Deng ZJ, Luo XN, Miao XP. Automatic cage building with quadric error metrics. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 26(3): 538–547 May 2011. DOI 10.1007/s11390-011-1153-4

Automatic Cage Building with Quadric Error Metrics

Zheng-Jie Deng[1,2,3] (邓正杰), Xiao-Nan Luo[1,2,3] (罗笑南), and Xiao-Ping Miao[4,*] (苗晓萍), *Member*, *IEEE*

[1] *School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510006, China*

[2] *National Engineering Research Center of Digital Life, Guangzhou 510006, China*

[3] *Research Institute of Sun Yat-Sen University in Shenzhen, Shenzhen 518057, China*

[4] *School of Software, Sun Yat-Sen University, Guangzhou 510006, China*

E-mail: dzhengj@mail2.sysu.edu.cn; lnslxn@mail.sysu.edu.cn; miaoxp@mail.sysu.edu.cn

Received October 16, 2010; revised February 26, 2011.

Abstract Modern computer graphics applications usually require high resolution object models for realistic rendering. However, it is expensive and difficult to deform such models in real time. In order to reduce the computational cost during deformations, a dense model is often manipulated through a simplified structure, called cage, which envelops the model. However, cages are usually built interactively by users, which is tedious and time-consuming. In this paper, we introduce a novel method that can build cages automatically for both 2D polygons and 3D triangular meshes. The method consists of two steps: 1) simplifying the input model with quadric error metrics and quadratic programming to build a coarse cage; 2) removing the self-intersections of the coarse cage with Delaunay partitions. With this new method, a user can build a cage to envelop an input model either entirely or partially with the approximate vertex number the user specifies. Experimental results show that, compared to other cage building methods with the same number of vertex, cages built by our method are more similar to the input models. Thus, the dense models can be manipulated with higher accuracy through our cages.

Keywords cage, quadric error metrics, mesh simplification, self-intersection, deformation

1 Introduction

In order to maintain a convincing level of realism, modern computer graphics models are often created or acquired at a very high resolution. The huge number of vertices makes direct manipulation of the models tedious and computationally expensive. A common way to reduce computational cost is to build a similar but simple structure with much fewer vertices first[1-5], and then manipulate the dense model through the simpler structure. Much deformation work[6-11] performed deformations with such simpler structures (2D polygons for 2D models; 3D triangular meshes for 3D models). These simple structures, which envelop the original dense models, are called *cages*.

The main advantages of using cages in deformations are controlling simplicity and fast computational speed. For example, deforming a triangular dense mesh with a cage requires much less computational cost than deforming a dense mesh directly, because transforming its vertices requires merely a linear combination of the cage geometry using precalculated coordinates[12]. Due to these advantages, many deformation methods[10-11,13-14] choose to use cages even complicated ones. Deformation transfer work[15-16] also was done by using cages.

Cages are useful, but building cages interactively is very tedious and time-consuming, especially when the topological structure of the model is complex, such as the Dilo illustrated in Fig.1. Despite the growing needs and interests in cages, there is not much work on automatic cage building in literature. To the best of our knowledge, there are only two known methods in published literature — Xian's[17] and Ben-Chen's[15].

According to [6, 8-9], a cage should have the following two properties:

1) enveloping the original model;

2) no self-intersections.

A cage is considered better if it has fewer vertices and it is more similar to the original model.

In this paper, we introduce a method to build cages that satisfy the properties for both 2D polygons (2D simply connected models' outlines) and 3D triangular meshes.

Regular Paper

Supported by the NSFC-Guangdong Joint Fund under Grant Nos. U0735001, U0835004, U0935004, and the National Basic Research 973 Program of China under Grant No. 2011CB302204.

*Corresponding Author

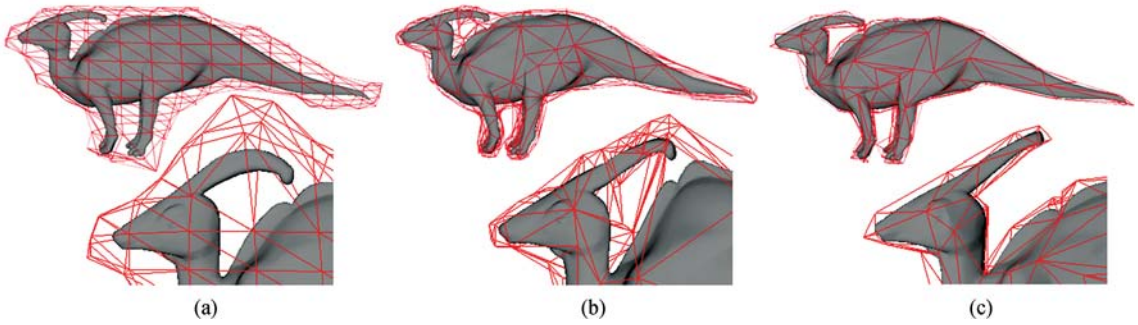


Fig.1. Dilo's cages built by Xian's, Ben-Chen's and our methods, respectively. The bottom row is using the cages to erect the Dilo's horn. (a) Xian's (173). (b) Ben-Chen's (288). (c) Ours (183). (Numbers in the parentheses are cages' vertex amounts.)

Our method consists of a simplification step and a cage self-intersection removal (CSIR) step. The simplification step iteratively simplifies the input model with quadric error metrics[2] (QEM) and quadratic programming, which eventually builds a coarse cage that envelops the input model. Moreover, two scalar functions on the vertex curvatures and the face normals are designed to adjust the simplification in order to maintain the similarity between the input model and the coarse cage. The CSIR step is performed only when the coarse cage has self-intersections. Self-intersections can be detected by methods like [18]. The CSIR step handles the intersections based on Delaunay partitions. The main contributions of this paper include:

1) An iterative method is proposed to automatically simplify an input model, where users can specify the exact number of vertices of the output coarse cage.

2) Two scalar functions about vertex curvatures and face normals are designed to maintain the shape similarity.

3) Based on Delaunay partitions a novel method has been developed to remove self-intersections in a coarse cage.

The rest of this paper is organized as follows. Section 2 introduces some previous work relative to our method. Our method is described in Section 3. Section 4 presents the experimental results. Comparisons with two typical cage building methods are also shown in this section. The conclusions are given in Section 5.

2 Previous Work

Cages are widely used in space deformations, such as [6, 8-9, 11]. In these techniques, points inside a cage are represented by their coordinates, such that manipulating the cage results in a smooth deformation of the enveloped model. Many deformations[10-11,13-14] used cages to help control a model. But they did not control the cages directly, instead they used a few control points to manipulate the cages. Deformation transfer work[15-16] is also based on cages, which demand high similarity between the cage and the original model.

Cages can be considered as a simplified version of the original model. Mesh simplification techniques have been studied for many decades, but to the best of our knowledge, there are only two automatic cage building methods[15,17] in published literature.

Ben-Chen *et al.*[15] used the approach of *simplification envelopes*[19], which guarantees that all vertices of the output mesh are within a user-specified distance of ε from the input mesh. Because the *simplification envelopes* do not consider whether the input mesh is enveloped or not, Ben-Chen *et al.*[15] built cages with an "offset" operation to enlarge the simplified mesh so that the original mesh fits inside. The cage built by [15] should be contained in a shell region with thickness 2ε, so the vertex amount of the cage can hardly be reduced. In addition, the Poisson reconstruction[20] used by [15] could not guarantee that the reconstructed mesh envelop the input mesh. Using this method, users have to build a cage recursively: reconstructing and simplifying for many times, especially for a complex model.

Xian *et al.*[17] applied *vertex clustering*[5,17], which is another commonly used simplification method. *Vertex clustering* divides the input mesh's bounding box into many grids, and then constructs a simplified mesh from the grid vertices selected according to the intersections of grids and the mesh. Xian *et al.*[17] improved the *vertex clustering* method to guarantee that the simplified mesh is a cage. But Xian *et al.* treated all vertices uniformly without considering the geometric features. Therefore, a small grid size causes the vertex amount of the cage to increase dramatically, while a big grid size causes some parts of the cage to fuse together.

Iterative simplification methods[1-4,21] are also widely used, which simplify the input mesh by a series of primitive simplification operations. Each primitive operation is determined by the least deviation error. Garland *et al.*[2] presented an iterative

simplification method, called *quadric error metric* (*QEM*) *simplification*. One advantage of the iterative simplification methods is that they are able to control the vertex amount of the simplified mesh exactly. But a common disadvantage of these iterative simplification methods is that the simplified mesh is not guaranteed to envelop the input mesh. The mesh simplification method proposed in this paper, though also based on QEM, successfully overcomes this problem of enveloping.

3 Cage Building

Let a model be an oriented simplicial surface (i.e., 2D polygons, 3D triangular meshes), that is $P = (V, F)$, where $V = \{v_i\}_{i \in I_V} \subset \mathbb{R}^d$ $(d = 2, 3)$ are the vertices and $F = \{f_i\}_{i \in I_F}$ are the simplicial face elements, namely edges in case of polygons in 2D, triangles in case of triangular meshes in 3D. I_V and I_F are the index sets of V and F. The outward normal to the oriented simplicial face f is denoted as $\boldsymbol{n}(f) = [n_x, n_y, n_z]$, $\|\boldsymbol{n}(f)\| = 1$.

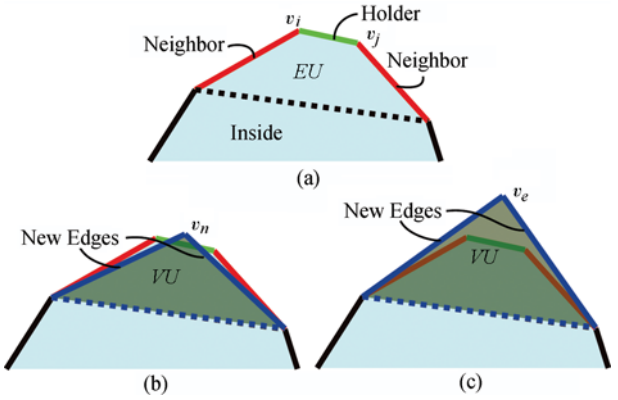


Fig.2. *EU* and *VU*. (a) One holder (green) and its neighbors (red) constitute an *EU*. (b) $VU(v_n)$, generated with QEM, not enveloping the *EU*. (c) One $VU(v_e)$ enveloping the *EU* for building cages.

Each vertex v_i and its first ring neighbors form a vertex-umbrella (VU) around v_i, denoted by $VU(v_i)$. Each edge (v_i, v_j) and its first ring neighbors form an edge-umbrella (EU) around the edge, denoted by $EU(v_i, v_j)$. The edge (v_i, v_j) is named as the *holder* of $EU(v_i, v_j)$. See Fig.2 for a 2D case.

The flowchart of our cage building method is illustrated in Fig.3. The simplification step inputs the original dense model, and outputs a coarse cage, which is the input to the CSIR step. The CSIR step removes the self-intersections of the coarse cage, and outputs the result cage. We will present the two steps in detail in the following two subsections.

3.1 Simplification

The simplification step is an iterative edge contraction method using QEM[2] to evaluate the deviation error of every contraction. Every contraction of an edge (v_i, v_j) is considered as replacing $EU(v_i, v_j)$ by $VU(v_n)$, where the borders of $EU(v_i, v_j)$ and $VU(v_n)$ are the same, and the replacing vertex v_n is obtained by a quadratic programming based on $EU(v_i, v_j)$, described in Subsection 3.1.1. To evaluate the deviation error, a symmetric matrix $\boldsymbol{Q}_i$ like QEM[2] is associated with each vertex v_i, and the deviation error at vertex v, denoted by $v = [v_x, v_y, v_z]$ (its homogeneous type is $\bar{v} = [v_x, v_y, v_z, 1]$), is the quadratic form $E_i(v) = \bar{v}\boldsymbol{Q}_i\bar{v}^{\mathrm{T}}$. For an edge (v_i, v_j), a new matrix $\boldsymbol{Q}_n = \boldsymbol{Q}_i + \boldsymbol{Q}_j$ is used to calculate the error at vertex v like

$$E_{ij}(v) = \bar{v}\boldsymbol{Q}_n\bar{v}^{\mathrm{T}}. \tag{1}$$

Though [2] does not mention the information, QEM could be easily extended to the 2D polygon case. With $v = [v_x, v_y]$, $n(f) = [n_x, n_y]$, the symmetric matrix $\boldsymbol{Q}_i$ can also be defined to evaluate the error for 2D polygon cases.

Note that the QEM simplification[2] uses the matrix $\boldsymbol{Q}_n$ not only to evaluate the deviation error, but also to calculate the new replacing vertex v_n (refer to [2] for details). The result of the QEM simplification is similar to the input model, but does not guarantee to envelop the input model (see Fig.2(b)).

3.1.1 Obtaining the Replacing Vertex

To build an enveloping cage, a new method of obtaining the replacing vertex should be designed, while it should guarantee each *VU* will cover the corresponding *EU* during replacing (see Fig.2(c)).

In order to achieve the enveloping, we firstly define a signed distance between a vertex v and a face f as:

$$D(v, f) = n(f)v^{\mathrm{T}} - n(f)v_i^{\mathrm{T}}, \quad v_i \in f.$$

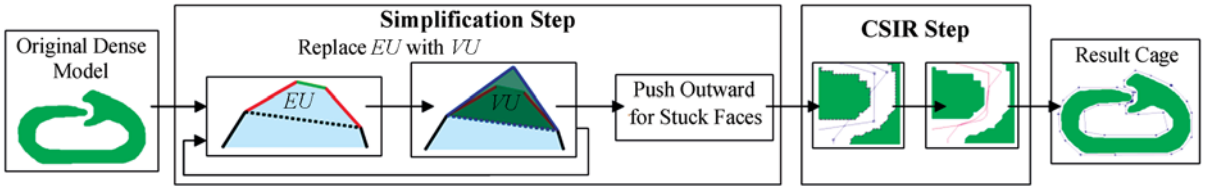


Fig.3. Flowchart of our cage building method.

With this definition, it is easy to check that

$$v \text{ is } \begin{cases} \text{outside } f, & \text{if } D(v,f) > 0; \\ \text{on } f, & \text{if } D(v,f) = 0; \\ \text{inside } f, & \text{if } D(v,f) < 0. \end{cases} \quad (2)$$

Then the new vertex replacing the edge (v_i, v_j) can be a solution, say v_e, of the following quadratic programming:

$$\min_v \sum_{f \in EU(v_i,v_j)} D(v,f),$$

s.t.

$$\forall f \in EU(v_i,v_j),\ D(v,f) \geqslant 0. \quad (3)$$

With (2) and (3), a vertex v_e, whose $VU(v_e)$ covers $EU(v_i,v_j)$, can be obtained. In order to maintain the fidelity of the input model, $D(v,f) = 0$ is allowed during simplifying, but a push-out post-treatment is applied afterwards.

Sometimes, v_e does not exist for some singular EUs. For example, in 2D case, the normals of the two neighbor edges of an edge are opposite vectors. For all singular cases, we choose v_e such that it is far away from the model (e.g., 1000 times of the radius of the model's bounding box).

Let $\{v_e\}$ be the set of the solutions of the quadratic programming (3) (running over all the edges). Then, a series of EU replacing can be done in an order of the errors $\{E_{ij}(v_e)\}$, similar to QEM. Because the new VU always covers the EU in each replacing step, a coarse cage is built after these simplifying iterations. Note that the coarse cage might have self-intersections, or have intersections with or stick to the input model. These violations need post-treatments and are discussed in Subsection 3.1.4 and Subsection 3.2, respectively.

By using $\{E_{ij}(v_e)\}$ to arrange the simplification, our method works well for many types of models. But it is not good enough, for example, in Fig.4(a) the cage does not reflect the model's shape, and in Fig.5(b) the cage contains two almost-opposite faces. This is because the QEM error evaluation scheme less cares the curvatures and the normals. The improvement will be described in Subsections 3.1.2 and 3.1.3.

3.1.2 Adjusting Errors by Vertex Curvatures

By using $\{E_{ij}(v_e)\}$, the cage may not reflect the shape of the input model, e.g., Fig.4(a). In [22], the shape of the model was well maintained by keeping the mean curvatures during deformations. This idea is applied here to designing a scalar function for vertex curvatures, called S_c, given by

$$S_c(v_e,v_i,v_j) = \|C(v_i)\| * \|v_e - v_i\| + \|C(v_j)\| * \|v_e - v_j\|, \quad (4)$$

where v_e is the replacing vertex of the edge (v_i, v_j), and $C(v)$ is the Laplacian coordinates[22] of the vertex v. Then, the simplifying order can be arranged by using $S_c(v_e,v_i,v_j)E_{ij}(v_e)$ as the deviation error. With S_c, the farther the v_e is from v_i, the bigger the $\|C(v_i)\|$ is, then the bigger the error is. Thus, the edge (v_i, v_j) is contracted later. This new alterative simplifying order can maintain the fidelity better, see Fig.4(b).

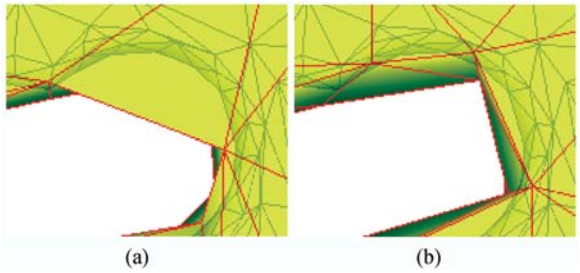


Fig.4. Cages built with the same vertex amount without/with S_c. The same part of each cage is shown embedding the input model. (a) Without S_c. (b) With S_c, maintaining the fidelity better.

3.1.3 Adjusting Errors by Face Normals

By using $\{E_{ij}(v_e)\}$, for 3D triangular meshes, the result VU could contain some face pairs with almost opposite normals. An example is shown in Figs. 5(b)∼5(c), where the arrows point out the

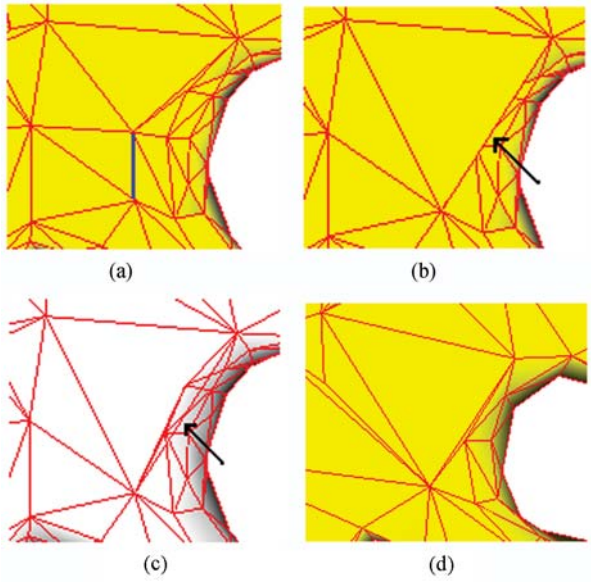


Fig.5. S_n effect observing through one contraction during building a cage. (a) Without/with S_n (1098). (b) Without S_n (1097). (c) Without S_n (1097), rendered by wireframes. (d) With S_n (500). (Numbers in parentheses are the cage's vertex amounts.) Without S_n, the blue edge in (a) is contracted to obtain (b), while the step is avoided with S_n.

almost-opposite face pairs. The dihedrals of such almost-opposite face pairs are near 0 or 2π. Such face pairs will result in a bad cage, since the two folder areas of such a pair impact the model with the almost same strength, but in the opposite direction. Therefore, another scalar function S_n is designed to avoid such face pairs

$$S_n(v_e) = \sum_{f \in VU(v_e)} \frac{1}{(\|\boldsymbol{n}(f) + \boldsymbol{n}(next(f))\|^2)}, \qquad (5)$$

where $next(f)$ is the face next to f in $VU(v_e)$ in counter-clockwise order around v_e. Then, the simplifying order can be arranged by using $S_n(v_e)E_{ij}(v_e)$ as the deviation error. When f and $next(f)$ are almost-opposite, $S_n(v_e)$ will be large enough to enlarge the deviation error, so this v_e will be selected later. As illustrated in Fig.5, with S_n, it avoids the step from Fig.5(a) to Fig.5(b), and does not generate such face pairs even after contracting more edges in Fig.5(d). For 2D polygons, S_n can be defined similarly. Fig.6 shows an example of the effect of S_c and S_n on a 2D monkey.①

3.1.4 Simplification Procedure

The pseudocode of the simplification step is given in Algorithm 1.

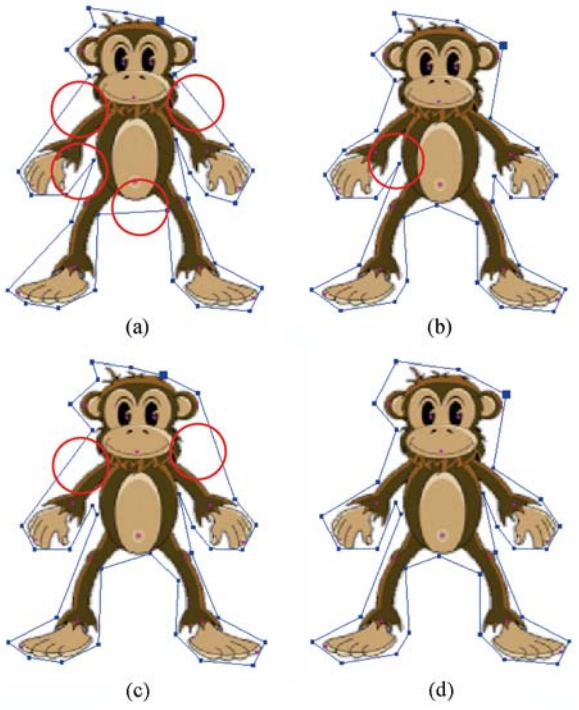


Fig.6. S_c and S_n. Cages built with the same vertex amount for a 2D monkey. (a) Without S_c and S_n. (b) Only S_c. (c) Only S_n. (d) With S_c and S_n. (The cage built with S_c and S_n presents the trunk better. Red circles mark the unsatisfactory parts of the other cages.)

Algorithm 1. Procedure of the Simplification

Input: a dense model M
Output: a coarse cage C enveloping M
Generate C by duplicating M;
foreach vertex v_i **do**
  Compute $\boldsymbol{Q}_i$ like QEM;
end
Build an empty heap H;
foreach edge e **do**
  FindVe(e, H);
end
while H can pop **do**
  Pop the least E edge e;
  Simplify e by replacing EU with VU in C (like QEM);
  foreach new edge e **do**
    FindVe(e, H);
  end
  foreach edge e adjoining to the new VU **do**
    FindVe(e, H);
  end
  if C meets user's constraints **then**
    Break;
  end
end
Procedure FindVe(e, H)
Look for v_e for replaced e (see (3));
Compute S_c (see (4)) and S_n (see (5));
Compute the replacing error E;
Insert e into H sorted by E.

During the simplification, it keeps checking whether the current coarse cage meets user's constraints (the vertex amount, or the tolerable error, etc.). Note that, the user's requirements may not always be satisfied. For example, the error of the edge on the heap top is extremely large, caused by a vertex far away from the model (as mentioned in Subsection 3.1.1). That means no edge can be contracted normally anymore. The simplification should be stopped.

A face of a cage is called stuck if it contains any point of the input model. Such faces can be found by checking if $D(v, f) = 0$ during the simplification. The problem of stuck faces in cages should be solved, because some applications[9] would be singular on them. When a stuck happens, a push-out post-treatment is applied, i.e., the stuck face is pushed outward with a

①The monkey model is taken from [11].

small distance along its normal. We use 1/1000 of the radius length of the model's bounding box. Such a small distance will not change the shape of the coarse cage too much.

3.2 Cage Self-Intersection Removal

The coarse cage built by the simplification step may have self-intersections, and even intersect with the input model. This violates the second property of a qualified cage. In this subsection, we illustrate how to remove the intersections in 2D case. Fig.7 demonstrates the removal procedure. It is straight forward to extend the idea to 3D case.

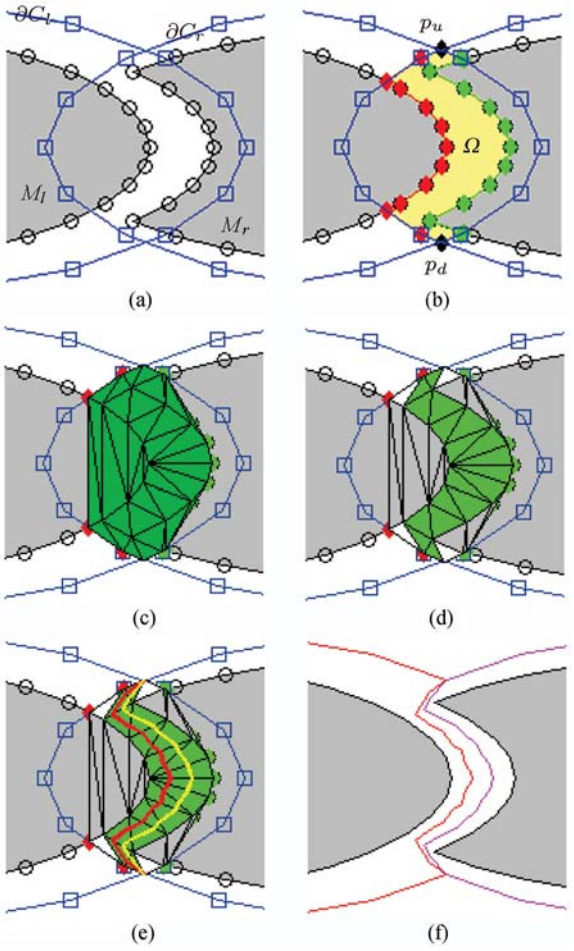


Fig.7. Cage self-intersection removal. (a) Intersection. (b) Ω, p_u, p_d. (c) Triangulation for $\partial\Omega$. (d) $\{\triangle l_i p r_i\}$. (e) Construction of broken lines. (f) Update of the cage.

The intersections can be detected by many methods, such as marching squares and the method presented in [18]. Let two areas of the coarse cage, C_l on the left and C_r on the right, be intersected, so $C_l \cap C_r \neq \varnothing$. C_l and C_r cover two areas of the model, denoted by M_l and M_r, respectively (e.g., Fig.7(a)). These areas' borders are denoted with $\partial *$, which are polygons in 2D case. Let C_l and C_r be small enough to make $C_l \cap C_r$ be a simply connected polygon, then $\partial C_l \cap \partial C_r$ has only two points, denoted by p_u and p_d. The un-intersecting cage's faces, separating M_l and M_r, should be in the simply connected polygon $\Omega = ((C_l \cap C_r) - M_l - M_r)$ (e.g., Fig.7(b)). $\partial\Omega$ is made by some vertices of $\partial C_l, \partial C_r, \partial M_l, \partial M_r$, and their intersecting points. In order to find the un-intersecting cage's faces and then remove the self-intersections of the coarse cage, we give the following theorem and its proof first.

Theorem 1. *Given C_l, C_r, M_l, M_r, Ω, p_u and p_d, then there is one broken line R from p_u to p_d, and R satisfies $R \subset \Omega$, $R \cap M_l = \varnothing$ and $R \cap M_r = \varnothing$.*

Proof. According to the simplification step, $p_u, p_d \notin M_l \cup M_r$, and $p_u, p_d \in C_l \cap C_r$, so p_u and p_d are two vertices of $\partial\Omega$. (See Fig.7(b).) Trace $\partial\Omega$, in counter-clockwise order from p_u to p_d, collects the meeting vertices into a set V_l (red vertices in Fig.7(b)), and then from p_d to p_u constructs the other set V_r (green vertices in Fig.7(b)). We perform a Delaunay triangulation on the vertices of $\partial\Omega$ to obtain a triangle set DT, whose edges form an edge set DE.

If $V_l = \varnothing$ or $V_r = \varnothing$, then the edge $(p_u, p_d) \subset \partial\Omega$, $(p_u, p_d) \cap M_l = \varnothing$ and $(p_u, p_d) \cap M_r = \varnothing$, so R can be the edge (p_u, p_d).

If $V_l \neq \varnothing$ and $V_r \neq \varnothing$, and the edge $(p_u, p_d) \in DE$, then R can be the edge (p_u, p_d). It should be satisfied that $R \cap M_l = \varnothing$ and $R \cap M_r = \varnothing$, or the edge (p_u, p_d) is segmented by some intersecting points.

If $V_l \neq \varnothing$ and $V_r \neq \varnothing$, but the edge $(p_u, p_d) \notin DE$, then $\exists p_s \in V_l$, $(p_u, p_s) \in DE$, and $\exists p_e \in V_r, (p_u, p_e) \in DE$. Let $E_u = \{(p_u, p) | (p_u, p) \in DE\}$, after sorting E_u around p_u from p_s to p_e in counter-clockwise order, we can find the last edge (p_u, l_1), where $l_1 \in V_l$, and the first edge (p_u, r_1), where $r_1 \in V_r$. Because Ω is simply connected, $\triangle l_1 p_u r_1 \in DT$. Given a constant $t \in (0, 1)$, one point $u_1 = l_1(1-t) + r_1 t$ can be obtained on the edge (l_1, r_1). Treat the segment (p_u, u_1) as the first part of R. For convenience, let $p_o = p_u$.

Then we can find $\triangle l_i p r_i \in DT$ $(i = 1, 2, \ldots)$, where $p \neq p_o$. According to vertex p,

$$\begin{cases} p \in V_l: & p_o = l_i,\ l_{i+1} = p,\ r_{i+1} = r_i, \\ & u_{i+1} = l_{i+1}(1-t) + r_{i+1} t; \\ p \in V_r: & p_o = r_i,\ l_{i+1} = l_i,\ r_{i+1} = p, \\ & u_{i+1} = l_{i+1}(1-t) + r_{i+1} t; \\ p = p_d: & u_{i+1} = p_d. \end{cases}$$

The segment (u_i, u_{i+1}) is set as the next part of R. If $p \neq p_d$, then continue to search the next triangle and the next segment $(i = i + 1)$, else stop and obtain an

R from p_u to p_d. This must happen, because p_d is one separator of V_l and V_r on $\partial\Omega$.

As $l_i \in V_l$ and $r_i \in V_r$, triangles $\{\triangle l_i p r_i\}$ (green triangles in Fig.7(d)) are in Ω. And the segments of R are abstracted in these triangles, so $R \cap M_l = \varnothing$ and $R \cap M_r = \varnothing$. □

With different t, different R can be obtained like the two broken lines (red and yellow) in Fig.7(e), which use 1/3 and 2/3, respectively. They are only sticking at p_u and p_d. Specially, if R is just the edge (p_u, p_d), the same R is used.

Then, a new cage can be constructed with the two Rs through splitting p_u and p_d, see Fig.7(f). In Fig.8, a 2D model's cage is built and the self-intersection is removed.

For 3D triangular meshes, one example of the CSIR step on 3D mesh is shown in Fig.1(c). In the simplification step, the approximate vertex amount is set as 173, equal to that of the cage built by Xian's method in Fig.1(a). After the simplification, the head of Dilo intersects with its back. After applying the CSIR step, a new no-intersection cage with 183 vertices is obtained. The result cage is used to deform the Dilo model, see Fig.1(c).

4 Results and Discussions

Through the simplification step and the CSIR step described in Section 3, we present a method of building 2D and 3D cages. Some examples are shown in Fig.1, Fig.6, and Figs. 8∼12, where we use Green Coordinates[9] for all deformations. As illustrated in Fig.9, a 2D monkey can be deformed with the cage built by our method.

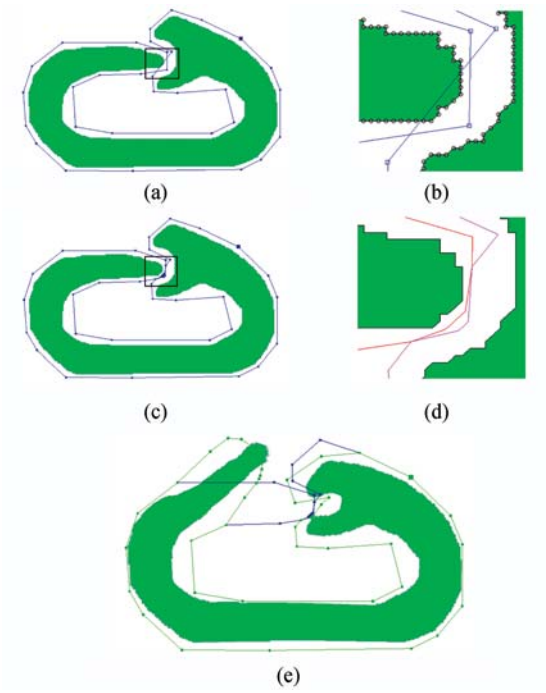


Fig.8. Removing self-intersections of a coarse cage of a 2D model (green). (a) The coarse cage (blue). (c) The cage after CSIR step. (b)/(d) The intersecting areas before/after the CSIR step. (e) Deforming the model with the green cage.

The comparisons among Xian's[17], Ben-Chen's[15] and our method are shown in Fig.1 and Fig.10. The models used in the two figures have various characteristics. The Dilo is an articulation model, the duck is

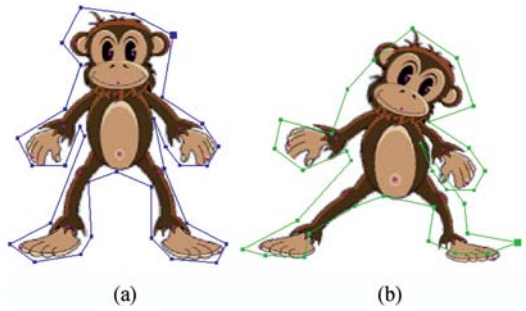


Fig.9. Using the cage built with S_c and S_n to do deformations. (a) Original monkey and cage. (b) Deformed monkey and cage.

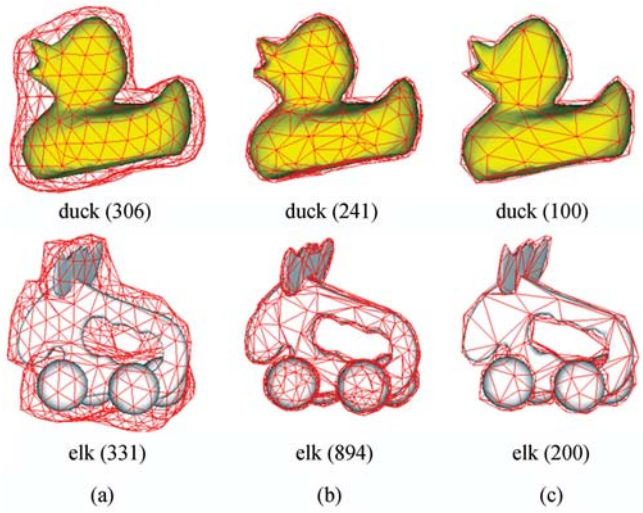


Fig.10. Comparisons of Xian's, Ben-Chen's and our results. (a) Xian's. (b) Ben-Chen's. (c) Ours. (The number in parentheses is the vertex amount for the corresponding cage.)

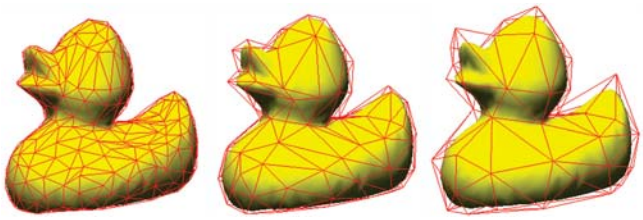


Fig.11. Building the duck cages with 306, 100, 60 vertices, respectively.

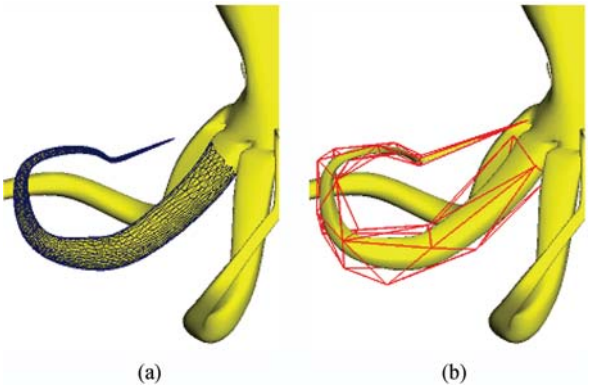


Fig.12. Building a partial cage. (a) Select one tentacle of an octopus. (b) The partial cage built by our method.

smooth, and the elk has a hole. As illustrated in Fig.1, it is nearly impossible to erect the Dilo's horn with the cage built by Xian's method. With the cage built by Ben-Chen's method, it deforms the Dilo's back dramatically at the same time. However, with the cage built by our method, it becomes easy. As illustrated in Fig.10, our method can use fewer vertices to build the cages.

The more similar the cages are to the original models, the more easily the users can manipulate the models. To compare the similarity of the cages built by the three methods, the well-known public domain Metro Tool[23-24] is employed, see Table 1. The symmetric Hausdorff distance and RMS error are widely used among graphics community.

Table 1. Using Metro Tool[24] to Compare the Cages Built by Xian's, Ben-Chen's (shortened by BC) and Ours Respectively

Model (vmnt)	Method (cage vmnt)	Metro Max/Mean Distance	Metro RMS	Metro Hausdorff
Dilo	BC (288)	0.041280/0.020043	0.020822	0.052654
(5524)	Xian (173)	0.126841/0.062284	0.066985	0.145208
	Ours (288)	0.019067/0.004937	0.005633	0.019067
	Ours (183)	0.034892/0.009145	0.010084	0.034892
Duck	BC (241)	3.524210/1.566573	1.647625	5.038621
(1000)	Xian (306)	13.592529/6.596735	6.805474	16.905624
	Ours (306)	1.843838/0.352616	0.415289	1.843838
	Ours (241)	2.018532/0.481728	0.556542	2.018532
Elk	BC (894)	5.229759/2.182084	2.253242	10.983293
(1250)	Xian (331)	27.702206/13.317059	13.992528	36.251770
	Ours (331)	5.984372/1.025486	1.268907	5.984372
	Ours (200)	9.717106/1.944199	2.352656	9.717106

Note: vmnt — vertex amount.

The comparison results show that, with the same vertex amount, the Hausdorff distances by our method are always smaller than those by the other two, and both the max and the mean distances are smaller too.

The cage's vertex amount is a trade-off between the manipulation's simplicity and delicateness: with fewer vertexes, the cages make the operation simple, but could not manipulate details; with more vertices, the cages make the operation complicated, but could adjust details. With our method, users can control the cage's vertex amount just by specifying a number, based on their applications, see Fig.11.

Because our method is local, i.e., each replacing is based on the area of *EU*, it can be flexibly used to build partial and/or entire cages (see Fig.12).

As illustrated in the previous figures, our method can build cages for both 2D polygons and 3D triangular meshes. The performance is illustrated in Table 2. The data are obtained by executing our program with single thread on the platform of Windows XP SP3, Intel Core 2 Duo CPU E7400@2.80 GHz, 3.46 GB memory, Intel G41 Express Chipset. We would like to mention that our program is based on the Graphite platform②, and the CGAL library③ is applied to the quadratic programming.

Table 2. Elapsing Time of the Simplification

Model (vmnt/emnt)	Cage (vmnt)	Build Heap (s)	Simplifying (s)	Per Edge (ms)
Dilo	288	28.419	105.854	20.216
(5524/16566)	256	28.524	107.181	20.346
Duck	306	5.088	14.357	20.687
(1000/2994)	241	5.085	15.583	20.531
Elk	331	6.622	18.904	20.570
(1250/3750)	200	6.598	21.174	20.166

Note: vmnt — vertex amount, emnt — edge amount.

Because every replacing involves multiple quadratic programmings, the simplification step is time consuming. The time elapsed in the CSIR step is much shorter than that in the simplification step. To resolve an intersecting area, with 32 vertices and 32 faces, costs about 0.03 seconds. And the CSIR step often needs not be applied. Since cage building is only one-time work, it is still acceptable even though our automatic method is a little time consuming.

5 Conclusions

We have presented a method of building cages for both 2D polygons and 3D triangular meshes. Our method consists of two steps: simplification step and CSIR step. The simplification step iteratively replaces

②http://alice.loria.fr/index.php/software/3-platform/22-graphite.html
③http://www.cgal.org/

EU by *VU* with QEM, while it solves quadratic programmings to obtain the replacing vertex. The simplification step eventually builds a coarse cage that envelops the input model. Self-intersections of the coarse cage are removed in the CSIR step through Delaunay triangulation/tessellation. With our method, users can build a cage easily to envelop a dense model either entirely or partially only by specifying an approximate vertex number according to the application needs. Our experimental results demonstrate the effectiveness of our method on a variety of models.

Up to now our result cages cannot reflect all joints of the input models exactly, so do the other two previous methods. This is considered as a limitation of the current auto cage-building methods. With such cases, it is a challenge to create realistic deformations. Another limitation is that the edges of a cage are not always related to the expected deformations that a user wants. In the future work, we would like to apply the joints of the input model or curves sketched by users, to overcome these limitations.

Acknowledgement We would like to thank Chuhua Xian for providing the cages generated by their method. We also thank Xiquan Shi, Dingyuan Liu, Zheng Li and Sixuan Zhong for some valuable comments on the early versions of the paper. The elk, duck, Dilo and octopus models are taken from the AIM@SHAPE shape repository.

References

[1] Wei J, Lou Y. Feature preserving mesh simplification using feature sensitive metric. *Journal of Computer Science and Technology*, 2010, 25(3): 595-605.

[2] Garland M, Heckbert P S. Surface simplification using quadric error metrics. In *Proc. the 24th Annual Conference on Computer Graphics and Interactive Techniques*, Los Angeles, USA, Aug. 3-8, 1997, pp.209-216.

[3] Garland M, Heckbert P S. Simplifying surfaces with color and texture using quadric error metrics. In *Proc. the Conference on Visualization 1998*, Research Triangle Park, USA, Oct. 18-23, 1998, pp.263-269.

[4] Hoppe H. New quadric metric for simplifying meshes with appearance attributes. In *Proc. the Conference on Visualization 1999: Celebrating Ten Years*, San Francisco, USA, Oct. 24-29, 1999, pp.59-66.

[5] DeCoro C, Tatarchuk N. Real-time mesh simplification using the GPU. In *Proc. Symposium on Interactive 3D Graphics* (*I3D*), Seattle, USA, Apr. 30-May 2, 2007, pp.161-166.

[6] Ju T, Schaefer S, Warren J. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics*, 2005, 24(3): 561-566.

[7] Floater M S. Mean value coordinates. *Computer Aided Geometric Design*, 2003, 20(1): 19-27.

[8] Joshi P, Meyer M, DeRose T, Green B, Sanocki T. Harmonic coordinates for character articulation. In *Proc. the ACM SIGGRAPH 2007*, San Diego, USA, Aug. 4, 2007, Article No.71.

[9] Lipman Y, Levin D, Cohen-Or D. Green coordinates. *ACM Transactions on Graphics*, 2008, 27(3): 1-10.

[10] Ben-Chen M, Weber O, Gotsman C. Variational harmonic maps for space deformation. *ACM Transactions on Graphics*, 2009, 28(3): 1-11.

[11] Weber O, Ben-Chen M, Gotsman C. Complex barycentric coordinates with applications to planar shape deformation. *Computer Graphics Forum*, 2009, 28(2): 587-597.

[12] Cohen-Or D. Space deformations, surface deformations and the opportunities in-between. *Journal of Computer Science and Technology*, 2009, 24(1): 2-5.

[13] Ju T, Zhou Q, Panne M, Cohen-Or D, Neumann U. Reusable skinning templates using cage-based deformations. *ACM Transactions on Graphics*, 2008, 27(5): 1-10.

[14] Huang J, Chen L, Liu X, Bao H. Efficient mesh deformation using tetrahedron control mesh. *Computer Aided Geometric Design*, 2009, 26(6): 617-626.

[15] Ben-Chen M, Weber O, Gotsman C. Spatial deformation transfer. In *Proc. the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, New Orleans, USA, Aug. 1-2, 2009, pp.67-74.

[16] Chen L, Huang J, Sun H, Bao H. Technical section: Cage-based deformation transfer. *Computers and Graphics*, 2010, 34(2): 107-118.

[17] Xian C, Lin H, Gao S. Automatic generation of coarse bounding cages from dense meshes. In *Proc. the IEEE International Conference on Shape Modeling and Applications* (*SMI*), Beijing, China, Jun. 26-28, 2009, pp.21-27.

[18] Jung W, Shin H, Choi B K. Self-intersection removal in triangular mesh offsetting. *Computer-Aided Design and Applications*, 2004, 1(1-4): 477-484.

[19] Cohen J, Varshney A, Manocha D, Turk G, Weber H, Agarwal P, Brooks F, Wright W. Simplification envelopes. In *Proc. the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, New Orleans, USA, Aug. 4-9, 1996, pp.119-128.

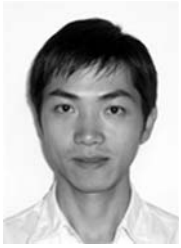
[20] Kazhdan M, Bolitho M, Hoppe H. Poisson surface reconstruction. In *Proc. the Fourth Eurographics Symposium on Geometry Processing*, Cagliari, Italy, Jun. 26-28, 2006, pp.61-70.

[21] Hussain M. Efficient simplification methods for generating high quality LODs of 3D meshes. *Journal of Computer Science and Technology*, 2009, 24(3): 604-613.

[22] Alexa M. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, 2003, 9(2): 105-114.

[23] Cignoni P, Rocchini C, Scopigno R. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 1998, 17(2): 167-174.

[24] Cignoni P. Metro. http://vcg.sourceforge.net/index.php/Metro. May 12, 2008 (cited on Sep. 13, 2010).



Zheng-Jie Deng is a Ph.D. candidate under the supervision of Prof. Xiaonan Luo in School of Information Science and Technology, Sun Yat-Sen University. His research interests include computer animations, graphics deformations and modeling.

Xiao-Nan Luo is a professor and Ph.D. supervisor of School of Information Science and Technology and the Chairman of Research Institute of Sun Yat-Sen University. His research interests include mobile computing, computer graphics & CAD and 3D CAD. He was granted the government special allowance by the State Council of China, and won the National Science & Technology Progress Prize awarded by the Ministry of Science and Technology of China and the National Science Fund for Distinguished Young Scholars granted by the National Natural Science Foundation of China.

Xiao-Ping Miao received her Ph.D. degree from School of Computing, National University of Singapore in 2009. She is currently a postdoc in School of Software, Sun Yat-Sen University. Her research interests include image processing and graphics modeling.