Vis Comput
DOI 10.1007/s00371-011-0595-6

ORIGINAL ARTICLE

Automatic cage generation by improved OBBs for mesh deformation

Chuhua Xian · Hongwei Lin · Shuming Gao

Abstract In cage-based deformation, the most tedious task is to construct the coarse cage bounding a model. Currently, the coarse cage is constructed mainly by hand, and the construction usually takes several hours, even longer. Therefore, it is important to develop a convenient method to generate the coarse cage bounding a model. In this paper, we devise a method to construct the coarse cage automatically using the improved OBB tree, while allowing the users to modify the cage easily. Firstly, the OBB tree bounding the model is generated, where we propose an improved OBB slicing rule to make the generated OBBs close to the model it contains. Secondly, the OBBs are adjusted and merged into a whole entity by the boolean union operation. Finally, the outer surface of the entity is extracted as the coarse cage. Empirical results demonstrate the effectiveness and efficiency of the automatic coarse cage-generation method.

Keywords Cage generation · Mesh deformation · Geometric design · Computer graphics

1 Introduction

In cage-based deformation, a coarse cage enclosing a model is required to be constructed in advance for manipulating the model. The mesh of the coarse cage should be sparse enough for easy manipulation, while its shape should be as close as possible to the model for generating desirable deformation result. In general, the coarse cage is constructed either by hand [1–5], or by subdividing a bounding box of the model [3]. However, the interactive method is very tedious and time-consuming, usually taking several hours, even longer. Furthermore, when the shape of the model to be deformed is complex, just as the *Octopus* model illustrated in Fig. 1, it is very hard to construct its coarse cage by hand. On the other hand, though it is easier to generate the coarse cage by subdividing the bounding box, the shape of the generated cage is far from that of the model to be deformed. Therefore, it is an important problem in cage-based deformation to construct the coarse cage with desirable shape automatically.

The *Oriented Bounding Box* (abbr. OBB) is presented in Ref. [6], and widely employed in the applications such as collision detection. In this paper, using the improved OBB tree, we develop a method to construct the coarse cage bounding a model automatically. Specifically, given a model, its OBB tree is first constructed, where we improve the OBB slicing rule to make the OBBs tightly enclose the model; next, these OBBs are registered and merged into a whole three-dimensional entity by the boolean union operation; finally, the outer surface of the 3D entity is extracted as the coarse cage. Figure 1 shows the automatically constructed coarse cage of the *Octopus* model using the method developed in this paper.

The automatic coarse cage-generation method can construct a coarse cage with desirable shape bounding a model

Electronic supplementary material The online version of this article (doi:10.1007/s00371-011-0595-6) contains supplementary material, which is available to authorized users.

C. Xian · H. Lin (✉) · S. Gao
State Key Lab. of CAD & CG, Zhejiang University, Hangzhou, 310058, P.R. China
e-mail: hwlin@cad.zju.edu.cn

C. Xian
e-mail: xianchuhua@cad.zju.edu.cn

S. Gao
e-mail: smgao@cad.zju.edu.cn

Published online: 21 April 2011

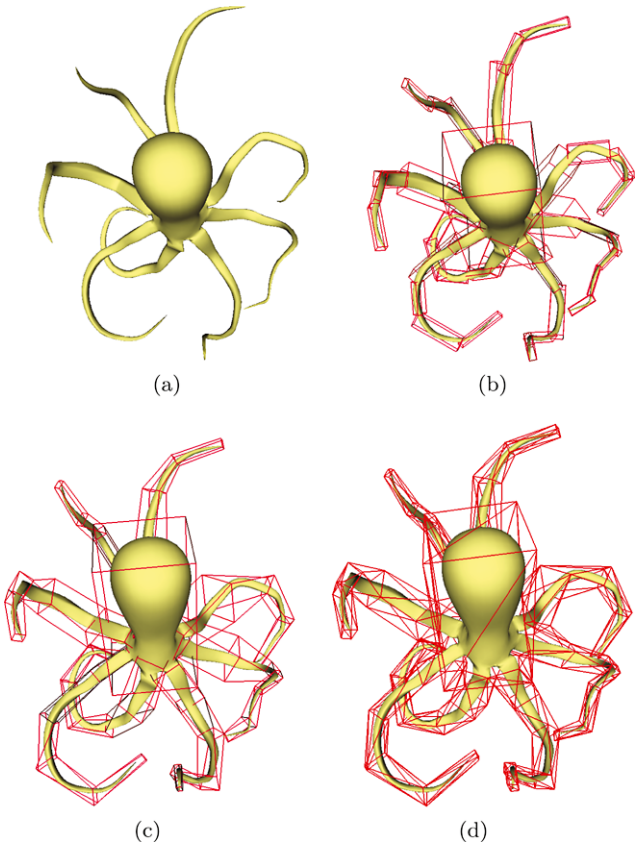

Fig. 1 Coarse cage generation for the *Octopus* model. (**a**) the *Octopus* model; (**b**) the OBBs; (**c**) the registered OBBs; (**d**) the coarse cage of the *Octopus* model generated by our method

automatically. However, if the users want to modify the automatically generated coarse cage, they just need to select the OBBs to be subdivided further. So the modification of the coarse cage is easy. The main contributions of this method include:

1. *Automation*: This method is automatic after the users set the two parameters for the termination condition (see Sect. 2.3), thus greatly reducing the users' interactive burden.
2. *Improved OBB slicing rule*: We improve the rule for slicing the OBBs, which makes the generated coarse cage enclose the model tightly.
3. *Hierarchical cages*: Different parameters in the termination condition (Sect. 2.3) lead to hierarchical cages.
4. *Local modification*: Users can modify the coarse cage locally and conveniently, by selecting and modifying a few OBBs.

The remainder of this paper is organized as follows. In Sect. 1.1, the related work is reviewed. In Sect. 2, we introduce the details of the coarse cage-generation algorithm. Section 3 discusses the algorithm in detail. Furthermore, Sect. 4 presents some results and experimental data. Finally, we conclude this paper in Sect. 5.

1.1 Related work

Invented by Sederberg et al. [7], free form deformation (FFD) is a well-known space-based deformation technique, where a regular lattice is employed to deform the model it contains. Alternatively, skeleton subspace deformation (SSD) is presented to facilitate the model deformation, by manipulating the skeleton of the model [8, 9].

To make the mesh deformation more efficient, the regular lattice in FFD is improved to the coarse cage with shape closer to the model, thus generating the cage-based deformation techniques [3–5]. Recently, Ju et al. [10] propose a cage-based deformation method for character animation by reusing skinning templates, where the coarse cage is constructed by piecing the pre-defined templates, guided by the skeleton of the model. Landreneau and Schaefer make real-time deformations of large models possible by reducing the weight set [11].

In Ref. [12], Xian et al. propose a method to generate the coarse bounding cage by uniform voxelization. This method first voxelizes the model, then extracts the outer sides of the feature voxels and optimizes it to produce the coarse cage. However, since the size of the voxels is uniform, the generated coarse bounding cage is usually too dense to be employed in the cage-based deformation.

2 Coarse cage-generation algorithm

In this section, we first list the coarse cage-generation algorithm in Algorithm 1, and explain the main steps in detail in the following sections. To ensure that the OBBs enclose the model strictly, they are all enlarged 1.1 times in our implementation.

2.1 Voxelization and point set generation

To voxelize the model M, we first calculate the initial OBB O of the model using Principal Component Analysis (PCA) of the mesh vertices. Next, the initial OBB O is divided into voxels with a pre-specified voxel size s. In this paper, we take the minimum length between two mesh vertices in M as the voxel size s.

After dividing the initial OBB O into voxels, they should be classified. First, the voxels intersecting the mesh M are taken as the *feature voxels*. Second, the other voxels are categorized as *inner voxels* or *outer voxels* by the scan-conversion algorithm [13]. Finally, the mesh vertices and the barycenters of the inner voxels constitute the point set P for calculating the OBBs. Noticeably, the OBB generation method presented in this paper performs just on the voxelization representation and the point set P it deduced, without requiring the mesh connectivity.

Algorithm 1: Coarse Cage-Generation Algorithm

1: Voxelize the model M, and generate the point set P, consisting of the mesh vertices and barycenters of the inner voxels (Sect. 2.1);
2: Calculate the OBB of the point set P using the *Principal Component Analysis* (PCA), and push the OBB on a stack;
3: **while** The stack is not empty **do**
4: Pop an OBB from the stack;
5: **if** The OBB does not satisfy the *termination condition* (Sect. 2.3) **then**
6: Slice the point set containing in the OBB into two parts using the *improved OBB slicing rule* (Sect. 2.2), construct two OBBs for them, and push the two OBBs to the stack;
7: **end if**
8: **end while**
9: Register the adjacent OBBs (Sect. 2.4);
10: Merge the OBBs into a whole entity using the boolean union operation (Sect. 2.4);
11: Extract the outer surface of the whole entity as the coarse cage bounding the model (Sect. 2.4).

It should be noted that an open model should be repaired to be closed in advance. Then, the above method can be applied to the repaired model to generate the inner voxels and the point set P. For the mesh repair methods, please refer to the survey paper Ref. [14].

The reason for performing PCA on the point set P, not the mesh vertices itself, is that the PCA result on the point set P is much closer to the original model than that on the mesh vertices, as illustrated in Fig. 2.

After the initial OBB O containing the mesh M is generated, we construct the *global* Cartesian coordinate system, by taking the barycenter of the OBB as the origin, and the three directional edges of the OBB as axes. Then, the coordinates of the points in the point set P are transformed into the new coordinate system.

In most cases, the OBB generated by PCA is smaller than the AABB (Axis Aligned Bounding Box). However, in some cases, the size of AABB is smaller than that of the OBB. In our implementation, we choose the smaller one of AABB and OBB as the bounding box.

2.2 Improved OBB slicing rule

In the original OBB construction method [6], each OBB is split by a plane which is perpendicular to the longest edge, and passes through the barycenter of the point set contained in the OBB. This simple rule does not take the shape of the model, which the OBB contains, into consideration. Visually, a geometric model is naturally segmented at the place

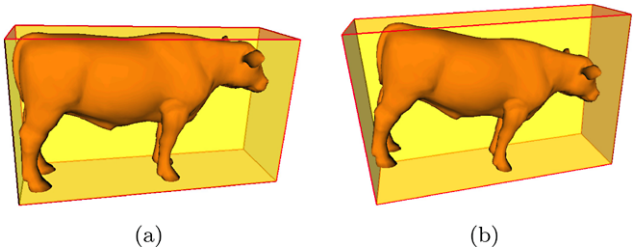

Fig. 2 Comparison of PCA results on the point set P and the mesh vertices. (**a**) PCA result on the point set P. (**b**) PCA result on the mesh vertices

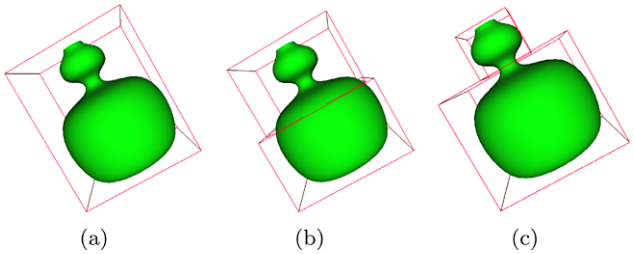

Fig. 3 Comparison of OBB slicing rules. (**a**) The model and its OBB; (**b**) result by the original OBB slicing rule [6]; (**c**) result by the improved OBB slicing rule developed in this paper

where its shape changes the most. Mimicking this practical rule, in this section, we present a carefully designed OBB slicing rule, which makes the splitting plane lie at the place with the greatest change in shape (see Fig. 3).

To this end, we first construct the *local* Cartesian coordinate system by taking the left-lower corner of the OBB as the origin, and the longest, the second longest, the shortest edges as x, y, z axes, respectively.

Next, define the *minimum cross section area function* $f(x)$, where the cross section is the intersection between the model M and the splitting plane at x. If the cross section at x has only one connected part, $f(x)$ is defined as the area of the part; otherwise, if it has two or more separated parts s_i^x, $i = 1, 2, \ldots, k$, $f(x)$ is taken as the minimum area of these parts (see Fig. 4). That is,

$$f(x) = \min_i \left\{ \text{area}\left(s_i^x\right), i = 1, 2, \ldots, k \right\}.$$

As illustrated in Fig. 4(c), after calculating the data points $\{(x_j, f(x_j)), j = 1, 2, \ldots, n\}$ on the area function $f(x)$, they are fitted by a piece of cubic uniform B-spline curve, which is taken as the area function $f(x)$. Figure 4(c) demonstrates the area function $f(x)$ of the model in Fig. 4(a), generated by B-spline curve fitting as aforementioned.

Evidently, the OBB of the model should be sliced at the place with the greatest change in shape. According to the geometric meaning of the function $f(x)$, the shape change of the model can be reflected by the graph of the function $f(x)$. The biggest jump in the graph corresponds to the greatest shape change of the model. Therefore, if the biggest jump

of $f(x)$ occurs at $x = x_0$, we will split the OBB, and the model it contains, at $x = x_0$, by a plane perpendicular to the x axis and passing at $x = x_0$.

To locate the place of the biggest jump, we first compute the interlaced local minimum and maximum points of $f(x)$. Suppose one of the local minimum point is $x_{\min}$, and the local maximum point near to it is $x_{\max}$. Then, the jump value at $x_{\min}$ is $J_{x_{\min}} = f(x_{\max}) - f(x_{\min})$. By simple comparison, the biggest jump can be located, supposing it is $J_{x^b_{\min}} = f(x^b_{\max}) - f(x^b_{\min})$. Then, the current OBB will be sliced into two sub-OBBs at $x = x^b_{\min}$.

The OBB slicing rule developed in this section chooses the longest axis as the first candidate. If it fails, the second longest is chosen, and finally the shortest one. It should be pointed out that if the shape of the model varies smoothly, the cross section area function f may have no jump. In this case, we employ the original slicing rule presented in [6] to split the OBB, that is, splitting the OBB at the barycenter of the point set the OBB contains.

In our implementation, we calculate the minimum cross section area function f based on the voxelization of the model. Specifically, to calculate the area function $f(x)$ at $x = x_0$, we just need to count the number of the inner and feature voxels that intersect the plane $x = x_0$, and take the number as the area value $f(x_0)$.

Afterwards, the two OBBs of the point sets in the two sub-OBBs are constructed using PCA, respectively, and pushed to the OBB stack.

2.3 Termination condition

In the coarse cage-generation algorithm, to determine whether an OBB is required to be split, we depend on the termination condition, which concerns the following factors.

Shape of the model As mentioned in Sect. 2.2, the cross section area function $f(x)$ reflects the shape variety of the model. The parameter T_1 will be closer to 1 if the mesh is smoother, where

$$T_1 = \frac{\min f(x)}{\max f(x)}. \tag{1}$$

Then, if T_1 is greater than a threshold, it means that the shape of the mesh contained in the OBB varies very little, so the OBB is not required to be split further. Otherwise, the OBB should be split.

Shape of the OBB However, if the model is very regular, such as a long cuboid, T_1 will equal 1 at the beginning. Hence, we also need to consider the shape of the OBB. Suppose $l_{\max}$ and $l_{\min}$ are the length of the longest and shortest edges of the candidate OBB, respectively. The other parameter for the terminal condition is,

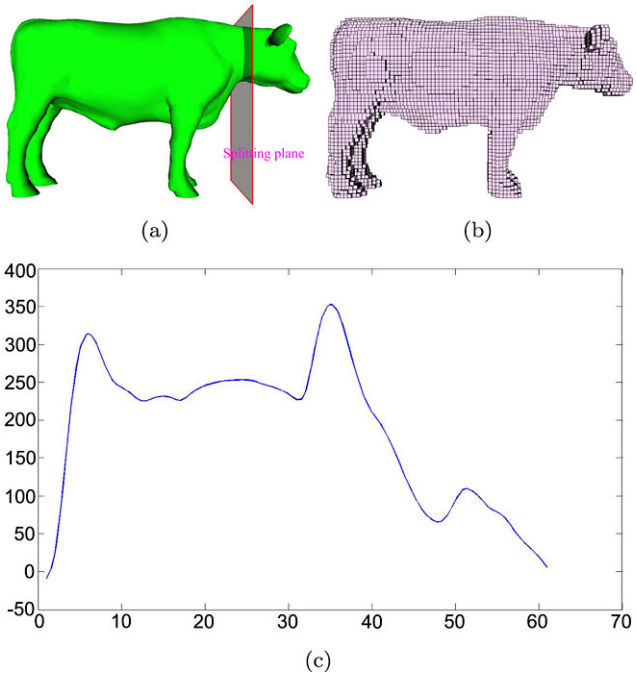

Fig. 4 Improved OBB slicing rule. (**a**) The *cow* model and the place with the greatest change in shape. (**b**) The voxelization of the *cow* model. (**c**) The minimum cross section area function $f(x)$ generated by B-spline curve fitting

$$T_2 = \frac{l_{\min}}{l_{\max}}. \tag{2}$$

If T_2 approaches 1, we will terminate the splitting of the OBB.

In conclusion, the termination condition for the OBB splitting is,

$$T_1 > \eta, \quad \text{and} \quad T_2 > \varsigma, \tag{3}$$

where η and ς are thresholds specified by users.

2.4 OBB registration and mesh generation

After all the OBBs cannot be split anymore, the binary OBB tree of the model is constructed. However, if we construct the coarse cage by combining the current OBBs using boolean union operation, the generated coarse cage is very uneven, and the distances between the mesh vertices of the coarse cage to the model it contains varies widely. To make the coarse cage smoother, we register and merge the adjacent OBBs with similar orientation and size before the OBB union, and improve the mesh quality of the coarse cage after the OBB union (see Sect. 2.5).

The pseudocode of the OBB registration and merging procedure is presented in Algorithm 2, and the details are explained in the following.

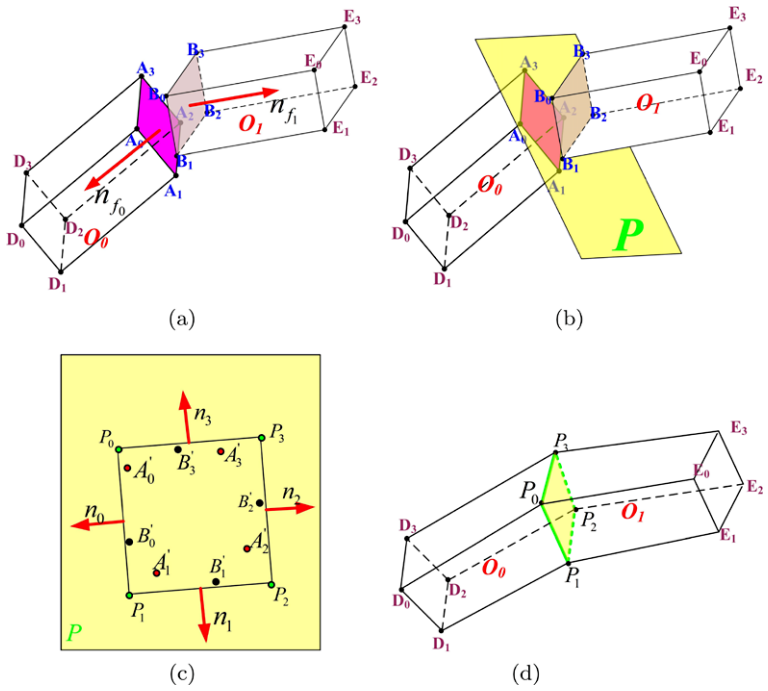

Fig. 5 OBB registration and merging. (**a**) The two adjacent OBBs; $F_{A_0A_1A_2A_3}$ and $F_{B_0B_1B_2B_3}$ are the two adjacent faces of the adjacent OBBs. (**b**) The plane P. (**c**) The projecting points and their oriented bounding box on the plane P. (**d**) The merged object

Algorithm 2: OBB Registration and Merging (Refer to Fig. 5)

1: Construct the plane P between the two adjacent OBBs;
2: Project the vertices $A_i, B_i, i = 0, 1, 2, 3$ of the faces F_0 and F_1 to the plane P, denoting the projecting points as $A'_i, B'_i, i = 0, 1, 2, 3$;
3: Generate the OBB F_b of the projecting points $A'_i, B'_i, i = 0, 1, 2, 3$ on the plane P;
4: Register the vertices of the faces F_0 and F_1 to that of F_b, respectively, thus getting the registration between the vertices of F_0 and F_1;
5: Merge the OBBs O_0 and O_1 by linking the corresponding vertices.

As shown in Fig. 5(a), O_0 and O_1 are two adjacent OBBs, with adjacent faces $F_0 = F_{A_0A_1A_2A_3}$ and $F_1 = F_{B_0B_1B_2B_3}$. Suppose the normals of F_0 and F_1 are $\boldsymbol{n}_{F_0}$ and $\boldsymbol{n}_{F_1}$, and their areas are s_0 and s_1 ($s_0 \le s_1$), respectively. In our implementation, if,

- $\frac{s_0}{s_1} > 0.6$,
- the angle between $\boldsymbol{n}_{F_0}$ and $\boldsymbol{n}_{F_1}$ is greater than $\frac{2}{3}\pi$, and,
- the sub-models contained in the two adjacent OBBs are connected,

the two adjacent OBBs O_0 and O_1 should be registered and merged.

Denote the centers of F_i as $C_i, i = 0, 1$, respectively. To register the two OBBs O_0 and O_1, we construct a plane P, which passes the point $\frac{C_0+C_1}{2}$, and is perpendicular to the line C_0C_1 (see Fig. 5(b)). Afterwards, the points A_j, B_j are projected to the plane P, generating the projecting points $A'_j, B'_j, j = 0, 1, 2, 3$ (Fig. 5(c)). Furthermore, the 2-dimensional OBB $F_b = F_{P_0P_1P_2P_3}$ bounding these projecting points is constructed on the plane P.

The bounding box F_b is taken as the context for registering the two OBBs O_0 and O_1. They are registered by searching the correspondence between the vertices of F_b and F_0, and that between F_b and F_1.

In the following, we only present the method for registering F_b and F_0. The registration between F_b and F_1 is similar. Suppose the normalized outward normals of the four side faces of the OBB O_0 are $\boldsymbol{n}_i^0, i = 0, 1, 2, 3$, respectively, and the normalized outward normals of the four edges of F_b are $\boldsymbol{n}_i, i = 0, 1, 2, 3$ (Fig. 5(c)), respectively. Clearly, there are four possible correspondences between the vertices of F_b and F_0. We choose the correspondence that makes the sum of the inner products between the corresponding normals maximal as the registration between F_b and F_0, i.e.,

$$\max_{j=0,1,2,3} \sum_{i=0}^{3} \boldsymbol{n}_i \cdot \boldsymbol{n}^0_{((j+i) \bmod 4)}.$$

After registering the two OBBs O_0 and O_1, they are merged into one entity by linking the corresponding vertices. Suppose the vertex A_i of F_0 corresponds to the vertex P_i of $F_b, i = 0, 1, 2, 3$; on the other hand, in the OBB O_0, the vertex A_i connects the vertex $D_i, i = 0, 1, 2, 3$ (Fig. 5(b)). Then, we link P_i and $D_i, i = 0, 1, 2, 3$ (Fig. 5(d)). Similarly, for the OBB Q_1, we also link the vertex P_i, which

corresponds to the vertex B_i, and the vertex E_i, which connects the vertex B_i in the OBB O_1, forming four lines $P_i E_i, i = 0, 1, 2, 3$ (Fig. 5(d)). In this way, the two OBB O_0 and O_1 are merged into one entity (Fig. 5(d)).

Finally, an intersection test is performed to check whether the merged object intersects the model it contains, or with the other OBBs. If so, the current OBB merging is invalid and discarded; if not, the merged object is retained.

Until now, we have merged the adjacent OBBs with similar orientation and size, if the merged object intersects with neither the model, nor the other OBBs. However, in general, there still remain some OBBs, which cannot be merged, either because their orientation or size is greatly different from its adjacent OBBs, or because the merged object intersects with the model or other OBBs. Therefore, the boolean union operation is employed to combine these non-merged OBBs and the merging entities into a whole entity, whose outer faces constitute the initial coarse cage. In our implementation, we employ CGAL [15] to perform the boolean union operation.

2.5 Mesh improvement

Generated by union of OBBs, the mesh of the initial coarse cage contains many *cap triangles*, *needle triangles*, and *high valence vertices*. Thus, a post-processing is required to improve the mesh quality of the coarse cage. Notably, in the improvement of the mesh quality, we need to ensure the improved cage intersects neither with the model, nor with the other OBBs.

In our implementation, we define a *cap triangle* as one with an angle larger than $\frac{8}{9}\pi$, and a *needle triangle* as one with the ratio between its shortest and longest edge less than 0.2. Additionally, a vertex is called a *high valence vertex* if its valence is greater than 8.

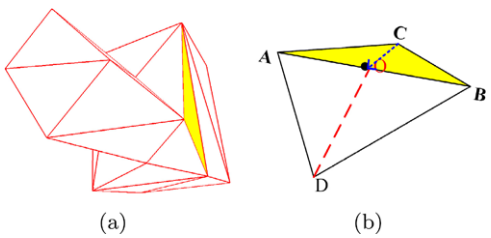

Fig. 6 Eliminate the cap triangle. (**a**) The cap triangle (in *yellow*) in a model. (**b**) Split the cap triangle and its adjacent triangle

To improve the mesh of the cage, we first deal with the cap triangles (Fig. 6). Suppose $\triangle ABC$ is a cap triangle, and $\triangle ABD$ is its adjacent triangle. The largest angle of $\triangle ABC$ is $\angle C$, and AB is the edge subtending $\angle C$. To eliminate the cap triangle, we split the triangles $\triangle ABC$ and $\triangle ABD$ by linking two edges DO and CO, where O is the middle point of the edge AB. The new triangles $\triangle ACO$ and $\triangle BCO$ are taken as two needle triangles, handled in the next step. Note that splitting the triangles does not change the shape of the cage, so it is ensured that the cage does not intersect the original model.

Next, we search the needle triangles in the new cage and clean them up by vertex merging (Fig. 7). Suppose $\triangle ABC$ is a needle triangle, and $\angle A$ is its smallest angle. To remove the needle triangle, there are two options for merging. One moves the vertex B to C, where B is the *merged vertex*, and C is the *fixed vertex*. The other moves C to B, where C is the merged vertex, and B is the fixed vertex. Moreover, to guarantee the cage after vertex merging does not intersect the model, we need to perform further testing. That is, if the fixed point lies outside all the newly generated triangles, then the new cage does not intersect the mesh, the vertex merging is valid, and the needle triangle can be removed. Otherwise, if the fixed point lies inside the newly generated triangles, the merging is invalid. If the two merging options are both invalid, the needle triangle cannot be removed. This way, we guarantee that the cage after improvement does not intersect the model. Noticeably, to keep the OBB structure, we do not merge the two vertices which belong to the same OBB.

Finally, the high valence vertices are identified in the new cage and handled. To do so, we sort the angles adjacent to the high valence vertices in ascending order, and deal with just the angles less than 20 degree in order. As illustrated in Fig. 8, an angle $\angle A$ has two edges e_1 and e_2, corresponding to two pairs of adjacent triangles. We flip each edge in each pair of triangles, and compute the smallest angle of the newly generated pair of triangles, denoted by α_1 and α_2, respectively. Then, we identify the bigger one between α_1 and

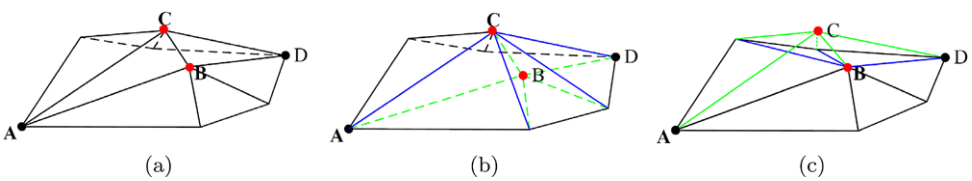

Fig. 7 Merge the needle triangles. (**a**) $\triangle ABC$ is a needle triangle and B, C need to merge. (**b**) Moving C to B is the valid merging. In this case, the newly generated faces (in *blue*) are outside the original coarse cage, guaranteeing that the new faces do not intersect the model it contains. (**c**) Moving C to B is the invalid merging. In this case, the newly generated faces (in *blue*) are inside the original coarse cage, so it is possible for the new faces to intersect the model it contains

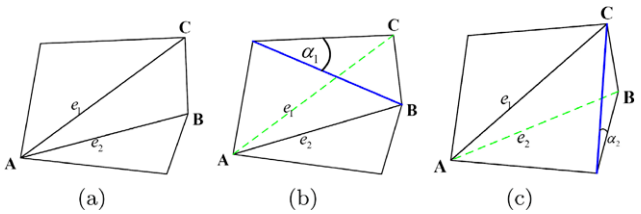

Fig. 8 Edge flip for high valence vertex. (**a**) A high valence vertex. (**b**) Flip one pair of triangles, and the minimum angle is α_1. (**c**) Flip the other pair of triangles with minimum angle α_2

α_2, and suppose it is α_1. If α_1 is bigger than $\angle A$, we leave the flip of edge e_1, and cancel the flip of e_2. Otherwise, both flips are invalid and canceled. Similarly, to ensure the new cage after edge flip does not intersect the model, we only flip the concave edges.

It is ensured in each of the above three operations that the newly generated mesh faces do not intersect the model. Moreover, we also need to check whether the newly generated mesh faces intersect the other mesh faces of the coarse cage after each of the three operations. If so, the operation is invalid, and the result is discarded.

3 Discussion

In this section, we discuss some issues related to the coarse cage-generation algorithm presented in this paper.

3.1 Computation of the OBBs for spherical and cylindrical shapes

First, the orientation of an OBB is unstable for spherical or cylindrical shapes, due to the use of PCA. In fact, in determining the orientation of the OBB containing a spherical shape, the three eigenvalues of the PCA matrix are the same. Similarly, in determining the orientation of the OBB enclosing a cylindrical shape, two eigenvalues of the PCA matrix are the same. When we first encounter these cases, i.e., two or three eigenvalues which are the same, we save the orientation of the OBB. After the model the OBB contains is divided into sub-parts, and two or three eigenvalues of the PCA matrix for calculating the OBB of the sub-part are still equal to each other, we make the orientation of the OBB for the sub-part consistent with that of its parent OBB.

3.2 Comparison to the original OBB slicing rule

The OBB slicing rule is the key to the OBB generation. The original OBB slicing rule [6] splits an OBB by a plane, which is perpendicular to the longest edge, and passes the barycenter of the point set contained in the OBB. As stated above, this simple slicing rule does not take into consideration the shape of the model which the OBB contains. In contrast, the improved OBB slicing rule presented in this paper takes advantage of the shape change of the model contained in the OBB. It splits the OBB at the place with the greatest shape change (see Fig. 3), imitating human beings' recognition procedure. Therefore, the OBBs generated by the improved slicing rule is more "natural" than that generated by the original rule. Figure 9(a) shows the OBBs generated by the original slicing rule, and Fig. 9(b) shows the OBBs by the improved slicing rule developed in this paper.

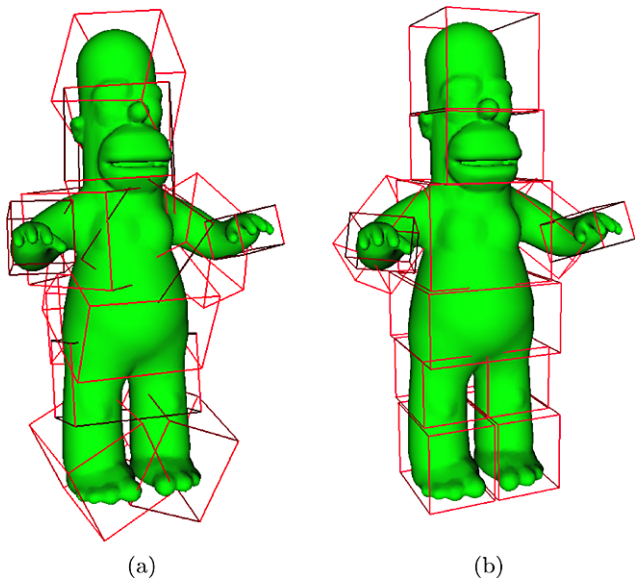

Fig. 9 Comparison between the original and improved slicing rules. (**a**) OBBs generated by the original slicing rule [6]; (**b**) OBBs generated by the improved slicing rule developed in this paper

3.3 Handling point clouds

As stated in Sect. 2.1, the improved OBB generation algorithm presented in this paper performs just on the voxelization representation of the original model and the point set P it deduced, without requiring the mesh connectivity. So, the improved OBB generation algorithm can handle even a point cloud model, which has no connectivity information. In Fig. 10, the OBBs (Fig. 10(a)) and coarse cage (Fig. 10(b)) of a point cloud model are generated by the improved OBB generation algorithm. In the generation of the OBBs bounding the point cloud, the feature voxels are taken as the voxels that contain the data points.

3.4 Robustness to the model transformation

Since the minimum cross section area function $f(x)$ is defined as the minimum area of the disconnected parts, and the minimum area changes a little during a slight rigid transformation of the model, the OBBs generated by the improved

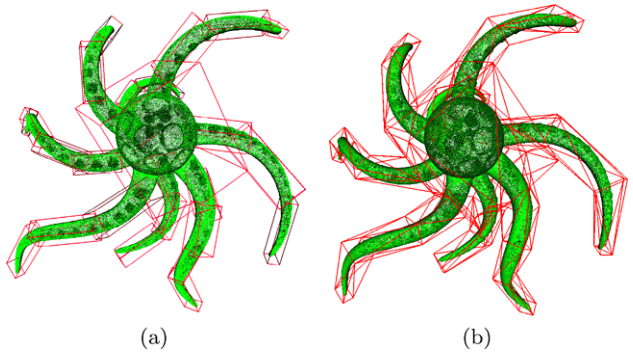

Fig. 10 The OBBs and coarse cage generation for the point cloud of the *Octopus* model. (**a**) The point cloud and OBBs of the *Octopus* model; (**b**) the coarse cage of the point cloud

slicing rule is robust to the model transformation. In other words, the generated OBBs by the improved slicing rule will move along with the transformed model. Figure 11 demonstrates the OBBs and coarse cages of the *woman* model before and after model transformation.

3.5 Influence of the parameters in the termination condition

The coarse cage-generation algorithm is nearly fully automatic, except that the users are required to input the two parameters η and ς for the termination condition (See 2.3). As pointed out in Sect. 2.3, both the parameters have clear geometric meanings, so they are easy to be understood and determined. The parameter ς controls the shape the OBBs, while η regulates the shape of the sub-models contained in the OBBs. Based on our experience, the coarse cage-generation algorithm can produce an acceptable result in most cases by setting $\eta = 0.75$, and $\varsigma = 0.5$.

Furthermore, it can lead to hierarchical cages by choosing different values of these parameters. Specifically, two hierarchical cages are illustrated in Fig. 12, with $\eta = 0.75$, $\varsigma = 0.3$ in Figs. 12(a) and 12(b), and $\eta = 0.8$, $\varsigma = 0.6$ in Figs. 12(c) and 12(d), respectively.

3.6 Comparison with the method in Ref. [12]

Moreover, in Fig. 13, we compare the method developed in this paper with that in [12]. As aforementioned, because the voxels employed in the method in [12] are both uniform in size, and parallel to the coordinate frame, the generated coarse cage is usually too dense to be used in mesh deformation. In this example, Figs. 13(a) and 13(b) are the cage generated by the method in Ref. [12], displayed from two views. Figure 13(c) are the OBBs bounding the *dino* model, and Fig. 13(d) is the coarse cage generated by the method developed in this paper. Evidently, though the cage in Fig. 13(a) is denser than the one in Fig. 13(d), it fails to separate the legs, while the cage in Fig. 13(d) does. On the other hand, though the coarse cage with higher resolution in Fig. 13(b) can separate the two legs, it is too dense to be employed in mesh deformation.

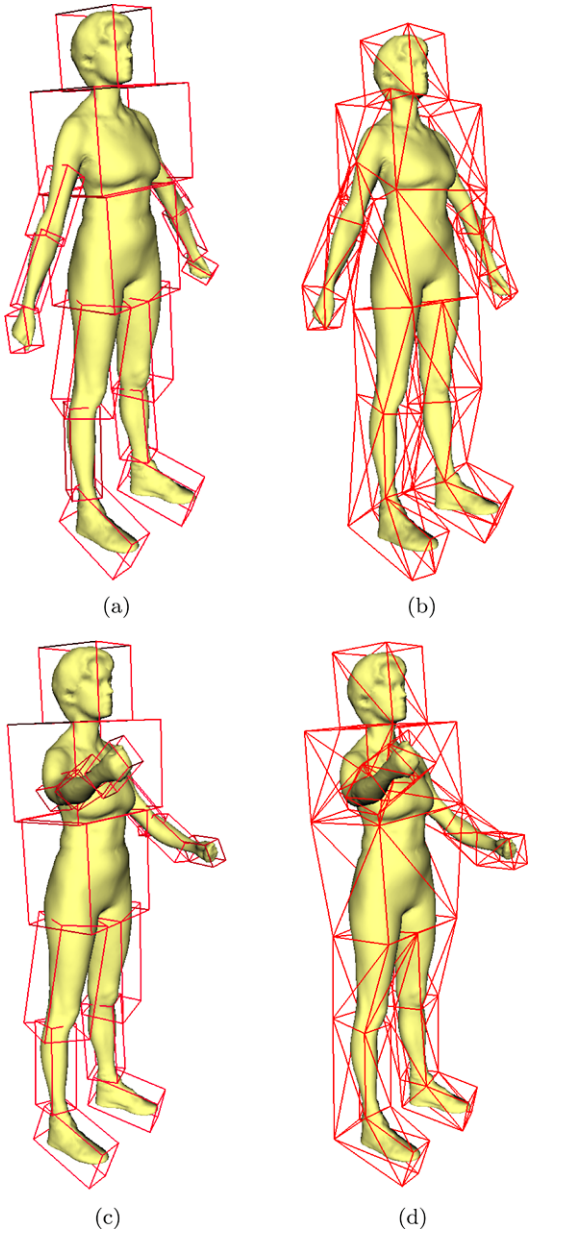

Fig. 11 The OBBs and coarse cage move along with the transformed model. (**a**, **c**) The OBBs of the *woman* model before and after transformation; (**b**, **d**) The coarse cages of the *woman* model before and after transformation

Moreover, the scaled minimum, average and maximum distances between the cage and the model contained by it in Fig. 13(b) are, 0.013, 0.053, 0.029, respectively, while these data for the cage and model in Fig. 13(d) are 0.003, 0.058, 0.028, respectively. The scaled distance means the real distance over the diagonal length of the initial OBB containing the model.

3.7 Limitations

One of the limitations of the coarse cage-generation method is that the generated coarse cage relies on the initial voxelization of the given model M, which is hard to be determined reasonably. The voxelization is one of the key ingredients in the coarse cage-generation method, which has a great impact on determining the slicing position. Currently, the voxel size is chosen as the minimum distance between two mesh vertices empirically. A more reasonable method for determining the resolution will be developed in the future.

Another limitation of our method is about the twisting between adjacent OBBs. When the shape is close to a sphere or cylinder, the 2nd and 3rd axes of PCA can be rather unstable. Also, the orientation of the parent OBB may not always be the one suitable for the children OBB. Hence, twisting can still happen even with the remedy proposed in Sect. 3.1. Ideally, some rotation of the 2nd and 3rd axes of adjacent OBBs should be applied in a post-process to ensure a low-twist at their junction.

Additionally, our method cannot deal with open models directly. To handle an open model, it should be closed firstly, and then voxelization can be performed on the closed model.

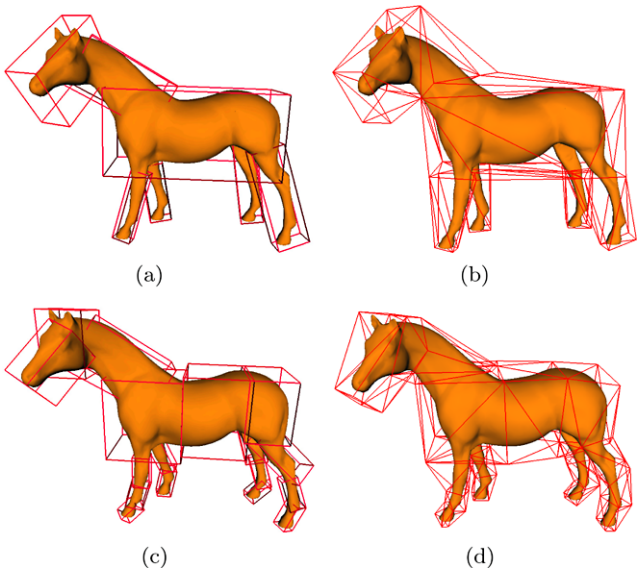

Fig. 12 *Horse* and its hierarchical cages. (**a**) The OBBs with $\eta = 0.75$ and $\varsigma = 0.3$; (**b**) the coarse cage with $\eta = 0.75$ and $\varsigma = 0.3$; (**c**) the OBBs with $\eta = 0.8$ and $\varsigma = 0.6$; (**d**) the coarse cage with $\eta = 0.8$ and $\varsigma = 0.6$

4 Results and applications

The automatic coarse cage-generation algorithm developed in this paper is implemented with VC++ 2008 and OpenGL, and runs on a PC with Core 2^{TM} 2.6 GHZ CPU and 4 GB memory in a single thread. Some empirical examples are illustrated in this section, and Table 1 presents the empirical data. In all of the examples, except that in Fig. 17, the thresholds for the termination condition (3) are chosen as $\eta = 0.75$, $\varsigma = 0.5$.

Figure 9 and Fig. 14 illustrate the procedure for generating the coarse cage for the model *homer*. Figure 9(b) demonstrates the OBBs bounding the model *homer*. Figure 14(a) shows the OBBs after registration and merging. After merging the OBBs by the union operation, the initial coarse cage is generated in Fig. 14(b). Furthermore, the mesh quality of the coarse cage is improved in Fig. 14(c). From Fig. 14, it can be seen that the two close legs are separated successfully using the automatic coarse cage-generation method.

Figure 15 and Fig. 16 demonstrate the capability for the coarse cage-generation algorithm to handle models with complex shapes. Figure 15(a) and Fig. 16(a) are the OBBs containing the models; Fig. 15(b) and Fig. 16(b) are the coarse cages of the *shark* and *dancer* models, which capture the salient features of the two models faithfully.

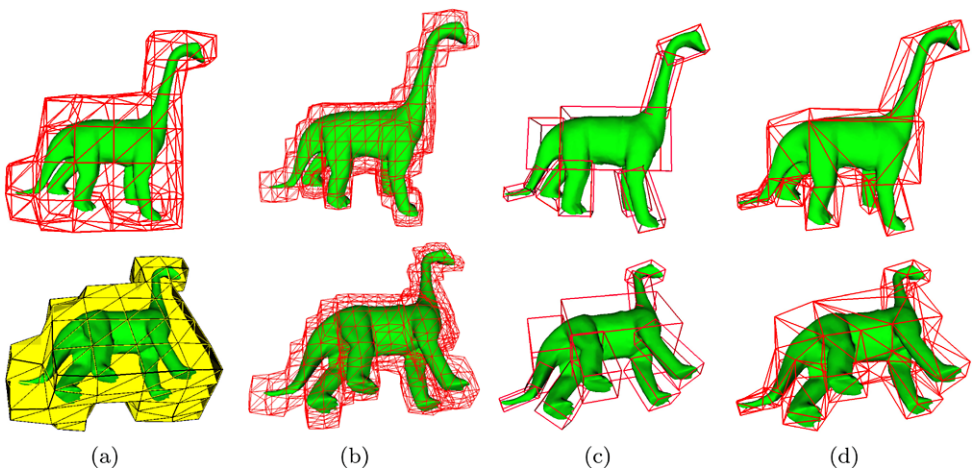

Fig. 13 Comparison between the method in Ref. [12] and the method developed in this paper. (**a**) The coarse cage generated by the method in [12] using a low resolution, where the legs of *dino* cannot be separated. (**b**) The legs of *dino* can be separated using a higher resolution, but the cage has too many vertices (#vertices: 599) to be used in the model deformation. (**c**) The OBBs. (**d**) The coarse cage generated by the method in this paper

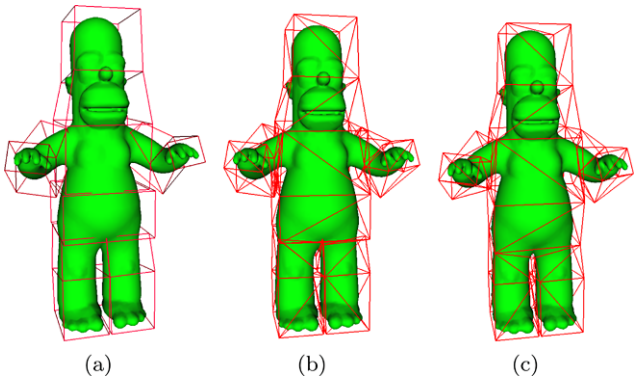

Fig. 14 Coarse cage generation for the *homer* model. (**a**) The OBBs after registration; (**b**) the initial coarse cage by uniting all OBBs and triangulating the outer face; (**c**) the coarse cage after mesh quality improvement

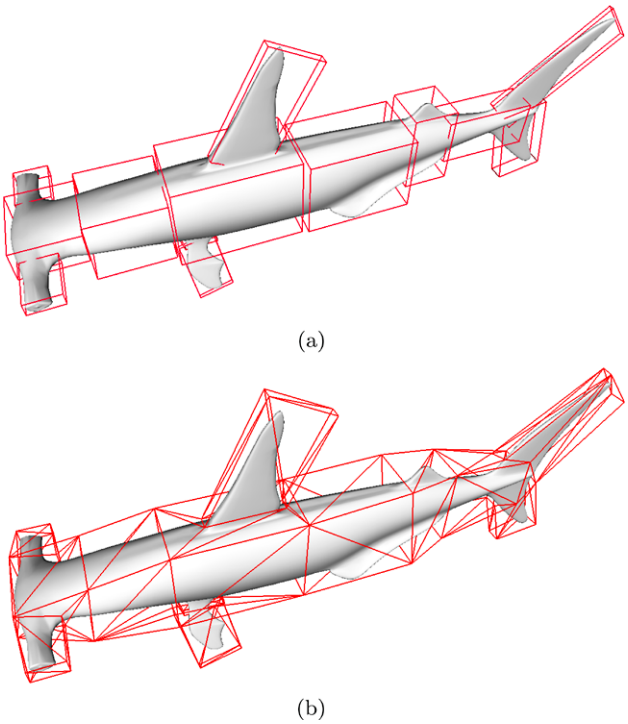

Fig. 15 *Shark* and its cage. (**a**) The OBBs. (**b**) The coarse cage

Moreover, Fig. 1 presents the generation of the coarse cage of the *Octopus* model which has a complex shape by the method developed in this paper. Note that it is very hard to produce the coarse cage of the *Octopus* model by hand, while it is generated successfully by our method. Figure 1(a) is the *Octopus* model. Figure 1(b) and 1(c) are the OBBs before and after registration and merging. Finally, Fig. 1(d) shows the coarse cage generated by our method automatically.

On the other hand, the shape of the *torus* model in Fig. 17 is regular, and the improved OBB slicing rule degenerates to the original rule developed in Ref. [6]. Figure 17(a) is

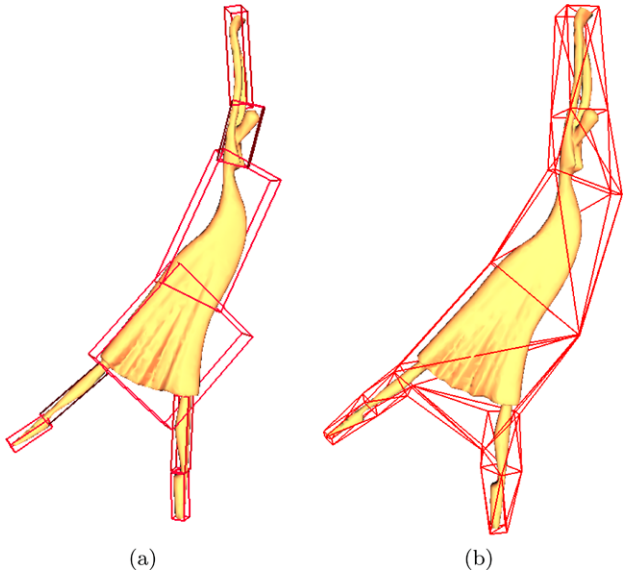

Fig. 16 *Dancer* and its cage. (**a**) The OBBs, (**b**) the coarse cage

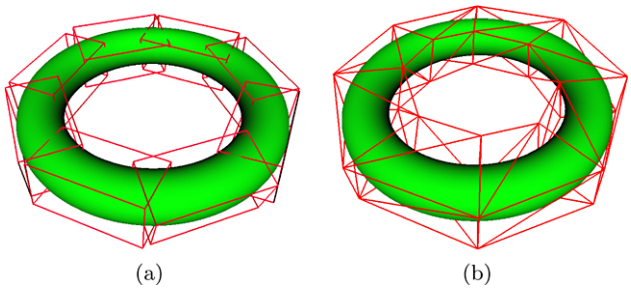

Fig. 17 *Torus* and its cages. (**a**) The OBBs, (**b**) the coarse cage

the OBBs bounding the model; Fig. 17(b) shows the coarse cage, which approximates the original model very well.

As stated above, the coarse cage can be easily modified by the user. If the users need to modify the coarse cage locally, they can just choose the corresponding OBBs, which are divided further to refine the local mesh (See Fig. 18).

Finally, we show some deformation examples in Figs. 19, 20, and 21, using the automatically generated coarse cages by the method developed in this paper, and Green Coordinates [4], where the original models are shown in Figs. 1, 12, and 14, respectively. In cage-based deformation, the shape of the cage influences the deformation result greatly. The closer the cage is to the original model, the better the deformation result. Since the coarse cage capture the salient features of the original models faithfully, the deformation results are desirable (Figs. 19, 20, and 21).

In conclusion, Table 1 lists the empirical data of the models and their cages. The second and third columns of Table 1 are the numbers of the mesh vertices and the cage vertices, respectively. The fourth column are the numbers of the OBBs enclosing the original model. The fifth to seventh

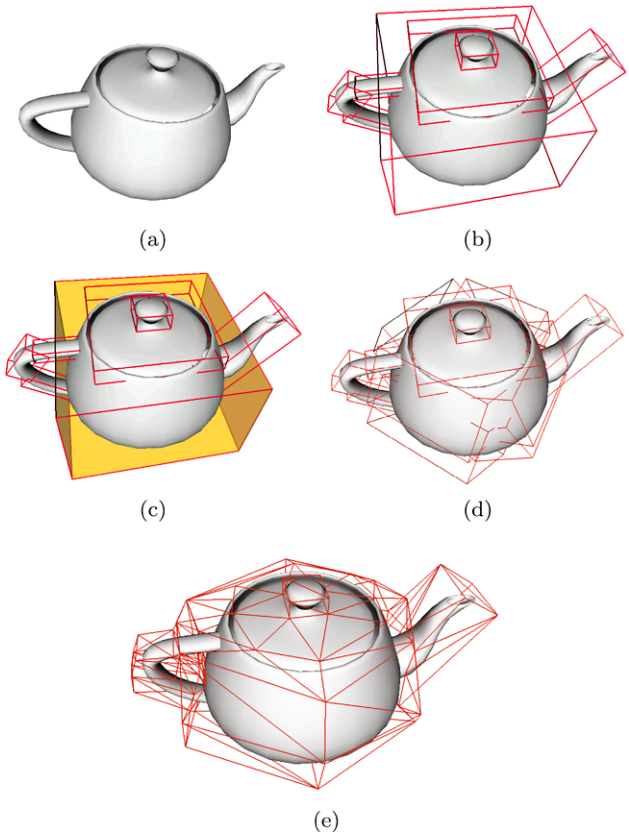

Fig. 18 Modification of the coarse cage. (**a**) The *teapot* model; (**b**) the initial OBBs; (**c**) select the OBB (in *yellow*) where the mesh need to be refined; (**d**) the refined OBBs; (**e**) the refined coarse cage

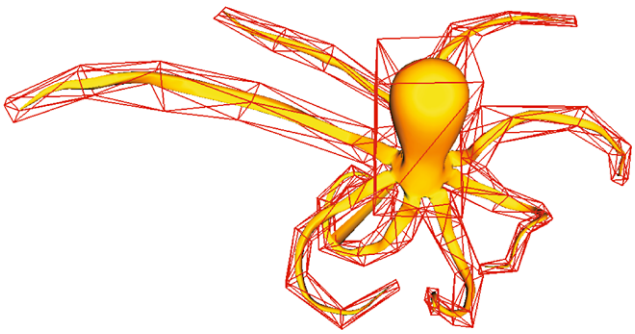

Fig. 19 Deformation result of the *octopus* model, using the coarse cage generated in Fig. 1

columns list the scaled minimum, average, and maximum distances between the coarse cage and the model, i.e., the real distance over the diagonal length of the initial OBB of the model. Finally, the last column shows the time cost in generating the coarse cages. From Table 1, we can see that the number of the vertices of the coarse cage is greatly reduced, while it captures the salient features of the model it contains.

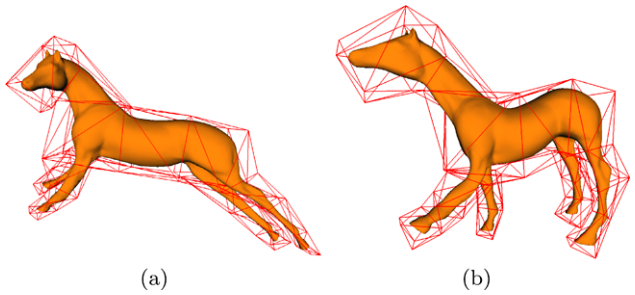

Fig. 20 Two deformation results of the *horse* model, using the coarse cage displayed in Fig. 12(d)

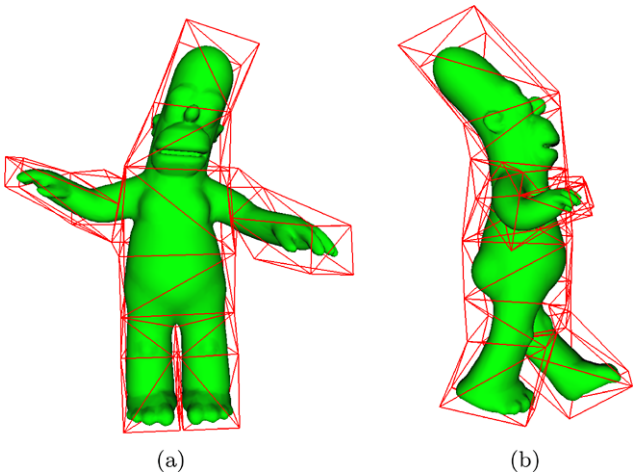

Fig. 21 Two deformation results of the *homer* model, using the coarse cage displayed in Fig. 14(c)

5 Conclusion

In this paper, we develop an automatic coarse cage construction method for cage-based deformation. Given a model, its OBBs is first constructed using the principal component analysis and the improved OBB slicing rule. Next, the OBBs are registered and merged into a whole entity using the boolean union operation, while its surface is extracted as the coarse cage. Finally, the mesh quality of the coarse cage is improved. Using this method, the coarse cage of a model is constructed usually in a few tens of seconds, thus saving users' labor greatly. Moreover, the local modification of the coarse cage is very easy. If users want to get a finer mesh at some place of the cage, they just need to choose the corresponding OBBs and slice them further. By merging the modified OBBs again, we get the refined cage. As future work, we want to improve the method to generate the control mesh for subdivision surface fitting and polycube parameterization.

Acknowledgements We thank the reviewers for their helpful comments. This paper is supported by Natural Science Foundation of China (Nos. 60970150, 60736019, 60933008), and Zhejiang Provincial Natural Science Foundation of China (No. Y1090416).

Table 1 Data on the automatic coarse cage-generation algorithm

Models	#Vert.	#Cage vert.	#OBBs.	Min	Ave.	Max	Time (s)
Octopus (Fig. 1)	5059	204	48	0.0008	0.120	0.043	1040.1
Octopus (Fig. 10)	149671	163	28	0.0007	0.115	0.036	1082.3
Woman (Fig. 11)	25172	78	15	0.007	0.130	0.059	332.5
Horse (Fig. 12(b))	5560	53	7	0.004	0.144	0.070	45.7
Horse (Fig. 12(d))	5560	111	11	0.002	0.140	0.053	69.1
Dino (Fig. 13)	5903	74	10	0.003	0.058	0.028	76.4
Homer (Fig. 14)	5103	64	12	0.008	0.132	0.069	44.7
Shark (Fig. 15)	4820	80	13	0.004	0.106	0.045	43.8
Dancer (Fig. 16)	2396	43	8	0.001	0.047	0.019	20.0
Torus (Fig. 17)	5566	32	8	0.052	0.063	0.057	21.5

Vert.: the number of the mesh vertices
Cage vert.: the number of the coarse cage vertices
OBBs: the number of the OBBs in the improved OBB tree
Min: the scaled minimum distance between the cage and the model
Ave.: the scaled average distance between the cage and the model
Max: the scaled maximum distance between the cage and the model
Time (s): the cost time in second in generating the coarse cage

References

1. Floater, M.S., Kos, G., Reimers, M.: Mean value coordinates in 3d. Comput. Aided Geom. Des. **22**, 623–631 (2005)
2. Ju, T., Schaefer, S., Warren, J.: Mean value coordinates for closed triangular meshes. In: Proceedings of SIGGRAPH 2005, pp. 561–566 (2005)
3. Joshi, P., Meyer, M., DeRose, T., Green, B., Sanocki, T.: Harmonic coordinates for character articulation. ACM Trans. Graph. **26**(3) (2007)
4. Lipman, Y., Levin, D., Cohen-Or, D.: Green coordinates. ACM Trans. Graph. **27**(3) (2008)
5. Huang, J., Chen, L., Liu, X., Bao, H.: Efficient mesh deformation using tetrandron control mesh. In: Proceedings of ACM Solid and Physical Modeling 2008, pp. 241–247 (2008)
6. Gottschalk, S., Lin, M.C., Manocha, D.: Obbtree: A hierarchical structure for rapid interference detection. In: Proceedings of SIGGRAPH 1996, pp. 171–180 (1996)
7. Sederberg, T.W., Parry, S.R.: Free-form deformation of solid geometric models. In: Proceedings of SIGGRAPH 1986 (1986)
8. Magnenat-Thalmann, N., Laperriére, R., Thalmann, D.: Joint-dependent local deformations for hand animation and object grasping. In: Proceedings on Graphics Interface'88, Edmonton, Alberta, Canada, pp. 26–33 (1989)
9. Lewis, J.P., Cordner, M., Fong, N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, pp. 165–172 (2000)
10. Ju, T., Zhou, Q.Y., van de Panne, M., Cohen-Or, D., Neumann, U.: Reusable skinning templates using cage-based deformations. ACM Trans. Graph. **26**(5) (2008)
11. Landreneau, E., Schaefer, S.: Poisson-based weight reduction of animated meshes. Comput. Graph. Forum **28**(2), 1–10 (2009)
12. Xian, C., Lin, H., Gao, S.: Automatic generation of coarse bounding cages from dense meshes. In: IEEE International Conference on Shape Modeling and Applications (SMI) 2009, Beijing, P.R. China, June (2009)
13. Hearn, D., Baker, M.P.: Computer Graphics, C Version, 2nd edn. Prentice Hall, Englewood Cliffs (1997)
14. Ju, T.: Fixing geometric errors on polygonal models: a survey. J. Comput. Sci. Technol. **24**(1), 19–29 (2009)
15. CGAL, Computational Geometry Algorithms Library, http://www.cgal.org

Chuhua Xian is currently a Ph.D. candidate of State Key Laboratory of CAD&CG, Zhejiang University, P.R. China. His research interest is 3D model deforming, geometry modeling and CAD/CAE integration.

Hongwei Lin is an associate professor in State Key Laboratory of CAD&CG, Zhejiang University, China. He received his BSc from Department of Applied Mathematics at Zhejiang University in 1996, and Ph.D. from Department of Mathematics at Zhejiang University in 2004. He worked as a communication engineer from 1996 to 1999. His current research interests are in computer aided geometric design, computer graphics, and image process.

Shuming Gao is a professor of the State Key Laboratory of CAD&CG, Zhejiang University. He received his Ph.D. degree from the Applied Mathematics Department of Zhejiang University in 1990, and was a visiting scholar and a visiting professor in the Design Automation Laboratory of Arizona State University, respectively, in 1996 and 2001. His research interests include product modeling, CAX integration, collaborative design, virtual reality in design and manufacturing, MEMSCAD, etc.