A short survey on recent methods for cage computation

Pascal Laube and Georg Umlauf

Institute for Optical Systems, University of Applied Sciences Constance, Germany

***Abstract*—Creating cages that enclose a 3D-model of some sort is part of many preprocessing pipelines in computational geometry. Creating a cage of preferably lower resolution than the original model is of special interest when performing an operation on the original model might be to costly. The desired operation can be applied to the cage first and then transferred to the enclosed model. With this paper the authors present a short survey of recent and well known methods for cage computation. The authors would like to give the reader an insight in common methods and their differences.**

I. Introduction

Due to the ever increasing amount of highly detailed 3D models algorithms that can handle large scale models and perform operations in acceptable time are a necessity. Since these models often consist of meshes with many thousand vertices, algorithms need to be highly optimized. Aside the possibility of optimizing each algorithm for speed it is also possible to simplify the problem itself. Many applications today use lower resolution versions of the respective models for evaluation and then apply the result to the high resolution model. These low resolution approximations that enclose the original model are called cages or envelopes. Fig. 1 shows an example model of an elephant and its cage.

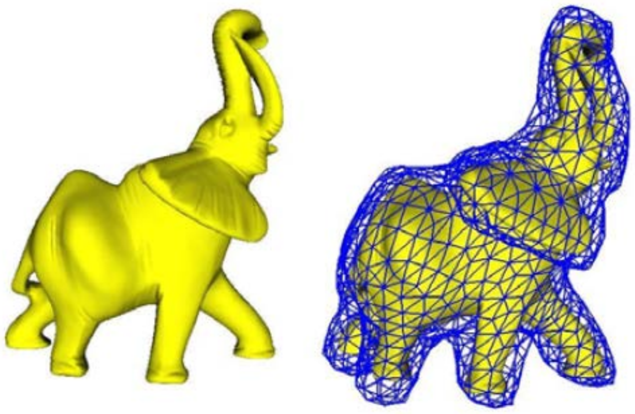
Fig. 1: Model of an elephant on the left and the model enclosed in a cage on the right [1].

There exist many different applications for cages and envelopes. The most prominent application is model deformation. When deforming a model, the deformation functional has to be applied to the whole model domain. In this case computational complexity depends on surface complexity as e.g. number of vertices or faces in a mesh. Applying the deformation to a simpler cage that encloses the original model and projecting the deformation onto the fine mesh, after applied to the cage, reduces the computational cost significantly. In [2] Lipman et al. propose the usage of Green Coordinates for cage-based space deformation. Debunne et al. [3] use multiresolution tetrahedral meshes to guarantee a certain framerate for the deformation of visco-elastic deformable objects. A fine mesh with physical properties is embedded in a tetrahedral grid to simplify deformation computation by the finite element method in [4].

Another important application that is directly related to deformation is contact and collision detection. A very common bounding structure for collision detection are spheres. James and Pai [5] use a Bounded Deformation Tree to perform collision detection on large amounts of objects using spheres. Dingliana and O'Sullivan [6] propose a multiresolution scheme detecting collisions based on level of detail when using spherical cages.

Other applications of cages and envelopes include e.g. projections of complex functions onto bounded models [7] or fast realistic rendering of objects based on high resolution texture but low resolution meshes [8].

In each area where cages are applied there exist task-specific requirements. Aside these task-specific properties there exist properties that are generally beneficial when applying cages. Cages should

- not self-intersect,
- not intersect the original model,
- be homeomorphic to the model enclosed,
- follow the model as close as possible while being a simplified version of the model.

This survey paper of methods for caging and enveloping will give the reader a quick overview of the topic and recent methods. The structure of this paper is as follows. In section two, short summaries of related methods are presented. Section three will compare the different methods while we conclude in section four.

II. Related methods

This section will give short summaries of caging methods from the areas of "Simplification and Flow", "Voxelization and Multigrid-Methods" and "Offset Surfaces".

A. Simplification and Flow

An approach for nesting multiresolution meshes is proposed by Sacht et al. [9]. The proposed algorithm does not present new results on surface simplification since the approach is independent of the used simplification. As input a number of polyhedra with varying resolution and a fitness function are needed. The polyhedra can be overlapping but need to be watertight. Taking a mesh of high resolution $\hat{M}_0$ and k decimated meshes $\hat{M}_1, \ldots, \hat{M}_k$ the method will output a sequence $M_1, \ldots, M_k$ of nested meshes, where M_{i-1} is strictly contained in M_i. $M_1, \ldots, M_k$ will be created minimizing a user-defined energy function E. Nesting is ensured by only operating on two meshes at a time starting with the finest and second finest mesh. In each step a finer mesh F is embedded inside a mesh C that was derived from an input mesh $\hat{C}$. The method consists of two main steps: In the first step the vertices of the finer mesh F are moved along a flow inwards to minimize the total signed distance to $\hat{C}$ and in a second step this mesh $\bar{F}$ will then be re-inflated back to F pushing $\hat{C}$ out of the way to become C. See Fig. 2 for a 2D example of the process. Flowing $\bar{F}$ inside $\hat{C}$ is done by minimization of

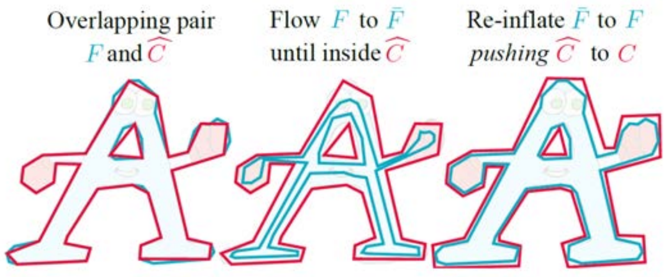
Overlapping pair
F and $\widehat{C}$
Flow F to $\bar{F}$
until inside $\widehat{C}$
Re-inflate $\bar{F}$ to F
pushing $\widehat{C}$ to C

Fig. 2: Main steps of the algorithm by Sacht et al. [9]

the unsigned distance function $d(\mathbf{p})$ integrated over all points $\mathbf{p}$ of the deforming surface

$$\Phi(F) = \int_F s(\mathbf{p}) d(\mathbf{p})\, dA \tag{1}$$

where $s(\mathbf{p})$ is the sign modulation. Minimization is performed by gradient descent with fictitious time t:

$$\frac{\partial \bar{\mathbf{f}}}{\partial t} = -\nabla_f \Phi(\bar{F}).$$

Vertex positions $\bar{\mathbf{f}}$ in $\bar{F}(t)$ will flow inside the coarse mesh by taking small steps in the opposite gradient direction. Since it is not always guaranteed that there will be no intersections, the possibility of first expanding the coarse mesh is proposed. The coarse mesh will then flow outwards creating some distance between the finer mesh and itself. Contact forces are introduced to prevent self-intersections. The problem of self-intersection is a common problem in mesh expansion. This is why inward flow or shrinkage of the fine mesh is preferred.

In a next step $\bar{F}$ needs to be re-inflated to recover the original position F. While re-inflating, $\hat{C}$ needs to be pushed outwards so that there are no collisions of $\bar{F}$ and $\hat{C}$. The previous steps of flowing F inwards can now be reversed to gradually inflate back to F. Since each step back to F is a positional change in some time step Δt this can be expressed in terms of velocity. Defining the re-inflation in terms of velocity makes the use of physical simulations for contact detection possible. The method [10] can be used out of the box since it takes mesh vertices as well as desired velocities and outputs adjusted velocities. By assigning infinite mass to $\bar{F}$ one can make sure that the fine mesh will return to its original position F. If [10] fails the slower but more robust method [11] can be used.

Much like Sacht et al. [9], Sander et al. [8] use the concept of simplifying the original model and flowing this simplification away from the original to compute cages. Sander et al. render models as coarse cages to reduce rendering complexity. Their approach is based on the concept of progressive nested hulls. They start by first defining the interior volume of the model. The decision if a point p lies inside the volume $\mathcal{V}(M)$ of some model M is based on the winding number. If one takes a ray from p to infinity and tracks its intersections with M one can decide if p lies inside M. For an intersection of the ray with an inner side of a face the winding number is increased by one while for intersections with an outer side it is decreased by one. Points with positive winding number will lie inside the model.

The progressive hull algorithm is strongly based on the original progressive mesh by Hoppe [12]. In the context of simplifying a model to create a cage the original mesh M^{i+1} has to be fully enclosed by M^i so that $M^i \subseteq M^{i+1}$. This relation can be ensured by introducing inequality constraints for the position of the unified vertex in an edge collapse. This unified vertex V is constrained to lie outside the model volume $\mathcal{V}(M^{i+1})$ after an edge collapse (refer to Fig. 3 for the setup). The position of V after the collapse can be found

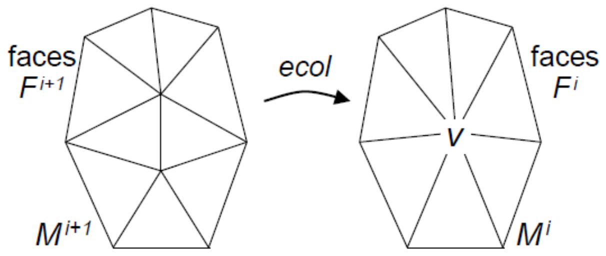

Fig. 3: Edge collapse [8].

by using linear programming to solve the resulting inequality constraints and minimizing the resulting volume enclosed by M^{i+1} and M^i. This ensures that the cage will enclose the model tightly. Additionally cost metrics can be applied when collapsing edges to ensure mesh quality.

B. Voxelization and Multigrid-Methods

Xian et al. [13] use an improved Oriented Bounding Box (OBB) tree [14] to create coarse cages. OBB structures are often applied to the problem of collision detection. Xian et al. start by computing an initial OBB O for the model at hand. O is then subdivided by voxelization. In a classification step the voxels are classified as inner voxels, outer voxels and feature voxels. Voxels that contain part of the model mesh are feature voxels while inner voxels lie within the model and outer voxels lie outside of the model. A point set P is created that contains the mesh vertices as well as the barycenters of inner voxels. Based on P the initial OBB can be recomputed by Principal Component Analysis to get a tighter fit on the model. To further divide the OBB information about the object shape is considered. The division takes place in regions of largest change in shape (see figure 4). Xian et al. define the

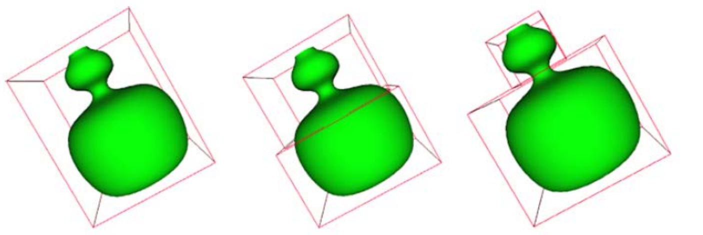
Fig. 4: Initial OBB on the right, OBB simply split in half in the middle and OBB split at locations of largest shape change on the right [13].

change in shape as cross section area function differences. The cross section area is defined as the area enclosed by the intersection of the model and a splitting plane parallel to the OBB faces composed of the smaller two OBB dimensions. The function of cross section area change is built by moving the splitting plane through the model. The location of largest change in shape is then defined as the biggest jump in the cross section area change function. A split of the initial OBB at this location results in further sub-OBBs for which the procedure is repeated until certain termination conditions are satisfied. The so created OBBs represent a first coarse cage of the model. This cage is refined by registration and merging of adjacent OBBs. Two adjacent OBBs are registered by first projecting the corner points A and B of the two adjacent faces onto an intermediate plane between them. A 2D-OBB is computed that encloses the resulting projection points A' and B' on the intermediate plane. Projection points A' and B' are then registered onto the corner points of the 2D-OBB. As a final step the two adjacent OBB are linked considering the registration of A' and B' in the intermediate plane. A triangulation can be created by simply splitting quads at diagonals. The resulting triangular mesh is then re-meshed testing for intersections of the coarse cage and the model in every step.

In [1] Xian et al. propose cage computation by first voxelizing the model to enclose. Like in [13] voxels are categorized as outer, inner and feature voxels. The resolution of the voxel grid will later on define the number of vertices of the coarse cage. The faces of the feature voxels that coincide with faces of outer voxels build a first rough approximate coarse cage. This initial cage might not be 2-manifold. At non-2-manifold edges voxels are attached while at non-2-manifold vertices a vertex-split operation is employed. For triangulation the surface quads are split at their diagonal. For further smoothing of the cage Xian et al. use an adapted mean curvature flow method [15]. Movement of vertices in the smoothing process is based on the curvature vector $H\mathbf{n}$ where H is the curvature and $\mathbf{n}$ the normal at a vertex. This vector points outwards on convex vertices while pointing inwards on concave vertices. Based on the information of outer and inner voxels Xian et al. computer an additional vector ∇d that always points away from the mesh. If the angle θ between $-H\mathbf{n}$ and ∇d lies between 0 and $\frac{\pi}{2}$ the vertex is moved along ∇d. If θ lies between $\frac{\pi}{2}$ and π it is moved along the normal $\mathbf{n}$. Additionally, the distance of each moved vertex to the model is tested at each step. If the vertex falls inside the model it is moved in the direction of ∇d. To create homeomorphic cages, the resolutions of the initial voxelization is iteratively increased until homeomorphism is given.

In [4] a very simple grid based approach is used for deformation simulation. The mesh is embedded in a hexahedral grid. A first, very coarse grid is further decomposed by an octree structure. The octree depth is defined by the user. Each of the so created voxels will inherit the mechanical properties of the enclosed polygons. This fine voxelization is then transformed back to a coarse approximation. By connecting eight voxels on each level one receives the next lager voxel of the octree. Deformation properties of the fine voxelization can be applied to each coarser level by recursive calculation. By using this approach deformation properties like e.g. stiffness of different materials can be merged on a coarser level.

C. *Offset Surfaces*

Ben-Chen et al. [7] create a bounding cages for the purpose of deformation transfer. In a first step they create a set of points along the surface. This can be any kind of surface as long as it is possible to assign normals to the created surface points. Next the points will be enveloped by using the Poisson reconstruction algorithm of Kazhdan et al. [16]. The resulting mesh or envelope E is simplified. This is done by using progressive mesh [12]. The surface is simplified until a user defined threshold is reached. Then for each remaining vertex a new offset position is computed by flowing each vertex of E along its normal direction outwards with predefined step size s. Ben-Chen et al. compute the vertex normal as the area-weighted average of normals of the vertices adjacent faces. This process is then repeated until the desired number of faces is reached. Ben-Chen et al. also state that the same step size s at all points can lead to self-intersections in regions that are close to each other like legs of a human model. For this case multiple user defined step sizes at different locations should be applied.

In [17] Shen et al. present an algorithm to create envelopes of polygonal soups based on approximation by moving least-squares. For a normal least squares fit one would have

$$\begin{bmatrix} \mathbf{b}^T(\mathbf{p}_1) \\ \vdots \\ \mathbf{b}^T(\mathbf{p}_N) \end{bmatrix} \mathbf{c} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix},$$

with points $\mathbf{p}_i, i \in [1, \ldots, N]$, basis functions $b(x)$, the values ϕ_i at points $\mathbf{p}_i$ and the unknown coefficients c. The authors introduce a weight function $w(x)$ into the normal equation of the standard least-squares formulation. $w(x)$ is a distance function by which one can regulate approximation behavior up to interpolation. The least-square fit then becomes

$$\begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) & & \\ & \ddots & \\ & & w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} \mathbf{b}^T(\mathbf{p}_1) \\ \vdots \\ \mathbf{b}^T(\mathbf{p}_N) \end{bmatrix} \mathbf{c} = \begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) & & \\ & \ddots & \\ & & w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix},$$

with $w(x, p_i) = w(||x - p_i||)$. Since the moving-least squares method is a point based method one needs to adapt the concept for polygons. Taking points along the polygons and performing a point-based approximation (especially very tight approximations) leads to bumps and dimples along the surface. This even is the case when using quadrature points. To handle these problems Shen et al. propose using the least-squares method not to blend points of each polygon but to blend functions associated to the polygons. The standard normal equation can be built based on these functions and becomes

$$\begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) & & \\ & \ddots & \\ & & w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \mathbf{c} = \begin{bmatrix} w(\mathbf{x}, \mathbf{p}_1) & & \\ & \ddots & \\ & & w(\mathbf{x}, \mathbf{p}_N) \end{bmatrix} \begin{bmatrix} S_1(\mathbf{x}) \\ \vdots \\ S_N(\mathbf{x}) \end{bmatrix},$$

where $S_i(\mathbf{x})$ is the polygonal function. Figure 5 shows some point-based and function-based approximations. For a more detailed explanation please consider reading [17].

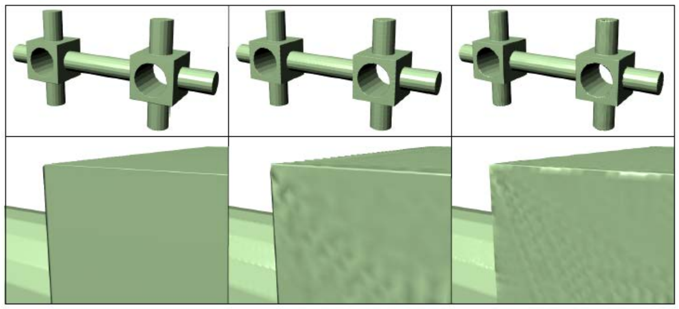
Fig. 5: The left column shows an exemplar approximation with polygonal constraints while the examples in the middle and on the right show results of point constraint examples with different densities of scattered points [17].

III. Method comparison

This section compares the different methods explained in the previous chapter. The methods will be evaluated based on the properties influencing quality defined in section one.

A problem of high priority when computing cages is the prevention of intersections between the model and the cage. While all caging methods aim to prevent these model-cage-intersections, ensuring absence of intersections (AOI) results in significant increases of computational costs. For Sacht et al. [9] AOI comes built in by the methods for physical simulation [10] and [11]. For these methods to work Sacht et al. need to ensure that the finer mesh does not intersect the coarser mesh after the inward flowing process. The methods [1] and [13] need to make sure that the cage does not intersect the model at different steps in the caging process. In [13] AOI is obtained by accepting a smaller cage resolution while in [1] vertices are pulled out of the model in the smoothing process which leads to other problems like larger cage-model-distance and possible self-intersection. For grid-based methods it is very easy to ensure AOI since they are based on the bounding box containing the model. Using the outer surface of a bounding grid or voxelization makes getting AOI simple but lacks a close resemblance of the model. The methods of [7] and [17] for offset surfaces do not include proper handling of model-cage-intersections. Deciding on intersections of the model and the cage is left to the user. For [17] it is clear that the focus of the presented work did not lie on creating cages that strictly enclose a model which is not the case in [7] where the cage generation method simply seems sufficient for the task at hand. In [8] ensuring AOI is a vital part of the method since it is introduced as a constraint into linear programming.

There exists a multitude of problems arising from cage self-intersections (SI) like unintended deformations in applications of contact detection and deformation. Preventing SI in the process of cage computations is very difficult. Since Sacht et al. [9] use a collision detection methods from physical simulation the re-inflation of the fine mesh leads to SI-free meshes. For the case where they first inflate the coarse mesh they include contact forces to prevent SI. In [1] the first approximate coarse cage (the triangulated voxelization) leads to a SI-free bounding cage. In the smoothing step the cage is forced to reside strictly outside of the enclosed model. Since in this step the cage is not checked for SIs this might lead to non-SI-free bounding cages. In [13] as well as in [1] there is no explicit SI-test or condition included in the cage computation process. Since both methods depend on user-defined thresholds for the resolution of the cages SIs might be resolved by larger resolution. This of course leads to higher cage resolution which might not be desired. When using a grid or voxelization SIs are excluded by the grid-structure itself. For methods like [4] or [1] (without the smoothing step) the problem would not be SI but merging of the bounding cage in regions where the finer model is separated like between the legs of a humanoid model. This merging can lead to problems in deformation or collision detection. The same is true for methods computing offset surfaces like [17] or [7]. Here the cage merges in areas that are very close rather than creating self-intersections. Every method that is able to create non-SI-cages like [9] and [8] needs to introduce a specific step in the computation process to handle SIs. To the authors knowledge there currently exists no method that implicitly creates non-SI-cages.

Creating cages that are homeomorphic to the enclosed model is an important feature in cage computation. Nearly all of the mentioned methods are able to keep homeomorphism. Since none of the presented methods explicitly ensure homeomorphism it is intrinsic to the methods themselves. The only presented method that is unable to create homeomorphic cages is [4] since it keeps empty voxels in the grid-structure and handles them later in the propagation of deformation information through the grid. For all other methods homeomorphism is a question of cage resolution. With smaller cage resolution holes may be filled and by that topological information about the enclosed model will be lost. Figure 6 shows an example from [7] where the homeomorphism is lost. Homeomorphism can only be guaranteed by iteratively decreasing cage resolution toward the desired threshold and checking for homeomorphism at each step of the iteration.

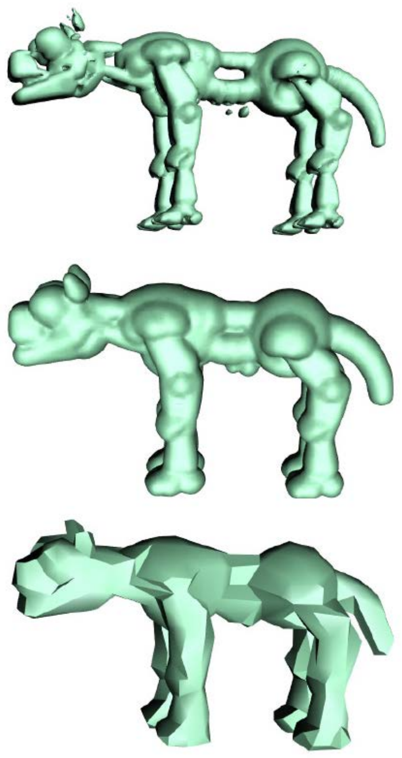
Fig. 6: Top: Initial triangular mesh; middle: Offset surface; bottom: Simplified offset surface. [8].

Staying as close as possible to the original model while resembling the models shape is important to many applications like projection of functionals from cage to model. Sacht et al. [9] propose using an energy term that penalizes total volume between cage and model. This way a very tightly fitting cage can be computed. For methods that depend on voxelization or a grid the tightness of the fit can be controlled by selecting a proper resolution. In contrast to [9] the tightness of the fit and the resolution of the bounding cage are directly correlated which is an undesirable property. A comparable method would be [8] where the tightness of the fit and cage resolution are coupled in terms of that the tightness will vary between different levels of resolution. In [17] the tightness of the resulting offset surface can be controlled by a user defined parameter. In case that the offset surface is then triangulated like in [7] this control is lost.

IV. Conclusion

Which method to use strongly depends on the task at hand. While voxel- and grid-based methods produce bounding cages that more loosely enclose a model they may be sufficient for some collision detection or deformation tasks. The main advantage of these methods is their small computational complexity. Methods to create offset surfaces like [7] and [17] are able to produce very tight cages but to the cost of higher computational complexity resulting from the surface approximation itself. Using them as a preliminary stage to receive a triangulated cage only seems feasible if the computational cost is of no concern. Methods that use mesh simplification and flow like [9] and [8] are able to produce tight cages but also at the price of high computational cost. For [9] costly step in the process is the collision detection while for [8] linear programming seems to be the bottleneck.

Since for many cases cage computation is part of the preprocessing pipeline the impact of computational cost might be neglected.

References

[1] C. Xian, H. Lin, and S. Gao, “Automatic generation of coarse bounding cages from dense meshes,” in *Shape Modeling and Applications, 2009. SMI 2009. IEEE International Conference on*. IEEE, 2009, pp. 21–27.

[2] Y. Lipman, D. Levin, and D. Cohen-Or, “Green coordinates,” in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 78.

[3] G. Debunne, M. Desbrun, M.-P. Cani, and A. H. Barr, “Dynamic real-time deformations using space & time adaptive sampling,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 31–36.

[4] M. Nesme, P. G. Kry, L. Jeřábková, and F. Faure, “Preserving topology and elasticity for embedded deformable models,” in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3. ACM, 2009, p. 52.

[5] D. L. James and D. K. Pai, “Bd-tree: output-sensitive collision detection for reduced deformable models,” *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 393–398, 2004.

[6] J. Dingliana and C. O’Sullivan, “Graceful degradation of collision handling in physically based animation,” in *Computer Graphics Forum*, vol. 19, no. 3. Wiley Online Library, 2000, pp. 239–248.

[7] M. Ben-Chen, O. Weber, and C. Gotsman, “Spatial deformation transfer,” in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2009, pp. 67–74.

[8] P. V. Sander, X. Gu, S. J. Gortler, H. Hoppe, and J. Snyder, “Silhouette clipping,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 327–334.

[9] L. Sacht, E. Vouga, and A. Jacobson, “Nested cages,” *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, p. 170, 2015.

[10] T. Brochu and R. Bridson, “Robust topological operations for dynamic explicit surfaces,” *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 2472–2493, 2009.

[11] S. Ainsley, E. Vouga, E. Grinspun, and R. Tamstorf, “Speculative parallel asynchronous contact mechanics,” *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, p. 151, 2012.

[12] H. Hoppe, “Progressive meshes,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 99–108.

[13] C. Xian, H. Lin, and S. Gao, "Automatic cage generation by improved obbs for mesh deformation," *The Visual Computer*, vol. 28, no. 1, pp. 21–33, 2012.

[14] S. Gottschalk, M. C. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 171–180.

[15] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 317–324.

[16] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, 2006.

[17] C. Shen, J. F. O'Brien, and J. R. Shewchuk, "Interpolating and approximating implicit surfaces from polygon soup," in *ACM Siggraph 2005 Courses*. ACM, 2005, p. 204.