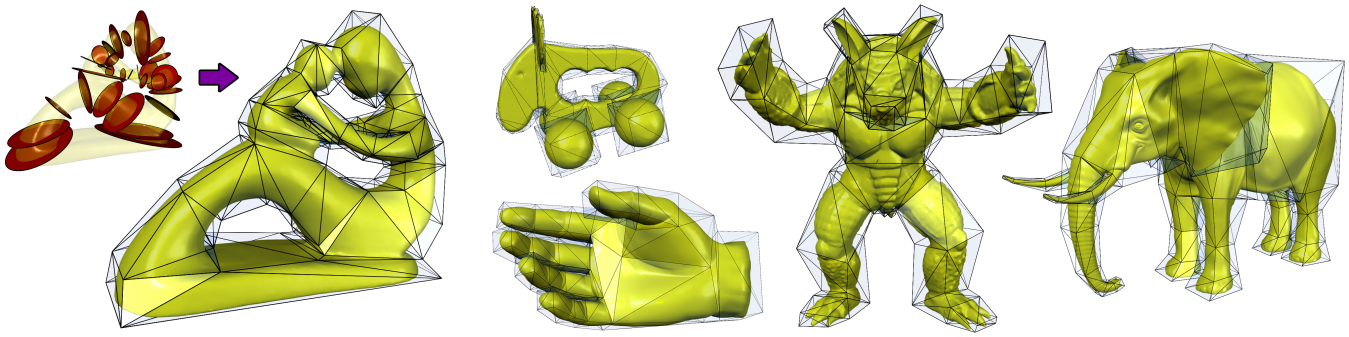


Figure 1: *Examples of cages generated by our approach. The leftmost image illustrates the user-specified cut slides for the Fertility model. The results by our method are comparable to artist-crafted cages in terms of quality.*

Abstract

Many previous efforts have been focused on generating optimal coordinates for cage deformation; cage generation for 3D models has been relatively understudied. We introduce an efficient complete pipeline to generate high quality cages for 3D models with arbitrary topological complexities, including high genus models and those without perceptible skeletal structures. Specifically, starting from user-specified cut slides, our method automatically optimizes the consistent, orthogonal orientations of cage cross sections. Then, through automated cage meshing and refining, it can further improve the cage quality by tackling the cage coverage issue and bounding the input model with a controllable tightness. Our experiments demonstrate this approach is efficient and robust to handle a variety of 3D models including human-like, animal, and high genus models.

Keywords: cage generation, cage-based deformation, and geometric deformation

Concepts: •**Computing methodologies** → *Mesh geometry models;*

1 Introduction

Among various geometric deformation methods, cage-based deformation has gained increasing popularity, due to its intuitiveness, simplicity, and performance. A cage is a low-resolution control mesh that closely bounds the high-resolution model. When the cage is manipulated, its deformation will be smoothly propagated to the enveloped model using pre-computed cage coordinates or weights [Ju et al. 2005; Joshi et al. 2007; Lipman et al. 2008]. Despite many advantages of cage-based deformation, currently cages in these previous works are constructed manually. Indeed, designing high-quality cages for any given 3D meshes is still a non-trivial under-studied research problem [Jacobson et al. 2014].

As pointed out by Jacobson *et al.* [2014], a high-quality cage for mesh deformation is expected to have the following desired qualities:

- It needs to be *low resolution*. A low-resolution cage with a small number of control vertices would help to reduce the amount of manual efforts in the mesh deformation process.
- It needs to *fully* and *often tightly bound* the enveloped model. The cage coordinates are well defined in the 3D space bounded by the cage, and often users do not want 3D geometry outside the cage to be deformed by it. Therefore, a tightly bounded cage would be desired in general [Nieto and Susín 2013].
- Its structure needs to *respect the topology* of the enveloped model. In this way, users can intuitively identify which parts of the cage to manipulate in order to achieve the desired deformation on the enveloped model.

Inspired by the above challenges, in this paper we propose an efficient and practical method to construct high quality cages for input meshes. Its core idea is to first obtain a set of cut slides, construct an initial cage via the automated orientation optimization of cage cross sections and cage meshing, and finally refine the cage by fixing the coverage issue and providing a controllable tightness. To this end, the resulting cages by our method have the aforementioned desired qualities. With our method, users can efficiently build high quality cages for various 3D models including high genus models and those without perceptible skeletal structures at an interactive rate on an off-the-shelf computer. Through many experiments, we demonstrate the efficiency and generality of our approach.

In this work we choose to focus on the design of an *interactive* pipeline for cage generation, instead a fully automatic solution, due to the following two main reasons. 1) It is difficult to design a

*e-mail:bbinh85@gmail.com
†email: zdeng4@uh.edu

I3D '17, February 25 - 27, 2017, San Francisco, CA, USA
ISBN: ACM 978-1-4503-4886-7/17/03$15.00
DOI: http://dx.doi.org/10.1145/3023368.3023369

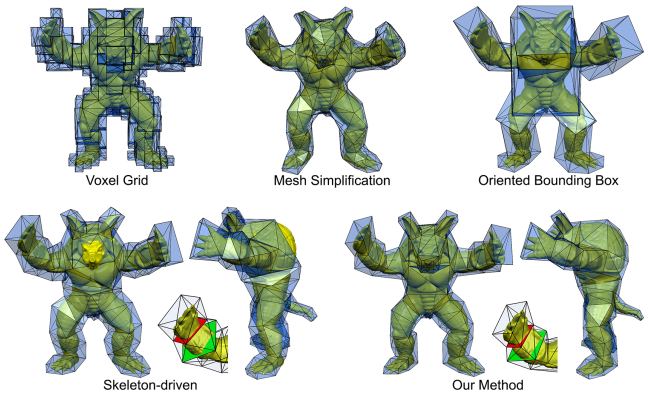


Figure 2: *Various cages are generated for the Armadillo model by our method and some previous methods, including voxel-grid based [Ben-Chen et al. 2009], mesh simplification based [Deng et al. 2011], oriented bounding box based [Xian et al. 2012], and skeleton-driven [Chen and Feng 2014].*

well-generalized, fully automatic approach to generate *high quality* cages for any complex models with arbitrary topologies, due to the varied complexities of 3D models. Therefore, instead, we focus on an efficient approach with affordable user interaction, even for novice users, to handle 3D models with a range of complexities, from common human-like models to those without perceptible skeletal structures, and to high genus models. 2) In practice, intuitive user interaction is a key element of most computer animation tools, since often the fully automated results may not meet the expectation perfectly. As a result, animators would still need user interaction to utilize their professionally trained skills, as well as to facilitate their artistic creativities, in order to improve the results. For a given 3D model, with our approach users typically need no more than several minutes to generate its high quality cage that is comparable to artist-crafted cages in terms of quality. Fig. 1 shows some example cages generated by our approach.

This work makes the following main technical contributions: (1) An efficient and complete pipeline is proposed to build high quality cages for a variety of 3D models, from human-like models to complex models including high genus models and those without perceptible skeletal structures. (2) An optimization algorithm is proposed to robustly solve the consistent and orthogonal orientations of cage cross sections in a global manner, which significantly outperforms previous methods that typically solve the same challenge in a heuristic way. (3) Novel cage meshing and refinement algorithms are proposed to allow users to intuitively control the tightness of the generated cage as well as to improve the quality of the cage structure.

2 Related Work

Cage based deformation. Researchers have proposed various methods to compute generalized barycentric coordinates with respect to the vertices of the bounding cage, including Mean Value Coordinates [Ju et al. 2005; Floater et al. 2005] and its variants [Li et al. 2013], Harmonic Coordinates [Joshi et al. 2007], and Green Coordinates [Lipman et al. 2008]. To improve the efficiency or controllability of cage-based deformation, researchers further developed novel methods to combine cages with different types of coordinates [García et al. 2013], and combine cages with bones and points via bounded bi-harmonic weights [Jacobson et al. 2011].

Cage generation. In recent years, a number of automatic or semi-automatic techniques have been proposed to generate cages for 3D models. These methods can be roughly categorized into four types (Fig. 2): template based, simplification based, bounding shape based, and skeleton-based, described below.

Template based cage generation methods [Yang et al. 2013; Ju et al. 2008] assemble a set of cage templates with pre-defined topologies to generate a closely bounded cage for a given 3D model. However, one common limitation of these approaches is to require a set of carefully pre-defined templates to cover various topologies; even so, they cannot be easily generalized to handle complex models since it is non-trivial to design suitable templates for complex joint areas beforehand.

Simplification based methods [Ben-Chen et al. 2009; Deng et al. 2011; Sacht et al. 2015] simplify the expanded version of the input model to produce a low-resolution cage. However, users are not able to intuitively control the topology of the resulting cage during this simplification process. Therefore, the vertex distribution of the resulting cage may not meet the users' requirement.

Bounding shape based approaches (e.g., [Xian et al. 2012]) build the cage by refining the bounding shape of the input model. It builds the cage for an input mesh by first generating oriented bounding box (OBB) for each mesh part and then registering the OBBs together. But it suffer from the following limitations: First, OBB is suitable for cylinder-like mesh parts; for joint areas where the deformation can be complex, the OBB is too rigid to provide the desired controllability. Second, the resulting cages cannot guarantee to fully bound the input model, because the partial cages (OBBs) are constructed independently, and the inconsistency among their orientations can cause potential twisting artifacts.

The *skeleton-based approach* [Chen and Feng 2014] first utilizes a user-sketched skeleton to guide the creation of many partial cages and then stitches them together. However, it solves the orientations of the partial cages in a heuristic manner, which is less robust and could cause noticeable twisting artifacts on the resulting cages for certain 3D models. Also, the cages by this method cannot guarantee to fully bound the input model (refer to the Armadillo model in Fig. 2). Furthermore, its applicability for handling complex models such as high genus models has not been demonstrated.

Efforts have also been done to construct cages from sequence data. For example, along the spirit of reverse engineering, researchers have proposed automatic methods to reconstruct a cage control sequence (i.e., cage parameters), given an animated mesh sequence [Thiery et al. 2012; Chen and Feng 2014] or a calibrated multi-view image sequence [Savoye and Franco 2010].

3 Method Overview

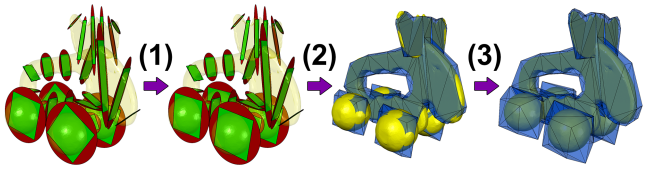


Figure 3: *Three main steps in our pipeline: (1) orientation optimization, (2) cage meshing, and (3) cage refinement. At the step (1), cut slides are illustrated as red ellipses, and the optimized cross sections are illustrated as green rectangles.*

The input of our method includes a given 3D model and a set of *cut slides* (§3.1). The input models considered in this work are only watertight manifolds, which can be represented as triangle

meshes. The cut slides are typically specified by users, holding the main frame for the cage. Each cut slide contains four automatically computed cage vertices that make a nearly rectangular *cage cross section* (or called *cross section* for brevity in this paper). In other words, a cross section is the intersection region of a cut slide and the cage, and all cross sections together divide the whole cage into parts.

For every change on the set of cut slides, our method recomputes a new cage at an interactive rate. In one computing cycle, our method performs three main steps as illustrated in Fig. 3. First, we find the optimal orientation of each cross section (green) so that it aligns well with the cut slides (red) as well as other neighboring cross sections to avoid cage twisting (§3.2). Then, we perform a meshing operation to generate the triangular mesh topology of the cage (§3.3). Finally, we refine the control vertices of the cage to ensure it fully and tightly bounds the input model (§3.4).

3.1 Cut Slides

We represent a cut slide k by an ellipse in 3D space as illustrated in Fig. 4. Each cut slide is determined by: a center $\mathbf{c}_k$ inside the input model, a unit normal vector $\vec{n}_k$ of the plane containing the ellipse, and two radii $\vec{u}_k$ and $\vec{v}_k$ such that $\vec{n}_k$ is in parallel with $\vec{u}_k \times \vec{v}_k$. The two radii represent the two principal directions of the cut slide, and they are later used to align the orientation of the cage cross section contained in k (§3.2).

The set of planes containing cut slides, called *cut slide planes*, segments the input model into *parts*, each of which is a 3D volume enclosed by the model and the cut slide planes. To obtain the parts, we first intersect each cut slide plane with the model and remove all the triangles on the intersection. Each intersection is a closed loop that encloses the center $\mathbf{c}_k$ on the cut slide plane. Then, we identify each part as a connected component in the remaining triangle-triangle graph (the dual graph of the mesh). To simplify the implementation, we disallow any two cut slides to intersect inside the model; this constraint is enforced by disallowing any pair of intersection loops to share the same triangle.

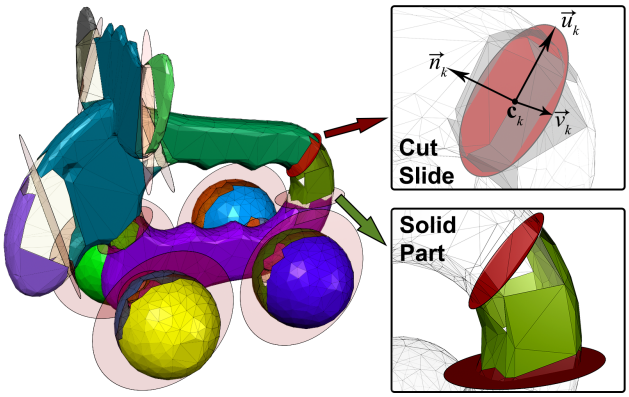


Figure 4: *Cut slides (red ellipses) segment the model into parts (color coded). Note that those triangles intersected with the cut slides are not considered to be a portion of any parts.*

We generate the cut slides from user-drawn strokes on the input model, and each cut slide is computed by fitting an ellipse to sample points on the strokes. Specifically, we first fit a plane to the sampled points. Then, we project these points to the plane and fit a 2D ellipse to the projections [Fitzgibbon et al. 1999]. Finally, the 2D ellipse is transformed back to 3D space.

Although automating the generation of cut slides could be possible (e.g., perform automated mesh segmentation [Shamir 2008] and then use the vertices on the segment boundaries for ellipse fitting), we did not investigate our approach in this direction, as the user interaction in our approach is very affordable and intuitive. Furthermore, it provides users full flexibility and controllability, especially for handling high genus models and other complex ones, e.g., the *fertility* model and the *elk* model (shown in Fig. 1 and Fig. 4) or other results shown in Fig. 17. Detailed discussion on various cut slide generation strategies can be found in §4.

3.2 Orientation Optimization

For each cut slide, its two radii are generally good local features to align its corresponding cage cross section. However, if the cut slide is sufficiently close to a circle, i.e. $|\vec{u}_k| \approx |\vec{v}_k|$, these two vectors are not robust enough for a good alignment, for example, the *armadillo* hand (shown at the bottom of Fig. 2) or the *armadillo* thigh (shown at the right side of Fig. 5). Thus, we need to find the optimal orientation of the cross section, that is, a rotated version of the two radii $\vec{u}_k, \vec{v}_k$ around the center $\mathbf{c}_k$.

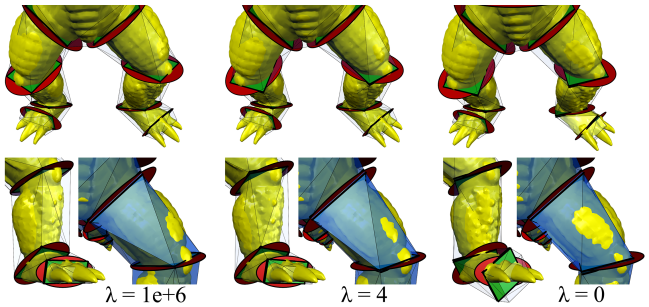
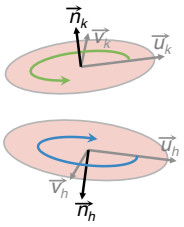


Figure 5: *The optimal cross section orientations with the alignment strength parameter λ decreasing from left to right.* **Left:** *Cross sections are completely aligned to the radii of the cut slides ($\lambda = 1e+6$); the cage at the thigh (blue) is twisted due to the orientation inconsistency between the cut slide at the knee and the cut slide near the hip.* **Right:** *The orientations of neighboring cross sections are smoothed out (no alignment, $\lambda = 0$); the cage has a good planar-facet structure but the cross section at the foot (green) does not align well with the model.* **Middle:** *$\lambda = 4$ makes a good balance between alignment and smoothness.*

We solve this problem by employing a technique similar to [Knöppel et al. 2013] to find a *smooth aligned directional field*, where the soft smoothness constraints between *neighboring cross sections* help to generate a robust solution that can avoid cage twisting (shown in the middle of Fig. 5). We say two cut slides or two cross sections are neighbors if they bound the same part. However, compared to [Knöppel et al. 2013], our problem is more general since our space is not limited to 2D manifold. If two neighboring cross sections have nearly opposite normal vectors, the local coordinates at these cross sections would need to be flipped over to establish a parallel transportation (refer to the inset figure). In this case, a direct application of the power of complex numbers [Knöppel et al. 2013] would make the orientations of these cross sections rotate in opposite directions.



Instead, we represent the orientation by 2D vectors and represent the rotation by a 2×2 matrix, as illustrated in Fig. 6. For each cut

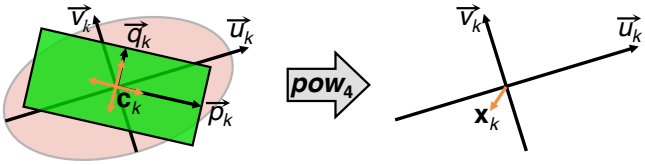


Figure 6: *In the local coordinate system of each cut slide, we take the 4th power of the orientations $\vec{p}_k, \vec{q}_k$ to make a unique representation $\mathbf{x}_k$, since p_k is a $\pi/2$ rotation of q_k. Refer to the main texts for the definition of the operator pow_4.*

slide k, we set up a 2D local coordinate system, where the origin is the centroid $\mathbf{c}_k$ and the two axes are the radii $\vec{u}_k$ and $\vec{v}_k$, respectively. In this coordinate system, we represent the optimal orientation of the rectangular cross section by two orthogonal vectors $\vec{p}_k, \vec{q}_k \in \mathbb{R}^2$ which are parallel to the edges of the cross section.

Let $pow_4(\vec{v})$ be an operator on $\vec{v}$ with the following four steps:

1. Normalize $\vec{v}$ to have a unit norm;
2. Convert the unit norm vector to a complex number in a component-wise way;
3. Take the 4$^{\text{th}}$ power of the complex number;
4. and convert the result from step 3 to a vector form in a component-wise way.

Since p_k is essentially a $\pi/2$ rotation of q_k, we only need to work on $\mathbf{x}_k$, the 4$^{\text{th}}$ power of $\vec{p}_k, \vec{q}_k$ (depicted as the orange vectors in Fig. 6), where:

$$\mathbf{x}_k = pow_4(\vec{p}_k) = pow_4(\vec{q}_k) = pow_4(-\vec{p}_k) = pow_4(-\vec{q}_k)$$

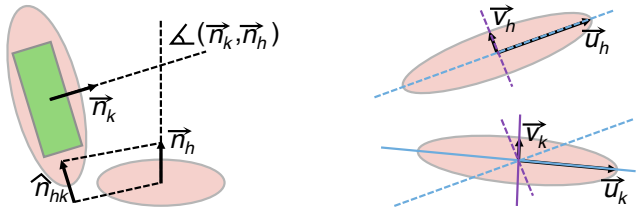


Figure 7: *Two cases of the smoothness constraint: (***left***) two cut slides are almost perpendicular to each other, and (***right***) two cut slides are almost parallel to each other. Refer to the main texts for the calculation in both cases.*

To the end, we formulate the orientation optimization problem as the minimization of an energy function, E, described in Eq. (1). It is defined as the sum of the smoothness energy, ${E^S}_{kh}$, and the alignment energy, ${E^A}_k$. The smoothness energy (Eq. (2)) measures the orientation consistency between two neighboring cut slides, and the alignment energy of each cut slide (Eq. (5)) measures the matchness between the orientation of its corresponding cross section and its radius vector.

$$E = \sum_{\forall \text{neighbors } k,h} {E^S}_{kh} + \sum_{\forall k} {E^A}_k \tag{1}$$

In the above equation, E is a linear quadratic function, which can be minimized by solving for the zero gradient. Then, we transform the optimal solution of all $\mathbf{x}_k$ back to the form of $\vec{p}_k, \vec{q}_k$ (Fig. 6), by converting $\mathbf{x}_k$ to a complex number and then solving its 4$^{\text{th}}$ root.

Finally, we rescale both $\vec{p}_k$ and $\vec{q}_k$ by only fitting the two radii of the cut slide ellipse to its corresponding cage cross section.

Smoothness energy. The smoothness energy between two neighboring cross sections h and k is defined as follows:

$${E^S}_{kh} = \begin{cases} w_{kh} \left| \mathbf{x}_k - pow_4(\hat{\mathbf{n}}_{hk}) \right|^2 & \text{if } \angle(\vec{n}_k, \vec{n}_h) \in [\frac{\pi}{4}, \frac{3\pi}{4}], \\ w_{kh} \left| {\mathbf{R}_{kh}}^4 \mathbf{F}_{kh} \mathbf{x}_k - \mathbf{x}_h \right|^2 & \text{otherwise.} \end{cases} \tag{2}$$

In the above Eq. (2), the following two cases are considered:

- If h and k are almost perpendicular to each other (illustrated in the left panel of Fig. 7), i.e. the angle between two normal vectors $\angle(\vec{n}_k, \vec{n}_h)$ is in the range of $[\frac{\pi}{4}, \frac{3\pi}{4}]$, we match the orientation of k with that of h. Specifically, by projecting $\vec{n}_h$ to the plane of k, we obtain its projection $\hat{\mathbf{n}}_{hk}$. Then, we take the $pow_4(\hat{\mathbf{n}}_{hk})$.
- If h and k are almost parallel to each other (illustrated in the right panel of Fig. 7), we represent the rotational difference between h and k by the matrix $\mathbf{R}_{kh}\mathbf{F}_{kh}$. The rotation matrix $\mathbf{R}_{kh} \in \mathbb{R}^{2\times 2}$ represents the transformation between the local coordinates of k and h. We compute $\mathbf{R}_{kh}$ by projecting the two lines containing $\vec{u}_h$ and $\vec{v}_h$ to the plane of k (blue and purple dashed lines in the right of Fig. 7), and then find the best rotation matrix to rotate these lines to the two lines containing $\vec{u}_k$ and $\vec{v}_k$ (blue and purple solid lines in the right of Fig. 7). Note that the two projection lines might not be perpendicular to each other; thus, we compute the best rotation matrix by performing Singular Value Decomposition (SVD). The flipping matrix $\mathbf{F}_{kh} \in \mathbb{R}^{2\times 2}$ is computed in Eq. (3), and right multiplying it with $\mathbf{R}_{kh}$ can change the rotation direction. At this step, the 4$^{\text{th}}$ power of the combined transformation, ${\mathbf{R}_{kh}}^4\mathbf{F}_{kh}$, is only performed on the rotation part, i.e., taking ${\mathbf{R}_{kh}}^4$, since ${\mathbf{F}_{kh}}^4 = \mathbf{I}$ cancels out the flipping (the combination of 4 flips makes an identity transformation).

$$\mathbf{F}_{kh} = \begin{bmatrix} 1 & 0 \\ 0 & sign(\vec{n}_k \cdot \vec{n}_h) \end{bmatrix} \tag{3}$$
$$\text{where: } sign(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{otherwise.} \end{cases}$$

The weight w_{kh} controls the strength of each smoothness constraint, which is computed in Eq. (4). w_{kh} favors cut slides with large cross section areas $s_k + s_h$, a small area difference $|s_k - s_h|$, and a small centroid-to-centroid distance $(\mathbf{c}_k - \mathbf{c}_h)^2$.

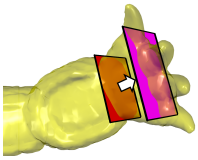
$$w_{kh} = \frac{s_k + s_h - |s_k - s_h|}{(\mathbf{c}_k - \mathbf{c}_h)^2} \tag{4a}$$
$$\text{where: } s_k = |\vec{u}_k|^2 + |\vec{v}_k|^2 \tag{4b}$$
$$s_h = |\vec{u}_h|^2 + |\vec{v}_h|^2 \tag{4c}$$

Alignment energy. For each cut slide k, we compute its alignment energy in Eq. (5), which matches the orientation of its corresponding cross section with the radius vector $\vec{u}_k$. In the local coordinate system, this is equivalent to matching $\mathbf{x}_k$ to the vector $pow_4([1\ 0]^{\mathsf{T}}) = [1\ 0]^{\mathsf{T}}$. The alignment weight w_k favors the magnitude difference between $\vec{u}_k$ and $\vec{v}_k$, i.e., if the cut slide ellipse was less similar to a circular shape, the alignment is stronger; and vice versa. The parameter λ controls the strength of the alignment constraint, or called the trade off between alignment and smoothness. Fig. 5 depicts the effect with different λ values. We choose $\lambda = 4$ in our experiments in this work.

$$E^A{}_k = w_k \left| \mathbf{x}_k - [1\ 0]^\mathsf{T} \right|^2 \tag{5a}$$

$$\text{where: } w_k = \lambda \frac{\left(|\vec{u}_k| - |\vec{v}_k|\right)^2}{|\vec{u}_k|^2 + |\vec{v}_k|^2} \tag{5b}$$

3.3 Cage Meshing



Before performing cage meshing, we add one *rectangular cap* for each leaf part (i.e., bounded by only one cage cross section), for example, the hands or feet of the *armadillo* model. The cap is simply generated by pushing the (only) rectangular cross section towards the centroid of the part and rescaling the rectangle to fit with the model (refer to the inset figure). Note that the four vertices of the rectangular cross section are automatically generated from its corresponding cut slide (ellipse), as described in §3.1.

Then, we perform meshing on each part as follows: First, we project the corners and edges of all of its cross sections to a sphere, whose center is the geometric center of all of its cross sections (defined as the centroid of all of its cross sections). Then, we compute the 3D Delaunay triangulation of the projected corners and edges. Finally, we perform edge flipping to further optimize the triangulation.

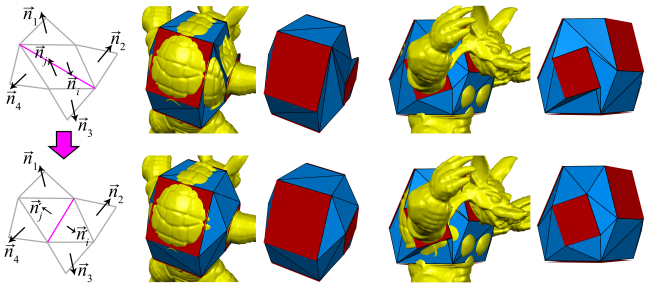


Figure 8: *The cage corresponding to a part before edge flipping (top) and after edge flipping (bottom).*

We flip an edge if flipping it would make the cage smoother. We elaborate this flipping condition as minimizing the difference among the normals of neighboring cage faces. Specifically, as illustrated in Fig. 8, we flip the purple edge if:

$$\measuredangle(\vec{n}_i, \vec{n}_1) + \measuredangle(\vec{n}_i, \vec{n}_2) + \measuredangle(\vec{n}_j, \vec{n}_3) + \measuredangle(\vec{n}_j, \vec{n}_4) + \measuredangle(\vec{n}_i, \vec{n}_j) > \measuredangle(\vec{n}'_i, \vec{n}_2) + \measuredangle(\vec{n}'_i, \vec{n}_3) + \measuredangle(\vec{n}'_j, \vec{n}_1) + \measuredangle(\vec{n}'_j, \vec{n}_4) + \measuredangle(\vec{n}'_i, \vec{n}'_j)$$

where $\vec{n}_x$ denotes the normal vector of a triangle x, and $\measuredangle(\cdot,\cdot)$ denotes the angle between two vectors. Note that a triangle x may also belong to a cross section since we process parts individually (shown in red color in Fig. 8). In this case, we consider the outward normal of the cross section as the normal of the triangle x. Also, since the cross sections establish a frame for the cage, we never flip edges on them.

3.4 Cage Refinement

After performing the above cage meshing, we refine cage vertices to make the cage fully bound the input model. Our cage refinement (illustrated in Fig. 9) pushes the cage vertices outward while keeping a minimal change on the normals of all the cage triangles, i.e., maximally preserving the orientations and ratios of the cross sections produced in §3.2. We can also control the distance between the surface of the cage and the input model (i.e., the tightness of the cage) by adjusting a parameter Δ. We choose $\Delta = 0$ in most of our experiments in this work, i.e., the generated cages tightly bound the models; however, we can also easily relax the cage tightness via Δ, as illustrated in Fig. 10.

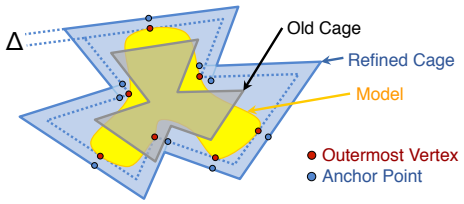


Figure 9: *The cage is refined by pushing its vertices outward to bound the input model at a distance of Δ, while keeping a minimal change on the normals of all the cage triangles.*

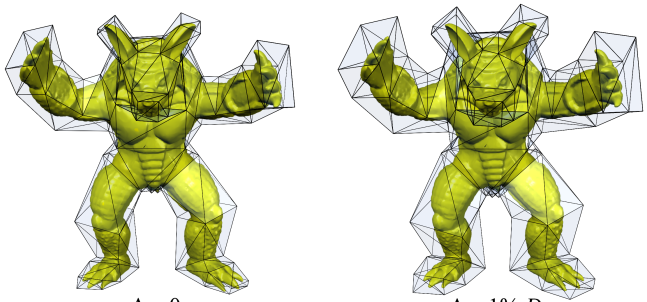


Figure 10: ***Left:*** *a tightly bounded cage.* ***Right:*** *a loosely bounded cage. Here D_{BB} denotes the diagonal length of the bounding box of the input model.*

To refine the cage, we first find the *outermost model vertex*, $\widehat{i}(t)$, for each cage triangle t (Eq. (6)). The outermost model vertex is the one with the largest *signed distance* to t, that is, in Eq. (6), $d(\mathbf{v}_i, t) = (\mathbf{v}_i - \mathbf{g}_t) \cdot \mathbf{n}_t$, where $\mathbf{g}_t$ denotes the barycenter of t and $\mathbf{n}_t$ denotes the normal vector of t. Intuitively, if the signed distance $d(\mathbf{v}_i, t)$ is non-negative, vertex i is outside the cage and we should push out t to cover i. Since the cut slides, as well as the corresponding cross sections, divide the model into the same set of parts, we only need to find the outermost vertex that belongs to the same part P with t, that is, P is the part such that $t \in P$. This makes our algorithm fast and robust.

$$\widehat{i}(t) = \arg\max_{i \in P} d(\mathbf{v}_i, t) \tag{6}$$

Assume the refined version of the cage triangle t is t'. From the outermost vertex $\widehat{i}(t)$, we then compute an *anchor point* $\mathbf{a}_{t'}$ for each cage triangle t (Eq. (7)). Intuitively, we want $\mathbf{a}_{t'}$ to be on the same plane with t' that tightly bounds the model. Here, we consider the following two cases:

- In Eq. (7a): if the outermost vertex $\widehat{i}(t)$ is outside the cage, or almost outside the cage within a tolerance of $-\Delta$, we push out $\widehat{i}(t)$ at a distance of Δ to make the anchor point.
- In Eq. (7b): Otherwise (that is, the outermost vertex $\widehat{i}(t)$ is inside the cage), we simply let the barycenter of the cage triangle t, $\mathbf{g}_t$, be the anchor point.

$$\mathbf{a}_{t'} = \begin{cases} \mathbf{v}_{\widehat{i}(t)} + \Delta \mathbf{n}_t & \text{if } d(\mathbf{v}_{\widehat{i}(t)}, t) > -\Delta, \quad (7a) \\ \mathbf{g}_t & \text{otherwise.} \quad (7b) \end{cases}$$

Since t' keeps the normal vector $\mathbf{n}_t$ and contains $\mathbf{a}_{t'}$, the equation of plane containing t' is $(\mathbf{x} - \mathbf{a}_{t'}) \cdot \mathbf{n}_t = 0$. Let $F(j)$ be the set of all the cage triangles adjacent to the cage vertex j. We compute the refined cage vertex j by solving the linear least squares in Eq. (8), where $\mathbf{u}_j^{(k)}$ denotes the position of the current cage vertex j, and $\mathbf{u}_j^{(k+1)}$ denotes the position of the refined cage vertex j. As some cage vertices j have more than 3 adjacent cage triangles, i.e., $|F(j)| > 3$, the refined cage vertices computed by this least squares solution might change the normals of some cage triangles. We minimize these changes by weighting the error of each cage vertex j by the inverse of its squared distance to the anchor point $\mathbf{a}_{t'}$, i.e., $\frac{1}{|\mathbf{u}_j^{(k)} - \mathbf{a}_{t'}|^2}$.

$$\mathbf{u}_j^{(k+1)} = \arg\min_{\mathbf{x}} \sum_{t \in F(j)} \frac{1}{|\mathbf{u}_j^{(k)} - \mathbf{a}_{t'}|^2} \big((\mathbf{x} - \mathbf{a}_{t'}) \cdot \mathbf{n}_t\big)^2 \quad (8)$$

The above least squares fitting of cage vertices might not keep all the anchor points to be exactly on the refined cage triangles. As a result, the refined cage cannot guarantee to bound the model. However, as the signed distances of the outermost model vertices are decreased, we can repeat the above refinement process, including finding new anchor points and solving the least squares in Eq. (8), until a maximal number of iterations are reached or the maximal signed distance of all the outermost vertices reaches a threshold. Specifically, in most of our experiments in this work, the maximum number of iterations is set to 30, and the termination criterion is the difference between the negative of the maximal signed distance and Δ is no more than 0.1% of D_{BB} (the diagonal length of the bounding box of the model), as described in Eq. (9). Note that when the cage is getting loosened, the maximal signed distance, $\max_{\mathbf{v}_i \in P, t \in P} d(\mathbf{v}_i, t)$, could become a negative value.

$$|-\max_{\mathbf{v}_i \in P, t \in P} d(\mathbf{v}_i, t) - \Delta| \leq 10^{-3} D_{BB} \quad (9)$$

In Fig. 11, we show an example of how the maximal signed distance and cage coverage are changed over all iterations until convergence. In this example, the fast decreasing of the maximal signed distance demonstrates the good convergence rate of our method. Note that for an incomplete cage, e.g., when users only make an insufficient number of cut slides, our iterative refinement process might not converge. For this reason, we only run 10 iterations of cage refinement while making cut slides, and then, we run the full refinement process at the end to produce the final result.

4 Results and Discussion

Cut slide generation strategies. We use three common strategies to make cut slides as shown in Fig. 12:

1. **Cross section cut.** This is the most common strategy where the cut slide is a cross section of a branch of the model, i.e., it is perpendicular to the medial axis of the model. In general, cross section cuts are put at a local minimum (e.g., near a joint) or a local maximum (e.g., near a muscle bulge). For most of articulated character models, where the body and limbs can be clearly seen, we only need to make cross section cuts as shown in Fig. 13. Most of the cut slides for human-like models (Fig. 14) are also cross section cuts.

2. **Sharp corner cut.** This type of cut can be placed at the sharp turn of the medial axis, such as the heels of the Armadillo model or the corners of the Fertility model's base (Fig. 1). In general, a sharp corner cut makes a diagonal cross section on the cage, which connects two nearly perpendicular branches of the model, e.g., the foot and the leg.

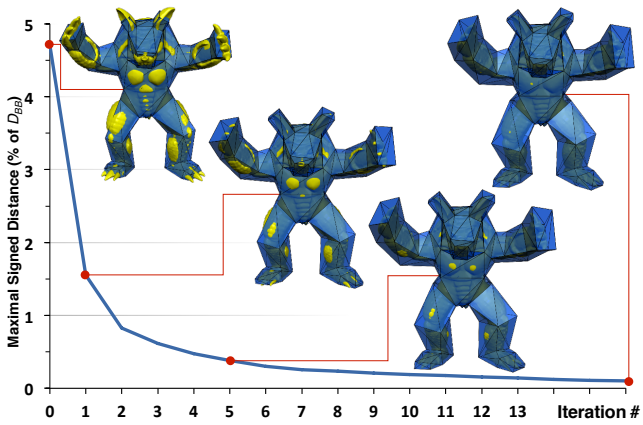


Figure 11: *The convergence of our iterative refinement for a cage that tightly bounds the Armadillo model ($\Delta = 0$). The maximal signed distance is decreased to $0.00099 \times D_{BB}$ after 16 iterations. D_{BB} denotes the diagonal length of the bounding box of the model.*

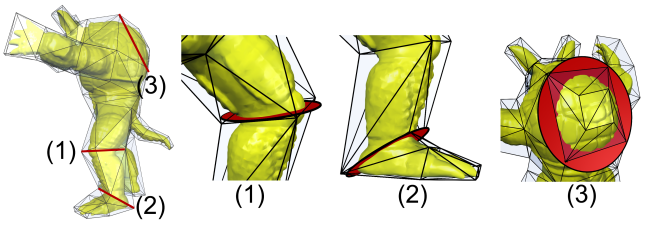


Figure 12: *Three common strategies of making cut slides: cross section cut (1), sharp corner cut (2), and bump cut (3).*

3. **Bump cut.** This type of cut is used to cover a bump on the model, such as the back of the Armadillo model. Unlike the two previous cut slide types, a bump cut does not rely on the medial axis at the bump (there is no medial axis or the medial axis is intangible). For this reason, bump cut slides are generally made by visual judgments. In Fig. 15, we show examples where bump cuts are important to generate reasonable cages.

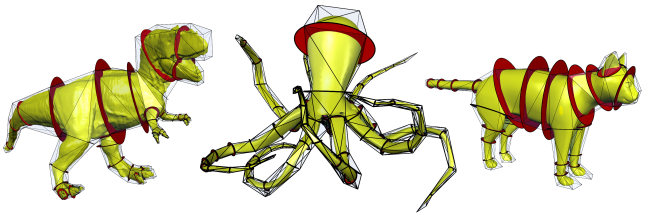


Figure 13: *Cages for different articulated characters can be generated by making cross section cuts perpendicular to the medial axis of the model. The cage for the Tyrannosaurus model contains details for fingers and toes. The cage for the Octopus model has a large number of branches (its head and 8 arms).*

Flexibility. The combination of the above cut slide strategies with user interaction allows us to generate high quality cages for a variety of 3D models, as shown in all of our results (Figures 1, 13 to 15 and 17). Note that even for a small class of models such as human-like models, the design of their cages may vary significantly as shown in Fig. 14. This challenge makes template-based cage generation methods [Ju et al. 2008; Yang et al. 2013] fall short easily. In practice, it is also non-obvious to generate similar results

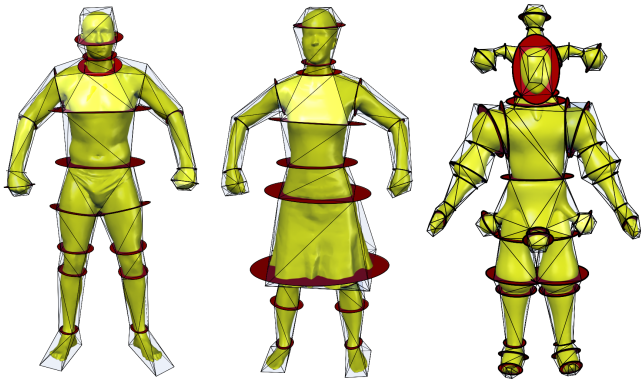


Figure 14: *Human-like models might have very different structures. Our interactive method offers flexible customizations for cage generation such as the skirt (middle) or the balls on the costume of the clown (right).*

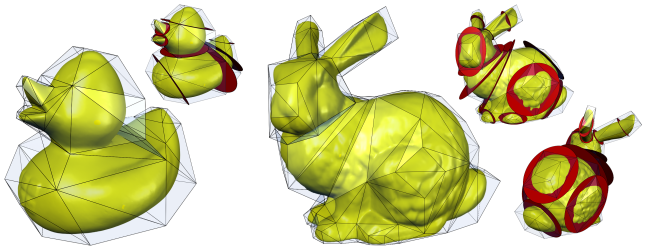


Figure 15: *Examples of bump cut slides: A cut to cover the breast of the Duck model and many cuts to cover the face, the tail, the back, two legs, and four feet of the Bunny model. Without making these bump cut slides, the generated cages could not reasonably cover the input models.*

with fully automatic methods, since non-trivial parameters tuning efforts often require visual feedback, and a set of global parameters to fit all the parts of the model may not exist. For example, we need to set up different resolutions to generate cage details for body and fingers of the Tyrannosaurus model (Fig. 13), including using different grid resolutions [Ben-Chen et al. 2009], different mesh resolutions [Deng et al. 2011], different sizes of bounding boxes [Xian et al. 2012], or different cross section sampling rates on the skeleton [Chen and Feng 2014].

Based on the cages generated by our approach, animators can conveniently incorporate them into deformation editing in various animation software packages (e.g., Autodesk Maya) and then efficiently sculpt various poses for the given 3D models. Fig. 16 shows Maya snapshots of a cage-based deformation application after the cages, generated by our approach, are incorporated into the Maya software via an in-house developed plug-in. In this cage-based deformation application, we used the Bounded Biharmonic Weights [Jacobson et al. 2011] to generate deformation coordinates, and computed the rotation of cage vertices by minimizing the As-rigid-as-possible energy [Sorkine and Alexa 2007].

High genus models. Since our orientation optimization step (§3.2) can handle the arbitrary topological connection of parts, we can naturally generate cages for high genus models, such as the Fertility model in Fig. 1, or more challenging models as shown in Fig. 17. Utilizing user interaction allows the preservation of the full topology of the model (e.g., the Children model), or allows the removal of small topological features or noise on the model (e.g., small holes on the Buddha model).

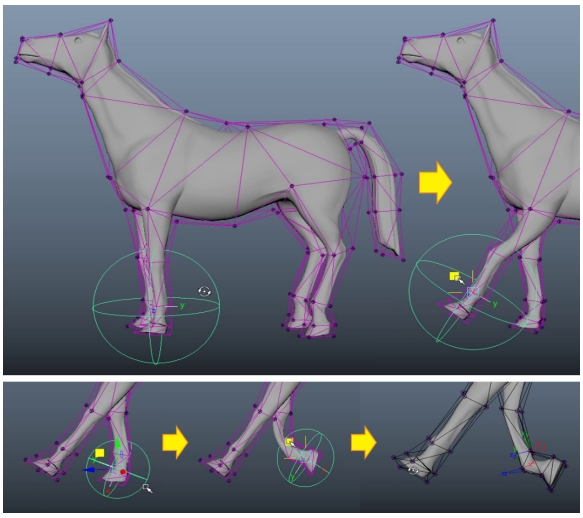


Figure 16: *Snapshots of a cage-based deformation application in Autodesk Maya, where the cages generated by our approach are incorporated into Maya via an in-house developed plug-in.*

Comparison with the nested cage method. The nested cage method [Sacht et al. 2015], considered as the state of the art in cage generation, focuses on a fully automatic solution, while the emphasis of our method is *interactivity* and *intuitive user control*. Therefore, they are complementary to each other; it is not fair to do direct comparison between them. However, as pointed out in [Sacht et al. 2015], for certain complex 3D models, with the nested cage method a coarse cage could collide with itself during inflation, which may create a pinch to cause the algorithm to fail. By contrast, our approach can provide a practical solution to avoiding such issues, since our method allows users to intuitively add or adjust cut slides to influence the resulting cage.

It is noteworthy that the cage expansion mechanism used in the nested cage method [Sacht et al. 2015] can be potentially used as an alternative to the cage refinement step (§3.4) in our approach. In addition, the sketch-based interface for quad-meshing [Takayama et al. 2013] could provide a potentially interesting alternative for user interaction in our approach: instead of specifying cut slides, the users can also interactively specify curve networks on the mesh [Takayama et al. 2013] to guide the generation of the initial cage.

Comparison with manual cage generation. We invited an animator with intermediate modeling experience to manually build a cage for the Armadillo model, which has a similar structure to the cage generated by our approach. Despite the simplicity of the input model, the manual cage building took about 2 hours for him to craft a cage with 128 vertices and 252 triangles. In particular, the majority of his time was spent on vertex positions adjustment, edges splitting, and faces extruding in Maya. In contrast, using our approach, it only took about 2 minutes for him to make 23 cuts slides, most of which were generated by only one stroke without requiring any adjustment. As the design of the cage was given beforehand, trial-and-error was not needed in both tasks. However, it is noteworthy that, if trial-and-error is needed, the full manual method would need much more trial-and-error time than our approach.

Runtime performance. Since most of the steps in our pipeline work on the low resolution cage, instead of the high resolution input model, the runtime and memory footprint of our approach are small. Our orientation optimization (§3.2), cage meshing (§3.3), and cage vertices fitting can achieve linear time and memory ef-

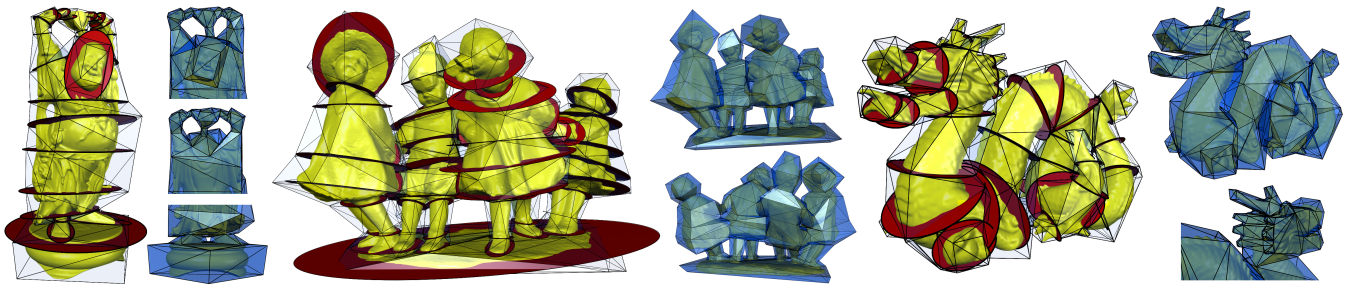


Figure 17: *Our method can handle high genus models (from left to right: Buddha, Children, and Dragon), thanks to its orientation optimization (§3.2) that can work on the arbitrary topological connection of parts. Utilizing user interaction helps to ignore small holes on the Buddha model. Note that all the cages in this figure tightly bound the models (Δ is in the range between 0 and 0.001), and certain rendering artifacts on the cages are mainly caused by the unavoidable numerical error during the computing process.*

ficiencies with respect to the number of cut slides (or the number of cage vertices/triangles). The only step that consumes significant computing time is to find the outermost model vertices (Eq. (6) in §3.4), and its runtime is proportional to the resolution of the input model and the number of cut slides. Among our experiments in this work, the most complex result is the Children model (Fig. 17), which includes 724,742 vertices, 1,449,512 triangles, and 53 cut slides in the final cage. For this model, the orientation optimization and cage meshing steps took 136 ms and the cage refinement step took 1,590 ms for 10 iterations. Its total memory footprint was only 950 MB with the use of double precision floats. The experiment ran on a 4-core Intel i7 computer with our C++ implementation using the *Eigen* library [Guennebaud et al. 2015] for numerical computing tasks and the *libigl* library [Jacobson et al. 2015] for geometry processing tasks. We also used OpenMP to find the outermost model vertices in parallel.

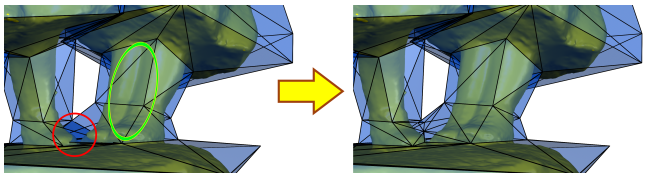


Figure 18: *Limitations of unbounded, planar cut slides. (left) The planar cut slides cannot separate some parts, e.g. two legs (circled in green), self-intersection between different parts of the cage (circled in red). The right shows the corrected cage.*

Limitations. The main limitation of our approach is that it only uses unbounded, planar cut slides. This type of cut slides may limit the separability of the model and could initially create a self-intersecting cage at certain unusual scenarios. As shown in the left of Fig. 18, for the complex Children model, with unbounded cut slides, we cannot make one cut slide perpendicular to the legs and another one parallel to the legs (to split two legs) while preventing these cut slides from intersecting each other. However, even if such a self-intersection happens on certain complex models, it can be fixed using the following scheme: first detect the cage self-intersection [Alliez et al. 2015], tetrahedralize the internal volume of the cage [Si 2015], and finally remove non-boundary triangles in the tetrahedralization [Jacobson et al. 2013], which results in the corrected cage. One example is shown in the right of Fig. 18.

As cross sections in the cage are fixed to the planes of cut slides, the rotations of the cross sections or the cage refinement could make a branch of the cage intersect with its other branches. In addition, as we only model the cage cross sections as rectangles, we can only generate a box-like cage for a cylindrical branch of the model, which might not be able to produce a high resolution, quad-dominant cage via cage subdivision. In the future, we would like to address it by modeling the cut slides as non-planar polygons.

5 Conclusion

In this paper, we have introduced an interactive cage generation approach given a single 3D model as the input. Our approach is capable of utilizing user-provided cut slides to generate results which are comparable to artist-crafted cages. The resulting cages have nearly rectangular cross sections with consistent orientations to avoid cage twisting (§3.2). With a good convergence to the input model, the resulting cages also bound the input model with a user-specified tightness (§3.4).

Our interactive solution offers a significant saving of manual effort while it is still flexible enough to handle inputs with various structures and topologies (§4). With very low time and memory complexities, our solution can be easily integrated into modeling and animation tools, making an important complement to many existing cage-based deformation coordinate generations.

The main limitations of our approach are due to the employment of unbound, planar cut slides in our current approach. This may limit the separability of the model and could initially create a self-intersecting cage at certain unusual scenarios. To address such issues, as the future work we plan to investigate the possible use of non-planar cut slides (e.g., the curve networks used in [Takayama et al. 2013]) as the user interaction. Lastly, open research problems still exist in automated cage generation. For example, although quite a few cage generation approaches have been proposed to date, there does not exist a systematic or quantitative measure to quantify the quality of cages in deformation applications. In order to develop such quantitative measures, a rigorous user study for comparing cages by various methods would be needed.

Acknowledgements

The authors would like to thank Li Wei for useful discussion, and thank anonymous SI3D reviewers for their constructive comments. This research is supported in part by NSF IIS-1524782 and the Natural Science Foundation of China (NSFC) grant (No. 61328204).

References

ALLIEZ, P., TAYEB, S., AND WORMSER, C. 2015. 3D fast intersection and distance computation. In *CGAL User and Reference Manual*, 4.7 ed. CGAL Editorial Board.

BEN-CHEN, M., WEBER, O., AND GOTSMAN, C. 2009. Spatial deformation transfer. In *SCA'09*, ACM, 67–74.

CHEN, X., AND FENG, J. 2014. Adaptive skeleton-driven cages for mesh sequences. *Computer Animation and Virtual Worlds 25*, 3-4, 447–455.

DENG, Z.-J., LUO, X.-N., AND MIAO, X.-P. 2011. Automatic cage building with quadric error metrics. *Journal of Computer Science and Technology 26*, 3, 538–547.

FITZGIBBON, A., PILU, M., AND FISHER, R. B. 1999. Direct least square fitting of ellipses. *IEEE Trans. Pattern Anal. Mach. Intell. 21*, 5 (May), 476–480.

FLOATER, M. S., KÓS, G., AND REIMERS, M. 2005. Mean value coordinates in 3D. *Comput. Aided Geom. Des. 22*, 7 (Oct.), 623–631.

GARCÍA, F. G., PARADINAS, T., COLL, N., AND PATOW, G. 2013. *cages:: A multilevel, multi-cage-based system for mesh deformation. *ACM Trans. Graph. 32*, 3 (July), 24:1–24:13.

GUENNEBAUD, G., JACOB, B., ET AL., 2015. Eigen v3.2.4. http://eigen.tuxfamily.org.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4 (July), 78:1–78:8.

JACOBSON, A., KAVAN, L., AND SORKINE-HORNUNG, O. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph. 32*, 4 (July), 33:1–33:12.

JACOBSON, A., DENG, Z., KAVAN, L., AND LEWIS, J. 2014. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*.

JACOBSON, A., PANOZZO, D., ET AL., 2015. libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3 (July).

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3 (July), 561–566.

JU, T., ZHOU, Q.-Y., VAN DE PANNE, M., COHEN-OR, D., AND NEUMANN, U. 2008. Reusable skinning templates using cage-based deformations. *ACM Trans. Graph. 27*, 5 (Dec.), 122:1–122:10.

KNÖPPEL, F., CRANE, K., PINKALL, U., AND SCHRÖDER, P. 2013. Globally optimal direction fields. *ACM Trans. Graph. 32*, 4 (July), 59:1–59:10.

LI, X.-Y., JU, T., AND HU, S.-M. 2013. Cubic mean value coordinates. *ACM Trans. Graph. 32*, 4 (July), 126:1–126:10.

LIPMAN, Y., LEVIN, D., AND COHEN-OR, D. 2008. Green coordinates. *ACM Trans. Graph. 27*, 3 (Aug.), 78:1–78:10.

NIETO, J. R., AND SUSÍN, A. 2013. Cage based deformations: a survey. In *Deformation models*. Springer, 75–99.

SACHT, L., VOUGA, E., AND JACOBSON, A. 2015. Nested cages. *ACM Transactions on Graphics (TOG) 34*, 6, 170.

SAVOYE, Y., AND FRANCO, J.-S. 2010. Cage-based tracking for performance animation. In *ACCV 2010*, vol. 3, 1903–1914.

SHAMIR, A. 2008. A survey on mesh segmentation techniques. In *Computer graphics forum*, vol. 27, Wiley Online Library, 1539–1556.

SI, H. 2015. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw. 41*, 2 (Feb.), 11:1–11:36.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, 109–116.

TAKAYAMA, K., PANOZZO, D., SORKINE-HORNUNG, A., AND SORKINE-HORNUNG, O. 2013. Sketch-based generation and editing of quad meshes. *ACM Trans. Graph. 32*, 4 (July), 97:1–97:8.

THIERY, J.-M., TIERNY, J., AND BOUBEKEUR, T. 2012. Cager: Cage-based reverse engineering of animated 3d shapes. *Computer Graphics Forum 31*, 8, 2303–2316.

XIAN, C., LIN, H., AND GAO, S. 2012. Automatic cage generation by improved obbs for mesh deformation. *The Visual Computer 28*, 1 (Jan.), 21–33.

YANG, X., CHANG, J., SOUTHERN, R., AND ZHANG, J. J. 2013. Automatic cage construction for retargeted muscle fitting. *The Visual Computer 29*, 5, 369–380.