High-quality Cage Generation Based on SDF

H. Qiu[1] , W. Liao[2] and R. Chen[†2]

[1]Zhejiang Sci-Tech University, China
[2]University of Science and Technology of China, China

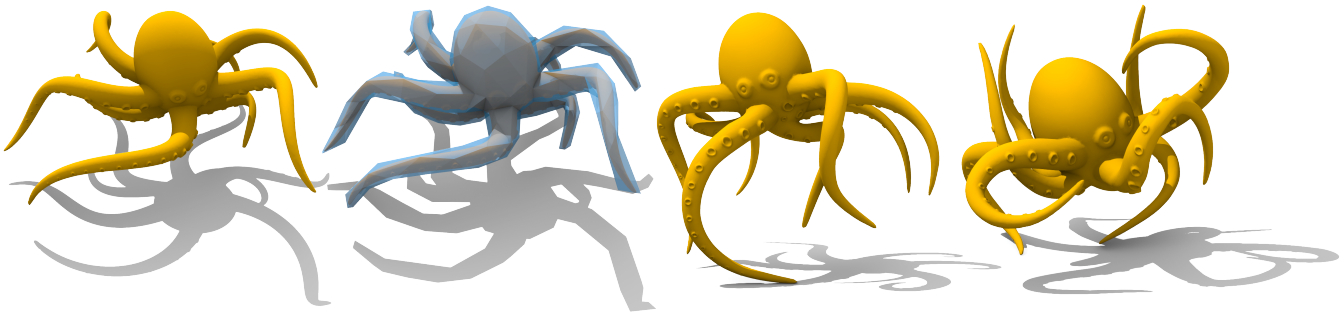


Figure 1: *A cage generated by our method for the octopus shape. The two images on the left are the input shape and the generated cage. The two images on the right show the high-quality cage based deformation results of [LCH*21].*

Abstract

Cages are widely used in various applications of computer graphics, including physically-based rendering, shape deformation, physical simulation, etc. Given an input shape, we present an efficient and robust method for the automatic construction of high quality cage. Our method follows the envelope-and-simplify paradigm. In the enveloping stage, an isosurface enclosing the model is extracted from the signed distance field (SDF) of the shape. By leveraging the versatility of SDF, we propose a straightforward modification to SDF that enables the resulting isosurface to have better topological structure and capture the details of the shape well. In the simplification stage, we use the quadric error metric to simplify the isosurface and construct a cage, while rigorously ensuring the cage remains enclosing and does not self-intersect. We propose to further optimize various qualities of the cage for different applications, including distance to the original mesh and meshing quality. The cage generated by our method is guaranteed to be strictly enclosing the input shape, free of self-intersection, has the user-specified complexity and provides a good approximation to the input, as required by various applications. Through extensive experiments, we demonstrate that our method is robust and efficient for a wide variety of shapes with complex geometry and topology.

CCS Concepts

• ***Computing methodologies*** → ***Computer graphics; Mesh models;***

1. Introduction

In computer graphics, it is commonplace to engage in modeling and processing shapes characterized by intricate geometries to cater to diverse applications such as animation, physical simulation, and rendering. Within these scenarios, high-resolution models are typically employed to meticulously capture the intricate details of complex geometries, thereby enhancing precision and, consequently, the quality of these applications. However, performing computations directly on such high-resolution models in graphics can lead to significant computational complexity, as it often involves intricate and nonlinear optimization problems. Conversely, users often require interactive or even real-time computational performance, particularly when frequent geometric edits are essential. Therefore, researchers have proposed various geometric proxy structures to reduce computational complexity and expedite calculations. Cages represent one prevalent geometric proxy, offering an intuitive representation of the original geometric structure.

† corresponding author

In short, a cage is a coarse structure that surrounds and visually and geometrically approximates a given shape, allowing computations to transfer from a high-resolution model to a coarse proxy, namely the cage itself. For example, in cage based deformation, when the cage is manipulated, the embedded fine shape can be smoothly deformed accordingly, using the generalized barycentric coordinates [FKR05, JSW05, JMDG07, LLCO08, WPG12]. In shape transfer, the pose of a source model can be transferred to a target model through the shape mapping space generated by the cage [BCWG09, CHSB10]. In collision detection, conservative culling can dramatically reduce computation time [PT03]. The key to achieving high performance in all these applications lies in effectively leveraging the cage structure to approximate complex shapes, thus reducing the dimensionality of the problem.

Although cages are widely used in geometric processing, generating a cage that meets requirements has always been a challenge. In many cage-based research studies [JSW05, JMDG07, LLCO08], the cages are manually constructed, utilizing modeling tools, which is tedious and time-consuming. This makes it difficult for non-professionals to grasp, thereby impacting the application and promotion of cage-based techniques.

Existing cage generation algorithms are heuristic mesh simplification methods that aim to create a coarse proxy mesh that encloses the original mesh as closely as possible. However, they lack a well-defined objective function, such as surface approximation error and geometric feature preservation, to guide the generation of high-quality cages. Recently, Sacht et al. [SVJ15] proposed an method for automatically generating cages using a distance flow, which reliably ensures the quality of the generated cages. As it pointed out, an ideal cage should generally possess the following properties:

1. It is strictly enveloping the shape.
2. It is free of self-intersection.
3. It is geometrically and topologically close to the shape.
4. It is of low complexity.

In various applications of cages, properties 1. and 2. are the most crucial ones. For instance, when computing various types of generalized barycentric coordinates [FKR05, JSW05, JMDG07], the exterior of the cage is poorly defined and can lead to local infinite values. However, constructing a cage that strictly enveloping the input shape is a very challenging problem. As stated in [SVJ15, Appendix A], we can easily construct counterexamples that cause methods using continuous flow to strictly enclose a shape to fail.

This paper proposes a universal and robust cage generation algorithm based on the perspective of strictly enclosing the input shape, leveraging the versatility of signed distance field (sdf). Our method is consist of the following three steps:

Isosurface extraction In this step, we extract an isosurface from the SDF of the shape. With a positive isovalue, the isosurface naturally satisfies properties 1. and 2.. The flexibility of SDF allows us to further modify it as a general shape aware implicit function, so that the extracted isosurface satisfies property 3. (Section 3.1).

Simplification A cage should describe the shape of the model with as few degrees of freedom as possible. Simply extracting isosurfaces from the SDF results in overly dense meshes. Therefore, we need to simplify this isosurface, while ensuring that properties 1. and 2. remains intact. This step iteratively simplifies the isosurface by edge collapsing. Different from Garland and Heckbert [GH97], we solve a constrained quadratic program, that allows us to eventually build a coarse cage that strict envelopes the original model and is self-intersection free. We propose a practically feasible solution for the constrained quadratic program, allowing the simplification process to iteratively continue until the user-specified number of vertices is reached. Unlike the simplification strategy in [SVJ15], we start with an initial mesh that envelopes the input shape, allowing us to strictly ensure that this property is maintained throughout each simplification step.

Optimization For the last step, we propose a general optimization framework to improve various qualities of the cage, using a combination of vertex relocation and edge flipping operations. In this step, we also strictly enforce the enveloping and no self-intersection constraints using the IPC barrier function proposed by Li et al. [LFS*20] (Section 3.3).

Our method is easy to implement. We experiment our method on a variety of input shapes, which show that it is effective, robust and can suit the needs of different applications.

2. Related Work

As an efficient and flexible method for shape editing and animation generation, free-form deformation (FFD) was first proposed by Sederberg and Parry [SP86]. It uses a Bézier volume control lattice as the proxy geometry for the target shape. Although various extended FFD methods can utilize irregular hexahedral lattice, the flexibility of deformation is constrained by the fixed topological shape of the control lattice, limiting the flexibility of deformation due to interpolation and smoothness constraints. Therefore, the exploration of spatial deformation techniques with control lattices possessing more degrees of freedom was triggered. Spatial deformation methods [FKR05, JSW05, JMDG07, LLCO08, WPG12] based on generalized barycentric coordinate interpolation are one popular approach in this regard. The control meshes no longer has a fixed topology and geometry but instead approximates the shape of the target model with fewer vertices, known as a 2-manifold mesh, referred to as a cage. Although cages are widely used in geometric processing, generating cages that meet requirements has always been a challenging task.

Recently, many methods of semi-automatically or automatically generating cage have been proposed. Chen and Feng [CF14] first utilizes an adaptive cross-section-based method to construct a cage from an initial skeleton structure, which is obtained through an sketching interface. Later, Le and Deng [LD17] suggested using some user-specified cut slides to optimize the consistent, orthogonal orientations of cage cross sections, greatly improving the cage quality. Calderon and Boubekeur [CB17] introduced a novel method for approximating bounding shapes and generating a bounding proxy that tightly fits the input shape. However, these tasks involve user interaction, which can be tedious for the user and does not allow to generate cages in batch for a set of mesh.

Xian et al. [XLG12] construct the cage automatically by refining

the bounding shape of the input shape. It first generates an oriented bounding box (OBB) tree for the shape, and then merges the OBBs into a whole entity using the union operation. However, OBB relies on an initial voxelization of input shape, which is hard to be determined reasonably. Additionally, the resulting cages are not guaranteed to envelope the input model, because there exists twisting among two joint parts of OBB.

The majority of automatic cage generation method is to simplify and expand the input mesh to produce the coarse proxy. Progressive decimation, first proposed by Sander et al. [SGG*00], applies edge collapse operator and carefully places the new vertices outside the input mesh by solving a linear program with linear inequality constraints. Deng et al. [DLM11] also uses edge collapses, but obtains the positions for the new vertices by solving a quadratic program with inequality constraints. Ben-Chen et al. [BCWG09] proposed a heuristic method for building cages around the input shape by repeating the offset, reconstruction and simplification operations. However, these methods do not guarantee to strictly enclose the target model, while simplification-based methods are prone to self-intersections in concave regions of the model, such as "U-shaped" areas.

Sacht et al. [SVJ15] proposed a method for producing series of cages with strict nesting in-between, by adding two steps of flow and contact-aware optimization after mesh decimation. It pointed out that distance flow may not necessarily be reasonable for all models. Because there exists some singular points without flow direction when more than two ridge-lines meet. Jiang et al. [JSZP20] provided a more intuitive explanation of the conditions for the existence of singular points, stating that they can only be bypassed without being processed.

In this work, we propose to construct cages using the isosurface of SDF, which is the most widely used implicit surface representation in computer graphics [BA05, XB14], as it allows easy determination of whether a point is inside or outside the shape based on the sign of SDF. In general, the implicit shape representation is popular for surface reconstruction [OBA*03] and rendering [CK10], and we show that it can be useful for other geometry processing applications as well.

In Figure 2, we compare the cages generated by the state-of-the-art methods and our method for two input shapes. It can be seen that the cages generated by the competing methods do not fit the input shape as well as ours under the same resolution.

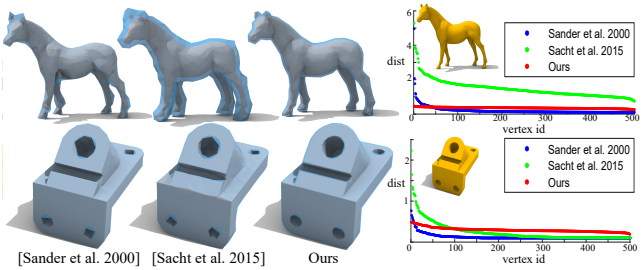


Figure 2: *A comparison of various cages generated for the horse shape and anchor shape using the methods from some previous works and ours. Each of the shown cages contains about* 500 *vertices. The last column records the distances from the vertices of the cages generated by the three different methods to the input shape. The horizontal axis represents the vertex index, which is sorted in descending order of distance.*

3. Method

Our method accepts inputs in the form of 3D solids or watertight surfaces. For other types of inputs, such as point clouds or triangle soups, one can first use mesh reconstruction methods like Poisson reconstruction [KBH06] to generate a fitted watertight surface before applying our method. In a nutshell, our method follows the envelope-and-simplify paradigm. Intuitively, we first expand the surface to envelope the shape, and then simplify the expanded surface to obtain the resulting cage.

Given a watertight surface represented by a triangle mesh $\mathcal{M} = (\mathcal{V}, \mathcal{T})$, with vertices $\mathcal{V}$ and faces $\mathcal{T}$. To obtain a geometry that can enclose the surface, the natural approach is to expand the mesh surface, i.e., to offset each vertex along its normal direction, as Ben-Chen et al. [BCWG09] did. This simple strategy works in 2D, where the offset curve is guaranteed to contain, i.e. envelope, the input curve and is free of self-intersection, as long as the offset distance is small enough. However, this is no longer true in 3D, as the normals of vertices are not always well-defined in three dimensions. Even on rugged and complex vertices, moving the vertex in any direction can result in intersections with the original mesh. Sacht et al. [SVJ15] provided an example of a star-shaped figure to illustrate the existence of such vertices in 3D..

Therefore, in this work, we utilize implicit shape representation, particularly the signed distance field (SDF), and use the isosurface to envelop the input shape. Furthermore, we can locally modify the SDF so that the resulting isosurface will be homeomorphic to the input surface, without having to specify tiny isovalues, especially for input shapes with nearly-touching parts.

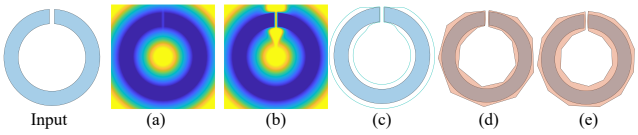


Figure 3: *The pipeline of our method. Given an input shape, our method will sequentially compute the SDF (a), modify the SDF (b), extract the isosurface (c), apply simplification to obtain a cage (d) and optionally perform further optimization (e).*

In general, the extracted isosurfaces are as dense as, or even more dense than the input surface mesh. To obtain a low-complexity cage, we need to simplify the isosurface, and apply further optimizations to improve its quality. Figure 3 shows the pipeline of our method.

3.1. Isosurface Extraction

In this step, we first convert the input shape representation into the SDF to the surface of the shape $\mathcal{M}$, which is assumed to be water-

tight. More specifically, the SDF is given by function $\xi : \mathbb{R}^3 \to \mathbb{R}$,

$$\xi(x) = \begin{cases} -d_{\mathcal{M}}(x), & x \in \mathcal{I} \\ d_{\mathcal{M}}(x), & x \in \mathcal{I}^c \end{cases}, \tag{1}$$

where $\mathcal{I}$ is the interior of $\mathcal{M}$, while $\mathcal{I}^c$ denotes its complement, and $d_{\mathcal{M}}(x) := \min_{y\in\mathcal{M}} \|x-y\|$ is the undirected distance from a point to the surface. $\xi(x)$ measures how far a point x is away from the surface $\mathcal{M}$ and the sign distinguishes whether x is inside or outside the surface. We further define a projection operator $\mathcal{P}_{\mathcal{M}}(x) := \arg\min_{y\in\mathcal{M}} \|x-y\|$. For any point x, the operator $\mathcal{P}_{\mathcal{M}}$ projects x to a point on the surface which is closest to x. SDF possess numerous favorable properties, with one of the most significant being its continuity and ability to effectively distinguish between inside and outside of meshes.

Through the SDF, it is straightforward to extract the isosurface of the mesh, which is a surface consisting of points with a constant field value c: $\mathcal{S}_c = \{p|\xi(p) = c\}$. As a special case, $\mathcal{S}_0$ coincides with the original surface $\mathcal{M}$. Due to the continuity of the SDF, the isosurface has no self-intersection and is strictly enveloping the original mesh for any $c > 0$.

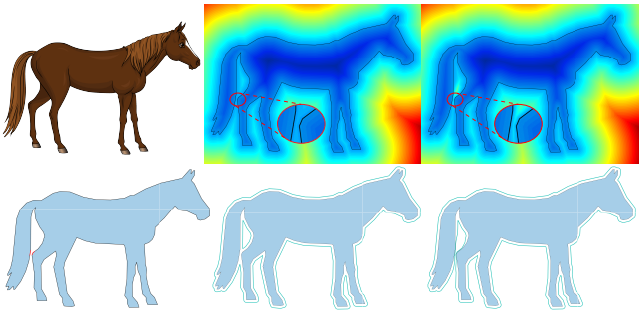
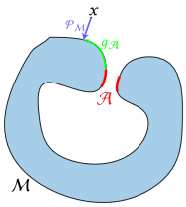


Figure 4: *An example of isosurface extraction in 2D. The left column shows the input horse shape. Top-center and top-right show a visualization of the SDF and our modified version. Bottom-center and bottom-right show the isosurfaces extracted from the corresponding fields. In the modified SDF, the distance in the zoom-in area are magnified locally, allowing us to extract an isosurface that is homeomorphic to the input.*

While the isosurface envelops the input mesh well, it often exhibits geometric and topological inconsistencies with the input mesh, particularly in nearly-touching parts on the mesh, as an example in Figure 4 shows. Such topological inconsistencies can lead to significant issues in cage based deformation, such as the inability to independently edit regions of the mesh with different semantic meanings. A simple strategy is to reduce the selection of the c value, but this would rely on a highly dense discrete SDF. This would significantly increase computational resource consumption and result in the generation of more high-degree isosurfaces, constraining our subsequent simplification steps in Section 3.2. Hence, we propose a simple local modification to the SDF which allows the isosurface to cut through the narrow channel between these nearly-touching parts, and make sure the topology of the enclosed surface is kept. We first try to locate the regions of the surface which are nearly-touching with some other regions of the surface, and then amplify the SDF locally near these regions.

To locate the nearly-touching regions of the surface, we identity and construct a set of active points, by comparing the interior (geodesic) distance and the Euclidean distance between each pair of points. Let $g(x,y)$ be the geodesic distance between x and y, i.e. through the interior of the surface $\mathcal{M}$, $d(x,y) = \|x-y\|$ be the Euclidean distance, and denote the neighborhood of a point $p \in \mathcal{M}$ in $\mathcal{M}$ as $B^{\mathcal{M}}_{p,r} := \{x|d(x,p) < r\}$. A point p is an active point if there exists a point $q \in B^{\mathcal{M}}_{p,r}$ such that $g(p,q) > l$, where r and l are user specified parameters, with r being related to the isovalue c and l related to the diameter of $\mathcal{M}$.



Let $\mathcal{A}$ be the set of all active points (see inset), whose existence indicates that there are narrow channels between some different parts of the shape, which will not be separated by the isosurface. Therefore, we locally modify the SDF near the set $\mathcal{A}$ so that the extracted isosurface will be closer to $\mathcal{A}$ and can separate the nearly-touching parts. Intuitively, we multiply the SDF by a radial function which quickly increases near the active points so that isosurfaces becomes more densely distributed there. Formally, we construct the radial function using the projection operator $\mathcal{P}_{\mathcal{M}}$ and geodesic distance g.

Let $g_{\mathcal{A}}(p) = \min_{q\in\mathcal{A}} g(p,q)$ be the distance from a point on the surface to the set $\mathcal{A}$. We define the following distance function $\hat{d}$ for $x \in \mathbb{R}^3$,

$$\hat{d}(x) = g_{\mathcal{A}}\big(\mathcal{P}_{\mathcal{M}}(x)\big) + d_{\mathcal{M}}(x). \tag{2}$$

For simplicity, we choose a variant of the Gaussian kernel function as the radial function:

$$\phi(x) = (\alpha - 1)e^{-\beta^2\hat{d}^2(x)} + 1, \tag{3}$$

where $\alpha > 1$ and β are two parameters related to the diameter of $\mathcal{M}$. ϕ obtains its maximum value α when $\hat{d}(x) = 0$, and stays close to 1 for sufficiently large $\hat{d}$. To this end, we modify the SDF as $\hat{\xi}(x) = \phi(x)\xi(x)$, where $\hat{\xi}$ is kept close to ξ in most regions, except near the active point, for which the value of $\hat{\xi}$ becomes much larger. From the modified SDF, we can now extract the isosurface $\mathcal{S}_c = \left\{p|\hat{\xi}(p) = c\right\}$. Since our modified 3D scalar field remains continuous, the extracted isosurface $\mathcal{S}_c$ is free of self-intersections. Figure 4 shows an example of isosurfaces extracted from the original SDF and the modified SDF for the planar case. Figure 7 shows an example for a 3D shape input.

3.2. Simplification

For the majority of applications, the cage should be of low complexity. However, the isosurface that we extracted above has similar, and often higher, complexity comparing to the input surface mesh. Hence, we need to apply simplification to the isosurface, in order to obtain an usable cage.

In the standard mesh simplification routine, edges are collapsed iteratively, with a quality measure, e.g. QEM [GH97], guiding the

collapsing order. In each iteration, the edge e, with the lowest cost among all remaining edges is chosen, and collapsed to a new vertex p, whose position is determined by solving a quadratic program to minimize the collapsing cost $\Delta(v) = v^T Q v$, with Q being a symmetric matrix and v is the position of vertex p. We denote $\mathcal{M}_p$ ($\widehat{\mathcal{M}}_p$) as the surface before (after) edge e is collapsed to the point p.

The QEM simplification can produce a high fidelity approximation to the original mesh. However, it does not guarantee that the simplified mesh is:

1. strictly enveloping the original mesh $\mathcal{M}$;
2. free of self-intersection.

which are important properties that the cage should satisfy, as we have pointed out. Therefore, we transform the unconstrained quadratic program into a constrained problem, similar to the formulation proposed by Xian et al. [XLG12],

$$\begin{aligned} &\arg\min_v v^T Q v \\ \text{s.t.} \quad &N(p) \cap \mathcal{M} = \emptyset \\ &\widehat{\mathcal{M}}_p \text{ is self-intersection free} \end{aligned} \tag{4}$$

where $N(p)$ contains all the triangles in $\widehat{\mathcal{M}}_p$, which are incident to vertex p.

The quadratic problem now becomes much more difficult to solve, due to the added hard constraints. We propose a heuristic solution guided by the SDF. While it is not guaranteed to succeed, it turns out to be highly effective in practice. In the rare case that our solution fails, or even there exists no solution to problem (4) for some edges, we set the cost of these edge to infinity to avoid collapsing them and allow the simplification process to continue with other edges.

In order to solve problem (4), we initialize p with its position determined by the unconstrained problem $\arg\min_v v^T Q v$, and then try to project the solution into the feasible region under the hard constraints. Formally, we introduce a function Ψ which measures the potential for vertex p to satisfy the constraints and then update p by iteratively applying gradient descent to Ψ until the constraints are met. Note that it is extremely rare in practice for $N(p)$ to self-intersect, hence to simplify the problem, we ignore this situation when designing Ψ.

Note that if $N(p)$ falls outside (or onto) the surface $\mathcal{M}_p$, then the obtained $\widehat{\mathcal{M}}_p$ is guaranteed to satisfy both of the constraints in problem (4). Hence, we construct Ψ in a way similar to Sacht et al. [SVJ15]. We define Ψ as the signed distance to $\mathcal{M}_p$ integrated over $N(p)$,

$$\Psi(p) = -\int_{x \in N(p)} \xi_p(x) dA, \tag{5}$$

where $\xi_p(x)$ denotes the signed distance to $\mathcal{M}_p$. The integral can be approximated with a finite sum over a sample set x_i,

$$\Psi(p) = \sum_{i=1 \cdots h} w_i \xi(x_i), \tag{6}$$

where h is the sample size and w_i is the area weight.

We minimize Ψ with gradient descent, which stops when the constraints are satisfied (or if the iteration count exceeds a user specified number). The cost of this edge is then set to $\Delta(v) = v^T Q v$ (or $\Delta(v) = \infty$). Note that, after an edge collapses, the cost of all the incident edges will be updated, therefore even if an edge is not collapsible in one iteration, it may be collapsed later after its cost gets updated. Figure 5 compares the simplification results of an isosurface with and without the no intersection constraints.

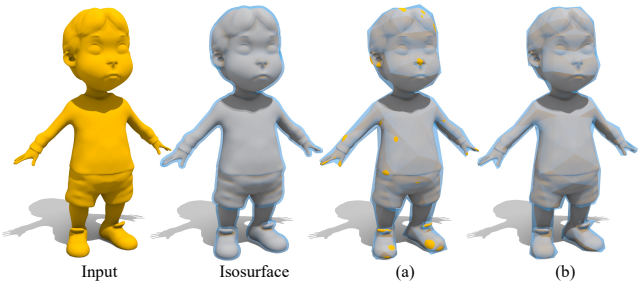


Figure 5: *A comparison of cages produced by simplifying the isosurface with (b) and without (a) the no intersection constraints. The bright yellow region in (a) indicates that the cage does not fully enclose the input shape.*

3.3. Cage Optimization

We can further optimize various qualities of the cage, to suit the needs of different applications. Let $E(\widehat{\mathcal{M}}, \mathcal{M})$ be a quality measure for the cage, with exact expression depending on the application. In the following, we propose two local operations to improve the cage quality. During the execution of each operation, we ensure that the enveloping property is maintained and no self-intersection occurs.

Vertex Relocation In order to strictly enforce the no self-intersection constraints, we include the IPC barrier function $B(v)$ proposed by Li et al [LFS*20] in the energy $E(v) = E(\widehat{\mathcal{M}}, \mathcal{M}) + \lambda B(v)$ with $B(v)$ given by,

$$B(v) = \sum_{i \in C} b(d_i, \hat{d}), \tag{7}$$

where v is the vertex to be relocated, $\hat{d}$ is a user specified distance that the barrier repulsions starts to response to, C contains all the non-incident point-triangle and non-adjacent edge-edge pairs between $N(p)$ and $\mathcal{M} \cup \widehat{\mathcal{M}}$, d_i is the unsigned distance between the pair i and $b(d, \hat{d}) = \max\big((d - \hat{d})^2 \ln(\frac{\hat{d}}{d}), 0\big)$. The IPC barrier effectively avoids the intersection between any primitive pair from C in the process of vertex relocation.

We choose the gradient descent to optimize the energy for vertex relocation, in which a line search is performed to obtain a proper step size along the descent direction such that it

1. reduces the energy,
2. all pairs in C do not intersect

In order to satisfy condition (2), we use the continuous collision detection (CCD) to compute the maximum collision free step for all the pairs in C.

Edge Flipping. We can also improve the quality of the cage by flipping edges. In this case, we only need to check if the quality improves with the edge flipping while making sure that the constraints stay satisfied.

Cage Quality We briefly introduce the cage quality measures that we have experimented with.

- Meshing quality. One of the widely used energies in geometry processing is the MIPS energy, proposed by Hormann and Greiner [HG12]. Given a pair of compatible triangulation $\mathcal{M}$ and $\widehat{\mathcal{M}}$, the MIPS energy is defined in terms of the Jacobians of the piecewise linear mapping between them,

$$E(\widehat{\mathcal{M}},\mathcal{M}) = \sum_{T\in\hat{\mathcal{T}}} \frac{\mathrm{Tr}(J_T^T J_T)}{\det(J_T)} \tag{8}$$

  By choosing equilateral triangles as the reference, MIPS can be used to measure the meshing quality of the cage, which allows us to push the triangles of the cage to be well-shaped during the optimization.
- Cage Spacing. In general, it is preferred that the cages are enveloping the shape as tightly as possible. Yet, there are some applications which requires having some spacing/distance between the cage and shape. For example, when performing shape deformation using the boundary element method [CW17,LCH*21], to avoid singularities of the mapping near the cage, it is typical that the cage is constructed away from the deforming shape. Hence, we propose the cage spacing energy, which allows us to roughly specify the distance between the cage and the surface,

$$E(\widehat{\mathcal{M}},\mathcal{M}) = \int_{x\in\widehat{\mathcal{M}}} (\xi(x)-h)^2 dA \tag{9}$$

  with h being the specified distance. It is worth noting that while the isovalue in Section 3.1 can be set to match the target distance h. However, practical constraints arise due to the discretization of the SDF through grid interpolation, which limits the selection of the isovalue, particularly preventing very small values. Despite this limitation, we allow for more flexibility in choosing the final target distance h. Figure 6 shows a comparison of a cage before and after optimizing the cage spacing energy. Figure 8 shows a series of cages obtained with different cage spacings.

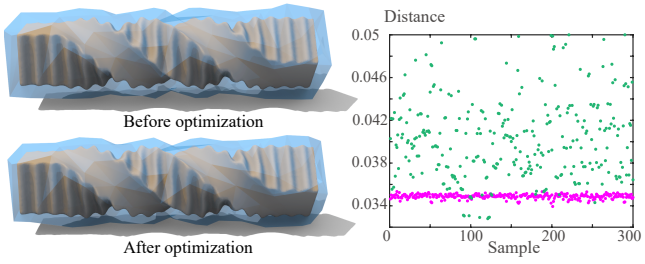


Figure 6: *A comparison of a cage before and after optimizing the cage spacing energy. We specify h as* 0.35. *It can be seen that after the optimization, the distance between the samples on the cage and the input mesh is evenly distributed around* 0.35.

4. Experimental Results

We implemented our method in C++, and performed the experiments on a Windows 10 machine with an Intel i5-8500 CPU 3.0GHz with 32GB RAM. As preprocessing, we normalize all the input meshes by scaling them into unit circle. For all the results presented in this paper, we use the following default parameters unless stated otherwise. We set the isovalue c to 0.01. The thresholds used to determine whether a point pair is in the different semantic part of the model are $r = 0.025$, $l = 0.15$. The parameters for the radial function are $\alpha = 3$ and $\beta = 8$. The weight for barrier function is $\lambda = 0.1$. The SDF and its gradient are computed using OpenVDB [Ope21]. Additionally, OpenVDB provides an implementation of the Marching Cubes algorithm [LC87], which can be used to extract isosurfaces from a 3D scalar field. In the final optimization process, to balance efficiency, we execute these two optimization steps only once.

Figure 7 shows that for a shape with nearly-touching parts, extracting the isosurface directly from the SDF will result in having different topology from the input shape. In contrast, our modified SDF allows to extract isosurface and generate cage homeomorphic to the input isosurface.

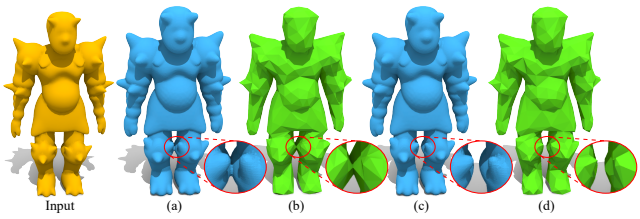


Figure 7: *Isosurfaces extracted from the SDF (a) and the modified SDF (c) and the resulting cages (b and d accordingly). The cage generated from the modified SDF has the nearly-touching parts separated, therefore is shape-aware.*

For certain applications, it is preferable to have a cage whose distance from the input mesh is evenly distributed and controllable. For example, in the cage based deformation [CW17, LCH*21], it is needed to have some offset between the cage and the deforming shape in order to avoid singularities with the harmonic basis functions. With our method, it is easy to control the offset distance by specifying the cage spacing h in the optimization step. Figure 8 shows multiple cages obtained with different cage spacings. Figure 1 shows the deformation results of [LCH*21] of the octopus shape by utilizing the cage generated by our method.

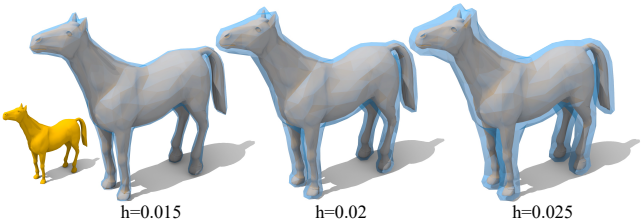


Figure 8: *Cages with different spacings h to the input shape.*

For some applications such as collision proxies, there is a need to have a multi-resolution hierarchy of cages with strict nesting structure. By repeating our method on the generated cages, it is easy to construct a sequence of cages with the coarser layers strictly encaging the finer layers, nesting one another. Figure 9 shows an example.

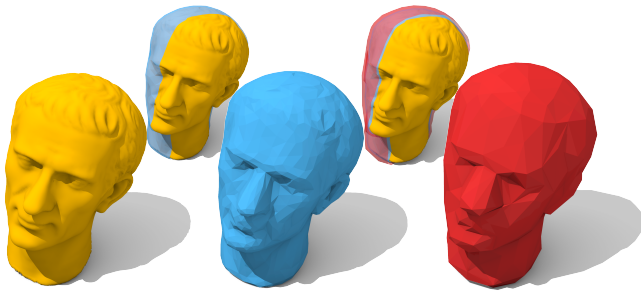


Figure 9: *The man head shape with two nested cages. We generate a cage (blue) from the input shape, and then rerun our algorithm (on the blue cage) to create another cage (red).*

Figure 10 compares the cages between those generated by [SVJ15] and those from our method. It can be seen that our results can better capture the appearance of the input shape with fewer vertices. In Figure 2, we compare the results of our method

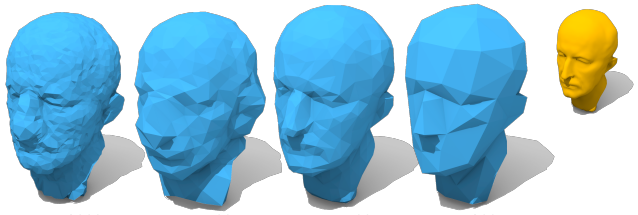


Figure 10: *A comparison of cages with different resolutions generated by Nested Cage [SVJ15] and our method. The two images on the left show cages generated by Nested Cage, which contain* 3000 *and* 500 *vertices respectively. The 3rd and 4th images show the cages generated by our algorithm with* 500 *and* 200 *vertices.*

with Sacht et al. [SVJ15] and Sander et al. [SGG*00]. This figure shows that our cage ensures the distance to the input shape remains within an appropriate range. In many 3D barycentric coordinate applications, such as Green's barycentric coordinates [LLCO08], if the cage vertices are too close to the input shape, it may cause singularities in the barycentric coordinate values. Maintaining a suitable distance helps to prevent this issue. In Figure 11, we compare a hand made cage from [JSW05] and a series of cages generated from our method for the Armadillo shape. It can be seen that our results have similar quality to the hand made cage, and they are still acceptable at very low resolutions. Finally, we include an additional set of cages generated for various shapes in Figure 12.

5. Summary and Discussion

We have presented a simple, efficient and robust cage generation method. Following the envelop-and-simplify paradigm, an isosurface is first extracted from the signed distance field to the input shape, which is guaranteed to be strictly enveloping, free of self-intersection and geometrically and topologically close to the input shape, allowing it to be easily simplified to a coarse cage structure in the second stage and further optimization to the quality can be performed if needed. Furthermore, the versatility of SDF as a shape-aware implicit function allows us to design a simple modification so that the extracted isosurface is homeomorphic to the input shape, which is important for many applications.

Maximally simplified cage For most applications, our simplification step can produce cages that are sufficiently coarse, i.e. having few enough vertices. If there is a need to further compress the cage, it can be achieved by either increasing the isovalue of the surface or repeating the simplification and optimization steps.

Limitation If the shape has extremely close-by parts, then the extracted surface will not be able to separate these parts in practice. This is due to that, the SDF has finite resolution with its underlying representation, which cannot be increased indefinitely.

Future work We have demonstrated an interesting and practical application of the implicit shape representation, in particular, SDF. In the future, it is interesting to see how this representation can help with other geometry processing problems.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. This research was supported by the National Natural Science Foundation of China (62072422, 52405298).

References

[BA05] BAERENTZEN J., AANAES H.: Signed distance computation using the angle weighted pseudonormal. *IEEE Transactions on Visualization and Computer Graphics 11*, 3 (2005), 243–253. doi:10.1109/TVCG.2005.49. 3

[BCWG09] BEN-CHEN M., WEBER O., GOTSMAN C.: Spatial deformation transfer. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, Association for Computing Machinery, p. 67–74. doi:10.1145/1599470.1599479. 2, 3

[CB17] CALDERON S., BOUBEKEUR T.: Bounding proxies for shape approximation. *ACM Trans. Graph. 36*, 4 (jul 2017). doi:10.1145/3072959.3073714. 2

[CF14] CHEN X., FENG J.: Adaptive skeleton-driven cages for mesh sequences. *Computer Animation and Virtual Worlds 25*, 3-4 (2014), 445–453. doi:https://doi.org/10.1002/cav.1577. 2

[CHSB10] CHEN L., HUANG J., SUN H., BAO H.: Cage-based deformation transfer. *Computers & Graphics 34*, 2 (2010), 107–118. doi:10.1016/j.cag.2010.01.003. 2

[CK10] CAMPEN M., KOBBELT L. P.: Polygonal boundary evaluation of minkowski sums and swept volumes. *Computer Graphics Forum 29* (2010). 3

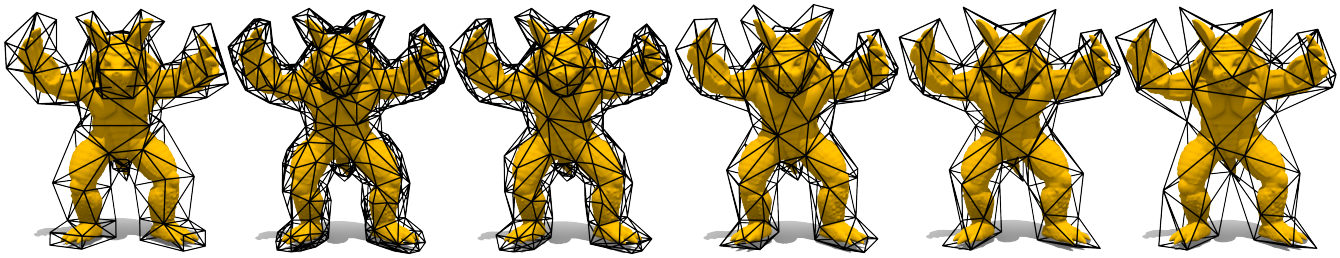


Figure 11: *A comparison between a hand made cage from [JSW05] and our results with five different resolution. The running time for our method to produce the shown cages from left to right are 34.9s, 40.4s, 58.1s, 69.4s and 79.7s respectively.*

[CW17] CHEN R., WEBER O.: Gpu-accelerated locally injective shape deformation. *ACM Trans. Graph. 36*, 6 (nov 2017). doi:10.1145/3130800.3130843. 6

[DLM11] DENG Z.-J., LUO X.-N., MIAO X.-P.: Automatic cage building with quadric error metrics. *Journal of Computer Science and Technology 26*, 3 (2011), 538. doi:10.1007/s11390-011-1153-4. 3

[FKR05] FLOATER M. S., KÓS G., REIMERS M.: Mean value coordinates in 3d. *Computer Aided Geometric Design 22*, 7 (2005), 623–631. 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 209–216. doi:10.1145/258734.258849. 2, 4

[HG12] HORMANN K., GREINER G.: Mips: An efficient global parametrization method. *Curve and Surface Design: Saint-Malo 2000* (11 2012), 10. 6

[JMDG07] JOSHI P., MEYER M., DEROSE T., GREEN B.: Harmonic coordinates for character articulation. *ACM Transactions on Graphics 26*, 3 (2007), 71. 2

[JSW05] JU T., SCHAEFER S., WARREN J.: Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics 24*, 3 (2005), 561–566. 2, 7, 8

[JSZP20] JIANG Z., SCHNEIDER T., ZORIN D., PANOZZO D.: Bijective projection in a shell. *ACM Trans. Graph. 39*, 6 (nov 2020). doi:10.1145/3414685.3417769. 3

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson Surface Reconstruction. In *Symposium on Geometry Processing* (2006), Sheffer A., Polthier K., (Eds.), The Eurographics Association. doi:10.2312/SGP/SGP06/061-070. 3

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1987), SIGGRAPH '87, Association for Computing Machinery, p. 163–169. doi:10.1145/37401.37422. 6

[LCH*21] LIAO W., CHEN R., HUA Y., LIU L., WEBER O.: Real-time locally injective volumetric deformation. *ACM Trans. Graph. 40*, 4 (jul 2021). doi:10.1145/3450626.3459794. 1, 6

[LD17] LE B. H., DENG Z.: Interactive cage generation for mesh deformation. In *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2017), I3D '17, Association for Computing Machinery. doi:10.1145/3023368.3023369. 2

[LFS*20] LI M., FERGUSON Z., SCHNEIDER T., LANGLOIS T., ZORIN D., PANOZZO D., JIANG C., KAUFMAN D. M.: Incremental potential contact: Intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph. 39*, 4 (jul 2020). doi:10.1145/3386569.3392425. 2, 5

[LLCO08] LIPMAN Y., LEVIN D., COHEN-OR D.: Green coordinates. *ACM Transactions on Graphics 27*, 3 (2008). 2, 7

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. *ACM Trans. Graph. 22*, 3 (jul 2003), 463–470. doi:10.1145/882262.882293. 3

[Ope21] OPENVDB: an academy award-winning open-source c++ library. https://www.openvdb.org/, 2021. Accessed: 29-October-2021. 6

[PT03] PLATIS N., THEOHARIS T.: Progressive hulls for intersection applications. *Computer Graphics Forum 22*, 2 (2003), 107–116. doi:https://doi.org/10.1111/1467-8659.00653. 2

[SGG*00] SANDER P. V., GU X., GORTLER S. J., HOPPE H., SNYDER J.: Silhouette clipping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 327–334. doi:10.1145/344779.344935. 3, 7

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1986), SIGGRAPH 1986, Association for Computing Machinery, pp. 151–160. 2

[SVJ15] SACHT L., VOUGA E., JACOBSON A.: Nested cages. *ACM Transactions on Graphics 34*, 6 (2015). 2, 3, 5, 7

[WPG12] WEBER O., PORANNE R., GOTSMAN C.: Biharmonic coordinates. *Computer Graphics Forum 31*, 8 (2012), 2409–2422. doi:https://doi.org/10.1111/j.1467-8659.2012.03130.x. 2

[XB14] XU H., BARBIČ J.: Signed distance fields for polygon soup meshes. In *Proceedings of Graphics Interface 2014* (CAN, 2014), GI '14, Canadian Information Processing Society, p. 35–41. 3

[XLG12] XIAN C., LIN H., GAO S.: Automatic cage generation by improved obbs for mesh deformation. *The Visual Computer 28*, 1 (2012), 21–33. Electronic Supplementary Material The online version of this article (doi: 10.1007/s00371-011-0595-6) contains supplementary material, which is available to authorized users. doi:10.1007/s00371-011-0595-6. 2, 5

Figure 12: *Additional results of cages generated from various shapes. Each triplet shows: input shape, a slice through the cage, and the cage.*