Uniform Offsetting of Polygonal Model based on Layered Depth-Normal Images

Yong Chen*

Department of Industrial and Systems Engineering
University of Southern California

Charlie C. L. Wang

Department of Mechanical and Automation Engineering
The Chinese University of Hong Kong

* Corresponding Author: 213-740-7829, yongchen@usc.edu.

Abstract

Uniform offsetting is an important geometric operation for computer-aided design and manufacturing (CAD/CAM) applications such as rapid prototyping, NC machining, coordinate measuring machines, robot collision avoidance, and *Hausdorff* error calculation. We present a novel method for offsetting (grown and shrunk) a solid model by an arbitrary distance *r*. First, offset polygons are directly computed for each face, edge, and vertex of an input solid model. The computed polygonal meshes form a continuous boundary; however, such boundary is invalid since there exist meshes that are closer to the original model than the given distance *r* as well as self-intersections. Based on the problematic polygonal meshes, we construct a well-structured point-based model, *Layered Depth-Normal Images* (LDNI), in three orthogonal directions. The accuracy of the generated point-based model can be controlled by setting the tessellation and sampling rates during the construction process. We then process all the sampling points in the model by using a set of point filters to delete all the invalid points . Based on the remaining points, we construct a 2-manifold polygonal contour as the resulted offset boundary. Our method is general, simple and efficient. We report experimental results on a variety of CAD models and discuss various applications of the developed uniform offsetting method.

Keywords: Offset surfaces, geometric modeling, trimming self-intersections, layered depth normal images, point-sampled geometry.

1. Introduction

Offsetting a solid *S* by a distance *r* into a grown or shrunken version of *S* has been precisely defined for point sets in *Euclidean* space $\boldsymbol{E}^2$ or $\boldsymbol{E}^3$ (Rossignac and Aristides A. Requicha 1986). As shown in Figure 1, suppose a ball with radius *r* is defined as b_r, we can define the two offsetting operations as:

(1) *S grown by r* as $S\uparrow_r = S \oplus b_r$, and (2) *S shrunk by r* as $S\downarrow_r = S \otimes b_r$,

where a special case of the *Minkowski* sum of *A* and *B*, denoted $A \oplus B$, is defined as $C = A \oplus B = \{a + b \mid a \in A, b \in B\}$, and a special case of the *Minkowski* difference, denoted $A \otimes B$, is $\overline{A \oplus B}$.

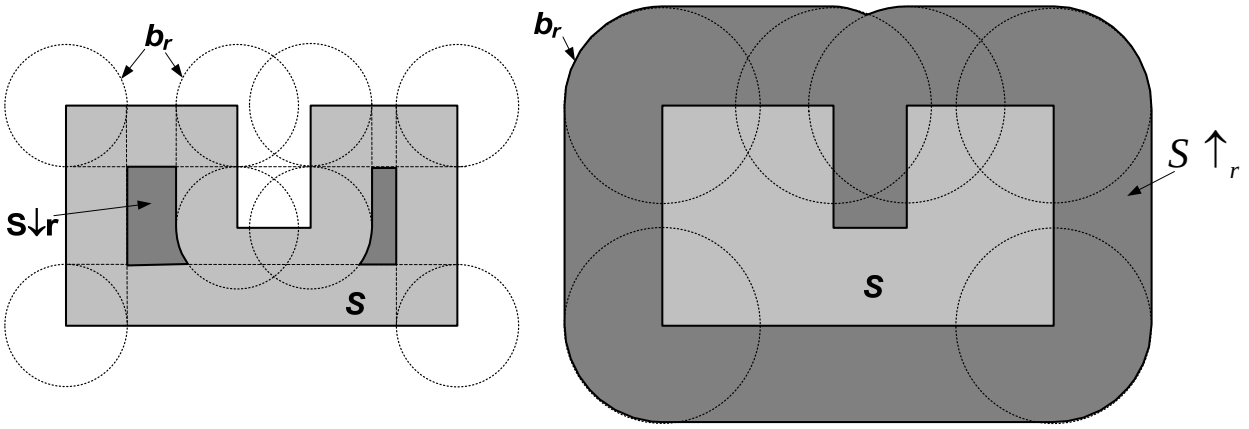


Figure 1. Offsetting a solid *S* by a distance *r*.

Offsetting problems belong to a class of geometric problems that are fundamental and significant to various computer-aided design and manufacturing (CAD/CAM) applications such as shelling, filleting and rounding of 3D models, tool path generation for 3D NC machining, rapid prototyping and coordinate measuring machines, tolerance analysis for assemblies, and robot path planning (Rossignac and Aristides A. Requicha 1986) (Pham 1992) (Maekawa 1999) (Chen et al. 2005). Since boundary representation (b-rep) is one of the most popular representations of 3D solid, we will focus on an input solid defined as a polygonal model. Our goal in this paper is to compute a uniform offsetting model for an arbitrary offset distance to a polygonal model, which has no defects such as gaps, holes, and self-intersections.

Although the offsetting operation is mathematically well defined, computing an offset model for a given solid has proven to be difficult. Position changes by an offset distance generally lead to self-intersections and consequently topological changes. Therefore trimming invalid offset surfaces in the polygonal model is required, which is usually computationally complex and numerically instable. Many degenerate cases between vertices, edges and surfaces need to be carefully considered in implementation. To avoid the difficulties, previous work based on volumetric approach (Chen, Wang et al. 2005; Varadhan and Manocha 2006; Lien 2007; Pavic and Kobbelt 2008) and sampling point approach (Lien 2007) have been presented. Instead of directly trimming the offset polygonal model, these approaches first generate volumetric grids and sampling points to approximate the offset model. Then distance field computation is used to calculate the minimum distance of a point to the original boundary, which is compared with the offset distance for its inside/outside property. Finally an offset model can be reconstructed from the volumetric representations.

1.1 Our Approach

In this work, we follow the volumetric approaches on computing the offset boundary of a given solid. However, different from the aforementioned volumetric approaches that are based on distance field computation, our method is based on directly computing offset boundary, converting the boundary into structurally sampled points and accordingly filtering the sampling points for reconstructing offset contour. An illustration of our method for a 3-dimensional (3D) solid model is shown in Figure 2. We first compute a set of offset surfaces directly from the vertices, edges and triangles of the input model. The offset surfaces form a continuous boundary (refer to Figure 2.a). However, the generated offset surfaces may have self-intersections and there are surfaces that are closer to the original model than the distance *r*. To trim the

invalid offset meshes, we construct a well-structured point representation, named *Layer Depth-Normal Images* (LDNIs) (Chen and Wang 2008), to sample the offset meshes (refer to Figure 2.b). The accuracy of the generated LDNIs models can be controlled during the construction process. We then process all the sampling points in the offset LDNIs by using a set of point filters. Accordingly all the invalid points are identified and discarded from the offset LDNIs. The remaining points after the filtering process are shown in Figure 2.c. Finally, we reconstruct an offset contour from the processed LDNIs model by using an adaptively sampling (Chen and Wang 2008) and manifold-preserving contouring methods (Wang and Chen 2008). The computed uniform offsetting model is shown in Figure 2.d with a magnified view (***A***~***D***) for each major step of the method.

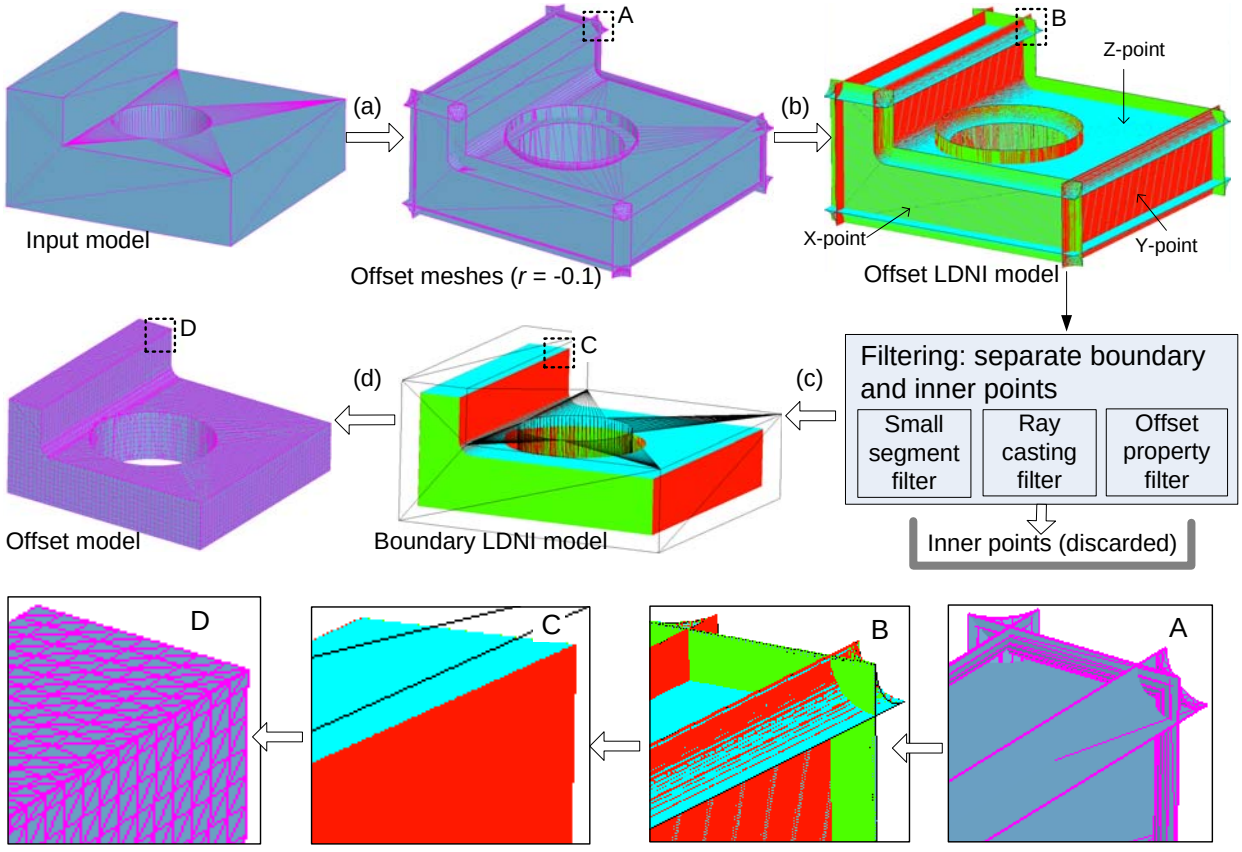


Figure 2. An overview of our method. First, a continuous offset boundary is constructed; we then sample the boundary from three axes to compute a point-based model; the sampling points are filtered and inner points are discarded; finally the remaining boundary points are used to construct the contour of offset model.

1.2 Key Contributions:

We present a novel approach to compute the uniform offsetting boundary for an input polygonal model and an offset distance. Our approach is based on the mathematical definition of offsets. It has several benefits which are listed as follows.

- **General**: We provide a practical method to offset general 3D surfaces represented in polygonal meshes. It can handle an arbitrary offset distance to generate both grown and shrunk models. The challenge of detecting singularities and eliminating self-intersections is handled by the point-based representation. Hence the topology of the reconstructed offset model can be quite different from that of the input model.

- **Accurate**: The resulting polygonal model is a close approximation of the exact offset boundary. Its approximation error is bounded by the tessellation density and the sampling resolution used in constructing the LDNIs model. Based on a mesh tiling technique, the LDNIs resolution can be set sufficiently small and satisfy the requirements of most CAD/CAM applications.
- **Efficient**: Our method directly computes the offset surfaces and accordingly a related LDNIs model. For each sampling point, we judge its inside/outside property directly based on a ray casting test instead of the minimum distance computation form the original model. Therefore the performance of our method is less related to the offset distance; while other offset approaches (Chen, Wang et al. 2005; Pavic and Kobbelt 2008) will dramatically slow down for a bigger offset distance. In addition, several key steps in processing a LDNIs model can be parallelized.
- **Simple**: Our method is mainly based on LDNIs. Comparing to other methods such as the ones based on the boundary representation, our method is relatively easy to implement.

To illustrate the benefits, we present two offset examples that are generated by our method in Figures 3 and 4 respectively. In Figure 3, both the grown and shrunk models of a complex dragon model (publically available from Stanford 3D Scanning Repository) are given. Notice the topological changes in the offset models. In Figure 4, both the grown and shrunk models of an engineering model (a hub) are given, where the sharp corners and edges in the offset boundary are well captured.

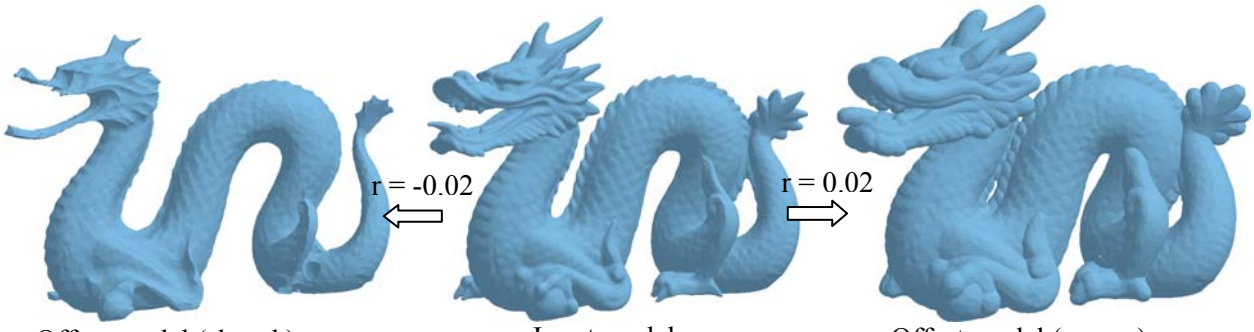


Figure 3. Screen capture of the offset results for a dragon model.

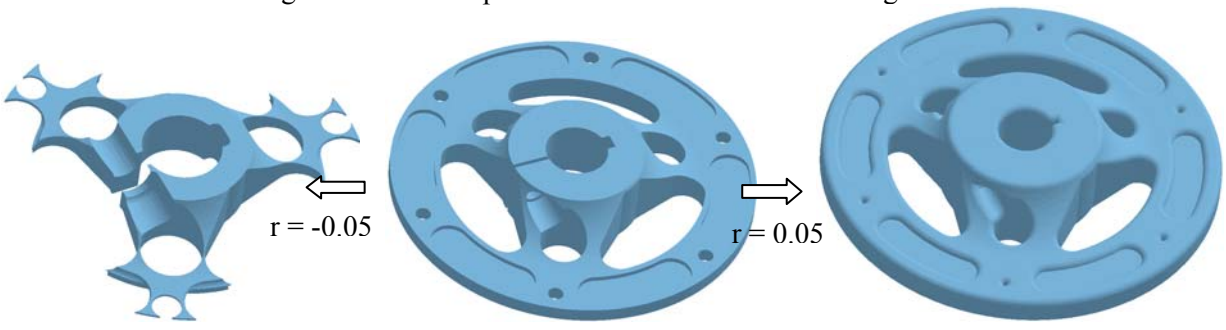


Figure 4. Screen capture of the offset results for a hub model.

2. Related Work

Earlier approaches (Rossignac and Aristides A. Requicha 1986) (Farouki 1985) (Satoh and Chiyokura 1991) (Frosyth 1995) first compute a superset of offset surfaces by offsetting vertices into spheres, edges into cylinders, and faces into parallel faces. Then, they trim that superset by subdividing its elements at their

common intersections and by deleting the pieces that are too close to the original solid. The computational complexity and numeric difficulty of trimming makes these approaches difficult to be implemented robustly.

To avoid the computational difficulties in surface trimming, some approximation approaches have also been investigated. For example, Qu and Stucker (Qu and Stucker 2003) presented an offset method based on moving triangle vertices while maintaining the same topology. The approach of calculating the position of offset vertices was presented. However, such an approach is limited to sufficiently small offset values since topological changes due to self-intersections are not considered.

Another approximation approach is to convert the offsets of a 3D model into the offsets of 2D contours. For example, Lam *et al.* (Lam, Yu et al. 1997) described an approach based on slicing geometries into 2D contours and offsetting each slice contour based on pixels. McMains *et al.* (McMains, Smith et al. 2000) presented an algorithm for building thin-walled parts in fused deposition modeling (FDM) machine. Geometries are sliced first followed by creating 2D offset contours. Allen and Dutta (Allen and Dutta 1998) also developed an algorithm for building thin shell surfaces with minimum supports in layered manufacturing processes. However, these approaches are mainly used for special applications such as layered manufacturing. No offset surface model is constructed.

A ray-rep representation and related computation method have been developed for offsets, sweeps, and Minkowski operations (Hartquist, Menon et al. 1999). A ray-rep is a set of line segments that lie inside the solid and generated by clipping a regular grid of lines against a solid model. The ray-rep representation stores only depth values of intersection points in one ray direction. In comparison, the LDNIs-based method uses both depth values and normals of sampling points that are generated in three orthogonal directions. Also the reconstruction of offset model was not considered in (Hartquist, Menon et al. 1999).

An offset method based on distance volume and fast marching method was presented for CSG models (Breen and Mauch 1999). The approach calculates the shortest distance to the CSG model at a set of points within a narrow band around the evaluated surfaces. Additionally, a second set of points, labeled as the zero set which lies on the CSG model's surfaces, are computed. A point in the zero set is associated with each point in the narrow band. Once the narrow band and zero set are calculated, a fast marching method is employed to propagate the shortest distance and the closest point information out to the remaining voxels in the volume. The approaches of calculating distance maps and their representations can also be found (Gibson 1999; Frisken, Perry et al. 2000). Based on the idea of computing unsigned distance fields, Kim et al. (Kim, Varadhan et al. 2003) presented a five-stage pipeline to approximate the swept volume of a polyhedron along a given trajectory. The major steps include classifying the grid points on a uniform grid, using fast marching front propagation and reconstructing volume from iso-surfaces.

More recently, Chen et al. (Chen, Wang et al. 2005) presented a point based offsetting approach, in which, uniform sampling points are first generated from the original model. Then for each point, a set of offsetting lines are constructed to mark all intersected voxel grids and offset points valid/invalid; finally offset surfaces are reconstructed from the marked voxels and offset points. Varadhan and Manocha (Varadhan and Manocha 2006) presented an algorithm to approximate the 3D Minkowski sum of polyhedral objects. The union of pairwise convex Minkowski sums is computed by generating a voxel grid, computing signed distance on the grid points and performing isosurface extraction from the distance field. Lien (Lien 2007) used sampling points to compute the Minkowski sum boundary based on the normal filter and the

collision detection technique. However, the approach can only generate a set of points instead of meshes. Pavic and Kobbelt (Pavic and Kobbelt 2008) presented a volumetric approach based on identifying octree cells whose minimum distance to the original model is less than the offset distance while the maximum distance to the original model is larger than the offset distance. The identified cells, which intersect the offset surfaces, are iteratively subdivided until it reaches the finest resolution level. Finally the offset surfaces can be extracted from the generated octree cells.

3. Principle of the LDNI-based Offsetting Method

Our method is based on the mathematical properties of offsets. Suppose ∂S is the topological boundary of a set *S*. Nadler (Nadler 1978) define the regularized offset of a regular set *S* by a positive distance *r* as $S \uparrow^* r = \{p : d(p,S) \le r\}$, where $d(p,S) = \inf_{q \in S} \|p - q\|$ and *inf* denotes the greatest lower bound. From this definition, $\partial(S \uparrow^* r) \subset \{p : d(p,S) = r\}$. For $p \notin S, d(p,S) = d(p,\partial S)$. The regularized negative offset of a non-empty *S* is defined as the complement of the positive offset of the complement of *S*. So the analogous result in terms of point/set distances for a negative offset of solid *S* is $\partial(S \downarrow^* r) \subset \{p : d(p, c^* S) = r\}$, where c^* denotes regularized complement. Since $S \downarrow^* r$ can be directly derived from $S \uparrow^* r$, we will focus on $S \uparrow^* r$ from now on and briefly mention $S \downarrow^* r$.

The principle of our method is to first generate a super set $\{p : d(p,\partial S) = r\}$ and then calculate the offset surfaces $\partial(S \uparrow^* r)$ from it. For a regularized set *S* defined as a polygonal model, its boundary $\partial S = V(S) \cup E(S) \cup F(S)$ where *V*(*S*), *E*(*S*), and *F*(*S*) refer to the vertices, edges and faces of *S* respectively. Correspondingly, for point $q \in \partial S$, we can calculate the set $\{p : d(p,q) = r\}$ as $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$, where $V\|_r^+$, $E\|_r^+$, and $F\|_r^+$ are the positive normal offset of *V*(*S*), *E*(*S*), and *F*(*S*) respectively.

(1) Faces *F*(*S*): Suppose *f* is a face of *S* and $q \in f$. We can construct the set $F\|_r^+$ by displacing each point $\boldsymbol{q}$ a distance *r* along the unit normal $\boldsymbol{n}$ of *f*, i.e., let $\boldsymbol{p} = \boldsymbol{q} + r\boldsymbol{n}$.

(2) Edges *E*(*S*): Suppose *e* is an edge of *S* and $q \in e$. The two neighboring faces of *e* are f_1 and f_2. We can construct the set $E\|_r^+$ by constructing a cylinder centered at *e* with radius *r*. The cylinder can be bounded by $f_1\|_r^+$ and $f_2\|_r^+$ because the points outside the bounded portion are closer to f_1 and f_2.

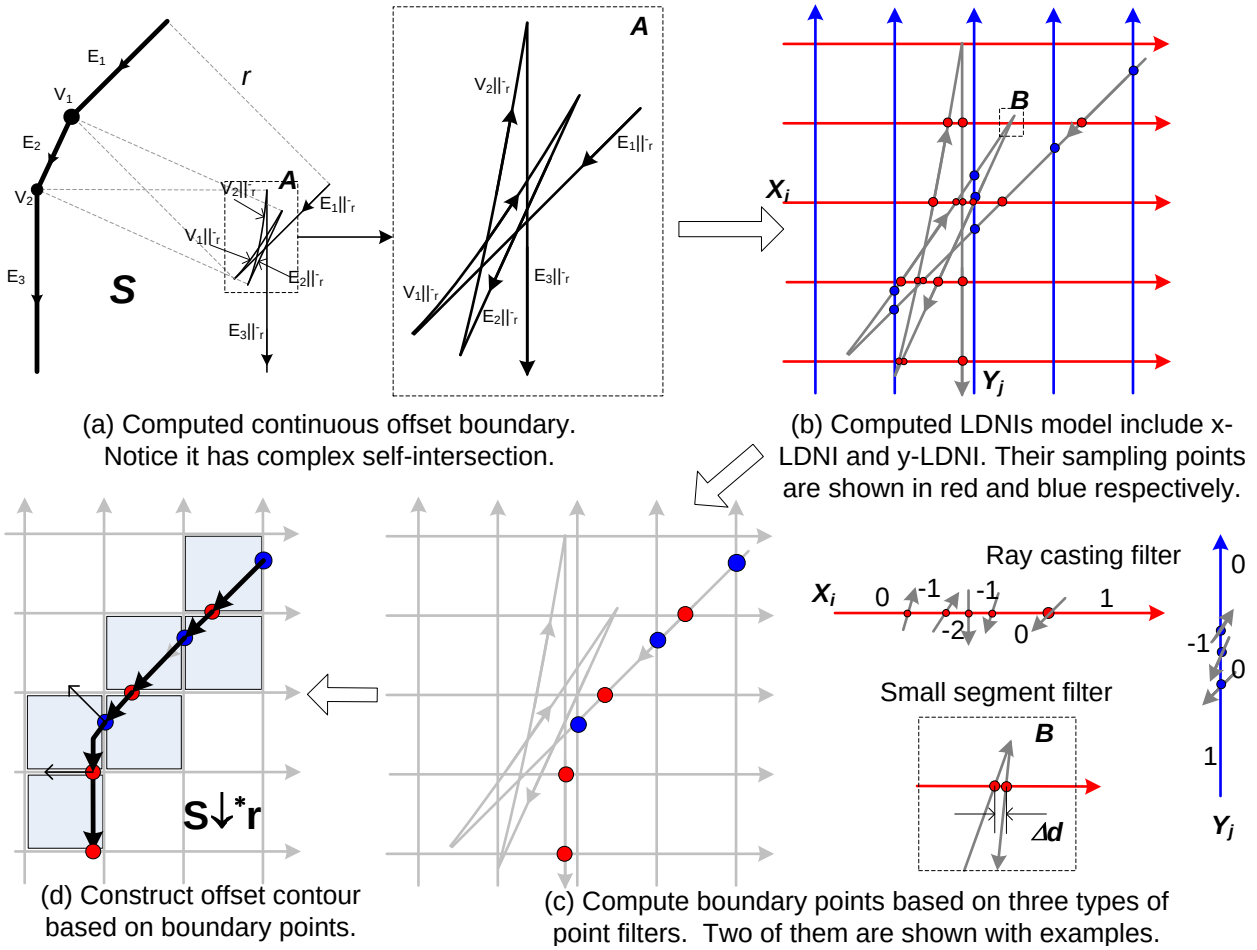
(3) Vertices *V*(*S*): Suppose *v* is a vertex of *S* and $q = v$. The neighboring edges of *v* are e_1, e_2, …, e_i. We can construct the set $V\|_r^+$ by constructing a sphere centered at *v* with radius *r*. The sphere can be bounded by $e_1\|_r^+$, $e_2\|_r^+$,…, $e_i\|_r^+$ because the points outside the bounded portion are closer to the neighboring faces.

Therefore, $\partial(S \uparrow^* r) \subseteq F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$. That is, any point $p \in \partial(S \uparrow^* r)$ must be attained from $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$. Notice all points of $F\|_r^+$, $E\|_r^+$, $V\|_r^+$ are at a distance *r* from some points of ∂S (suppose q_i); however some of them may be at a smaller distance to other points of ∂S (suppose q_k). When *p* is an invalid point, $q_i\|_r^+$ and $q_k\|_r^+$ will intersect each other. Such self-intersection is a core challenge to be addressed. We define **inner points** as the sampling points on $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$ whose minimum distance to $\partial(S)$ is less than the offset distance *r*, and **boundary points** as all the sampling points on $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$ that are also on $\partial(S \uparrow^* r)$. Therefore, the essence of our method is to remove the inner points from the set $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$ such that the boundary points can be computed to approximate the boundary of $S \uparrow^* r$.

The uniform offsetting considered in this paper is a special case of general *Minkowski* operations. Guibas et al (1983) presented a framework which converts *Minkowski* operations into convolution operations.

A set of concepts such as polygonal tracings and state counting functions have been proposed and the related properties have been studied. Our offsetting approach shares the similarity to their framework on computing the offset boundary by a polygonal tracing tour and judging its ray casting values based on winding numbers.

We illustrate the aforementioned principle by using a 2D case as shown in Figure 5. Suppose a portion of the boundary for defining a 2D region S is shown in Figure 5.a. It includes three edges ($E_1 \sim E_3$) and two vertices (V_1, V_2). To offset $\partial(S)$ by a distance r (shrunk), we can compute $e\|_r^-$ for each edge of $E(S)$ and $v\|_r^-$ for each vertex of $V(S)$. Notice $V\|_r^- \cup E\|_r^-$ form a continuous boundary (refer to a magnified view ***A*** in Figure 5.a-right). It also has multiple self-intersections. We use a well-structured point representation (LDNIs) to sample the boundary $V\|_r^- \cup E\|_r^-$. Figure 5.b gives a two-dimensional illustration of the constructed LDNIs model, where the red dots indicates the points recorded on the *x*-LDNI and the blue ones illustrate the points on the *y*-LDNI. Hence each pixel of a LDNI contains a sequence of *Hermite* data that specify the depths from the intersections to the viewing plane and the unit normal vector of the sampled surface at the intersection point. Furthermore, all the depths of a pixel are sorted in the ascending order.



(a) Computed continuous offset boundary. Notice it has complex self-intersection.

(b) Computed LDNIs model include x-LDNI and y-LDNI. Their sampling points are shown in red and blue respectively.

(c) Compute boundary points based on three types of point filters. Two of them are shown with examples.

(d) Construct offset contour based on boundary points.

Figure 5. A 2D illustration example of S shrunk by a distance r.

A set of point filters have been developed to process all the LDNI points. For example, the ray casting test values of pixels $\boldsymbol{X_i}$ and $\boldsymbol{Y_j}$ are shown in Figure 5.c. A point will be deleted if its two neighboring regions do not have the ray casting value of (0, 1) or (1, 0). As another example, the two LDNI points as shown in

view ***B*** define a small segment. The two points will be deleted if $\Delta d < \varepsilon$, where ε is a small tolerance such as 10^{-5}. After the filtering process, the remaining LDNI points are shown in Figure 5.c-left. Notice the exact surface normals at these points are also known. Hence based on the *Hermite* data at these points, a manifold contour can be computed from the cells defined by the intersections of the rays related to x-LDNI and y-LDNI pixels. Such a contour is an approximated boundary of $\partial(S\downarrow^{*} r)$ (refer to Figure 5.d).

We further illustrate our method by using a 2D contour sliced from a 3D model as shown in Figure 6.a. The offset boundary $V\|_r^- \cup E\|_r^-$ of its *internal loop 1* is shown in Figure 6.b. As illustrated in two magnified views ***A*** and ***B*** in the figure, complex self-intersections exist in the offset boundary. As discussed before, we handle them by converting the boundary into a LDNIs model and using point filters to process the sampling points. The filtered LDNIs points are shown in Figure 6.c-top. The offset boundary of its *internal loop 2* and related LDNI points are shown in Figure 6.c-middle. Notice if only the ray casting filter as shown in Figure 5.c is used, wrong judgment can be made in classifying points. For example, the resulted points based on such a filter are shown in Figure 6.c-right. For such inner points, another type of filter, offset property filter, can correctly remove them. Accordingly, the reconstructed contour to approximate $\partial(S\uparrow^{*} r)$ is shown in Figure 6.d.

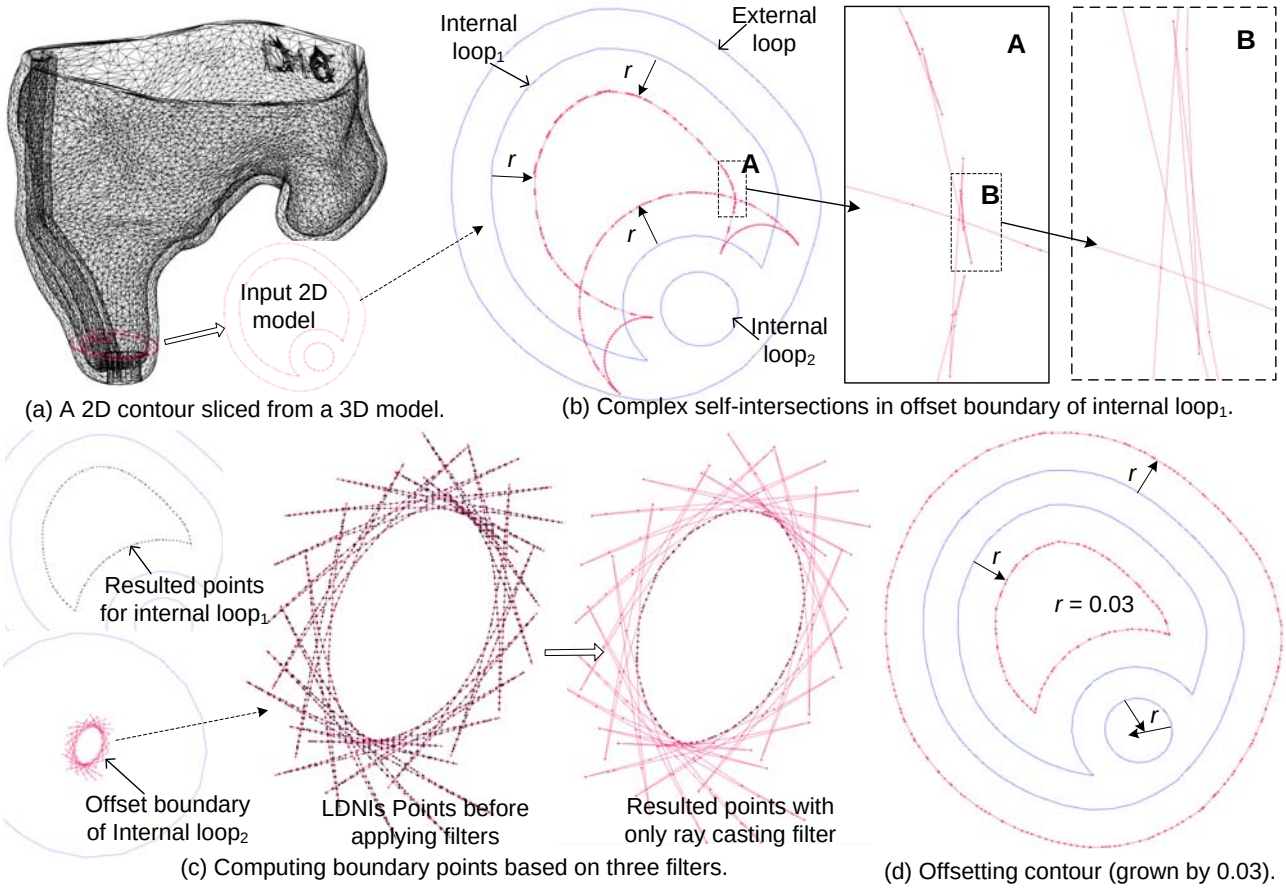
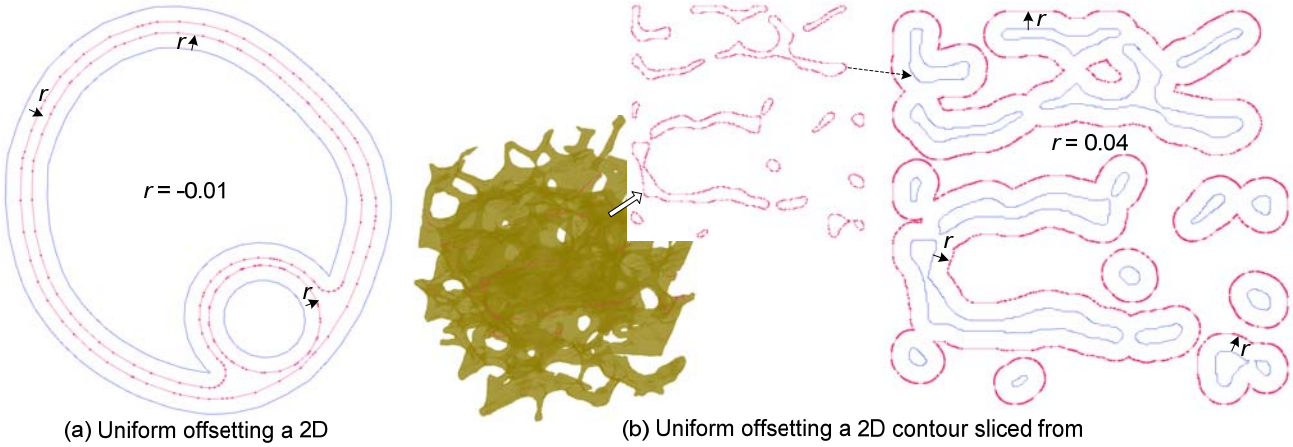


Figure 6. An example of a 2D contour grown by a distance *r*.

Similarly, the contour for $\partial(S\downarrow^{*} r)$ can be computed. A test example based on shrinking the same input model is shown in Figure 7.a. The uniform offsetting of a more complex contour is shown in Figure 7.b,

in which a 2D contour with 18 loops is sliced from a cancellous bone structure model. The offset contour computed by our method for such an input model is shown in Figure 7.b-right.



(a) Uniform offsetting a 2D contour by 0.01 (shrunk).

(b) Uniform offsetting a 2D contour sliced from a scanned bone model by 0.04 (grown).

Figure 7. An illustration of *S* shrunk by a distance *r* and a more complex 2D example.

Our method is based on an approximation approach. Hence it suffers from the same shortcomings as all other volumetric approaches. That is, any features (gaps or shells) whose size is smaller than the LDNIs resolution may not be captured in the computed offset model. However, compared to other volumetric approaches that are only based on cells or voxles, our approach is much more accurate because: (i) we can accurately compute the offset boundary $V\|_r^- \cup E\|_r^-$; (ii) we accurately compute the ray casting points on $V\|_r^- \cup E\|_r^-$ along a set of rays. Therefore, the boundary points on the rays related to LDNIs pixels are accurate (both their positions and normals). Although in practical implementation, a tessellated version is computed instead of an exact $V\|_r^- \cup E\|_r^-$, the accuracy actually can be controlled by choosing different density in tessellation. Consequently the computed offset contour is a close approximation of the exact offset boundary on features whose sizes are bigger than the LDNIs resolution. A volume tiling technique has been developed to ensure a sufficiently small LDNIs resolution (Chen and Wang, 2008). Since the small features that may be missed in our offset contour are usually unmanufacturable, our approach is appropriate for most CAD/CAM applications.

The remainder of the paper is organized as follows. The approach of computing candidate offset meshes is presented in Section 4. The approach of determining valid offset points is presented in Section 5. The approach of reconstructing an offset contour from the processed LDNIs model is presented in Section 6. We discuss the implementation techniques related to our method in Section 7. The experimental results of various test cases are presented in Section 8. We also discuss some applications of our offsetting method in the section. Finally, conclusion with future work is drawn in Section 9.

4. Compute Continuous Offset Boundary

Offsetting a face, a edge and a vertex into a face, a cylinder and a sphere respectively has been discussed before (Lee 1999; Kim, Lee et al. 2004). In our method, we further ensure the offset meshes $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$ will form a continuous boundary with a consistent surface orientation.

Theorem 4.1. *Let S be a regularized point set and ∂S be its boundary defined as a polygonal model. The offset meshes* $F\,||_r^+ \cup E\,||_r^+ \cup V\,||_r^+$ *as defined in Section 3 will form a continuous boundary with a consistent surface orientation.*

Proof. According to the definition of positive normal offset, we know:

(1) the offset $F\,||_r^+$ is a set of translated faces by an offset distance in their surface normal directions (refer to an example as shown in Figure 8.b);

(2) For a valid input model ∂S, each edge *e* must have two half-edges, he_1 and he_2, corresponding to its two neighboring faces f_1 and f_2. Edges $he_1\,||_r^+$ and $he_2\,||_r^+$ in $F\,||_r^+$ will be on the same cylinder centered at *e* with radius *r*. Further since he_1 and he_2 have opposite directions in ∂S, we must be able to construct a portion of a cylinder that is bounded by the reverse edge of $he_1\,||_r^+$ and $he_2\,||_r^+$ to seal the gap between $f_1\,||_r^+$ and $f_2\,||_r^+$. Hence $E\,||_r^+$ will form a continuous boundary with $F\,||_r^+$ (refer to an example as shown in Figure 8.c);

(3) For a valid input model ∂S, each vertex *v* has a set of neighboring edges e_1, e_2, …, e_n. The constructed cylinders in $E\,||_r^+$ related to e_i at *v* must have an edge that is on the great circle of the sphere centered at *v* with radius *r*. Further, all the edges will have a consistent orientation and we can use them to form a closed loop. Therefore, we must be able to construct a spherical region bounded by the reverse edges of the loop to seal the gap between $e_i\,||_r^+$ $(1 \le i \le n)$. Hence $V\,||_r^+$ will form a continuous boundary with $E\,||_r^+$ (refer to an example as shown in Figure 8.d-right). □

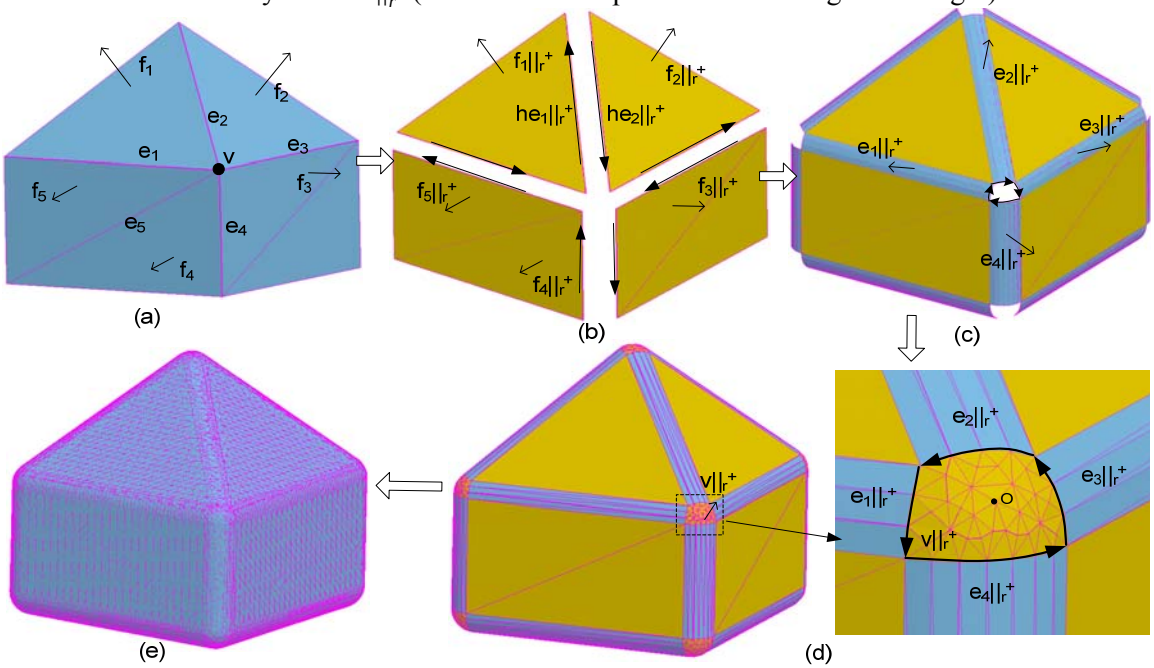


Figure 8. An illustration of offsetting faces, edges and vertices.

Notice the offset meshes may be degenerated. For example, edge e_5 in Figure 8.a has two neighboring faces with the same normal. Hence the cylinder portion related to e_5 is degenerated to a line; and the related edge $e_5\,||_r^+$ for *v* is degenerated into a vertex. For ∂S as shown in Figure 8.a, the offset boundary $F\,||_r^+ \cup E\,||_r^+ \cup V\,||_r^+$ is given in Figure 8.d. Based on our point-based method, the reconstructed contour by removing self-intersections and overlapping surfaces is shown in Figure 8.e.

The construction of $F\|_r^+$ and $E\|_r^+$ is straightforward since they are the translated faces of F and the truncated cylinders of E respectively; however, the construction of $V\|_r^+$ needs further consideration. The neighboring edges e_1, e_2, …, e_n around a vertex v can be both concave and convex. The offset meshes of such a complex vertex can be challenging to process. Basch et al. (1996) presented a notion of polyhedral tracings for convolutions between polyhedrons. Three types of polyhedral tracing that are on face, edge and vertex domains have been developed. It is illustrated that complex self-intersecting path may exist on the sphere related to the vertex. In our method, we mark such complex vertices and convert their offset meshes into structured sampling points. Two related filters (small segment and offset property) are then used to process the sampled points (refer to Section 5.2 and 5.4).

An example of such a complex vertex is shown in Figure 9.a. Among the 7 neighboring edges, e_2 and e_6 are concave while all other five edges are convex. The offset cylinders for all the edges are shown in Figure 9.b. The cylinders related to e_2 and e_6 are not shown in the figure since their face normals are opposite to the viewing direction. However, as shown in Figure 9.c, we can still form a closed loop on the sphere related to all the edges even thought the loop might have multiple self-intersections. The loop edges related to e_2 and e_6 are shown in dotted lines in Figure 9.c. In our method, a spherical region $v\|_r^+$ will be constructed, whose boundary Ψ is the reverse edges of the closed loop generated by edges e_1~e_7. Hence $v\|_r^+$ and $e_i\|_r^+$ will still form a continuous boundary along Ψ.

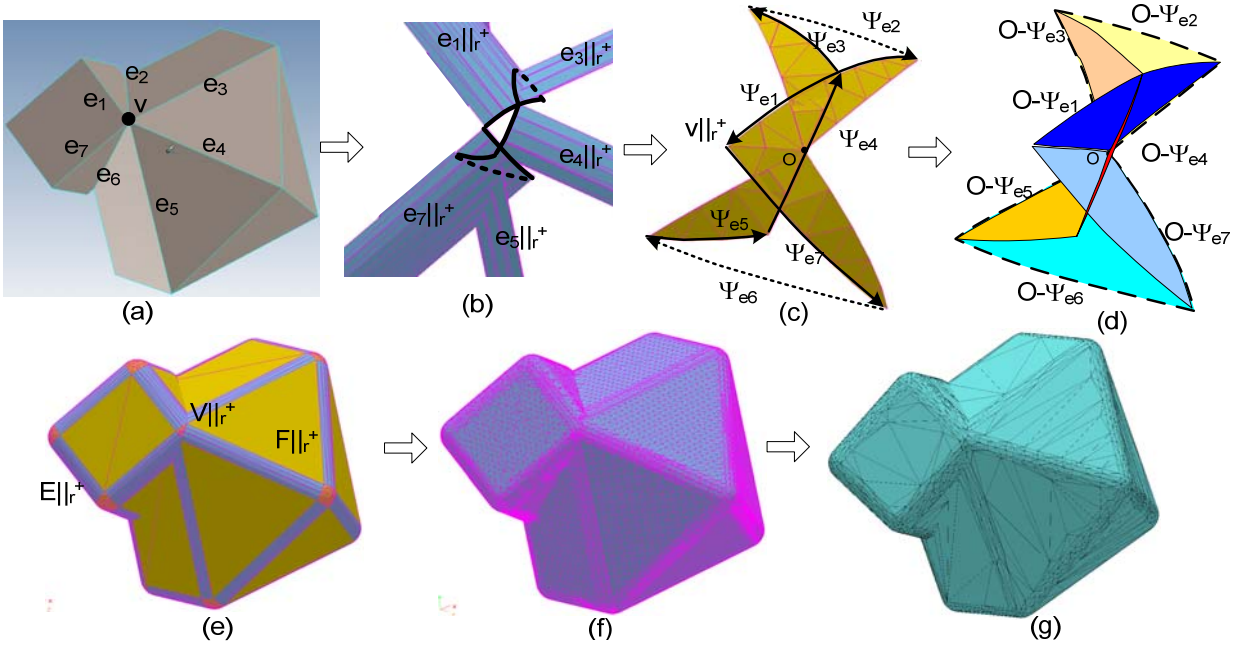


Figure 9. An illustration of offsetting a complex vertex.

Our approach of computing the offset meshes for an any given vertex is described as follows. Suppose a vertex v has a neighboring edge e_i with two neighboring faces f_1 and f_2. Based on a plane defined by the normals of f_1 and f_2, we can construct a great circle. Among the two arcs on the great circle, the loop edge Ψ_{ei} is always the smaller one. This is because: (i) If e_i is a convex or flat edge, the angle α between f_1 and f_2 must be $0^o < \alpha \le 180^o$. The positive normal offset of f_1 and f_2 will form an angle $\angle n_1 v n_2 = 180^o - \alpha$. Therefore, Ψ_{ei} is smaller than half of a great circle. (ii) If e_i is a concave edge, the angle α between f_1 and f_2 must be $180^o < \alpha < 360^o$. The positive normal offset of f_1 and f_2 will form an angle $\angle n_2 v n_1 = \alpha - 180^o$.

Therefore, Ψ_{ei} is still smaller than half of a great circle. Such an observation has also been presented in (Basch et al. 1996). In addition, all the computed arcs Ψ_{ei} will form a closed spherical loop Ψ (refer to the example as shown in Figure 9.c).

The spherical loop Ψ may have multiple self-intersections. Our approach of constructing $v\,||_r^+$ based on such a spherical loop is described as follows.

(1) Suppose C_{ei} is the center of an edge Ψ_{ei}. We first calculate point O' for Ψ based on their edge lengths as $O' = \frac{\sum_{i=1}^{n} || \Psi_{ei} || C_{ei}}{\sum_{i=1}^{n} || \Psi_{ei} ||}$. We then project O' onto the sphere to get a point O. Consequently a plane P that is tangent to the sphere at O can be generated. We then map all the vertices V_i of Ψ onto P (denoted as V_i') by ensuring point v, O, V_i, and V_i' in the same plane and the length OV_i' the same as the arc length OV_i. Consequently, based on such a mapping, the spherical loop Ψ is converted into a 2D planar loop Ψ'.

(2) We then use a triangulation method that is similar to the well-know XOR polygon filling algorithm to compute a triangulation of Ψ'. That is, we construct a triangulation for each edge Ψ_{ei}' individually. An example of such triangulation result for a boundary Ψ in Figure 9.c is shown in Figure 9.d. Notice to ensure the continuity of the offset meshes, some triangles $O\text{-}\Psi_{ei}$ may be flipped depending on the direction of Ψ_{ei} related to O (refer to $O\text{-}\Psi_{e2}$ and $O\text{-}\Psi_{e6}$). Therefore the constructed offset meshes for a vertex v may have overlapping surfaces.

(3) The planar triangulations are refined and mapped back to the sphere. The refinement of triangulation should be controlled to ensure the same triangulation boundaries are generated between $\text{O-}\Psi_{ei}$ of $V\,||_r^+$ and the related loop edge of $E\,||_r^+$. An example of the related $V\,||_r^+$ and $E\,||_r^+$ is shown in Figure 9.e.

(4) As discussed in Section 5, the offset meshes are then converted into sampling points and the point related to overlapping faces will consequently be filtered by a small segment filter. This is based on the property of the XOR polygon filling algorithm. That is, the inside/outside property of a point is determined by the ray casting values at a sampling point p. Two overlapping faces will not change the ray casting value at p if the two faces have flipped normals.

(5) In addition, we further identify all the sampling points P_i that are constructed by a complex vertex v (i.e. v has both concave and convex neighboring edges) and classify them as valid or invalid if its related face normal is the same or reverse to vP_i respectively. Such classification will be used in the offset property filter to remove internal shells, if any, that contain invalid points of complex vertices.

Based on the processed points, a manifold offset contour can be reconstructed (refer to an example as shown in Figure 9.f). The resulted contour can be further decimated as shown in Figure 9.g. Our approach of removing self-intersections and overlapping surfaces in the offset meshes $F\,||_r^+ \cup E\,||_r^+ \cup V\,||_r^+$ is discussed in the next section.

5. Compute Boundary Points based on LDNIs

Directly trimming self-intersections and overlapping surfaces based on the boundary representation is quite challenging to be implemented robustly. In our method, we use a point-based representation, layered depth normal images (LDNIs) to extract the boundary surfaces of $\partial(S \uparrow^* r)$. Based on a required resolution, we first generate a set of sampling points from the offset meshes. We then use three filters to separate the

points into two groups: boundary and inner points. All the inner points will be discarded; only boundary points will be used in reconstructing the offset contour.

5.1. Layered Depth Normal Images

The Layered Depth-Normal Images (LDNIs) is a point representation which sparsely encodes the shape of solid models in three orthogonal directions (Chen and Wang, 2008). A structural set of Layered Depth-Normal Images (LDNIs) consists of x-LDNI, y-LDNI and z-LDNI along *X*, *Y*, and *Z* axis respectively. The three images are located to let the intersections of their rays form the $w_X \times w_Y \times w_Z$ nodes of uniform grids in $\Re^3$. A LDNI in an axis is a sequence of two-dimensional images. For each pixel (*i*, *j*), we shoot a ray from its center along the axis and calculate the intersections of the ray and the surfaces under sampling. Consequently, for each pixel (*i*, *j*), we can build a sequence of four-tuple (d, n_x, n_y, n_z), where *d* specifies the depth from an intersection point *P* to the viewing plane, and $N_P(n_x, n_y, n_z)$ is the surface normal at *P*.

The construction of a LDNIs model for a polygonal model can be performed rather quickly with the aid of graphics hardware (Chen and Wang, 2008). We can set the viewing parameters by the working envelope, which is slightly larger than the bounding box of the model. An orthogonal projection is conducted for rendering so that the intersection points from parallel rays can be generated by the graphics hardware. In order to get an accurate surface normal, we encode a unique ID of every polygonal face into a RGB-color. After rendering all the faces by the encoded colors, we can easily identify a face that is intersected with a ray and accordingly retrieve its surface normal from the input model. The accuracy of a generated LDNIs model depends on the pixel width used in the rendering process. We can use volume tiling to achieve high accuracy requirement by splitting the bounding box of a model into multiple smaller tiles. A LDNIs model for each tile can then be generated and processed independently (either sequentially or in parallel).

Therefore, from the offset surfaces $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$, we can compute a LDNIs model, in which the sampling points can capture all the boundary information along pre-defined uniform grids. For a pixel (*i*, *j*) in a LDNI, we got a set of sampling points $P_1 \sim P_n$ within a range (d_{min}, d_{max}) along the axis. Point $P_1 \sim P_n$ can be sorted according to their distances from d_{min}. We can easily calculate the normal index number I_{Norm} for all the line segments of the ray (d_{min}, d_{max}). That is, I_{Norm} is an accumulated integer value along a ray such that for any point P_i with unit normal $\boldsymbol{N_{Pi}}$, we increase I_{Norm} by 1 if $N_{pi} \cdot N_{ray} < 0$ and decrease I_{Norm} by 1 if $N_{pi} \cdot N_{ray} > 0$. The unit normal $\boldsymbol{N_{ray}}$ is along the LDNI axis from d_{min} to d_{max}. Refer to an example in Figure 5.c. It is obvious that each line segment will have a unique integer value I_{Norm} (can be positive or negative). Such I_{Norm} value is called winding numbers in (Guibas et al. 1983).

We present three point filters for removing inner points therefore removing self-intersections. The first filter, named **small segment filter**, determines if a pair of sampling points comes from two overlapping surfaces and should be filtered. The second filter, named **ray casting filter**, determines if a sampling point is an inner point by judging its two neighboring I_{Norm} values. The third filter, named **offset property filter**, determines if a set of sampling points that form a shell are inner points by judging if any points are generated from invalid edges. The three filters are discussed in more details in Sections 5.2 ~ 5.4 respectively.

5.2. Small Segment Filter

Suppose for a pixel (*i*, *j*) in a LDNI, a set of sampling points $P_1 \sim P_n$ has been calculated and further sorted based on their distances to d_{min}. By going through $P_1 \sim P_n$, we can identify a pair of points P_i and P_j

that satisfy $|d_{Pi} - d_{Pj}| < \varepsilon$ and $N_{Pi} \cdot N_{Pj} < 0$, where ε is a small segment tolerance (e.g. 10^{-5} in our implementation) and $N_P(n_x, n_y, n_z)$ is the surface normal at P. If a pair of such points is identified, both points are removed from the LNDI. We first apply the small segment filter to delete the overlapping surfaces in $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$ that are generated by the same geometric element in $\partial(S)$ (refer to an example in Figure 9.c). The small segment filter will then be used to remove all the gaps or thin-shells whose thickness is less than ε in the offset model. Such small gaps and thin-shells are usually non-manufacturable in most CAD/CAM applications.

5.3. Ray Casting Filter

The ray casting filter is designed based on the normal index number I_{Norm}.

Proposition 5.1. *Suppose a point P_i along a ray (d_{min}, d_{max}) has two neighboring normal index numbers I_{Norm1} and I_{Norm2}. The difference $|I_{Norm1} - I_{Norm2}| = 1$.*

Proof. If a sampling point P_i is generated along a ray, we know $N_{pi} \cdot N_{ray} \neq 0$, where N_{Pi} is the face normal at P_i and $\boldsymbol{N_{ray}}$ is the unit normal of the ray. Therefore, $I_{Norm2} = I_{Norm1} + 1$ if $N_{pi} \cdot N_{ray} < 0$ and $I_{Norm2} = I_{Norm1} - 1$ if $N_{pi} \cdot N_{ray} > 0$. □

Proposition 5.2. *Suppose a LDNI is computed from a polygonal model that is two-manifold and regulated. I_{Norm} of any point P on a ray (d_{min}, d_{max}) is 0 or 1. Further P is inside the model if $I_{Norm}(P) = 1$; otherwise, it is outside the model.*

The boundary of *S grown by r* must lie in the outside of *S*; while the boundary of *S shrunk by r* must lie in the inside of *S*. Therefore, if an input polygonal model is two-manifold and regulated, points of $\partial(S \uparrow^* r)$ and $\partial(S \downarrow^* r)$ must lie in the regions that have $I_{Norm} = 0$ and $I_{Norm} = 1$ respectively (refer to an example as shown in Figure 10). This is also shown in the properties of winding number (Guibas et al. 1983).

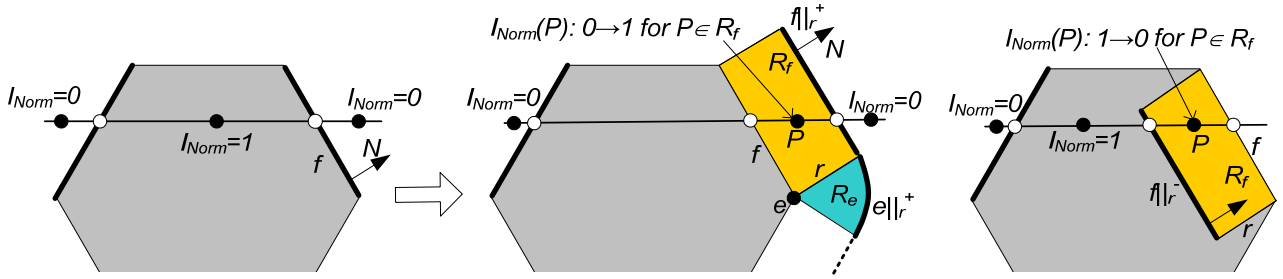


Figure 10. An illustration of offsetting a face and an edge.

Table 1. I_{Norm} Effect of different geometric elements on offset meshes.

Geometric Element of ∂S	**Change of I_{Norm}**	
	S grown by r	*S shrunk by r*
Face	+1	-1
Edge (Valid)	+1	-1
Edge (Invalid)	-1	+1
Vertex (Valid)	+1	-1
Vertex (Invalid)	-1	+1

Example of $f||_r^+$, $e||_r^+$ and $f||_r^-$ are shown in Figure 10.b and c. Regions R_f and R_e that are defined by (f, $f||_r^+$) and (e, $e||_r^+$) respectively are also shown in the figure. Obviously for a point P inside R_f or R_e, $I_{Norm}(P)$ will be changed by 1 due to the addition of the related offset meshes.

Proposition 5.3. *The offset meshes ($f||_r^+$, $e||_r^+$ or $v||_r^+$) of a geometric element (f, e or v) in $\partial(S)$ will change I_{Norm} by 1 for all the points in the region R defined by the offset meshes and the related geometric element. Further, the effects of such changes on I_{Norm} for different geometric elements are shown in Table 1.*

Since the offset faces $F\,||_r^+ \cup E\,||_r^+ \cup V\,||_r^+$ will change I_{Norm} of the points between $\partial(S)$ and $\partial(S \uparrow^* r)$, we can identify the boundary points of $\partial(S \uparrow^* r)$ by judging point's I_{Norm} values. Therefore we can avoid the distance field computation that is usually more computationally expensive. Notice I_{Norm} at a point P can be computed by accumulating the I_{Norm} changes defined by the related offset regions that contain P. For example, offset meshes $f_1||_r^+$, $e||_r^+$ and $f_2||_r^+$ for a given S *grown by* r are shown in Figure 11.a. The related offset regions R_{f1}, R_e, and R_{f2} are also shown Figure 11.b. Their effects on I_{Norm} changes can be identified from Table 1. Hence I_{Norm} of any given points (e.g. $P_1 \sim P_6$) in the offset result can be computed based on the original I_{Norm} values and related I_{Norm} changes. The computed results are shown in Figure 11.c. Accordingly some offset meshes can be identified as invalid (marked as '×') even though no distance computation is performed between the offset and original meshes.

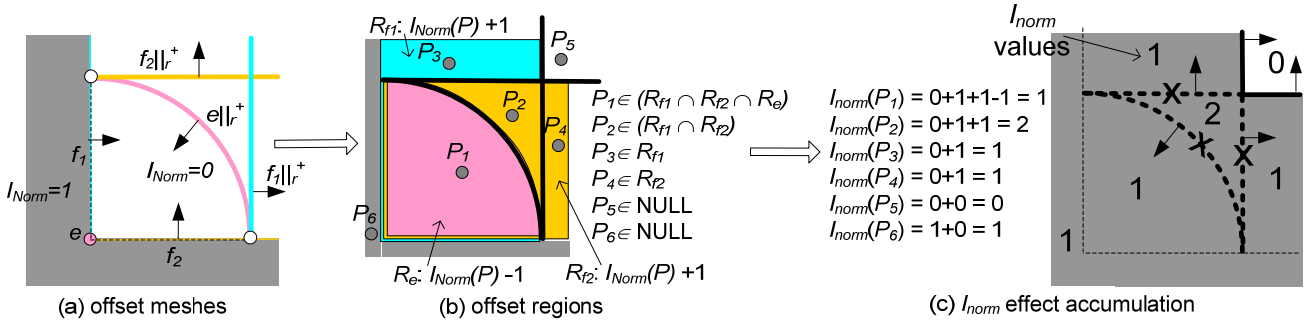


Figure 11. A classification of I_{Norm} changes with an illustration example.

Theorem 5.1. *Suppose a point P_i along a ray (d_{min}, d_{max}) has two neighboring normal index numbers I_{Norm1} and I_{Norm2}. P_i is a boundary point only if I_{Norm1} = 0 and I_{Norm2} = 1, or I_{Norm1} = 1 and I_{Norm2} = 0.*

Proof. (1) For S *grown by* r, P_i in LDNIs must be generated by the offset meshes of a face, a valid edge, or a valid vertex in order for it to be valid. In addition, P_i must be outside of S (i.e. the initial value of I_{Norm} = 0). Hence the offset meshes will increase I_{Norm1} of P_i from 0 to 1. Therefore, if no other offset faces affecting P_i, P_i is a boundary point with I_{Norm1} = 1 and I_{Norm2} = 0. However, if I_{Norm1} or I_{Norm2} is bigger than 1, P_i must lie within a region between another offset meshes and its related geometric element in $\partial(S)$. Therefore, the smallest distance from P_i to $\partial(S)$ must be smaller than r. Hence P_i is not a boundary point of $\partial(S \uparrow^* r)$. Notice P_i may also be generated by the offset meshes of an invalid edge or an invalid vertex, or located inside S. Such P_i is not a boundary point; however, we may still get I_{Norm1} = 0 *and* I_{Norm2} = 1 for P_i. We will address such cases in Section 5.4. (2) Similarly, we can prove for S *shrunk by* r. □

Therefore, based on the normal index number I_{Norm} related to offset meshes $F\,||_r^+ \cup E\,||_r^+ \cup V\,||_r^+$, the ray casting filter is designed as follows. For a LDNIs model, we process its x-LDNI, y-LDNI and z-LDNI separately (sequential or in parallel). For each LDNI, we go through each pixel (i, j) to sort points $P_1 \sim P_n$.

We then calculate I_{Norm} for each line segment along the ray. Finally, the calculated I_{Norm} can be used to delete all the inner points whose two neighboring I_{Norm1} or I_{Norm2} are not (0, 1) or (1, 0) from the LDNIs model.

5.4. Offset Property Filter

A sampling point P_i is not a boundary point if its I_{Norm1} and I_{Norm2} are not (0, 1) or (1, 0); however, P_i may still be an inner point even if its I_{Norm1} and I_{Norm2} are (0, 1) and (1, 0). This is due to the interaction of multiple offset surfaces. A 2D example in shown in Figure 6.c. A 3D illustration example is shown in Figure 12. For a simple cube in Figure 12.a, the offset surfaces and related LDNIs points are shown in Figure 12.b and Figure 12.c respectively. The processed sampling points after the first two filters are shown in Figure 12.d. Notice the points at the eight corners are inner points while their I_{Norm1} and I_{Norm2} are (0, 1). We will address such invalid shells by using an offset property filter. The resulted points after applying the offset property filter are shown in Figure 12.e and the related offset contour is shown in Figure 12.f.

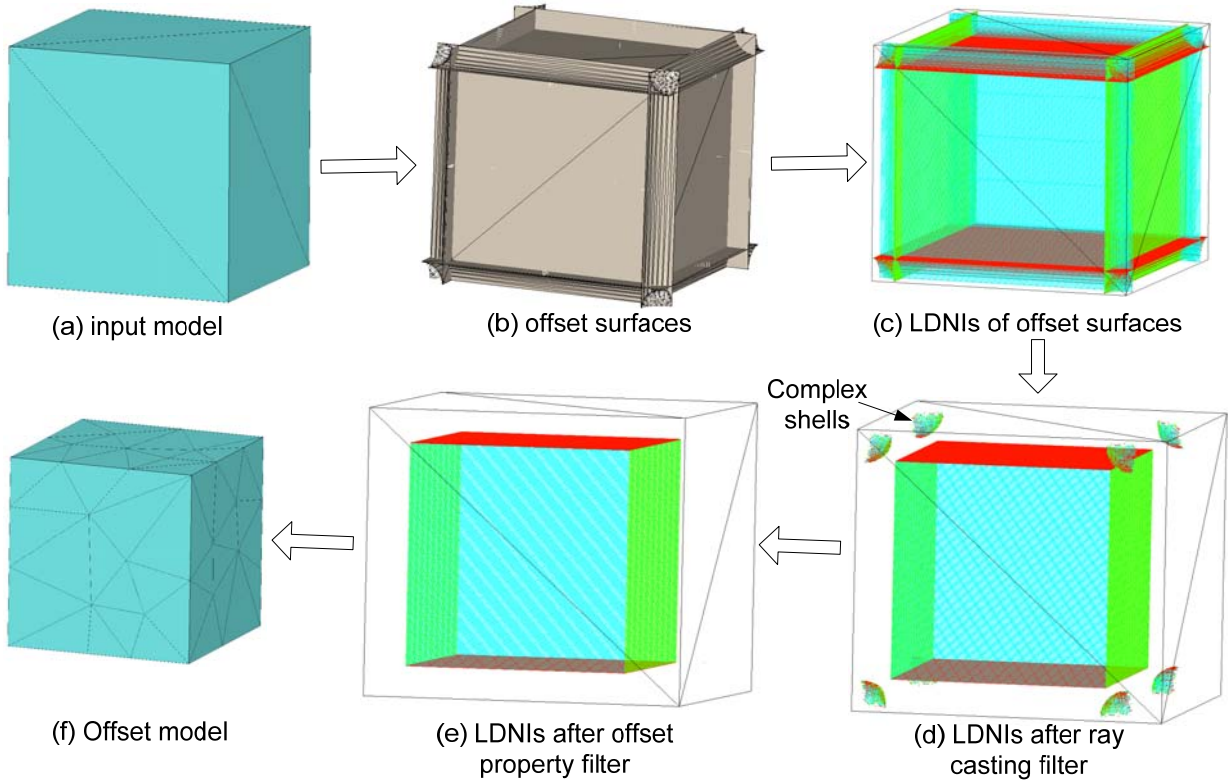


Figure 12. An illustration of removing inner points by filters.

Proposition 5.4. *All the LDNIs points with $I_{Norm1} = 0$ and $I_{Norm2} = 1$, or $I_{Norm1} = 1$ and $I_{Norm2} = 0$ will form one or multiple shells and each shell defines a separate volume (in the side of $I_{Norm} = 1$).*

Proof. After the small segment filter, the LDNIs points related to singular edges and non-manifold vertices will be removed. Therefore, $S\uparrow^* r$ and $S\downarrow^* r$ will be separated into individual volumes that are defined by one or multiple shells. □

Proposition 5.5. *Suppose a shell contains a set of points P_i with $I_{Norm1} = 0$ and $I_{Norm2} = 1$, or $I_{Norm1} = 1$ and $I_{Norm2} = 0$. If any of the point is an inner point, the whole shell is inside the offset distance to ∂S.*

Proof. We can define the offset $S\uparrow^* r$ and $S\downarrow^* r$ by the boundary $\partial(S\uparrow^* r)$ and $\partial(S\downarrow^* r)$ which have one or multiple shells. All the sampling points on the boundary are not inner points. Therefore, if some points on a candidate shell are inner points, the shell must not belong to $\partial(S\uparrow^* r)$ and $\partial(S\downarrow^* r)$. □

As discussed in Section 4, we require the offset meshes $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$ form a continuous boundary. Therefore the offset meshes $E\|_r^+$ and $V\|_r^+$ may flip their surface orientations due to self-intersections. In our method, we classify all the edges and vertices of ∂S based on such offset property. We define **invalid edges** as all the concave edges for *S grown by r* and all the convex edges for *S shrunk by r*, and all the other edges are **valid**. An example of *S grown by r* for an invalid edge is shown in Figure 11. Notice for an invalid edge *e*, the surface orientation of related $e\|_r^+$ is flipped (i.e. surface normals pointing to *e* instead of away from it). Hence we can classify all the related points of $E\|_r^+$ as **invalid or valid edge points**. Similarly as discussed in Section 4, we identify all the complex vertices of ∂S that has both concave and convex neighboring edges and process the related sampling points of $V\|_r^+$. We define **invalid vertex points** as the points whose normal is flipped (i.e. pointing to *v* instead of away from it), and all the other points are **valid**. Obviously for *S* grown by *r*, the effect of such invalid offset surfaces will decrease $I_{Norm}(P)$ by 1; while the effect of all other offset surfaces will increase $I_{Norm}(P)$ by 1 (refer to Table 1).

Therefore, due to the offset surfaces of invalid edges and vertices, there may exist some holes inside the region between ∂S and $\partial(S\uparrow^* r)$ for *S grown by r*, and some islands inside the region between ∂S and $\partial(S\downarrow^* r)$ for *S shrunk by r*. For an example as shown in Figure 12.d, I_{Norm} at the eight cubic corners will be changed by the offset regions of three faces and three invalid edges since $P \in (R_{f_1} \cap R_{f_2} \cap R_{f_3} \cap R_{e_1} \cap R_{e_2} \cap R_{e_3})$. Therefore, from the initial I_{Norm} value of 1 (i.e. inside ∂S for $S\downarrow^* r$), I_{Norm} will be modified based on the effects of related faces and edges (refer to an example as shown in Figure 11.c). Hence the points inside the shell will have I_{Norm} = 1 -1-1-1 +1+1+1 = 1. Consequently they will pass the ray casting filter.

Proposition 5.6. *Suppose a point P_i has I_{Norm1} = 0 and I_{Norm2} = 1, or I_{Norm1} = 1 and I_{Norm2} = 0. If P_i is an inner point, it must be within the offset regions of some invalid edges or vertices.*

Proof. If P_i is an inner point, it must be within the offset regions of some other geometric elements. As shown in Table 1, only an invalid edge or vertex can decrease its I_{Norm} value for *S grown by r* and increase its I_{Norm} value for *S shrunk by r*. Therefore, if none of the geometric elements are invalid edges or vertices, I_{Norm} > 1 for *S grown by r* and I_{Norm} < 0 for *S shrunk by r*. Hence these geometric elements must contain some invalid edges or vertices in order to make I_{Norm} = 0 or 1. □

Proposition 5.7. *Suppose a shell contains a set of points P_i with I_{Norm1} = 0 and I_{Norm2} = 1, or I_{Norm1} = 1 and I_{Norm2} = 0. If the shell is inside the offset distance to ∂S, some of the points P_i must be generated from the offset faces of invalid edges or vertices.*

Proof. If a shell is inside the offset distance to ∂S, there must exist some points P_i on the shell that are inner points. Therefore, P_i must be within the offset regions of some invalid edges or vertices in order for the point to have I_{Norm} = 1. □

Theorem 5.2. *A shell containing a set of points P_i belongs to the offset boundary $\partial(S\uparrow^* r)$ or $\partial(S\downarrow^* r)$ only if none of the points P_i are marked as invalid edge or vertex points.*

Therefore, the offset property filter can be designed as follows. We first mark all the invalid edge and vertex points in a generated LDNIs model. An invalid LDNI point can then be used as a seed for flood

filling operation to remove all the points in the same shell. Notice the ambiguity cases for flood filling are removed by the small segment filter by removing small gaps and shells. We can also implement the offset property filter in the reconstructed mesh level. That is, after extracting mesh surfaces from the LDNIs model, we cluster polygonal faces into shells by their connectivity. If some triangles of a shell are generated from invalid LDNI points, we will remove the shell from the resultant polygonal model. With such a filter, the complex shells as shown in Figure 6.c and Figure 12.d can be effectively removed.

The discussed three filters are sequentially applied to remove inner points in a LDNIs model. Consequently only boundary points will remain in the processed LDNIs model. From such a LDNIs model, a polygonal model can be reconstructed as discussed in the following section.

6. Reconstructing Offset Contour

The generated LDNIs model is an implicit representation of a solid defined by $S\uparrow^* r$ or $S\downarrow^* r$. However, most computer-aided manufacturing systems, such as computer numerical control (CNC) and rapid prototyping machines, require polygonal meshes as input CAD models. We briefly describe our approach of converting a LDNIs model into a polygonal model.

6.1. Adaptive Sampling for Cell Representation

As discussed in Section 5.1, the accuracy of a generated LDNIs model depends on the pixel width δ used in the construction process. Hence a small δ value will lead to a better resolution and, at the same time, a large number of sampling points. If we directly construct polygonal meshes from such a LDNIs model, the constructed polygonal model will have a larger number of triangles. Most of the triangles will be much smaller than the feature sizes of $\partial(S\uparrow^* r)$ and $\partial(S\downarrow^* r)$. Therefore, it is generally not efficient and practical to directly construct contours from a highly accurate LDNIs model.

In our method, we construct another type of implicit representation, an adaptive cell representation, from a LDNIs representation. The adaptive cell representation contains two types of cells, uniform cells and octree cells (Chen 2007). The uniform cells are used for rough sampling; for a uniform cell which has complex geometry such as small features, we then use octree cells to refine it. An adaptive sampling approach to construct a cell representation from a LDNIs representation including the approaches of handling volume titling is presented in (Chen and Wang 2008). During the adaptive sampling test, we calculate an error-minimizing point of a cell from all the sampling points within the cell and explicitly compare the approximation error with a given tolerance ε. If the approximation error is smaller than ε, we will use the calculated error-minimizing point in the contouring process; otherwise, we subdivide the cell until it reaches the finest level. In essence, we intelligently down-sample a LDNIs model into an adaptive cell model hence denser samples will be used only in a region with more complex geometries.

6.2. Manifold-Preserved Mesh Reconstruction

After a cell representation is constructed, we use a modified dual contouring method for reconstructing polygons (Wang and Chen 2008). Unlike the marching cube algorithm, the dual contouring algorithm will not generate cracks for an adaptive grid with different grid sizes. Further, two strategies to generate manifold-preserved mesh surfaces are presented for overcoming the topology ambiguity that may occur

inside the finest octree cells after the maximum subdivision. The constructed polygonal model is manifold with no gaps or overlapping surfaces.

7. Implementation Discussion

In the section, we discuss some implementation techniques related to the efficiency, accuracy and robustness of our method.

7.1. Decimation of Input Mesh

Mesh decimation has been extensively studied for computer graphics applications (Garland 1999). It produces a lower number of polygons to approximate an input solid. The approximation accuracy can be controlled by a decimation tolerance α. A mesh decimation algorithm will try to delete as many triangles as possible while ensuring the maximum approximation error is smaller than α. Therefore, we can first process an input polygonal model by using such a mesh decimation algorithm. The decimation tolerance α can be set as a fraction of θ where θ is the required offsetting accuracy. In most cases, the decimation of an input polygonal model can significantly reduce the number of geometric elements in ∂S and consequently improve the speed of generating offset surfaces $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$.

7.2. Approximation Errors of Cylinders and Spheres

For an input polygonal model, offset faces $F\|_r^+$ are exact. However, offset faces $E\|_r^+$ and $V\|_r^+$ are an approximation of the cylinders and spheres related to edges and vertices respectively. Suppose r is the offset distance. We know: (i) $E\|_r^+$: the maximum approximation error of a cylinder is $\lambda = (r - \sqrt{r^2 - 0.25 \bullet \xi^2})$ where ξ is the maximum edge length used in the approximation of an arc; (ii) $V\|_r^+$: the maximum approximation error of a sphere is $\lambda = (r - \sqrt{r^2 - 0.5 \bullet \xi^2})$ where ξ is the maximum edge length of a triangle for the approximation of a spherical patch. Therefore we can set ξ based on a required offsetting accuracy θ and offset distance r such that $\lambda < \theta$. By using a smaller ξ, we will get a better approximation of the cylinders and spheres; however more triangles will be generated in $E\|_r^+$ and $V\|_r^+$.

7.3. Constructing Offset Meshes for Different Offset Distances

Without considering the approximation errors of cylinders and spheres, the offset meshes with different offset distances $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$ actually have the same topological connections regardless of the size of offset distance. Therefore, if we calculate the offset meshes for an offset distance r_1 and save them as a polygonal model, we can directly scale all its polygonal vertices by a scale factor r_2/r_1 for another offset distance r_2. The scaled model can be used directly as the offset meshes for r_2. The approach can speed up the step of computing offset meshes especially for interactively displaying the results. In order to know the scale center, we store an additional tag for each polygonal vertex in the offset meshes. The tag value is an index number of an corresponding vertex in the input model ∂S. Notice if $r_2 >> r_1$, the approximation errors of a cylinder and a sphere may be large. Also the valid/invalid edges are different if r_1 and r_2 have a different sign. In those cases, we may recalculate the offset meshes.

7.4. Volume Titling for High Accuracy

In generating a LDNIs model for offset meshes $F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$, we can set a pixel width δ according to a required offset accuracy θ. However, if θ is rather small or the input model has a large

dimensional size, the required LDNI resolution may exceed the image resolution of graphics hardware (e.g. a typical resolution of 1280x1024). In these cases, we split the bounding box of the model into multiple smaller tiles. We display each tile and construct a LDNIs model independently. Each tile also needs a buffer region around its boundary to ensure the continuity of the reconstructed surfaces (Chen and Wang 2008). Some examples as shown in Section 8.1 require multiple tiles (e.g. a liberty and a tutor model).

7.5. Speedup via Parallelization

A significant advantage of our method is the simplicity of parallelizing it. The construction of a LDNIs model from a polygonal model can be aided by a graphics hardware such as *NVIDIA* GeForce 8800 GT that we used in our tests. Based on highly parallel structure, the graphics hardware can construct a LDNIs model rather quickly (usually within seconds). All the generated LDNIs models can be saved in a network-connected hard disk, which can be accessed from a cluster of PCs (Chen and Wang 2008). The LDNIs model of each tile can then be processed separately without other tiles' information. Therefore we can use a PC cluster to parallelize the tasks of computing boundary points from a LDNIs model and reconstructing offset boundary from the computed boundary points.

7.6. Verification of Polygonal Models

In our method, we require an input polygonal model as two-manifold. Therefore, the model should have no gaps. Otherwise, our method will fail. In addition, we further require each input polygon have a valid normal (that is, $\boldsymbol{N_f} \neq 0$). If a triangle has no area and consequently its normal is zero, we will also have difficulty in generating its offset surfaces to form a continuous boundary. Therefore, before using our approach, we need to verify an input model by sealing gaps and cleaning non-regular triangles (i.e. triangle area is 0).

In addition, the calculation of I_{Norm} for the sampling points of a LDNI model requires the offset meshes $F\,\|_r^+ \cup E\,\|_r^+ \cup V\,\|_r^+$ to form a continuous boundary. Due to the floating-point arithmetic in geometric computations, the corresponding vertices of $F\,\|_r^+$, $E\,\|_r^+$, and $V\,\|_r^+$ may have small numerical errors. Hence the constructed offset meshes need to be verified by sealing the boundary before computing LDNIs.

8. Experimental Results and Applications

We used C++ programming language with *Microsoft* Visual C++ compiler to implement the presented algorithm. In this section, we present our test results by highlighting the accuracy and the performance of our algorithm. We also present some applications based on the developed offsetting method.

8.1. Experimental Results

Accuracy. We used four simple models (a cube, a pyramid, a sphere, and a cylinder) to test the accuracy of our method. The models are selected because their offset results are known based on theoretical analysis. We fit the size of all the input models into a unit cube ($1\times1\times1$) and test the offset distance -0.1 (shrinking the model). Both the input models and the ideal offset models are constructed in a CAD software system. To compare the construct offset models by our method, a publicly available Metro tool (Cignoni, Rocchini et al. 1998) was used. The shape approximation errors are measured with reference to the unit length of the input model. The offset accuracy based on orthogonal distance between two comparing models is given in

Table 2. The maximum edge length ξ used in constructing offset faces $E\|_r^+$ and $V\|_r^+$ is 0.025. The related maximum approximation error λ is 0.00078 and 0.0016 for cylinders and spheres respectively. The pixel width δ used in constructing LDNIs is 0.005 (i.e. the image resolution required for the unit cube is 200x200). The tolerance ε used in the adaptive sampling is 0.001. Since the offsetting accuracy is quite satisfactory, the same setting is used in all other tests.

Table 2. Offset Accuracy Test Results.

Models	Offset Dist	Size	Polygonal Mesh Error		
			Maximum Error Distance (E_{max})	Average Error Distance (E_{mean})	Root Mean Square (RMS)
Cube	0.1	1.0 x 1.0 x 1.0	0.000005	0.000001	0.000002
Pyramid	0.1	1.0 x 1.0 x 1.0	0.001643	0.000004	0.000018
Sphere	0.1	1.0 x 1.0 x 1.0	0.001311	0.000224	0.000290
Cylinder	0.1	1.0 x 1.0 x 1.0	0.001263	0.000246	0.000381

Performance. We performed tests on geometries with various complexities and sizes, as well as various offset distances (both grown and shrunk). All the tests are done in a PC with a 2.4 GHz *Intel Core Quad* CPU Q6600 and 4GB *DRAM* running *Windows Vista*. The test results on algorithm performance are given in Table 3. Beside the information of the input models, the main memory requirements of our approach are also given. The running time of the three major steps of our method is presented, that is: (i) generating offset faces (refer to Section 4); (ii) generating and processing a LDNIs model (refer to Section 5); and (iii) reconstructing offset model (refer to Section 6). For input models with a relatively large size (e.g. a liberty and a tutor model), multiple tiles are used. The given running time is for all the tiles. As shown in the results, the step of generating and processing a LDNIs model takes the biggest portion of the running time, which can be sped up by using a parallelization approach.

Comparing the experimental results by our method and a point-based offsetting method (Chen, Wang et al. 2005), the required running time based on our method is much less (1/5~1/15 of running time) for the same test cases such as bunny and dragon. In addition, since the software application of (Lien 2007) is available online (*www.cs.gmu.edu/~jmlien*), we performed comparison tests with it by using the same PC and similar settings. The test results of three cases are shown as follows. Notice the running time of our software only includes the steps of generating offset faces and processing LDNIs since Lien's software can only generate boundary points. However, notice the comparison is somehow unfair since the method presented in (Lien 2007) can be used for general Minkowski sum while our approach is just for uniform offsetting.

Running Time	Octa-Flower (r=0.15)	Bunny (r= 0.06)	Tutor (r= 0.06)
Lien's Software	70.2 Second	200.0 Second	Out of memory
Our Software	7.6 Second	15.2 Second	39.4 Second

In addition to Figure 3 and Figure 4 for a dragon and a hub model, some additional screen captures of the generated offset results are shown in Figure 13 to 18.

Table 3. Algorithm performance of our test results.

Models	Tri # (K)	Size	Off-set Dist.	Memory		Running Time (Second)			
				Offset Tri # (K)	LDNIs Point # (K)	Generate Offset Faces	Generate & Process LDNIs	Construct Offset Model	Total
Accuracy Test Cases									
Cube (Fig. 10)	0.012	1 x 1 x 1	-0.1	0.8	153.6	0.008	0.69	0.30	1.0
Cylinder	0.4	1 x 1 x 1	-0.1	5.0	142.6	0.05	1.11	0.37	1.5
Sphere	10.2	1 x 1 x 1	-0.1	0.8	74.0	0.008	0.61	0.30	0.9
Pyramid	0.006	1 x 1 x 1	-0.1	20.3	120.4	0.26	3.04	0.41	3.7
Freeform Models									
Bunny (Fig. 13)	69.6	X: 0.93 Y: 0.93 Z: 0.72	0.02	105.0	140.2	1.25	6.85	1.50	9.6
			0.04	105.0	158.9	1.25	10.17	1.56	13.0
			0.06	105.0	176.7	1.24	13.96	1.67	16.9
			-0.02	105.0	99.1	1.24	7.58	1.32	10.1
			-0.04	105.0	76.0	1.25	14.12	1.10	16.5
			-0.06	105.0	62.0	1.25	33.72	0.87	35.8
Dragon (Fig. 3)	693.5	X: 1.23 Y: 0.87 Z: 0.55	0.02	286.5	186.4	3.24	31.77	2.82	37.8
			-0.02	286.5	105.0	3.31	32.96	2.07	38.3
Liberty (Fig. 12)	38.0	X: 2.09 Y: 2.09 Z: 6.93	0.02	1,198.6	824.8	12.84	350.3 (1x1x3)	19.78 (1x1x3)	82.9
			-0.02	1,198.6	1,700	12.67	434.33 (1x1x3)	20.29 (1x1x3)	467.3
Octa Flower (Fig. 16)	15.8	X: 1.38 Y: 1.37 Z: 0.98	0.02	52.5	217.2	0.61	2.07	1.68	4.4
			0.06	52.5	255.9	0.62	5.13	1.87	7.6
			0.15	52.5	342.5	0.62	7.04	2.17	9.8
Statue (Fig. 17)	5.0	X: 2.50 Y: 1.91 Z: 1.39	0.02	94.2	701.7	1.17	8.10	4.20	13.5
			-0.02	94.2	632.4	1.02	8.94	3.92	13.9
Statue +0.02 (Fig. 17)	14.2	X: 2.54 Y: 1.95 Z: 1.43	-0.02	172.7	664.3	1.88	16.42	4.05	22.4
Statue -0.02 (Fig. 17)	12.9	X: 2.46 Y: 1.87 Z: 1.35	0.02	156.6	666.3	1.72	23.72	3.99	29.4
Engineering Models									
Case 1 (Fig. 14)	0.044	X: 1.02 Y: 1.02 Z: 2.03	0.05	2.5	289.7	0.016	1.40	0.67	2.1
			-0.05	2.5	434.3	0.015	1.14	1.20	2.4
Case 2	0.38	X: 2.03 Y: 1.02 Z: 2.03	0.05	4.0	743.4	0.039	1.30	2.08	3.4
			-0.05	4.0	554.4	0.037	1.35	1.30	2.7
Tutor (Fig. 15)	1.4	X: 4.59 Y: 9.56 Z: 4.01	0.06	38.2	1,346	0.37	38.99 (2x4x2)	77.9 (2x4x2)	117.3
			-0.06	38.2	1,336	0.36	30.96 (2x4x2)	67.1 (2x4x2)	98.4
Hub (Fig. 4)	17.9	X: 2. 94 Y: 2.94 Z: 0.49	0.05	117.5	386.6	1.17	27.19 (2x2x1)	5.82 (2x2x1)	34.2
			-0.05	117.5	129.1	1.17	19.49 (2x2x1)	2.03 (2x2x1)	22.7

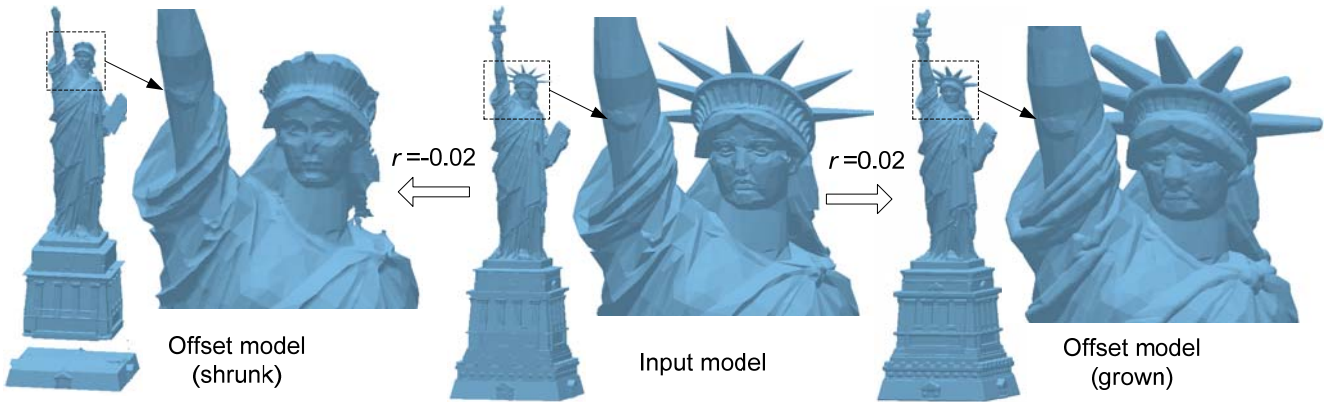


Figure 13. Screen capture of the offset results for a liberty model.

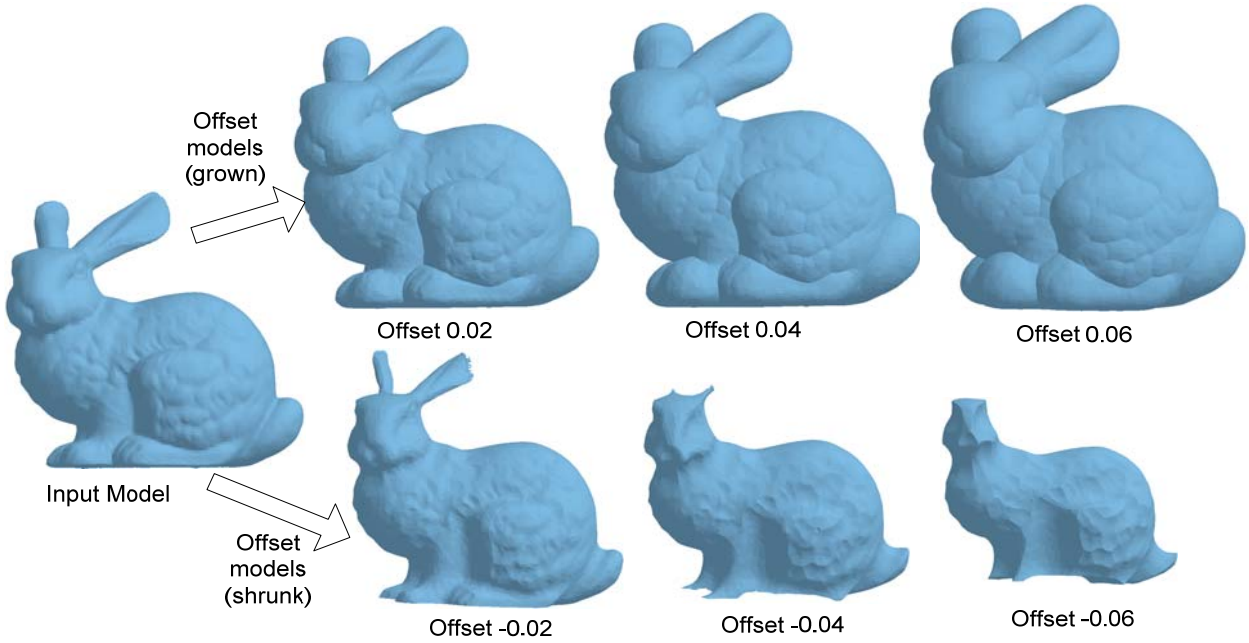


Figure 14. Screen capture of the offset results for a bunny model.

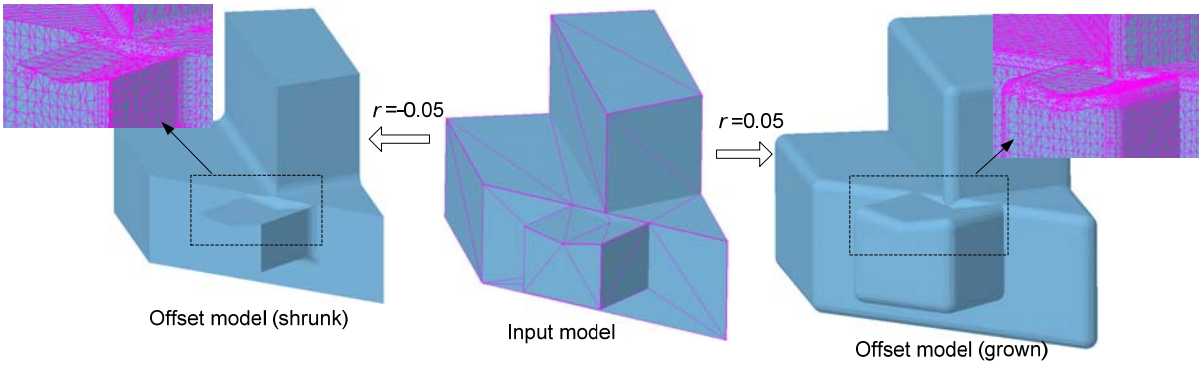


Figure 15. Screen capture of the offset results for a case1 model.

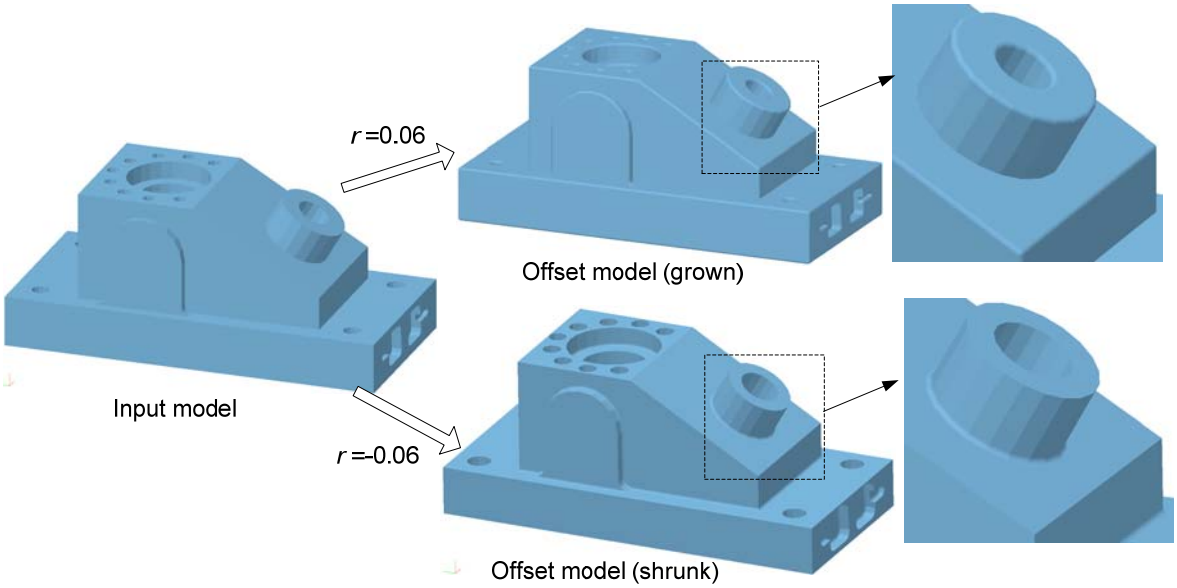


Figure 16. Screen capture of the offset results for a tutor model.

8.2. Applications

The presented uniform offsetting operation has a wide range of applications. We present some of these applications with our test results.

Single offsetting operation. The uniform offsetting can be used to create a thin shell of a solid model by simply merging an input model ∂S and an offset model $\partial(S\downarrow^{*} r)$ with flipped normals. In addition, we can add a wide variety of cellular structures inside the hollowed portion of the model for better physical properties (Chen 2007). The CAD model of the external thin-shells with complex internal structures can be manufactured by solid freeform fabrication (SFF) processes.

Multiple offsetting operations (same type). We use the uniform offsetting operation in the tool path planning for CNC machining. Due to the accuracy limitation of casting and forging processes, we need to enlarge a CAD model such that sufficient extra materials can be ensured for CNC machining. The uniform offsetting is ideal for such purpose since it can ensure the uniform cutting depth during the machining. An example based on an octa flower model is shown in Figure 17. The offsetting results related to different offset distance from 0.02 to 0.15 are shown in the figure. We put two pair of the models together to show the uniform cutting depth that can be achieved.

Multiple offsetting operations (different types): Rounds and fillets are transitional faces that are common in most machined, cast and molded parts. They are important mechanical design features that serve to relieve stress concentration, to simplify fabrication, and to improve appearance. We can use the uniform offsetting to automatically add fillets and rounds in a CAD model (Chen, Wang et al. 2005). That is, *S filleted by r* can be defined as $F_r(S) = S\uparrow_r\downarrow_r$. And *S rounded by r* can be defined as $R_r(S) = S\downarrow_r\uparrow_r$. An example of added fillets and rounds in a Beethoven statue model are shown in Figure 18. The algorithm performance of the offsetting operations is also given in Table 3. Compared to the experimental results presented in (Chen, Wang et al. 2005), the method based on LDNIs is much faster.

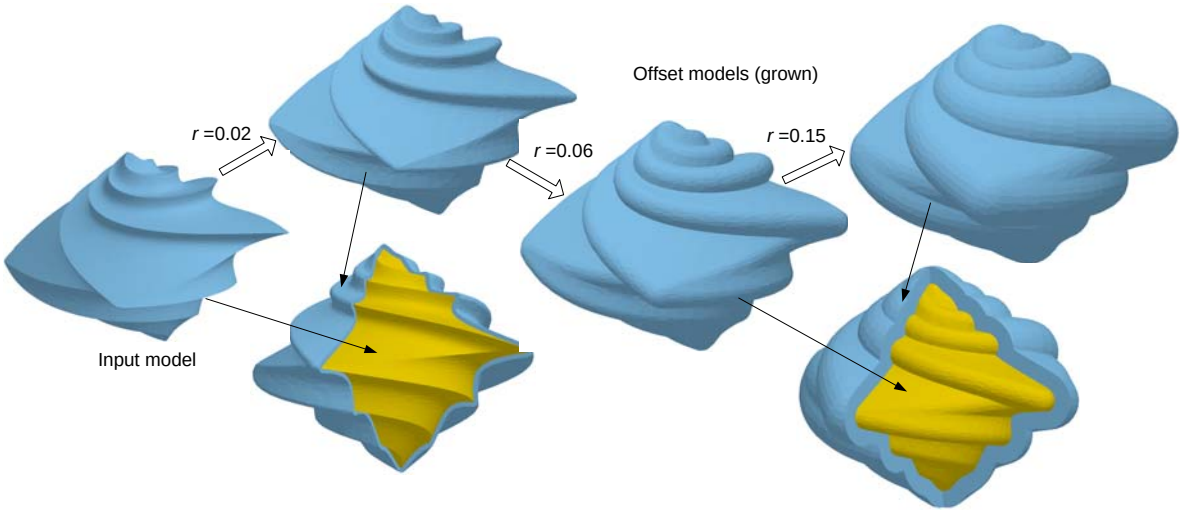


Figure 17. An example of multiple offsetting operations (same type).

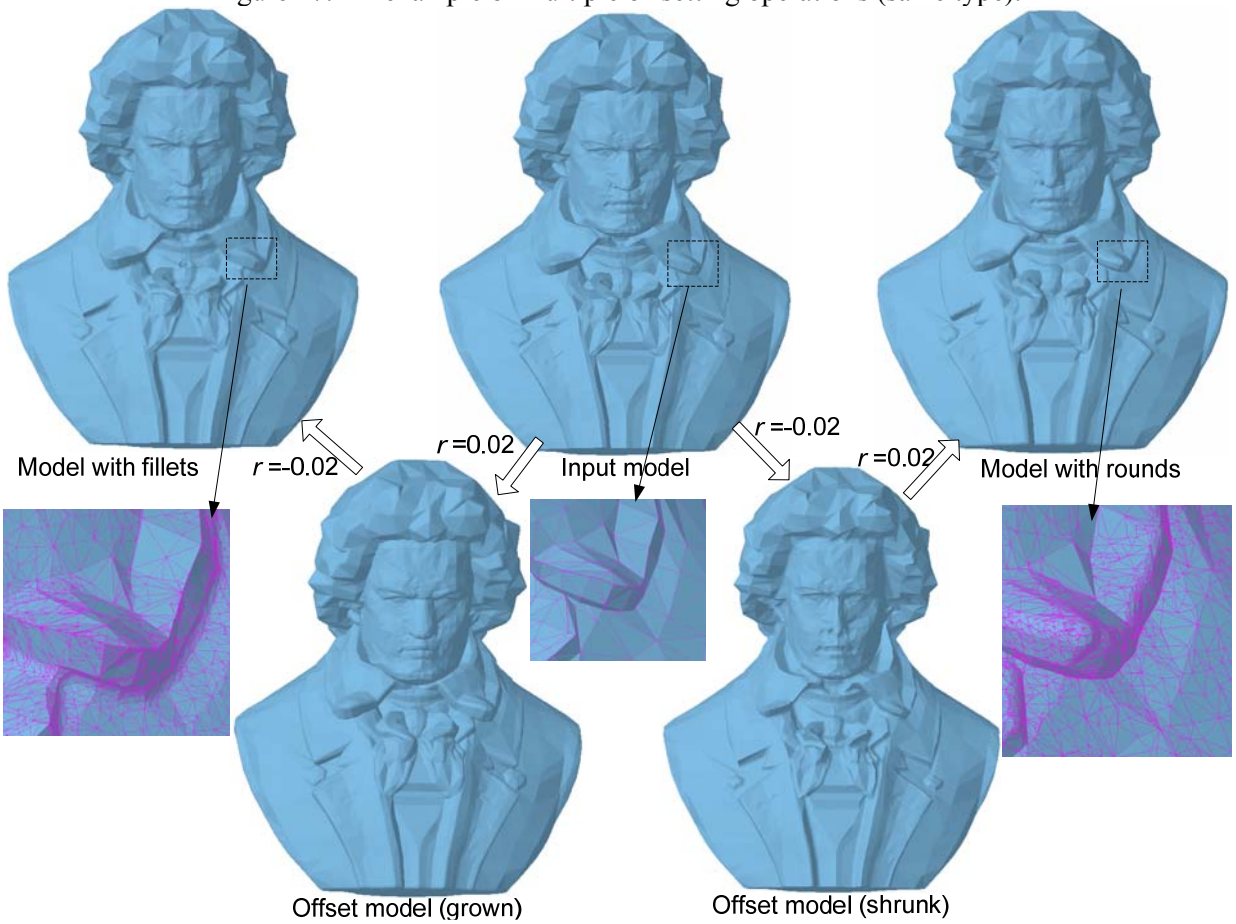


Figure 18. An example of multiple offsetting operations (different types).

9. Conclusion and Future Work

Uniform offsetting is a fundamental and significant geometric modeling operation. However, due to the dramatic topological changes in the offset solid, computing offset boundary is a rather challenging problem. We believe a promising approach for the offsetting operation is to compute an approximated boundary based

on point representations. In this paper, we presented a novel LDNIs-based uniform offsetting method for any input polygonal model and an arbitrary offset distance. In our approach, each face, edge, and vertex of an input solid model generates a set of offset faces which then form a continuous boundary. We construct a LDNIs model from the offset faces, which contains a set of well-structured sampling points. Accordinlgy three point filters have been developed to delete all the inner points. Finally the offset model can be reconstructed from the processed LDNIs model based on adaptive sampling and manifold-preserved contouring.

Our offsetting approach has several advantages which have not been provided by existing methods. Our approach is general that can handle both grown and shrunk operations for an arbitrary offset distance on freeform objects with complex geometry. The algorithms of our method are simple and can be easily implemented. The experimental results on a variety of CAD models have verified the effectiveness and efficiency of our algorithms.

Some future work we are investigating includes: (1) we are investigating approaches for improving our algorithm on handling solid models with degenerated data; (2) we are exploring the usage of our offsetting method in new applications; (3) we plan to extend our method to other solid operations such as general Minkowski operations.

References

Allen, S. and D. Dutta (1998). "Wall Thickness Control in Layered Manufacturing for Surfaces with Closed Slices." Computational Geometry: Theory and Application, Vol. 10, pp. 223-238.

Breen, D. E. and S. Mauch (1999). Generating Shaded Offset Surfaces with Distance, Closest-Point and Color Volumes. Proceedings of the International Workshop on Volume Graphics.

Basch, J., L. J. Guibas, G. D. Ramkumar, and L. Ramashaw (1996). Polyhedral Tracings and their Convolution. Algorithms for Robotic Motion and Manipulation, A. K. Peters, Ltd, Wellesley, MA.

Chen, Y. (2007). "3D Texture Mapping: A Microstructure Design Method for Rapid Manufacturing." Computer-aided Design and Application, Vol. 4, No. 6, pp. 761-771.

Chen, Y. (2007). "An Accurate Sampling-based Method for Approximating Geometry." Computer-Aided Design, Vol. 39, No. 11, pp. 975-986.

Chen, Y. and C. C. L. Wang (2008). Layer Depth-Normal Images for Complex Geometries - Part One: Accurate Modeling and Adaptive Sampling. Proceedings of ASME International Design Engineering Technical Conferences, Brooklyn, New York.

Chen, Y., H. V. Wang, D. W. Rosen, J. R. Rossignac (2005). Filleting and Rounding Using a Point-based Method. Proceedings of ASME International Design Engineering Technical Conferences, Long Beach, CA.

Cignoni, P., C. Rocchini, R. Scopigno (1998). "Metro: measuring error on simplified surfaces." Computer Graphics Forum, Vol. 17, No. 2, pp 167-174.

Farouki, R. T. (1985). "Exact Offset Procedures for Simple Solids." Computer-Aided Design, Vol. 2, No. 4, pp. 257-279.

Frisken, S. F., R. N. Perry, A. P. Rockwood, T. R. Jones (2000). Adaptively Sample Distance Fields: A General Representation of Shape for Computer Graphics. Proc. of ACM SIGGRAPH, New Orleans, LA.

Frosyth, M. (1995). Shelling and Offsetting Bodies. Proceedings of Third Symposium on Solid Modeling and Applications, Salt Lake City, Utah.

Garland, M. (1999). Quadric-Based Polygonal Surface Simplification. PhD Dissertation, Computer Science. Carnegie-Mellon University, Pittsburgh, PA.

Gibson, S. F. (1999). Calculating the Distance Map for Binary Sampled Data. TR99-26, Mitsubishi Electric Research Laboratory, Cambridge, MA.

Guibas, L. J., L. Ramshaw, and J. Stolfi (1983). A Kinetic Framework for Computational Geometry. In Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci., pp. 100-111.

Hartquist, E. E., J. P. Menon, K. Suresh, H. B. Voelcker (1999). "A Computing Strategy for Applications Invovling Offsets, Sweeps, and Minkowski Operations." Computer-Aided Design, Vol. 31, pp. 175-183.

Kim, S.-J., D.-Y. Lee, and M. Y. Yang (2004). "Offset Triangular Mesh Using the Multiple Normal Vectors of a Vertex." Computer-aided Design and Applications, Vol. 1, No. 1-4, pp. 285-291.

Kim, Y. J., G. Varadhan, M. C. Lin, and D. Manocha (2003). Fast Sweep Volume Approximation of Complex Polyhedral Models. ACM Symposium on Solid Modeling and Applications.

Lam, T. W., K. M. Yu, K. M. Cheung, and C. L. Li (1997). "Octree Reinforced Thin-Shell Rapid Prototyping." Journal of Materials Processing Technology, Vol. 63, pp. 784-787.

Lee, S. H. (1999). Offsetting Operations on Non-manifold Boundary Representation Models with Simple Geometry. ACM Symposium on Solid and Physical Modeling. Ann Arbor, MI, pp. 42-53.

Lien, J.-M. (2007). Point-based Minkowski Sum Boundary. Pacific Conference on Computer Graphics and Applications, Maui, Hawaii.

Maekawa, T. (1999). "An Overview of Offset Curves and Surfaces." Computer-Aided Design, Vol. 31, No. 3, pp. 165-173.

McMains, S., J. Smith, J. Wang, C. Sequin (2000). Layered Manufacturing of Thin-Walled Parts. Proceedings of ASME Design Engineering Technical Conference, Baltimore, Maryland.

Nadler, S. B. J. (1978). Hyperspaces of Sets. New York, Marcel Dekker.

Pavic, D. and L. Kobbelt (2008). "High-Resolution Volumetric Computation of Offset Surfaces with Feature Preservation." EUROGRAPHICS, Vol. 27, No. 2.

Pham, B. (1992). "Offset Curves and Surfaces: A Brief Survey." Computer-Aided Design, Vol. 24, No. 4, pp. 223-229.

Qu, X. and B. Stucker (2003). "A 3D Surface Offset Method for STL-format Models." Rapid Prototyping Journal, Vol. 9, No. 3, pp. 133-141.

Rossignac, J. R. and Aristides A. Requicha (1986). "Offsetting Operations in Solid Modelling." Computer Aided Geometric Design, Vol. 3, pp. 129-148.

Satoh, T. and H. Chiyokura (1991). Boolean Operations on Sets Using Surface Data. ACM SIGGRAPH: Symposium on Solid Modeling Foundations and CAD/CAM Applications, Austin, TX.

Varadhan, G. and D. Manocha (2006). "Accurate Minkowski Sum Approximation of Polyhedral Models." Graph. Models, Vol. 68, No. 4, pp. 343-355.

Wang, C. C. L. and Y. Chen (2008). Layer Depth-Normal Images for Complex Geometries - Part Two: Manifold-Preserved Adaptive Contouring. Proceedings of ASME International Design Engineering Technical Conferences, Brooklyn, New York.