Uniform offsetting of polygonal model based on Layered Depth-Normal Images

Yong Chen\textsuperscript{a,}\textsuperscript{*}, Charlie C.L. Wang\textsuperscript{b}

\textsuperscript{a} Epstein Department of Industrial and Systems Engineering, University of Southern California, 90089 Los Angeles, CA, United States
\textsuperscript{b} Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Hong Kong

**A B S T R A C T**

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 a 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 Image (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 two-manifold polygonal contour as the resulting 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.

© 2010 Elsevier Ltd. All rights reserved.

**A R T I C L E I N F O**

Article history:
Received 17 December 2008
Accepted 17 September 2010

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 $E^2$ or $E^3$ [1]. As shown in Fig. 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 $S \downarrow_r = S \ominus b_r$,

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

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 three-dimensional (3D) models, tool path generation for 3D NC machining, rapid prototyping and coordinate measuring machines, tolerance analysis for assemblies, and robot path planning [1–4]. 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 unstable. Many degenerate cases between vertices, edges and surfaces need to be carefully considered in the implementation. To avoid the difficulties, previous work based on volumetric approach [4–7] and sampling point approach [6] 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
Fig. 1. Offsetting a solid $S$ by a distance $r$.

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 3D solid model is shown in Fig. 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 Fig. 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 Image (LDNI) [8,9], to sample the offset meshes (refer to Fig. 2(b)). The accuracy of the generated LDNI models can be controlled during the construction process. We then process all the sampling points in the offset LDNI model using a set of point filters. Accordingly all the invalid points are identified and discarded from the offset LDNI. The remaining points after the filtering process are shown in Fig. 2(c). Finally, we reconstruct an offset contour from the processed LDNI model by using adaptively sampling [8] and manifold-preserving contouring methods [10]. The computed uniform offsetting model is shown in Fig. 2(d) with a magnified view ($A \sim D$) for each major step of the method.

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 LDNI model. Based on a mesh tiling technique, the LDNI 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 LDNI model. For each sampling point, we judge its inside/outside property directly based on a ray casting test instead of the minimum distance computation from the original model. Therefore the performance of our method is less related to the offset distance; while other offset approaches [4,7] will dramatically slow down for a bigger offset distance. In addition, several key steps in processing a LDNI model can be parallelized.
- **Simple**: Our method is mainly based on LDNI. 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 Figs. 3 and 4 respectively. In Fig. 3, both the grown and shrunk models of a complex dragon model (publicly available from Stanford 3D Scanning Repository) are given. Note the topological changes in the offset models. In Fig. 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.

2. Related work

Earlier approaches [1,11–13] 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 numerical 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 [14] 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 two-dimensional (2D) contours. For example, Lam et al. [31] described an approach based on slicing geometries into 2D contours and offsetting each slice contour based on pixels. McMains et al. [15] presented an algorithm for building thin-walled parts in a fused deposition modeling (FDM) machine. Geometries are sliced first followed by creating 2D offset contours. Allen and Dutta [16] 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 [17]. A ray-rep is a set of line segments that lie inside the solid and is 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 LDNI-based method uses both depth values and normals of sampling points that are generated in three orthogonal directions. Also the reconstruction of the offset model was not considered in [17].

An offset method based on distance volume and fast marching method was presented for CSG models [18]. 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 the 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 [19,28]. Based on the idea of computing unsigned distance fields, Kim et al. [30] 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 isosurfaces.

More recently, Chen et al. [4] 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 [5] 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 [6] 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 [7] 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 [20] defines the regularized offset of a regular set S by a positive distance r as $S^+ r = \{ p : d(p, S) \leq r \}$, where $d(p, S) = \inf_{p \in S} \| p - q \|$ and $\inf$ denotes the greatest lower bound. From this definition, $\partial(S^+ r) \subset \{ p : d(p, S) = r \}$. For $p \notin S$, $d(p, S) = (p, \partial S)$. 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 $\partial(S^- r) \subset \{ p : d(p, c S) = r \}$, where c denotes regularized complement. Since $S^+ r$ can be directly derived from $S^- r$, we will focus on $S^+ r$ from now on and briefly mention $S^- r$.

The principle of our method is to first generate a superset $\{ p : d(p, \partial S) = r \}$ and then calculate the offset surfaces $\partial(S^+ r)$ 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 \( F \cup E \cup V \) \( r \) away from the boundaries of \( S \) and \( F \), respectively.

1. Faces \( F(S) \): Suppose \( f \) is a face of \( S \) and \( q \in f \). We can construct the set \( F \cup E \cup V \) by displacing each point \( q \) a distance \( r \) along the unit normal of \( f \), i.e., let \( p = q + r \).

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 \( E \) \( r \) away from the edges of \( S \) 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 \( e_1, e_2, \ldots, e_p \). We can construct the set \( V \cup E \cup V \) \( r \) by constructing a sphere centered at \( v \) with radius \( r \). The sphere can be bounded by \( e_1 \) \( r \), \( e_2 \) \( r \), \ldots, \( e_p \) \( r \) because the points outside the bounded portion are closer to \( e_1 \), \( e_2 \), \ldots, \( e_p \).

Therefore, \( \partial(S \cup r) \subset F \cup E \cup V \cup \partial(S \cup r) \). That is, any point \( p \in \partial(S \cup r) \) must be attained from \( F \cup E \cup V \cup \partial(S \cup r) \). Note that all points of \( F \cup E \cup V \cup \partial(S \cup r) \) are at a distance \( r \) from some points of \( \partial S \) (suppose \( p \)). However, some of them may be at a smaller distance to other points of \( \partial S \) (suppose \( q_k \)). When \( p \) is an invalid point, \( q_k \) \( r \) and \( q_k \) \( r \) will intersect each other. Such a self-intersection is a core challenge to be addressed. We define inner points as sampling points on \( F \cup E \cup V \cup \partial(S \cup r) \) whose minimum distance to \( \partial(S \cup r) \) is less than the offset distance \( r \), and boundary points as all the sampling points on \( F \cup E \cup V \cup \partial(S \cup r) \) that are also on \( \partial(S \cup r) \).

The uniform offsetting considered in this paper is a special case of general Minkowski operations. Güibas et al. [21] 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 Fig. 5. Suppose a portion of the boundary for defining a 2D region \( S \) is shown in Fig. 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 \( r \) (shrunken), we can compute \( E \) \( r \) for each edge of \( E(S) \) and \( V \) \( r \) for each vertex of \( V(S) \). Note that \( V \cup E \cup V \) \( r \) form a continuous boundary (refer to a magnified view \( A \) in Fig. 5(a)-right). It also has multiple self-intersections. We use a well-structured point representation (LDNI) to sample the boundary \( V \cup E \cup V \) \( r \). Fig. 5(b) gives a 2D illustration of the constructed LDNI model, where the circles indicate the points recorded on the x-LDNI and the squares illustrates 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 set of point filters have been developed to process all the LDNI points. For example, the ray testing cast values of pixels \( X \) and \( Y \) are shown in Fig. 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 \( \varepsilon \) is a small tolerance such as \( 10^{-4} \). After the filtering process, the remaining LDNI points are shown in Fig. 5(c)-left. Note that 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 Fig. 5(d)).

We further illustrate our method by using a 2D contour sliced from a 3D model as shown in Fig. 6(a). The offset boundary $V \parallel_{-r} \cup E \parallel_{-r}$ of its internal loop 1 is shown in Fig. 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 LDNI model and using point filters to process the sampling points. The filtered LDNI points are shown in Fig. 6(c)-top. The offset boundary of its internal loop 2 and related LDNI points are shown in Fig. 6(c)-middle. Note that if only the ray casting filter as shown in Fig. 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 Fig. 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 Fig. 6(d).

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

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 LDNI resolution may not be captured in the computed offset model. However, compared to other volumetric approaches that are only based on cells or voxels, our approach is much more accurate because: (i) we can accurately compute the offset boundary $V \parallel_{-r} \cup E \parallel_{-r}$; (ii) we accurately compute the ray casting points on $V \parallel_{-r} \cup E \parallel_{-r}$ along a set of rays. Therefore, the boundary points on the rays related to LDNI pixels are accurate (both their positions and normals). Although in practical implementation, a tessellated version is computed instead of an exact $V \parallel_{-r} \cup E \parallel_{-r}$, the accuracy actually can be controlled by choosing different densities in tessellation. Consequently the computed offset contour is a close approximation of the exact offset boundary on features whose sizes are bigger than the LDNI resolution. A volume tiling technique has been developed to ensure a sufficiently small LDNI resolution [8]. 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 LDNI 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, an edge and a vertex into a face, a cylinder and a sphere, respectively has been discussed before [22,29]. In our method, we further ensure that the offset meshes $F^\parallel r^+ \cup E^\parallel r^+ \cup V^\parallel r^+$ will form a continuous boundary with a consistent surface orientation.

**Theorem 4.1.** Let $S$ be a regularized point set and $\partial S$ be its boundary defined as a polygonal model. The offset meshes $F^\parallel r^+ \cup E^\parallel r^+ \cup V^\parallel r^+$ 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 $F^\parallel r^+$ is a set of translated faces by an offset distance in their surface normal directions (refer to an example as shown in Fig. 8(b));

2. For a valid input model $\partial S$, 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 $he_1^\parallel r^+$ and $he_2^\parallel r^+$ in $F^\parallel r^+$ will be on the same cylinder centered at $e$ with radius $r$. Further since $he_1$ and $he_2$ have opposite directions in $\partial S$, we must be able to construct a portion of a cylinder that is bounded by the reverse edge of $he_1^\parallel r^+$ and $he_2^\parallel r^+$ to seal the gap between $f_1^\parallel r^+$ and $f_2^\parallel r^+$. Hence $E^\parallel r^+$ will form a continuous boundary with $F^\parallel r^+$ (refer to an example as shown in Fig. 8(c));

3. For a valid input model $\partial S$, each vertex $v$ has a set of neighboring edges $e_1, e_2, \ldots, e_n$. The constructed cylinders in $E^\parallel r^+$ 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 $e_i^\parallel r^+$ $(1 \leq i \leq n)$. Hence $V^\parallel r^+$ will form a continuous boundary with $E^\parallel r^+$ (refer to an example as shown in Fig. 8(d)-right).

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

The construction of $F^\parallel r^+$ and $E^\parallel r^+$ is straightforward since they are the translated faces of $F$ and the truncated cylinders of $E$, respectively; however, the construction of $V^\parallel r^+$ needs further consideration. The neighboring edges $e_1, e_2, \ldots, e_n$ 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. [23] presented a notion of polyhedral tracings for convolutions between polyhedrons. Three types of polyhedral tracings 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 Sections 5.2 and 5.4).

An example of such a complex vertex is shown in Fig. 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 Fig. 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 Fig. 9(c), we can still form a closed loop on the sphere related to all the edges even though the loop might have multiple self-intersections. The loop edges related to $e_2$ and $e_6$ are shown in dotted lines in Fig. 9(c). In our method, a spherical region $v^\parallel r^+$ will be constructed, whose boundary $\Psi$ is the reverse edge of the closed loop generated by edges $e_1, \sim e_7$. Hence $v^\parallel r^+$ and $e_i^\parallel r^+$ will still form a continuous boundary along $\Psi$.  

Fig. 8. An illustration of offsetting faces, edges and vertices.
Our approach of computing the offset meshes for 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 \( \Psi_{ei} \) is always the smaller one. This is because: (i) If \( e_i \) is a convex or flat edge, the angle \( \alpha \) between \( f_1 \) and \( f_2 \) must be \( 0^\circ < \alpha \leq 180^\circ \). The positive normal offset of \( f_1 \) and \( f_2 \) will form an angle \( \angle n_1 n_2 = 180^\circ - \alpha \). Therefore, \( \Psi_{ei} \) is smaller than half of a great circle. (ii) If \( e_i \) is a concave edge, the angle \( \alpha \) between \( f_1 \) and \( f_2 \) must be \( 180^\circ < \alpha < 360^\circ \). The positive normal offset of \( f_1 \) and \( f_2 \) will form an angle \( \angle n_1 n_2 = \alpha - 180^\circ \). Therefore, \( \Psi_{ei} \) is still smaller than half of a great circle. Such an observation has also been presented in \[23\]. In addition, all the computed arcs \( \Psi_{ei} \) will form a closed spherical loop \( \Psi \) (refer to the example as shown in Fig. 9(c)).

The spherical loop \( \Psi \) may have multiple self-intersections. Our approach of constructing \( v \parallel \Psi \) based on such a spherical loop is described as follows.

1. Suppose \( C_{ei} \) is the center of an edge \( \Psi_{ei} \). We first calculate point \( O' \) for \( \Psi \) 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 \( V \) of \( \Psi \) onto \( P \) (denoted as \( V' \)) by ensuring that points \( v, O, V_i, \) and \( V'_i \) are in the same plane and the length \( OV'_i \) is the same as the arc length \( OV_i \). Consequently, based on such a mapping, the spherical loop \( \Psi \) is converted into a 2D planar loop \( \Psi' \).

2. We then use a triangulation method that is similar to the well-known XOR polygon filling algorithm to compute a triangulation of \( \Psi' \). That is, we construct a triangulation for each edge \( \Psi'_{ei} \) individually. An example of such a triangulation result for a boundary \( \Psi \) in Fig. 9(c) is shown in Fig. 9(d).

Note that to ensure the continuity of the offset meshes, some triangles \( O-\Psi_{ei} \) may be flipped depending on the direction of \( \Psi_{ei} \) related to \( O \) (refer to \( O-\Psi_{e1} \) and \( O-\Psi_{e2} \)). 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 that the same triangulation boundaries are generated between \( O-\Psi_{ei} \) of \( V \parallel \Psi \) and the related loop edge of \( E \parallel \Psi \). An example of the related \( V \parallel \Psi \) and \( E \parallel \Psi \) is shown in Fig. 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 \( p \) 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 a 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 Fig. 9(f)). The resulted contour can be further decimated as shown in Fig. 9(g).

Our approach of removing self-intersections and overlapping surfaces in the offset meshes \( F \parallel \Psi \cup E \parallel \Psi \cup V \parallel \Psi \) is discussed in the next section.

5. Compute boundary points based on LDNI

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 Image (LDNI) to extract the boundary surfaces of \( S \cup r \). 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 Image (LDNI) is a point representation which sparsely encodes the shape of solid models in three orthogonal directions \[8\]. A structural set of Layered Depth-Normal Images consists of \( x \)-LDNI, \( y \)-LDNI and \( z \)-LDNI along \( X, Y, \) and \( Z \) axes, 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 \( \mathbb{R}^3 \). A LDNI in an axis is a sequence of 2D 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-tuples \((d, n_x, n_y, n_z)\), where \(d\) specifies the depth from an intersection point \(P\) to the viewing plane, and \(N_0(n_x, n_y, n_z)\) is the surface normal at \(P\).

The construction of a LDNI model for a polygonal model can be performed rather quickly with the aid of graphics hardware \cite{8}. 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 LDNI 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 LDNI model for each tile can then be generated and processed independently (either sequentially or in parallel).

Therefore, from the offsets surfaces \(F \parallel_d^+ \cup F \parallel_d^- \cup V \parallel_d^+\), we can compute a LDNI 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 \(P_i \sim P_j\), within a range \([d_{min}, d_{max}]\) along the axis. Point \(P_i \sim P_j\) 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 \(N_i\), we increase \(I_{norm}\) by 1 if \(N_i \cdot N_{ray} < 0\) and decrease \(I_{norm}\) by 1 if \(N_i \cdot N_{ray} > 0\). The unit normal \(N_{ray}\) is along the LDNI axis from \(d_{min}\) to \(d_{max}\). Refer to an example in Fig. 5(c). It is obvious that each line segment will have a unique integer value \(I_{norm}\) (can be positive or negative). Such an \(I_{norm}\) value is called winding number in \cite{21}.

We present three point filters for removing inner points, therefore removing self-intersections. The first filter, named small segment filter, determines whether a pair of sampling points comes from two overlapping surfaces and should be filtered. The second filter, named ray casting filter, determines whether a sampling point is an inner point by judging its two neighboring \(I_{norm}\) values. The third filter, named offset property filter, determines whether a set of sampling points that form a shell are inner points by judging whether any points are generated from invalid edges. The three filters are discussed in more detail in Sections 5.2.5–5.4 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 \(|d_{ij} - d_{pj}| < \varepsilon\) and \(N_i \cdot N_{pj} < 0\), where \(\varepsilon\) is a small segment tolerance (e.g. \(10^{-5}\) in our implementation) and \(N_0(n_x, n_y, n_z)\) 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 \(F \parallel_d^+ \cup F \parallel_d^- \cup V \parallel_d^+\), that are generated by the same geometric element in \(\partial(S)\) (refer to an example in Fig. 9(c)). The small segment filter will then be used to remove all the gaps or thin shells whose thickness is less than \(\varepsilon\) in the offset model. Such small gaps and thin shells are usually non-manufacturable in most CAD/CAM applications.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>(I_{norm}) effect of different geometric elements on offset meshes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric element of (dS)</td>
<td>Change of (I_{norm})</td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>Face</td>
<td>+1</td>
</tr>
<tr>
<td>Edge (Valid)</td>
<td>+1</td>
</tr>
<tr>
<td>Edge (Invalid)</td>
<td>-1</td>
</tr>
<tr>
<td>Vertex (Valid)</td>
<td>+1</td>
</tr>
<tr>
<td>Vertex (Invalid)</td>
<td>-1</td>
</tr>
</tbody>
</table>

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 that \(N_{pi} \cdot N_{ray} \neq 0\), where \(N_{pi}\) is the face normal at \(P_i\) and \(N_{ray}\) is the unit normal of the ray. Therefore, \(I_{norm1} = I_{norm1} + 1\) if \(N_{pi} \cdot N_{ray} < 0\) and \(I_{norm2} = I_{norm1} - 1\) if \(N_{pi} \cdot N_{ray} > 0\). □

**Proposition 5.2.** 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 inside 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 \(\partial(S^+ r)\) and \(\partial(S^- r)\) must lie in the regions that have \(I_{norm} = 0\) and \(I_{norm} = 1\), respectively (refer to an example as shown in Table 1). This is also shown in the properties of winding number \cite{21}.

Examples of \(f \parallel_d^+, e \parallel_d^+\) and \(f \parallel_d^-\) are shown in Fig. 10(b) and (c). Regions \(R_i\) and \(R_e\) that are defined by \((f, f \parallel_d^+), (e, e \parallel_d^+),\) and \((e, e \parallel_d^-)\), respectively, are also shown in the figure. Obviously for a point \(P\) inside \(R_i\) 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 \((f \parallel_d^+, e \parallel_d^+\) or \(v \parallel_d^+)\) of a geometric element \((f, e, v)\) 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 \(F \parallel_d^+ \cup F \parallel_d^- \cup V \parallel_d^+\) will change the \(I_{norm}\) of the points between \(\partial(S)\) and \(\partial(S^+ r)\), we can identify the boundary points of \(\partial(S^+ r)\) by judging the point’s \(I_{norm}\) values. Therefore we can avoid the distance field computation that is usually more computationally expensive. Note that the \(I_{norm}\) 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 \(f_1 \parallel_d^+, e \parallel_d^+, f_2 \parallel_d^-\) for a given \(S\) grown by \(r\) are shown in Fig. 11(a). The related offset regions \(R_1, R_e, R_2\) are also shown in Fig. 11(b). Their effects on \(I_{norm}\) changes can be identified from Table 1. Hence the \(I_{norm}\) 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 Fig. 11(c). Accordingly some offset meshes can be identified as invalid (marked as ‘\(\times\)’) even though no distance computation is performed between the offset and original meshes.

**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 \(I_{norm2} = 0\).
After the small segment filter, the LDNI points related to singular edges and non-manifold vertices will be removed. Therefore, \( S^{t\ast}r \) and \( S^{\downarrow\ast}r \) 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 \( P \) with \( I_{\text{Norm}}^1 = 0 \) and \( I_{\text{Norm}}^2 = 1 \), or \( I_{\text{Norm}}^1 = 1 \) and \( I_{\text{Norm}}^2 = 0 \). If any of the point is an inner point, the whole shell is inside the offset distance to \( \partial S \).

**Proof.** After the small segment filter, the LDNI points related to singular edges and non-manifold vertices will be removed. Therefore, \( S^{t\ast}r \) and \( S^{\downarrow\ast}r \) will be separated into individual volumes that are defined by one or multiple shells.

As discussed in Section 4, we require the offset meshes \( F \mid_1^{t\ast} \cup F \mid_1^{\downarrow\ast} \cup V \mid_2^{t\ast} \cup V \mid_2^{\downarrow\ast} \) form a continuous boundary. Therefore the offset meshes \( E \mid_1^{t\ast} \) and \( V \mid_2^{t\ast} \) may flip their surface orientations due to self-intersections. In our method, we classify all the edges and vertices of \( \partial S \) based on such an 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 Fig. 11. Note that for an invalid edge \( e \), the surface orientation of related \( e \mid_1^{t\ast} \) is flipped (i.e. surface normals pointing to \( e \) instead of away from it). Hence we can classify all the related points of \( E \mid_1^{t\ast} \) as invalid or valid edge points. Similarly as discussed in Section 4, we identify all the complex vertices of \( \partial S \) that has both concave and convex neighboring edges and process the related sampling points of \( V \mid_2^{t\ast} \). We define invalid vertex points as the points whose normal is flipped (i.e. pointing to \( e \) 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 \( I_{\text{Norm}}(P) \) by 1; while the effect of all other offset surfaces will increase \( I_{\text{Norm}}(P) \) 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 \( \partial S \) and \( \partial (S^{t\ast}r) \) for \( S \) grown by \( r \), and some islands inside the region between \( \partial S \) and \( \partial (S^{\downarrow\ast}r) \) for \( S \) shrunk by \( r \). For an example as shown in Fig. 12(d), \( I_{\text{Norm}} \) at the eight cubic corners will be changed by the offset regions of three faces and three invalid edges since \( P \in (R_{e_1} \cap R_{e_2} \cap R_{e_3} \cap R_{e_4} \cap R_{e_5} \cap R_{e_6}) \). Therefore, from the initial \( I_{\text{Norm}} \) value of 1 (i.e. inside \( \partial S \) for \( S \mid_2^{\downarrow\ast}r \)), \( I_{\text{Norm}} \) will be modified based on the effects of related faces and edges (refer to an example as

**Proof.** After the small segment filter, the LDNI points related to singular edges and non-manifold vertices will be removed. Therefore, \( S^{t\ast}r \) and \( S^{\downarrow\ast}r \) will be separated into individual volumes that are defined by one or multiple shells.

As discussed in Section 4, we require the offset meshes \( F \mid_1^{t\ast} \cup F \mid_1^{\downarrow\ast} \cup V \mid_2^{t\ast} \cup V \mid_2^{\downarrow\ast} \) form a continuous boundary. Therefore the offset meshes \( E \mid_1^{t\ast} \) and \( V \mid_2^{t\ast} \) may flip their surface orientations due to self-intersections. In our method, we classify all the edges and vertices of \( \partial S \) based on such an 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 Fig. 11. Note that for an invalid edge \( e \), the surface orientation of related \( e \mid_1^{t\ast} \) is flipped (i.e. surface normals pointing to \( e \) instead of away from it). Hence we can classify all the related points of \( E \mid_1^{t\ast} \) as invalid or valid edge points. Similarly as discussed in Section 4, we identify all the complex vertices of \( \partial S \) that has both concave and convex neighboring edges and process the related sampling points of \( V \mid_2^{t\ast} \). We define invalid vertex points as the points whose normal is flipped (i.e. pointing to \( e \) 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 \( I_{\text{Norm}}(P) \) by 1; while the effect of all other offset surfaces will increase \( I_{\text{Norm}}(P) \) 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 \( \partial S \) and \( \partial (S^{t\ast}r) \) for \( S \) grown by \( r \), and some islands inside the region between \( \partial S \) and \( \partial (S^{\downarrow\ast}r) \) for \( S \) shrunk by \( r \). For an example as shown in Fig. 12(d), \( I_{\text{Norm}} \) at the eight cubic corners will be changed by the offset regions of three faces and three invalid edges since \( P \in (R_{e_1} \cap R_{e_2} \cap R_{e_3} \cap R_{e_4} \cap R_{e_5} \cap R_{e_6}) \). Therefore, from the initial \( I_{\text{Norm}} \) value of 1 (i.e. inside \( \partial S \) for \( S \mid_2^{\downarrow\ast}r \)), \( I_{\text{Norm}} \) will be modified based on the effects of related faces and edges (refer to an example as
shown in Fig. 11(c). Hence the points inside the shell will have 
\( h_{\text{norm}} = 1 - 1 - 1 = 1 + 1 + 1 = 1 \). Consequently they will 
pass the ray casting filter.

**Proposition 5.6.** Suppose a point \( P_i \) has \( h_{\text{norm}1} = 0 \) and \( h_{\text{norm}2} = 1 \), 
or \( h_{\text{norm}1} = 1 \) and \( h_{\text{norm}2} = 0 \). 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 \( h_{\text{norm}} \) value for \( S \) grown by \( r \) 
and increase its \( h_{\text{norm}} \) value for \( S \) shrunk by \( r \). Therefore, if none of 
the geometric elements are invalid edges or vertices, \( h_{\text{norm}} > 1 \) for 
\( S \) grown by \( r \) and \( h_{\text{norm}} < 0 \) for \( S \) shrunk by \( r \). Hence these geometric 
elements must contain some invalid edges or vertices in order to 
make \( h_{\text{norm}} = 0 \) or 1. □

**Proposition 5.7.** Suppose a shell contains a set of points \( P_i \) with 
\( h_{\text{norm}1} = 0 \) and \( h_{\text{norm}2} = 1 \), or \( h_{\text{norm}1} = 1 \) and \( h_{\text{norm}2} = 0 \). If the 
shell is inside the offset distance to \( \partial S \), 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 \( \partial S \), 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 \( h_{\text{norm}} = 1 \). □

**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 
LDNI model. An invalid LDNI point can then be used as a seed for 
the flood filling operation to remove all the points in the same shell. 
Note that 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 LDNI 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 Figs. 6(c) and 12(d) can be 
effectively removed.

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

### 6. Reconstructing offset contour

The generated LDNI 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 LDNI model into a polygonal model.

#### 6.1. Adaptive sampling for cell representation

As discussed in Section 5.1, the accuracy of a generated LDNI 
model depends on the pixel width \( \delta \) used in the construction 
process. Hence a small \( \delta \) 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 LDNI 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 LDNI model.

In our method, we construct another type of implicit representa-
tion, an adaptive cell representation, from a LDNI representa-
tion. The adaptive cell representation contains two types of cells, 
uniform cells and octree cells [24]. 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 LDNI representation including the approaches of handling vol-
ume titling is presented in [8]. During the adaptive sampling test, 
we calculate an error-minimizing point of a cell from all the sam-
pling points within the cell and explicitly compare the approxima-
tion error with a given tolerance \( \epsilon \). If the approximation error is 
smaller than \( \epsilon \), 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 LDNI 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 [10]. 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 [25]. It produces a lower number of polygons to approximate an input solid. The approximation accuracy can be controlled by a decimation tolerance $\alpha$. A mesh decimation algorithm will try to delete as many triangles as possible while ensuring that the maximum approximation error is smaller than $\alpha$. Therefore, we can first process an input polygonal model by using such a mesh decimation algorithm. The decimation tolerance $\alpha$ can be set as a fraction of $\theta$ where $\theta$ is the required offsetting accuracy. In most cases, the decimation of an input polygonal model can significantly reduce the number of geometric elements in $\delta S$ and consequently improve the speed of computing offset surfaces $F \parallel^{+} r \cup V \parallel^{+} r$.

7.2. Approximation errors of cylinders and spheres

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

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 $F \parallel^{+} r \cup E \parallel^{+} r \cup V \parallel^{+} r$ 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 a corresponding vertex in the input model $\delta S$. Note that if $r_2 \gg r_1$, 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 different signs. In those cases, we may recalculate the offset meshes.

7.4. Volume titling for high accuracy

In generating a LDNI model for offset meshes $F \parallel^{+} r \cup E \parallel^{+} r \cup V \parallel^{+} r$, we can set a pixel width $\delta$ according to a required offset accuracy $\theta$. However, if $\theta$ 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 1280 × 1024). In these cases, we split the bounding box of the model into multiple smaller tiles. We display each tile and construct a LDNI model independently. Each tile also needs a buffer region around its boundary to ensure the continuity of the reconstructed surfaces [8]. 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 LDNI 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 LDNI model rather quickly (usually within seconds). All the generated LDNI models can be saved in a network-connected hard disk, which can be accessed from a cluster of PCs [8]. The LDNI 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 LDNI 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 to have a valid normal (that is, $N_i \neq \mathbf{0}$). 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 $I_{\delta S}$ for the sampling points of a LDNI model requires the offset meshes $F \parallel^{+} r \cup E \parallel^{+} r \cup V \parallel^{+} r$ to form a continuous boundary. Due to the floating-point arithmetic in geometric computations, the corresponding vertices of $F \parallel^{+} r$, $E \parallel^{+} r$, and $V \parallel^{+} r$ may have small numerical errors. Hence the constructed offset meshes need to be verified by sealing the boundary before computing the LDNI.

8. Experimental results and applications

We used the 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 × 1 × 1) 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 constructed offset models by our method, a publicly available Metro tool [26] 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 $\xi$ used in constructing offset faces $E$ and $V$ is 0.025. The related maximum approximation error $\lambda$ is 0.00078 and 0.0016 for cylinders and spheres, respectively. The pixel width $\delta$ used in constructing LDNI is 0.005 (i.e., the image resolution required for the unit cube is $200 \times 200$). The tolerance $\varepsilon$ used in the adaptive sampling is 0.001. Since the offsetting accuracy is quite satisfactory, the same setting is used in all other tests.

**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. Besides 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 the offset faces (refer to Section 4); (ii) generating and processing a LDNI model (refer to Section 5); and (iii) reconstructing the 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 LDNI 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 [4], the required running time based on our method is much less ($1/5 \sim 1/15$ of running time) for the same test cases such as bunny and dragon. In addition, since the software application of [6] 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. Note that the running time of our software only includes the steps of generating offset faces and processing LDNI since Lien’s software can only generate boundary points. However, note that the comparison is somehow unfair since the method presented in [6] can be used for the general Minkowski sum while our approach is just for uniform offsetting.

<table>
<thead>
<tr>
<th>Models</th>
<th>Offset dist.</th>
<th>Size</th>
<th>Polygonal mesh error</th>
<th>Average error distance ($E_{ave}$)</th>
<th>Root mean square (RMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube</td>
<td>0.1</td>
<td>1.0 $\times$ 1.0 $\times$ 1.0</td>
<td>0.000005</td>
<td>0.000001</td>
<td>0.000002</td>
</tr>
<tr>
<td>Pyramid</td>
<td>0.1</td>
<td>1.0 $\times$ 1.0 $\times$ 1.0</td>
<td>0.001643</td>
<td>0.000004</td>
<td>0.000018</td>
</tr>
<tr>
<td>Sphere</td>
<td>0.1</td>
<td>1.0 $\times$ 1.0 $\times$ 1.0</td>
<td>0.001311</td>
<td>0.000224</td>
<td>0.000290</td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.1</td>
<td>1.0 $\times$ 1.0 $\times$ 1.0</td>
<td>0.001263</td>
<td>0.000246</td>
<td>0.000381</td>
</tr>
</tbody>
</table>

**Single offsetting operation.** The uniform offsetting can be used to create a thin shell of a solid model by simply merging an input model $\partial S$ and an offset model $\partial(S \uparrow \gamma$ 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 [27]. 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. Uniform offsetting is ideal for such a purpose since it can ensure uniform cutting depth during machining. An example based on an octa-flower model is shown in Fig. 17. The offsetting results related to different offset distances from 0.02 to 0.15 are shown in the figure. We put two pairs 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 [4]. That is, $S$ filleted by $r$ can be defined as $F_r(S) = S \uparrow \downarrow \gamma r$, and $S$ rounded by $r$ can be defined as $R_r(S) = S \downarrow \uparrow r$. An example of added fillets and rounds in a Beethoven statue model is shown in Fig. 18. The algorithm performance of the offsetting operations is also given in Table 3. Compared to the experimental results presented in [4], the method based on LDNI is much faster.

### 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 that a promising approach for the offsetting operation is to compute an approximated boundary based on point representations. In this paper, we presented a novel LDNI-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 LDNI model from the offset faces, which contains a set of well-structured sampling points. Accordingly, three point filters have been developed to delete all the inner points. Finally, the offset model can be reconstructed from the processed LDNI 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.
Table 3
Algorithm performance of our test results.

<table>
<thead>
<tr>
<th>Models</th>
<th>Tri # (K)</th>
<th>Size</th>
<th>Offset dist.</th>
<th>Memory</th>
<th>Running time (s)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Offset tri # (K)</td>
<td>LDNI point # (K)</td>
<td>Generate offset faces</td>
</tr>
<tr>
<td>Accuracy test cases</td>
<td></td>
<td></td>
<td></td>
<td>Mem</td>
<td>Tri #</td>
<td>Offset dist.</td>
</tr>
<tr>
<td>Cube (Fig. 10)</td>
<td>0.012</td>
<td>1 × 1 × 1</td>
<td>−0.1</td>
<td>0.8</td>
<td>153.6</td>
<td>0.008</td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.4</td>
<td>1 × 1 × 1</td>
<td>−0.1</td>
<td>5.0</td>
<td>142.6</td>
<td>0.05</td>
</tr>
<tr>
<td>Sphere</td>
<td>10.2</td>
<td>1 × 1 × 1</td>
<td>−0.1</td>
<td>0.8</td>
<td>74.0</td>
<td>0.008</td>
</tr>
<tr>
<td>Pyramid</td>
<td>0.006</td>
<td>1 × 1 × 1</td>
<td>−0.1</td>
<td>20.3</td>
<td>120.4</td>
<td>0.26</td>
</tr>
<tr>
<td>Freeform models</td>
<td></td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Bunny (Fig. 13)</td>
<td>69.6</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Dragon (Fig. 3)</td>
<td>69.15</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Liberty (Fig. 12)</td>
<td>38.0</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Octa-flower (Fig. 16)</td>
<td>15.8</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Statue (Fig. 17)</td>
<td>5.0</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Statue +0.02 (Fig. 17)</td>
<td>142</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Statue −0.02 (Fig. 17)</td>
<td>12.9</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Engineering models</td>
<td></td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Case 1 (Fig. 14)</td>
<td>0.044</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Case 2</td>
<td>0.38</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Tutor (Fig. 15)</td>
<td>1.4</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
<tr>
<td>Hub (Fig. 4)</td>
<td>17.9</td>
<td></td>
<td></td>
<td>Mem</td>
<td>Mem</td>
<td>Mem</td>
</tr>
</tbody>
</table>
Fig. 13. Screen capture of the offset results for a liberty model.

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

Fig. 15. Screen capture of the offset results for a case 1 model.

Fig. 16. Screen capture of the offset results for a tutor model.
Some future work we are investigating are as follows. We are investigating approaches for improving our algorithm on handling solid models with degenerated data; we are exploring the usage of our offsetting method in new applications; we plan to extend our method to other solid operations such as general Minkowski operations.

Acknowledgements

The work is partially supported by the National Science Foundation grant CMMI-0927397, the Hong Kong RGC/GRF grant CUHK/417508, and the CUHK direct grant CUHK/2050400.

References


