Efficient Cage Generation by Region Decomposition

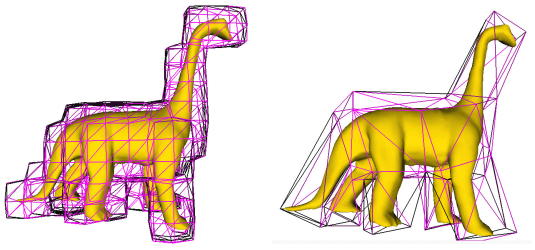
Abstract

Cage-based deformation has become popular method for shape deformation. To editing a shape, we need to build a cage to envelop the target model and this is a tedious work by manual work. In this paper, we develop an automatic method to generate the cage for a model using voxelization based decomposition. We first voxelize the input model, then using the seed filling algorithm to group the inner voxels. By dilating the inner voxel groups, we decompose the model into broad parts and narrow parts. Then we construct partial cages using different strategies and union them to get a cage. Experiment results demonstrate that our method is effective , efficient and robust to model transformation.

Keywords: cage generation, voxelization, region decomposition, mesh deformation

1 Introduction

With the advantages of simplicity, flexibility and speed, space-based deformation techniques are popular in recent years. The major feature of these techniques is to deform the objective model which is embeded in the ambient space [1, 2, 3, 4, 5, 6]. Space-based deformation can be applied to a polygonal mesh, point cloud or volumetric data, but it need to build a coarse cage or an embed graph [7] , then manipulate the cage or the embed graph to control the model. Although the cage plays an important role in the cage-based deformation, it is mainly constructed by manual currently or by subdividing a bounding box of the model [2]. However, the construction process is very tedious and time-consuming. And more serously, if the input model is complex, it is very hard or impossible to build such a cage. Therefore, it is meaningful to study how to generate a coarse cage from a complex model automatically.



(a) (b)

Figure 1: The cages of the dino model. (a)Cage generated using the method in [8]. It is too dense for manipulate with 500 vertices . (b)The cage of the horse generated by the proposed method and it has only 58 vertices.

Although many cage-based deformation techniques have been developed currently, there are few works about how to generate a cage from a given model. Guiding by the skeleton of the model, in [9], Ju et al. propose a cage generation method by piecing pre-defined templates. The major limitation of this method is that it need to build the template library. In [8], Xian et al. describe an automatically cage generation method based on voxelization. The main feature of [8] is that it extract the outerside of the surface voxels to get a coarse bounding cage and then smooth it.The drawback is that it is hard to control the size of the voxel and the generated cage is usually too dense for employing the cage-based deformation.

In this paper, we propose to use the voxelization-based method to decompose the input model into the broad parts and the scattered narrow parts. The partial cages for the broad parts are extracted from the surface vox-

els bounding them. And we use the *Oriented Bounding Box* (abbr. **OBB**) [10] to bound the scattered narrow parts and extracted the outer surface. Then we operate the boolean operation to union these partial cages . As described in [10], using the **OBB** can make the part cage for the scattered narrow parts of the model independent of the orientation, and it is robust for the model transformation.

1.1 Contribution

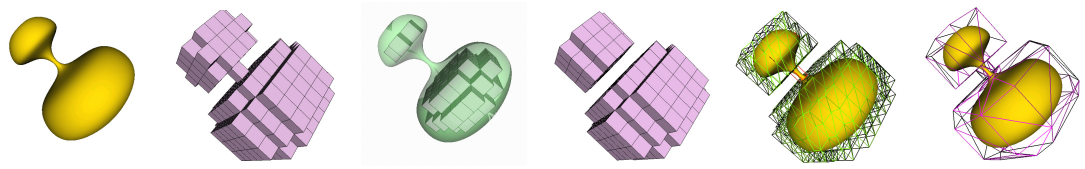
Our main contribution is developing a robust and very efficient method to generate a cage from a given model automatically, which provides many advantages over existing methods. The automatic process reduces many interaction operations by the user. We first voxelize the model with a proper size to divide the model into several parts. Since the voxel size is often large, the voxelization process is very fast. Not like the method in [8], we only extract the outside surface voxel from the broard parts. The remainder parts are bounded by **OBBs**. And finally, we union all the extracted mesh of the surface voxels on the major parts and **OBBs** of the scattered parts. The size of the voxel can be spcified by the user or it can be determined by iterative process using the guiding function. As the broad parts capture the main features of the model, and the **OBBs** catch the reminder details, our method can generate a proper coarse cage. We also try to develop a function to evaluate the cage. In addition, our method is quite simple to implementation and robust.

2 Background and Related Work

Space-based deformation techniques were first introduced into computer graphics by Sederberg and Parry in 1986 [11] and developed by Coquillart [12]. MacCracken and Joy extended it to arbitrary topology [13]. The major feature of these *Free-From deformation* techniques is construct a lattice which has a small number of control points. Then deform the model smoothly by manipulating the control points. The early FFD techniques used the regular lattice which can be constructed in a simple way, but as pointed out in [2], these methods make the deformation less flexible.

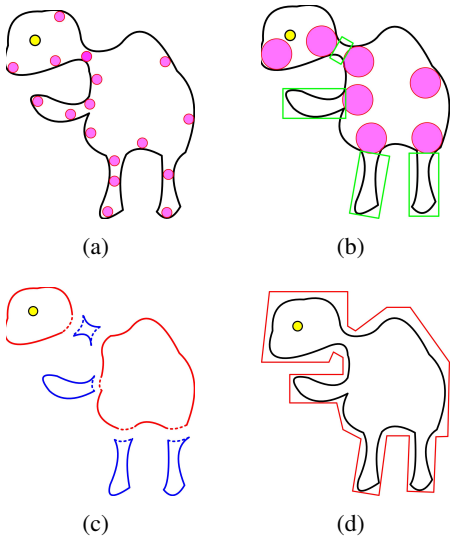
In recent ten years, significant work has been done on searching for a more general control polyhedron that enclose the model tightly and has more degrees of freedom. Ju [1] and Floater et al. [14] extended the Mean Value Coordinate from the 2D polygon [15] into 3D polyhedron and based on these coordinates Ju et al. [1] introduced a deformation method. The polyhedron can be regarded as ”cage”, so we call it ”cage-based” technique. Since cage-based methods is defined as the linear combination of the vertices of the target cage with a set of *coordinate functions* and these coordinates can be pre-computed, they are efficient in interactive deformation. And compared with the FFD techniques, these methods are more flexible. In the following years, many works focus on the*coordinates functions* for cage-based deformation techniques [2, 16, 4, 5, 17, 18, 6]. The object of these methods is to achieve smooth and shape-prevation deformation. In most of cage-based methods, the cage plays an important role for the deformation. A desirable cage is convenient for manipulation. However, most cages are constructed by manual, and there are few literatures about how to generate the cage automatically. Although building a bounding box and subdividing it [2] can get a coarse cage, it does not always work if the model is not simple.

Ju et al. [9] proposed cage-based deformation method to by reusing skinning templates, where the outside cage is constructed by piecing together the pre-defined templates. Although this method can generate cage automatically for character model, it needs the users provide the templates guiding by the skeleton of the model, and sometimes it may generates artifacts and need manual correction. Xian et al. [8] proposed a voxelization-based method to generate a coarse cage from a dense mesh. This method voxelizes the model first, then extracts outer surfaces of the feature voxels and optimizes it to generate a coarse cage. However, since the voxels are regular shape, they can not catch many detail features of the model, and the generated cage is often too dense for cage-based deformation.



(a) (b) (c) (d) (e) (f)

Figure 3: The process of the automatic cage generation(a)The input model. (b)The voxelization of the model.(c)Two inner voxel groups. (d)The extending groups of the two inner groups. (e)The initial cages for the two main parts. The "bottle-neck" part is bouned by an OBB in red. (f)The resultant cage after union operation and cage refinement.



(a) (b) (c) (d)

Figure 2: Cage generation for a 2D model. Using a circle with a small radius can round the whole model(a), while using a proper radius(b) can decompose the model into parts(c): the main parts in red and the "bottle-neck" parts in blue. (d) Shows the cage of the model by constructing the partial cages for each part with different strategies and union them.

3 Cage Generation Algorithm

Given a mesh model(or point cloud, volume data), human visual system departs it into major silhouette and slight parts by the concave crease [19]. As shown in Figure. 2(a), for a 2D model, if we use a circle with a small radius rolling inside the model, it can run around the whole model. But as the radius increasing, there will be some narrow areas that the rolling circle can not pass(see Figure. 2(b)). These "bottle-neck" areas decompose the model into main parts and the "bottle-neck" parts (the box in green in Figure. 2(c)). Obviously, the main parts catch the major parts of the model while the "bottle-neck" parts reflect the geometry protrusion and make main parts disconnected. Following this idea, similar with the case in 2D, we can use a rolling ball with a proper radius to decompose the 3D model. There are many methods to compute the rolling ball [20]. However, the computation of the rolling ball is rather tedious and time-consuming if the input model is complex, using an approximate but effective method is more practical.

In this section, we begin by describing our decomposition method based on voxelization, and then we will show how to build the partial cages for the parts using different strategies. Once the cage of every part is constructed, we will union them by boolean operation. Extension to the voxel's size determination will be also discussed.

The main steps of the proposed cage generation algorithm are listed as follows:

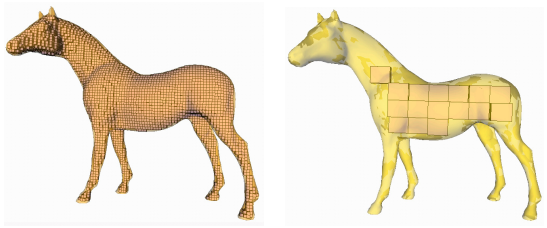
1: Voxelize the input model with a specified voxel size, decompose the model into main parts and narrow parts.
2: Generate partial cages by extracting the outer surface of the main groups with the feature voxels while using the OBBs to bound the scattered narrow parts, then triangulate them.
3: Merge all the partial cages by boolean operation.
4: Refine the cage.

The voxelization process is almost the same as the method in [8]. We will detail the decom-

position step, step 2 and step 4 in the following sections.

3.1 Model Decomposition Based on Voxelization

As described in [21, 22], the voxelization is an approximate representation of the model, and the voxels can be classified into *surface voxel*, *inner voxel* and the *outer voxel* [8]. For a closed 2D manifold model, the inner voxels can be regarded as the filling elements(see in Figure. 4). And if the voxel size is small enough, the innver voxels can fill almost the inside of the model(Figure. 4(a)). But as the voxel size increasing, some narrow areas only contains surface voxels and these areas make the model into *broad parts* and *narrow parts*. Figure. 4(b) shows the inner voxels with a large size only fill the body of the horse. Base on this property, we use the voxelization to approximate the rolling ball method.



(a) (b)

Figure 4: Voxelizing the horse model with different size of the voxel.(a) The inner voxels with a small size can fill the whole model, while (b)voxels with large size can not fill the legs and the head of the horse.

Let M be an input model represented by point cloud or polygon mesh. To voxelize M, we first calculate the OBB O using the Principle Component Analysis(PCA) to the input vertices. Then the OBB O is divided into voxels with a specified resolution or voxel size. In this paper, we use the method in [23] to voxelize the model. As the object of the voxelization is to decompose the model into parts, the size of the voxel is usually large. Since the points of the model are usually dense, we can also use the sampling method to accelerate the voxelization process.

Once the OBB O is divided into voxels, we use the scan-conversion algorithm [8] to classify the voxels into inner voxel and outer voxel. Then the inner voxels can be decomposed into disconnected groups using the seed filling algorithm. We use $G = \{g_i\}$ to record these inner groups. It should point out that the adjacent relations of the voxel using in the filling process is 26 neighborhood, including face-adjacency, edge-adjacency and vertex adjacency.

It can be seen that the voxels in g_i are not intersect with the model(see Figure 4 and Figure. 3(c)). To decompose the model using $G = \{g_i\}$, it needs to extend the inner voxels. For each g_i in G, we extend its outer most layer voxels to get a new group g_i^m through dilation. As g_i is the maximal connected set of the inner voxel, the dilation voxels in g_i^m of the outer most layer of g_i are all surface voxels, and these voxels will envelop g_i. For two different inner groups g_j, g_k and $j \neq k$, their corresponding dilation group may have same surface voxel, that means $g_j^m \cap g_k^m \neq \emptyset$. In this case, we merge these groups into a new one. Then we group the vertices inside the surface voxels of g_i^m and then get the broad parts of the model. We use P_i^b to represented the corresponding broad part for g_i^m. Figure. 3(d) shows the two surface voxel groups dilating from Figure. 3(c). We can see that these two disconnected groups contain most vertices of the input model.

Further, there are some surface voxels are not in $\{g_i^m\}$, and these voxels containing the vertices constitute the narrow parts. Like the method of getting getting $\{g_i\}$, the vertices of the narrow parts can be grouped using the seed filling algorithm. We use $\{P_i^n\}$ to represent the narrow parts, and $P_i^n \cap P_j^n = \emptyset$ if $i \neq j$.

Figure. 5(a) and 5(b) show an example of the decomposition. The screw driver model is decomposed into two parts with proper voxel size, and these two parts represent two different features of the model.

3.2 Partial Cage Construction

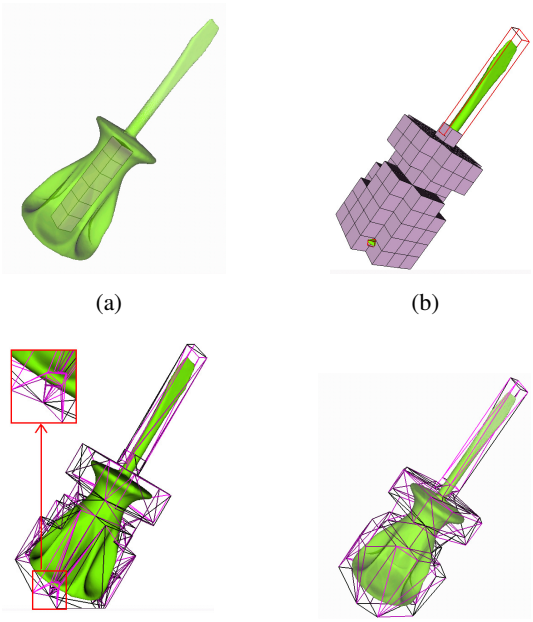
After decomposing the model and extracting all broad parts $\{P_i^b\}$ and narrow parts $\{P_i^n\}$, the next step is to construct the partial cage these parts using different strategies.

For the broad parts, we build the partial cage for P_i^b by extracting the outer surfaces of the voxels in g_i^m, just like the method in [8]. S-

ince g_i^m dilates from the inner voxels using the 26-neighborhood relationship, it does not contain the non-manifold connection cases. In [8], all outside surfaces of the surface voxel form a closed surface. But some voxels in g_i^m here may not be adjacent with the outside voxels as their neighbors are only inner and surface voxels. To ensure to get a closed cage from each g_i^m, the surface voxels of the narrow parts are signed as the outer voxels after the decomposition process.

To construct the partial cage for a narrow part, we first use an OBB [10] O_i to bound the vertices group P_i^n and then triangulate each outer face of O_i. We use the position of the vertices as the inputs, then the Principle Component analysis (PCA) method is applied to calculate the three major direction of OBB O_i and construct it.

As show in Figure. 3(e), the partial cages in green are extracted from the dilation groups in Figure. 3(d) while the narrow part bounded in an OBB is in red.



(a) (b) (c) (d)

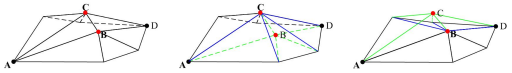
Figure 5: (a)The screw driver model and its inner voxels. (b)The inner voxels decompose the screw driver into several parts. (c)The initial cage. (d)The finnal cage after refinement.

3.3 Cages Generation and Refinement

After all partial cages for $\{P_i^b\}$ and $\{P_i^n\}$ are constructed, we employ CGAL [24] to perform the Boolean union operations to merge them into a whole cage. In CGAL, the mesh is converted to the intermediate model *Nef_polyhedron*, then the union operation is performed by the overload *plus* operator. And finally, it use the function *convert_to_Polyhedron* to convert the intermediate model to a surface mesh.

Usually, the quality of the initial coarse cage generated after the union operation is ugly since it may induce many narrow needle triangles and scraggly close vertices. As analyzed in [4], the more vertices and faces of the cage the slower for the editing. More ever, although the mesh quality of the cage has not obvious affects for the model editing [6], the more vertices the more difficult to manipulate the deformation. Figure. 5(c) shows an initial cage generated by simply union the partial cages. We can see that the quality of the cage is bad, and it has many close vertices which make it is not proper for directly interactive deformation .

To improve the mesh quality of the cage, we first merge the co-planar faces generated from the outside surface of the feature voxels. This merging process is quite simple, and the embedded space of the cage will not change since the merging faces are co-planar.



(a) (b) (c)

Figure 6: Merge the close vertices.(a)B and C are two vertices and their distance d_{BC} is less than a threshold δ. (b)Valid merging : move B to C, as B is inside the newly generated triangles. (c)Invalid merging : move C to B, as C is outside the newly generated triangles.

Next, we find out the close adjacent vertices and we clean up them by vertex merging(Figure. 6). Suppose B and C are two closed adjacent vertices, and there two manners to merge B, C. One move the vertex B to C, where B is the merged vertex and C is the fixed

vertex. The other is to reverse this operation on B and C. To guarantee the cage still envelop the original model, it need to perform further checking when merging the vertices. That is , if the merged vertex is inside all the newly generated triangles, the merging operation is valid (see Figure. 6(b)). Otherwise, if the merged vertex is outside one or more newly generated triangles, the operation is invalid (see Figure. 6(c)). In this way, the cage can be guaranteed that it does not intersect with the original model. In our implementation, we take the voxel size as the threshold δ to determine the close vertices for merging process.

3.4 Automatic Voxel Size Determination

In the cage generation process, the voxel size can be specified by user. Different size will generate different cage. In order to get a good cage, it may take a user without experience to try several times. In this section, we give a general solution to compute the adapted voxel size automatically.

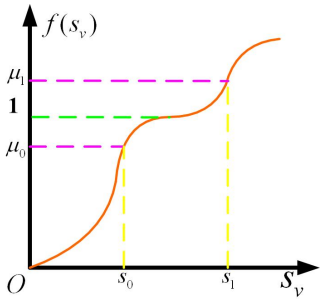


Figure 7: The guidance function

We first define a guidance function $f(s_v)$ as following:

$$f(s_v) = \frac{\frac{1}{n}\sum_i^n s_o}{s_v} \quad (1)$$

where s_o is minimal size of the edges of each OBB $\{O_i\}$, n is the number of the OBBs, and s_v is the size of the voxels. This guidance function can be interpreted qualifiedly as Figure. 7. That means if the voxel size is small, the inner voxels will almost fill the inner space of the model, and there is no generated OBBs, or the edge size of the OBB is very small. As the voxel size increasing, the volume of the narrow part will also expand, and meanwhile the edge size of the corresponding OBB increase. When the value of the guidance function $f(s_v) \in [\mu_0, \mu_1](\mu_0 < 1 < \mu_1)$, the voxel size approximately equal to the average size of the shortest edges of OBBs, which means the inner voxels can not pass through the narrow parts of the model. Base on this property, we design an algorithm to compute the adaptive voxel size automatically. This process can be summarized as Algorithm 1.

Input: :
pre-defined interval $[\mu_0, \mu_1]$. **Output**: :
voxel size s_v. Set s_c a large enough size,
s_f a small size, $s_v \leftarrow s_f$. **while** *true* **do**
  Voxelize the model, decompose it,
  extract the OBBs and compute
  $f(s_v)$;
  if $\mu_0 \leq f(s_v) \leq \mu_1$ **then**
    return s_v;
  end
  else if $f(s_v) < \mu_0$ **then**
    $s_f \leftarrow s_v$;
  end
  else if $f(s_v) > \mu_1$ **then**
    $s_c \leftarrow s_v$;
  end
  $s_v \leftarrow \frac{s_c + s_f}{2}$;
end

Algorithm 1: Pesudo-code for automatic voxel size determination.

4 Cage Evaluation

Since the cage is an coarse mesh to describe the shape of the model, it is improper to use the traditional rules to evaluate its quality, such as curvature [25], reflection lines [26], or specular shading. More ever, different applications require different cages. In [8], Xian et al. introduce the sparse factor to determine the initial resolution for voxelization. Actually, the sparse factor is an important indicator, and it is apparently not enough for cage evaluation. In this section, we try to give a criterion to evaluate the cage.

For a generated cage C of a model M, we define the evaluation function E_c as

$$E_c = (1 - \frac{N_c}{N_m})e^{1-\frac{V_c}{V_m}} \cdot S(C, M) \quad (2)$$

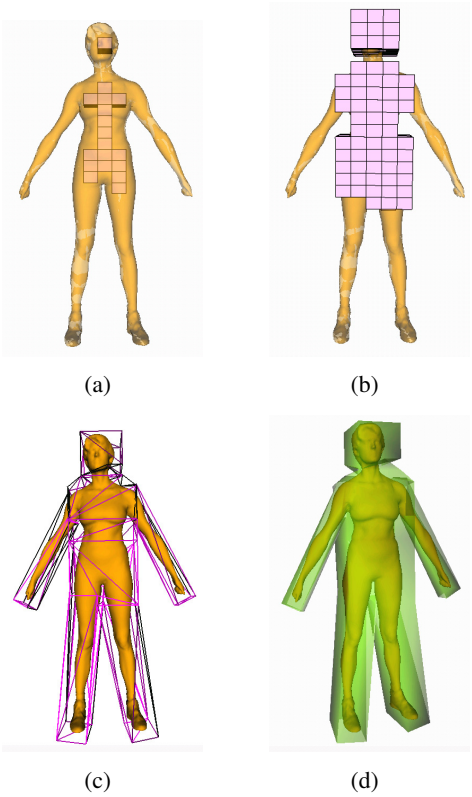


Figure 8: The woman model. (a)The inner voxels of the voxel. (b)The dilation voxels decompose the woman model into several parts. (c)The final cage. (d)The cage envelops the original woman model.

where N_c and N_m are the numbers of the vertices of C and M respectively, V_c is the volume of C while V_c is that of M, $S(C, M)$ is a function related with the application and $0 \leq S(C, M) \leq 1$.

From Equation. (2), we can see that the first term $(1 - \frac{N_c}{N_m})$ reflects the sparseness, and the second item $e^{1-\frac{V_c}{V_m}}$ reflects the tightness of the cage. For different applications, $S(C, M)$ can be defined in different ways. In this paper, we define $S(C, M)$ as the shape similarity assessment and we use the method in [27] to compute it.

5 Results and Discussion

We have implemented the proposed method with Visual C++ 2008 and OpenGL, and runs it on a PC with Dual Core 2^{TM} 2.3 GHZ CPU and 4GB memory in a single thread. Figure. 11 shows the automatic generated cages of some models.

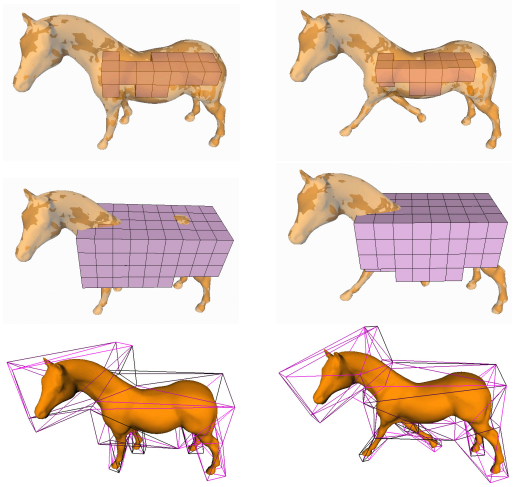


Figure 9: Horse model. The *left* shows the cage generated from the original model, while the *right* shows the cage for the model with little deformation.

Figure. 5(d) shows the cage for the screw driver after merging the close vertices. We can see that the cage capture the major features of the model. In Figure. 8, the outer voxels dilating from the inner voxel in Figure. 8(a) depart the arms and legs from its body successfully and base on this decomposition, we construct a proper cage for it and this cage envelop the target model entirely.

In Figure. 11, the dinosaur model has two inner groups. But after dilating these two groups, their surface voxels merge into only one group and decompose the dinosaur into body part, leg parts, arm parts, tail part and head part. While it is tedious to construct the partial cage for some areas of some animals, in Figure. 11, the cages of bunny and armadillo show that our method can successfully do these for the ears of these two models.

We compared the proposed method with [8]. In Figure. 1, we can see that our method can generate a more coarse cage , while the cage generated by the method in [8] is too dense to manipulate.

Our method is also robust to the model transform. Since our method use dilation voxels from the inner voxels to decompose the model, if we only manipulate the narrow parts of the model,

Table 1: Data of the experiment results

Model	Model Verts.	Cage Verts.	Resolution	Voxel size	Timing(in second)			E_c
					T_v	T_u	T_t	
Dino(Figure. 1)	5903	58	—	0.1	0.251	1.400	1.820	0.61
Screwdriver	27152	70	$5 \times 5 \times 16$	—	0.118	0.818	0.992	0.83
Woman	25172	67	$11 \times 31 \times 5$	—	0.115	0.342	0.359	0.73
Horse(original)	19851	57	—	0.081	0.029	1.065	1.118	0.67
Horse(deformed)	19851	67	—	0.081	0.029	0.901	0.953	0.67
Bunny	34835	94	$12 \times 12 \times 12$	—	0.152	1.486	1.731	0.86
Cow	2904	65	$9 \times 7 \times 3$	—	0.014	0.861	0.892	0.65
Armadillo	130838	139	$12 \times 21 \times 7$	—	0.499	2.881	3.601	0.82
Dinosaur	56194	115	$24 \times 31 \times 7$	—	0.260	6.680	7.057	0.72

the broad parts will change a little. That will not induce much change of the inner space and arrangement of the inner voxel. So the cage will not change a lot. Figure. 9 shows the cages of the horse model before and after deformation. And the result validate this conclusion. More ever, the voxel size of these two model in Figure. 9 is determined automatically using the algorithm. 1. We set $\mu_0 = 0.8$ and $\mu_1 = 1.2$ as the interval.

In Table. 1, we show the data of the experiment results. We show the time it takes to generate the cage. T_t is the total time for the whole cage generation process. T_v and T_u are the time for voxelization and union operation respectively. E_c is computed by the Equation. 2. From this table, we can see that the time mainly consume on the union step and the cage generation method proposed in this paper is efficient. Further more, as shown in the evaluation value, we can see that our method can constructive a good cage.

Discussion: the basic foundation for using the connected inner voxels to decompose the model into parts is that it assume the model contains constituted by broad and narrow parts. But if a model does not satisfy this condition, such as the Torus model in Figure. 10, it can not distinguish these parts and no OBB is generated. In this case, the proposed method will degenerate to the method in [8].

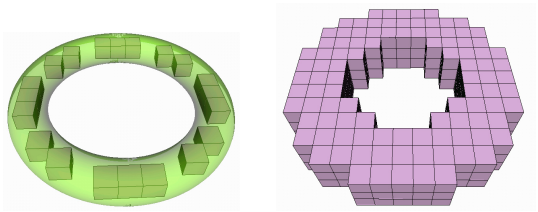


Figure 10: The inner and dilation voxels of the torus. There is no OBB generated from the model.

6 Conclusions

We have proposed a method to generate a cage automatically. For a input model, we assume it consists of broad and narrow parts, and we first use a specified size of voxel (or resolution) to voxelize it. Then the inner voxels are grouped by the 26-neighborhood connectivity and dilate to surface voxels to decompose the model into parts. Next, We construct the partial cages for the broad parts by extracting the outer faces of the surface voxels, and for the narrow parts, we use the OBBs to bound each and triangulate outside surface. Finally, all partial cages are unironed to a cage and we refine it by some rules. Further, we also give an automatic solution to determine the voxel size. Moreover, we discuss how to evaluate the cage and try to give an evaluation function. Our method work well for the model satisfies the broad-narrow consecution and the generated cages can be applied to cage-based deformation or subdivision surface fitting . Further, experiment shows that our method is robust to the model transformation. In our future work, we will study the work for generating cage for such models that do not have obvious broad and narrow parts.

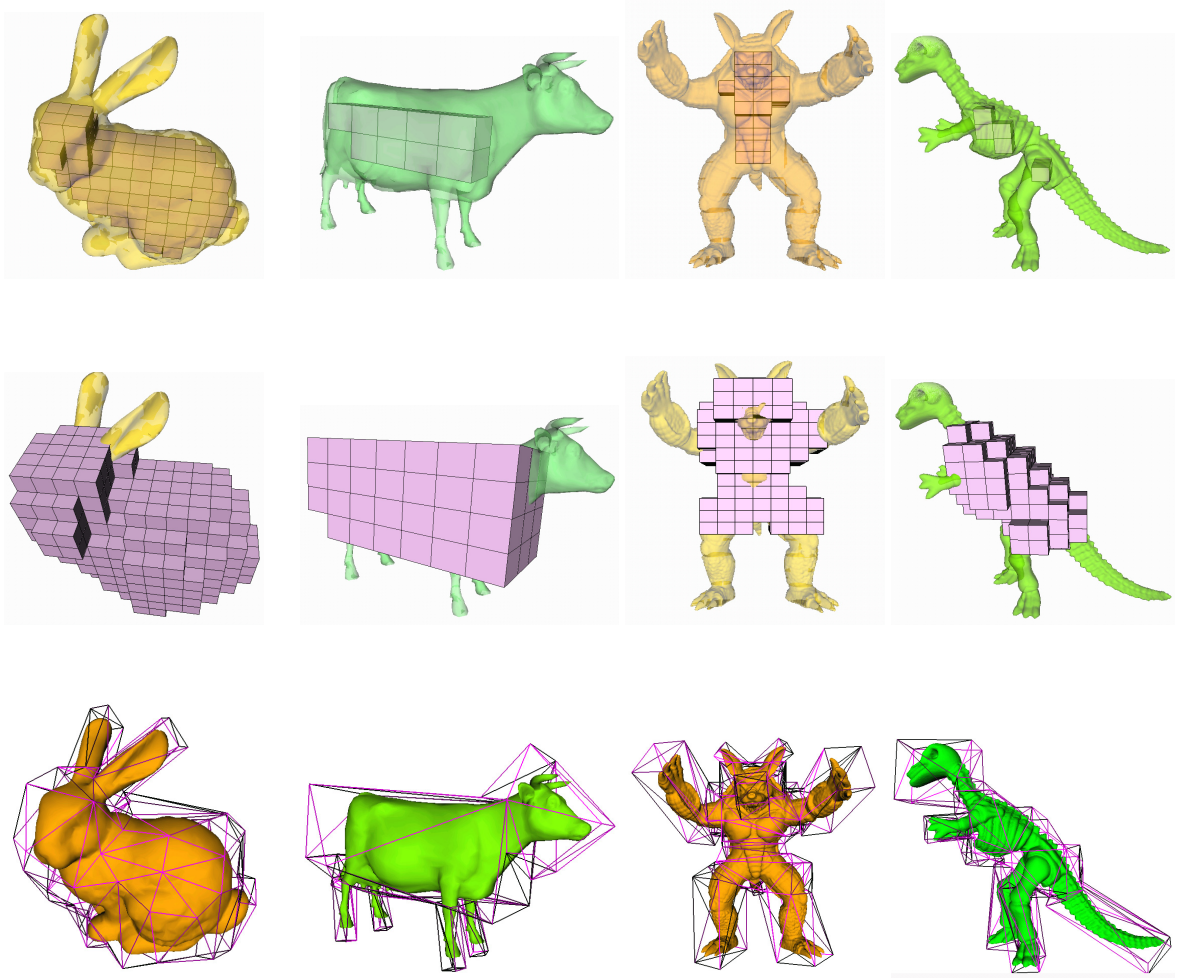


Figure 11: Results. The first layer shows the inner voxels of the models, while the second shows the dilation surface voxels in layer 1. The final cages are shown in the third layer

References

[1] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. In *Proceedings of SIGGRAPH 2005*, pages 561–566, 2005.

[2] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. *ACM Transaction on Graphics*, 26(3), July 2007.

[3] M. Botsch, M. Pauly, M. Wicke, and M. Gross. Adaptive space deformations based on rigid cells. *Computer Graphics Forum*, 26(3):339–347, 2007.

[4] Y. Lipman, D. Levin, and D. Cohen-Or. Green coordinates. *ACM Transaction on Graphics(TOG)*, 27(3), August 2008.

[5] J. Huang, L. Chen, X. Liu, and H. Bao. Efficient mesh deformation using tetrandron control mesh. In *Proceedings of ACM Solid and Physical Modeling 2008*, pages 241–247, 2008.

[6] M. Ben-Chen, O. Weber, and C. Gotsman. Variational harmonic maps for space deformation. *ACM Transactions on Graphics (TOG)*, 28(3):34, 2009.

[7] RW Summer, J. Schmid, and M. Pauly. Embedded deformation for shape manipulation. *ACM Transactions on Graphics*, 26(3):80, 2007.

[8] Xian Chuhua, Lin Hongwei, and Gao Shuming. Automatic generation of coarse bounding cages from dense meshes. In *IEEE International Conference of Shape*

Modeling and Applications (SMI) 2009, Beijing, June 2009.

[9] Tao Ju, Qian-Yi Zhou, Michiel van de Panne, Daniel Cohen-Or, and Ulrich Neumann. Reusable skinning templates using cage-based deformations. *ACM Transaction on Graphics (TOG)*, 26(5), 2008.

[10] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, New York, NY, USA, 1996. ACM.

[11] T.W. Sederberg and S.R. Parry. Free-form deformation of solid geometric models. *ACM Siggraph Computer Graphics(TOG)*, 20(4):151–160, 1986.

[12] Coquillart Sabine. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *Proceedings of SIGGRAPH 1990*, pages 187–196, New York, NY, USA, 1990. ACM.

[13] R. MacCracken and K.I. Joy. Free-form deformations with lattices of arbitrary topology. In *Proceedings of SIGGRAPH 1996*, pages 181–188. ACM New York, NY, USA, 1996.

[14] M. S. Floater, G. Kos, and M. Reimers. Mean value coordinates in 3d. *Computer Aided Geometry Design*, 22:623–631, 2005.

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

[16] Y. Lipman, J. Kopf, D. Cohen-Or, and D. Levin. GPU-assisted positive mean value coordinates for mesh deformations. In *Proceedings of Eurographics symposium on Geometry processing 2007*, pages 117–123. Eurographics Association, 2007.

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

[18] E. Landreneau and S. Schaefer. Poisson-based Weight Reduction of Animated Meshes. *Computer Graphics Forum*, 28(2), 2009.

[19] D. Hoffman and W. Richards. Parts of recognition. *Cognition*, 18(1-3):65–96, 1984.

[20] G. Kós, R.R. Martin, and T. Várady. Methods to recover constant radius rolling ball blends in reverse engineering. *Computer Aided Geometric Design*, 17(2):127–160, 2000.

[21] S.W. Wang and A.E. Kaufman. Volume sampled voxelization of geometric primitives. In *Proceedings of the 4th conference on Visualization'93*, page 84. IEEE Computer Society, 1993.

[22] D. Cohen-Or and A. Kaufman. Fundamentals of surface voxelization. *Graphical models and image processing*, 57(6):453–461, 1995.

[23] G. Varadhan, S. Krishnan, Y.J. Kim, S. Diggavi, and D. Manocha. Efficient max-norm distance computation and reliable voxelization. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 116–126. Eurographics Association, 2003.

[24] A. Fabri and S. Pion. CGAL: the Computational Geometry Algorithms Library. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 538–539. ACM, 2009.

[25] H. Burchard, J. Avers, W. Frey, and N. Sapidis. Approximation with aesthetic constraints. *Designing Fair Curves and Surfaces: Shape Quality in Geometric Modeling and Computer-Aided Design*, pages 3–28, 94.

[26] Y. Chen, K.P. Beier, and D. Papageorgiou. Direct highlight line modification on nurbs surfaces* 1. *Computer Aided Geometric Design*, 14(6):583–601, 1997.

[27] M. Elad, A. Tal, S. Ar, and I. Haifa. Content Based Retrieval of VRM L Objects-An Iterative and Interactive Approach. In *Multimedia 2001: proceedings of the Eurographics Workshop in Manchester, United Kingdom, September 8-9, 2001*, page 107. Springer Verlag Wien, 2002.