

# A Point-Based Offsetting Method of Polygonal Meshes

Yong Chen <sup>a\*</sup>  
Assistant Professor

Hongqing Wang <sup>b</sup>  
Sr. Software Engineer

David W. Rosen <sup>c</sup>  
Professor

Jarek Rossignac <sup>d</sup>  
Professor

<sup>a</sup> Epstein Department of Industrial and Systems Engineering, University of Southern California, Los Angeles, CA 90089

<sup>b</sup> 3D Systems Inc., 26081 Avenue Hall, Valencia, CA 91354

<sup>c</sup> The George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA 30332

<sup>d</sup> GVU Center, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332

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

## Abstract

We address the delicate problem of offsetting polygonal meshes. Offsetting is important for stereolithography, NC machining, coordinate measuring machines, robot collision avoidance, and Hausdorff error calculation. We introduce a new fast, and very simple method for offsetting (growing and shrinking) a solid model by an arbitrary distance  $r$ . Our approach is based on a hybrid data structure combining point samples, voxels, and continuous surfaces. Each face, edge, and vertex of the original solid generates a set of offset points spaced along the (pencil of) normals associated with it. The offset points and normals are sufficiently dense to ensure that all voxels between the original and the offset surfaces are properly labeled as either too close to the original solid or possibly containing the offset surface. Then the offset boundary is generated as the isosurface using these voxels and the associated offset points. We provide a tight error bound on the resulting surface and report experimental results on a variety of CAD models.

**Keywords:** Offsets, Tool Path Generation, Polygonal Meshes, Point-Sampled Geometry, Solid Modeling, Blending, Distance Fields.

## 1. Introduction

Polygonal meshes are the simplest and most popular representation of 3D models. They are supported by commercial CAD software and graphics systems. Different CAD formats can be easily converted to polygonal meshes. Real objects, such as teeth or bones in various medical applications, may be scanned into 3D models, producing a triangle mesh with a large number of triangles. As 3D scanners are used more widely, offsetting such a triangle mesh robustly and quickly becomes necessary for various computer-aided manufacturing (CAM) applications. Some CAM applications which directly use offsetting operation include tool path generation for 3D NC machining, rapid prototyping and coordinate measuring machines, tolerance and clearance analysis for assemblies, and robot path planning (Rossignac and Requicha 1986) (Pham 1992) (Maekawa 1999). In other CAM applications, offsetting operation is used to change the input solid model for constructing tools. For example, an enlarged model is required for casting or forging processes in order for the following machining processes to have enough allowances to generate the desired tolerances.

Offsets of solids, which are defined as grown or shrunken versions of the solid, have been defined precisely for point sets (Rossignac and Requicha 1986). Offset results are generated individually for surfaces, edges and vertices of a solid model and then combined by a Boolean union. But their computation for solids bounded by triangle meshes has proven difficult because of its computational complexity and numeric instability during the Boolean operation.

(1) Volumetric representations (Kaufman, Cohen et al. 1993) have become popular, partly because they avoid the need for the computation of geometric intersections and topological decisions based on geometric calculations. Furthermore, voxel data structures have been used (Sethian, 1996) to accelerate the computation of topological changes during the deformation of level set surfaces. The global consistency of the final surface is guaranteed by the level-set reconstruction.

(2) Point based representations of shapes and their combinations with voxel models to accelerate computations have gained popularity (Pauly, Keiser et al. 2003) (Adams and Dutre 2003).

(3) Digital signal processing (DSP) techniques are widely used in audio and image processing. An analog signal is first converted into a digital presentation with an analog to digital converter. After processing digital signal, it converts the processed digital data into analog signal again by a digital to analog converter.

The above observations motivated us to develop a point based offsetting method that uses voxels to resolve the topology of the resulting point set. In this paper, we propose a simple, yet effective approach for computing offsets of solids bounded by triangle meshes. Our approach is based on a precise mathematical definition of offsets, on volumetric data structures, and on point primitives. We present our data structures and a set of algorithms that are designed to combine the advantages of parametric surface models, volumetric models and implicit models. The main process of our approach illustrated by a simply cube example is presented in Figure 1. The basic definitions and mathematical properties of offsets are first given in Section 3. The five major steps in Figure 1 and related data structures are discussed in Section 4 to 6. Our offsetting method and its error and performance analyses are discussed in Section 7. We demonstrate the accuracy and the efficiency of our approach in different applications in Section 8. Conclusions and future work are given in Section 9.

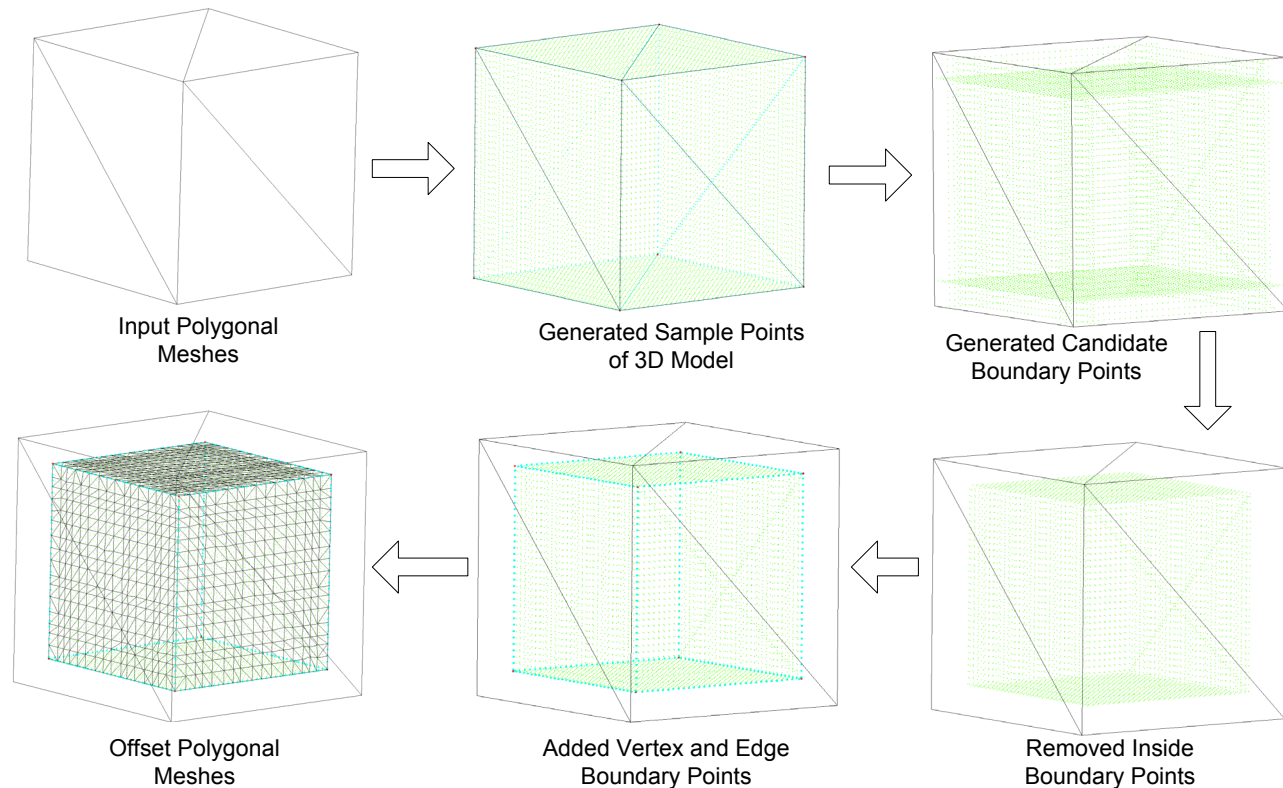


Figure 1. The Main Processes of Point-based Offsetting Method.

## 2. Related Work

Earlier approaches (Rossignac and Requicha 1986) (Farouki 1985) (Sato and Chiyokura 1991) (Frosyth 1995) first compute a superset of the offset surface 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 impractical or at least difficult to support reliably in an industrial strength system

More recently, Qu and Stucker presented an offset method based on moving triangle vertices (Qu and Stucker 2003). The same topology was used to construct offset results. The approach of calculating the position of offset vertices was presented. However, the solution is limited to sufficiently small offset values, for which local and global self-intersections do not happen or is not a main issue.

Some offsetting methods were developed by converting offsets of 3D geometry into offsets of 2D contours. Lam *et al.* described an approach based on slicing geometries into 2D contours and offsetting each slice contour based on pixels (Lam, Yu et al. 1997). McMains *et al.* presented algorithms for building thin-walled parts in fused deposition modeling (FDM) machine (McMains, Smith et al. 2000). Geometries are sliced first followed by creating 2D offset contours. Allen and Dutta developed an algorithm for building thin shell surfaces with minimum supports in layered manufacturing processes (Allen and Dutta 1998). This approach is mainly used for special applications such as layer based manufacturing, and cannot be used for other applications which require continuous offsetting results.

A ray representation method was used in (Hartquist, Menon et al. 1999). It is a set of line segment that lie inside the solid and generated by clipping a regular grid of lines against a solid model. Hartquist et al. proposed the strategy of using ray representations and parallel hardware as primary media for offsets, sweeps, and Minkowski operations.

In image processing, image dilation based on the neighborhood of pixels to calculate distance is common. Different distance metrics such as city-block and chessboard distance were used to generate distance transform images (Ritter 2001). Using the similar principle, Gurbuz and Zeid extended offsetting operations from 2D pixels to 3D voxels (Gurbuz and Zeid 1995). A cubic (chessboard neighborhood) was used to approximate the closed ball expansion. However, the offset errors in this approach are accumulated. So the total error can have the same magnitude order of offset distance regardless the voxel size.

A more advanced method based on distance volume and fast marching method was presented for offsetting 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 surface. Additionally, a second set of points, labeled the zero set, which lies on the CSG model's surface 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 closest point information out to the remaining voxels in the volume. Other works on calculating distance maps and their representations can also be found (Gibson 1999; Frisken, Perry et al. 2000). However, most of these works were developed for displaying purpose in computer graphics domain. No accuracy analysis of offsetting results was found in the literatures while it is a main concern for manufacturing purpose such as CNC and RP applications.

More recently, Kim (Kim, Varadhan et al. 2003) presented a five-stage pipeline to approximate the swept volume of a polyhedron along a given trajectory. The main approach was based on computing unsigned

distance fields on a uniform grid, classifying all grid points using fast marching front propagation and reconstructing volume from iso-surfaces. Various applications based on point primitives are also studied. Pauly et al. (2003) presented a point-based approach for interactively modeling shapes, and Adams and Dutre (2003) presented a point-based approach for interactive Boolean operations.

### 3. Basic Definitions and Formulation

The mathematical properties of offsets are the foundation of the methodology developed in this paper. We briefly review the properties of offsetting operations mainly based on (Rossignac and Requicha 1986). We then introduce the related discrete operations to derive our offset approaches. Finally we illustrate the offsetting process with a 2D example.

#### 3.1 Continuous Offsetting Operations

Offsetting operations are special cases of Minkowski sums and differences. Suppose two sets  $A$  and  $B$  are closed and regular subset of Euclidean space  $E^2$  or  $E^3$ . The Minkowski sum of  $A$  and  $B$ , denoted  $A \oplus B$ , is defined as  $C = A \oplus B = \{a + b \mid a \in A, b \in B\}$ , where “+” denotes the normal vector addition of two points. The

Minkowski difference, denoted  $A \otimes B$  is  $\overline{A \oplus B}$ . Suppose a ball with radius  $r$  is defined as  $b_r$ , we can define two offsetting operations,  $S$  grown by  $r$  as  $S \uparrow_r = S \oplus b_r$ , and  $S$  shrunk by  $r$  as  $S \downarrow_r = S \otimes b_r$  for any model  $S$ . In terms of point/set distance, Nadler (1978) define the regularized offset of a regular set  $S$  by a positive distance  $r$  as  $S \uparrow^* r = \{p : d(p, S) \leq r\}$ , where  $d(p, S) = \inf_{q \in S} \|p - q\|$  and *inf* denotes the *greatest lower bound*.

From this definition, if  $p$  is exterior to  $S$ , then the closet points  $q$  lie in  $\partial S$ , the topological boundary of  $S$ . That is,  $\partial(S \uparrow^* r) \subset \{p : d(p, S) = r\}$ . 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.

Suppose point  $p \in \partial(S \uparrow^* r)$ , the distance  $d(p, S)$  must be attained from some point  $q$  of  $\partial S$ . In this paper, we will use the symbols  $v(S)$ ,  $e(S)$ , and  $f(S)$  to refer to a vertex, an edge, or a face of  $S$ . And  $V(S)$ ,  $E(S)$ ,  $F(S)$  will refer respectively to the sets of all the vertices, edges and faces of  $S$ . As  $\partial S = V(S) \cup E(S) \cup F(S)$ , point  $q$  to make  $d(p, S) = r$  must be in one of the sets.

- (1) Faces  $F(S)$ : Suppose  $q \in F$ . We can construct the set  $F\|_r^+$  (positive normal offset of faces) by displacing each point  $q$  of  $F$  by a distance  $r$  along the unit normal  $n$  at  $q$ , i.e., let  $p = q + rn$ .
- (2) Edges  $E(S)$ : Suppose  $q \in E$  and the normals of two neighboring faces ( $f_1, f_2$ ) of  $q$  are  $n_1, n_2$ . We can construct the set  $E\|_r^+$  (positive normal offset of edges) by constructing an arc of radius  $r$  and center  $q$  in the normal plane for all points  $q$  of  $E$ . The arc is bounded by  $n_1$  and  $n_2$  (including the normal point), because points outside the arc are closer to  $f_1$  and  $f_2$ .
- (3) Vertices  $V(S)$ : Suppose  $q \in V$  and the neighboring edges of  $q$  are  $e_1, e_2, \dots, e_i$ . We can construct the set  $V\|_r^+$  (positive normal offset of vertices) by constructing a sphere region of radius  $r$  and center  $q$  for all points  $q$  of  $V$ . The sphere region is bounded by the positive normal offset of edges  $e_1 \sim e_i$ .

So  $\partial(S \uparrow^* r) \subset F\|_r^+ \cup E\|_r^+ \cup V\|_r^+$ . Notice all points of  $F\|_r^+, E\|_r^+, V\|_r^+$  are at a distance  $r$  from some points of  $\partial S$ ; however some may be at a smaller distance to other points of  $\partial S$ . Also the offsets of smooth surfaces need not be smooth because they may self-intersect, and the normal offsets of a surface need not even be a

surface (can be a point). In this paper, we used volumetric representation to handle these problems. Accordingly, the offsetting operations defined for continuous boundary cases need to be converted into discrete offsetting operations.

### 3.2 Discrete Offsetting Operations

Suppose  $C$  is a set of uniform grid cells (cubes) enclosing the bounding box of  $S \uparrow^* r$ . Each grid cell  $c$  is a set of points:  $\{(x, y, z) : x_{center} - \frac{x_{size}}{2} \leq x < x_{center} + \frac{x_{size}}{2}, y_{center} - \frac{y_{size}}{2} \leq y < y_{center} + \frac{y_{size}}{2}, z_{center} - \frac{z_{size}}{2} \leq z < z_{center} + \frac{z_{size}}{2}\}$  where  $(x_{center}, y_{center}, z_{center})$  is the center of  $c$ , and  $x_{size}, y_{size}, z_{size}$  are the size of  $c$ . We introduced an operation  $sp_c(A)$  for set  $A$  and cell  $c$  as taking sample points  $p$  such that  $p \in A \cap c$ . Obviously  $sp_c(A) = 0$  if  $A \cap c = 0$ . We further defined set  $SP_C(A)$  as points  $sp_c(A)$  for all  $c$  of  $C$ . Thus, the definition of offset can be extended from the continuous surfaces to discrete point sets.

Since  $SP_C(\partial(S \uparrow^* r)) \subseteq SP_C(F \uparrow_r^* \cup E \uparrow_r^* \cup V \uparrow_r^*) \subseteq SP_C(F \uparrow_r^*) \cup SP_C(E \uparrow_r^*) \cup SP_C(V \uparrow_r^*)$  by using the same sampling approach,  $SP_C(\partial(S \uparrow^* r))$  can be constructed from  $SP_C(F \uparrow_r^*)$ ,  $SP_C(E \uparrow_r^*)$ , and  $SP_C(V \uparrow_r^*)$ . This can be achieved by (1) for  $F(S)$ ,  $E(S)$ , and  $V(S)$  of a solid  $S$ , we discretize them by  $SP_C(F)$ ,  $SP_C(E)$ , and  $SP_C(V)$  respectively; (2) for each point of  $SP_C(F)$ ,  $SP_C(E)$ , and  $SP_C(V)$ , we discretize their positive normal offsets by  $SP_C(Line_r)$ ,  $SP_C(Arc_r)$ , and  $SP_C(Sphere_r)$  respectively. Suppose the sampling rate is  $SR_P$ , which is defined as the number of sampling points per unit length along an axis. Correspondingly the sample point size  $SS_{point}$  is  $1/SR_P$ . Two kinds of points, *boundary points* and *inside points*, are defined here. The *boundary points* are points  $p$  with  $d(p, q) = r$  for a point  $q$  on  $\partial S$ . Notice a boundary point  $p$  may not be on  $\partial(S \uparrow^* r)$  or  $\partial(S \downarrow^* r)$  if its distances to another point on  $\partial S$  is smaller than  $r$ . The *inside points* are points  $p$  with  $d(p, q) < r$ . The *inside points* are only used for the global verification of  $d(p, S) = r$  so boundary points that are not on offset boundaries are identified.

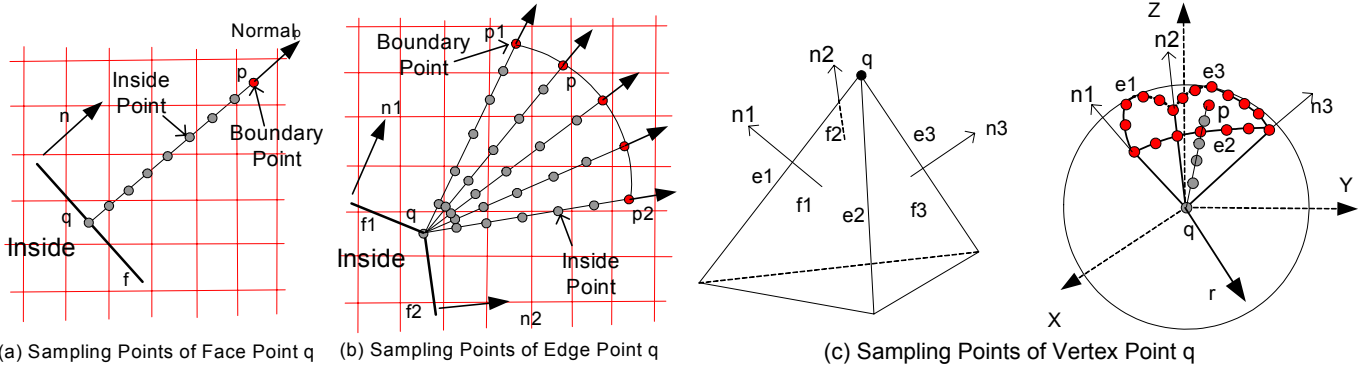


Figure 2. Discrete Offsetting Operations.

The discrete offsetting operations are shown in Figure 2 and explain below.

(1) Faces  $F(S)$ : For each point  $q$  of  $F$ , a boundary point  $p = q + r\mathbf{n}$ , where  $\mathbf{n}$  is the unit normal of  $F$  and  $r$  is the offset distance. Correspondingly, a set of inside points can be generated by sampling line  $pq$  using  $SR_P$  (Figure 2.a).

(2) Edges  $E(S)$ : For each point  $q$  of  $E$ , an arc with radius  $r$  is defined by the great circle in a sphere centered in  $q$ . The starting and ending points of the arc are determined by  $n_1$  and  $n_2$ , where  $n_1$  and  $n_2$  are the unit normals of two neighboring faces  $f_1$  and  $f_2$ . The arc is sampled first into some boundary points. For each boundary point  $p$  in the arc, a set of inside points can be generated by sampling line  $pq$  using  $SR_P$  (Figure 2.b).

(3) Vertices  $V(S)$ : For each point  $q$  of  $V$ , a region formed by a set of arcs is defined on a sphere with radius  $r$  centered in  $q$ . The boundary of the region is determined by  $e_1, e_2, \dots, e_i$  where  $e_1 \sim e_i$  are the neighboring edges of  $q$ . The region is sampled first into some boundary points. For each boundary point  $p$  in the region, a set of inside points can be generated by sampling line  $pq$  using  $SR_p$  (Figure 2.c).

As stated before, a point  $p$  of  $\partial(S \uparrow^* r)$  satisfies  $d(p, \partial S) = r$  if  $p \notin S$ . However some boundary points of  $F \parallel_r^+ \cup E \parallel_r^+ \cup V \parallel_r^+$  may be at a smaller distance to  $\partial S$ . So we further classify all the boundary points into two categories.

(1) *real boundary points*: points  $p$  with  $d(p, q) \geq r$  for all points  $q$  on  $\partial S$ ;

(2) *inside boundary points*: points  $p$  with  $d(p, q) < r$  for some points  $q$  on  $\partial S$ .

By judging points in  $SP_c(F \parallel_r^+) \cup SP_c(E \parallel_r^+) \cup SP_c(V \parallel_r^+)$  and deleting all *inside boundary points*, we can get all the remaining points as the *real boundary points* and use them to construct  $SP_c(\partial(S \uparrow^* r))$ . We then can approximate  $\partial(S \uparrow^* r)$  from the points of  $SP_c(\partial(S \uparrow^* r))$ . The detail approaches of the above process are presented in Section 5 and 6 with error analysis presented in Section 7.

### 3.3 A 2D Example on Offsetting Operations

Before finishing the section, we used a 2D polygon offsetting example to illustrate sets  $\partial S$ ,  $\partial(S \uparrow^* r)$ , and  $F \parallel_r^+ \cup E \parallel_r^+ \cup V \parallel_r^+$  in continuous domain, and related  $SP_c(\partial S)$ ,  $SP_c(\partial(S \uparrow^* r))$ , and  $SP_c(F \parallel_r^+) \cup SP_c(E \parallel_r^+) \cup SP_c(V \parallel_r^+)$  in discrete domain. Offsetting line segments of a closed polygon in 2D domain is actually a simplified version of offsetting polygons of a solid in 3D domain as discussed in this section. The boundary of a 2D polygon  $S$  is  $\partial S = L(S) \cup V(S)$ , where  $L(S)$  and  $V(S)$  refer to the sets of all the line segments and vertices of  $S$  respectively. We can construct set  $L \parallel_r^+$  (positive normal offset of line segments) by displacing each point  $q$  of  $L$  by a distance  $r$  along the unit normal  $n$  at  $q$ , i.e., let  $p = q + rn$  (Figure 3). Similarly we can construct set  $V \parallel_r^+$  (positive normal offset of vertices) by judging each vertex of  $S$ . Suppose the normals of two neighboring line segments ( $l_1, l_2$ ) of a vertex  $q$  are  $n_1, n_2$ . We can construct its positive normal offset as an arc  $p_2 p_1$  determined by  $n_1$  and  $n_2$  in a circle of radius  $r$  and center  $q$  (Figure 3).

For a polygon  $S$  as shown in Figure 3, set  $\partial S$  is all the boundary line segments and vertices. A set of uniform grid cells  $C$  encloses the bounding box of  $S \uparrow^* r$ .  $SP_c(\partial S)$  is all the sample points in  $C$  that  $\partial S$  intersects. We represented points of  $SP_c(V)$  by red dots and points of  $SP_c(L)$  by smaller blue dots in the figure. Set  $\partial(S \uparrow^* r)$  is the positive offset of  $S$  by distance  $r$ , which is shown as thicker lines in Figure 3. Sets  $L \parallel_r^+$  and  $V \parallel_r^+$  are all the normal offsets of  $L(S)$  and  $V(S)$ . Obviously  $L \parallel_r^+ \cup V \parallel_r^+$  is a superset of  $\partial(S \uparrow^* r)$ . Similarly,  $SP_c(\partial(S \uparrow^* r))$ ,  $SP_c(L \parallel_r^+)$ , and  $SP_c(V \parallel_r^+)$  are the sampling points of  $\partial(S \uparrow^* r)$ ,  $L \parallel_r^+$ , and  $V \parallel_r^+$  respectively. And  $SP_c(L \parallel_r^+) \cup SP_c(V \parallel_r^+)$  is also a superset of  $SP_c(\partial(S \uparrow^* r))$ , because some points  $p$  have  $d(p, \partial S) < r$ . We marked *real boundary points* and *inside boundary points* as small yellow dots and small cyan dots respectively in Figure 3. All the *real boundary points* can be used to construct  $SP_c(\partial(S \uparrow^* r))$ . As shown in a magnified view  $A$ , we also add some calculated *vertex boundary points* to  $SP_c(\partial(S \uparrow^* r))$  in order to construct sharp features in the offset model. Finally  $\partial(S \uparrow^* r)$  can be reconstructed from  $SP_c(\partial(S \uparrow^* r))$ .

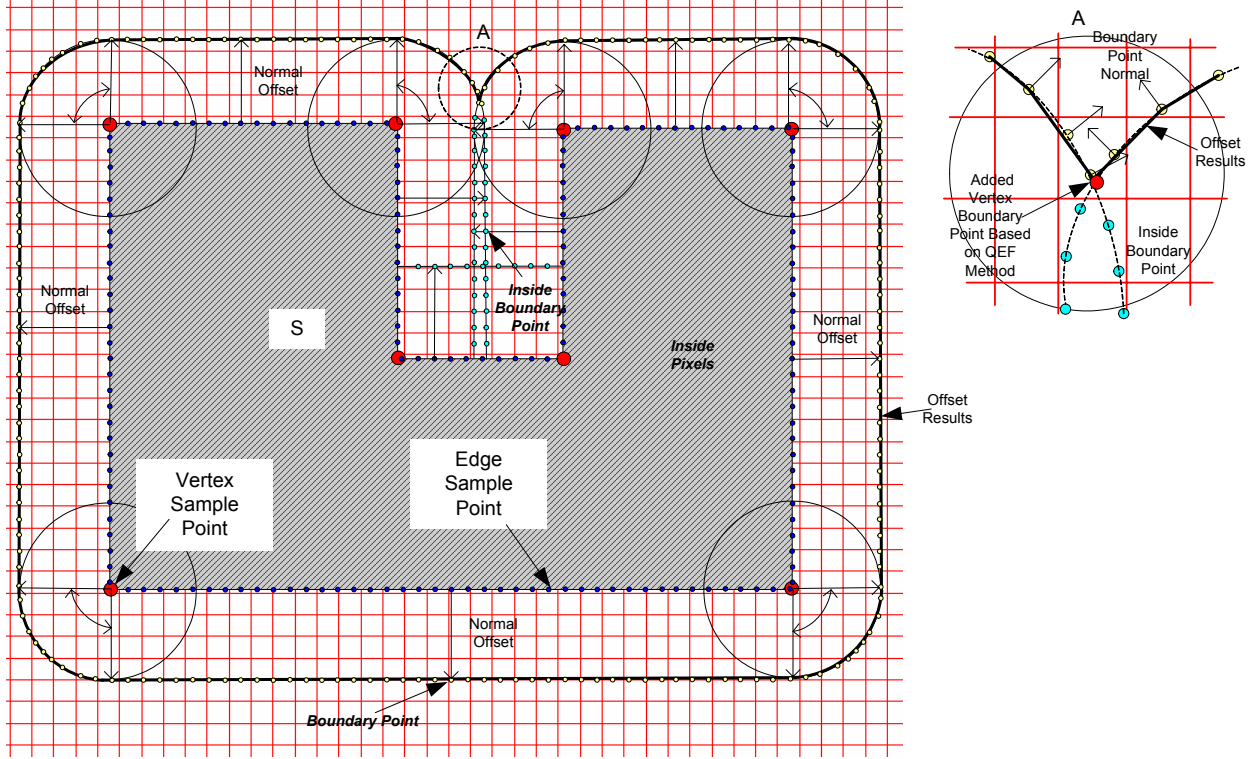


Figure 3. Positive Offset of a 2D Polygon by Using Point Set Approach.

Therefore, the major steps of our method are: (1) generate  $SP_c(F \parallel_r^+) \cup SP_c(E \parallel_r^+) \cup SP_c(V \parallel_r^+)$ ; (2) delete *inside boundary points*; (3) construct  $SP_c(\partial(S \uparrow^* r))$  and  $\partial(S \uparrow^* r)$ . We will discuss them in more details in Section 4, 5 and 6 respectively.

#### 4. Generating Point Sets

Without loss of generality, we can assume that a polygonal model consists entirely of triangular faces, since any non-triangular polygons may be triangulated in a pre-processing phase (O'Rourke 1998). The representation of a triangular surface model is well studied (Watt 2000). In this paper, we further assume an input surface model is pure complex and manifold.

Pfister *et al.* presented surfel (surface element or surface voxel) and related techniques to render complex geometric objects (Pfister, Zwicker et al. 2000). In this paper, we used similar principles in generating sample points from geometric models as those presented in (Pfister, Zwicker et al. 2000) in generating surfels. However, there are two differences to be noticed: (i) we sampled edges and vertices in addition to surfaces; (ii) we sample models from the center of a cell instead of its boundary due to the normal offsetting operations (refer to Section 3). Also the following two kinds of sample points are not needed for our purpose (Rossignac and Requicha 1986):

- (a) Points of concave edges for  $S \uparrow^* r$ , and points of convex edges for  $S \downarrow^* r$ ;
- (b) Points of vertices whose neighboring edges are all concave for  $S \uparrow^* r$ , and points of vertices whose neighboring edges are all convex for  $S \downarrow^* r$ .

In each sample point  $sp_i$ , we record whether it is generated by a vertex, an edge, or a triangle and related geometry elements in the triangular surface model. By connecting the sample points to the original surface model, we can get all the information needed in generating the positive normal offset of  $sp_i$  (Figure 2).

The approach to generate the boundary points from a sample point was presented in Section 3.2. For an offset distance  $r$ , a set of uniform sample points are first generated for a circle and a sphere with radius  $r$  (Du, Gunzburger et al. 2003). We used  $SS_{point}$  as the sample size to discretize the circle and sphere in our implementation. However, depending on the tolerance requirements, the sample size can be different from  $SS_{point}$ . Obviously, with smaller sample size, more sampling points will be generated (proportional to  $\frac{r}{SS_{circle/sphere}}$  for circle and  $(\frac{r}{SS_{circle/sphere}})^2$  for sphere) and the approximation error of the circle and sphere will decrease (proportional to  $\frac{r}{SS_{circle/sphere}}$ ).

In our implementation, we use a set of uniform grid cells to speed up the searching and updating points. So both point lists are actually hash-maps and each point has an index( $ix, iy, iz$ ) to a cell where the point locates. From the hash-map, we can find all the points within a cell by its index; we can also find all the points within some distance to a given point  $p$  by identifying the related cell  $c$  and searching the points within the neighboring cells of  $c$ . A good cell size  $SS_{cell}$ , based on our experience, is  $2*SS_{point}$ , which is used in our implementation.

After presenting the representation method of point sets and their generation approach, we will discuss the approach to process them in the next section.

## 5. Deleting Inside Boundary Points

As shown in Figure 3, some points in the generated set  $SP_C(F \parallel_r^+) \cup SP_C(E \parallel_r^+) \cup SP_C(V \parallel_r^+)$  are *inside boundary points* and need to be deleted from  $SP_C(\partial(S \uparrow^* r))$ . A global judging approach is required since some *inside boundary points* cannot be found based on local geometries. The basic approach we used is to go through each boundary point  $p$  in  $SP_C(F \parallel_r^+) \cup SP_C(E \parallel_r^+) \cup SP_C(V \parallel_r^+)$  to construct a “stick” (line segment  $pq$  as shown in Figure 2), and check all the cells and related boundary points to mark all the points within the “stick”. A brief description of the algorithm is given as follows.

Input: A list of boundary points.

Output: A list of boundary points with all *inside boundary points* deleted.

- (1) Construct cells from the input polygonal meshes with each cell classified as *Inside*, *Outside*, and *Boundary*;
- (2) Initialize all boundary points as *real boundary point*;
- (3) For each boundary point,
- (4)     Construct a stick;
- (5)     For each cell that the stick intersects,
- (6)         if the cell type is *Boundary*, *Inside* for positive offset, or *Outside* for negative offset, continue;
- (7)         Otherwise, check the cell related to the stick;
- (8)         If the cell is totally within the stick, mark it as *Inside* for positive offset, or *Outside* for negative offset;
- (9)         Otherwise, for each boundary point within the cell,
- (10)             if it is marked as *inside boundary point*, continue;
- (11)             Otherwise, if the boundary point is within the stick, mark the point as *inside boundary point*;
- (12) For each boundary point, if it is marked as *inside boundary point*, or located in a cell marked as *Boundary*, *Inside* for positive offset, or *Outside* for negative offset, delete it from the list.

In Step (1), we construct a set of uniform grid cells to speed up the judging process. This is because if a cell is totally within the offset distance  $r$ , we can classify all the points within the cell as *inside boundary point* without further judgments. The cells are represented in volumetric data structure (the same as voxels). Each cell denoted as  $Cell[i][j][k]$  has only a *Type* value which can be *Inside*, *Outside*, or *Boundary* (Kaufman, Cohen et al. 1993). Indices  $i, j, k$  are integer numbers to specify the position of the cell along  $x, y, z$  axis respectively. Various volumetric data structures have been studied. The simplest one is just to use an array. To reduce the memory requirements for small cell size, octree data structure can be utilized (Szeliski 1990). Another approach to reduce the memory requirement is using a hash-map to record only the cells within  $r$  distance from the polygonal meshes. However, this will require us to construct a two-side “stick” and process cells and points on both sides. Since each cell only takes two bits in our application, we used a simple array  $Cell[X_{Size} * Y_{Size} * Z_{Size}]$  in our implementation.

For a boundary point  $p$ , we know the related sample point  $q$  as  $p - normal(p) * r$ . Therefore in Step (4), a “stick” for  $p$  can be constructed as  $qp'$ , where  $p' = p + normal(p) * (r + 0.8 * SS_{Point})$ . Notice we only need one side of the stick since the cells in other side has been classified as *Boundary*, *Inside* for positive offset, or *Outside* for negative offset. We elongate the “stick” by  $0.8 * SS_{Point}$  to compensate for possible errors due to discrete effects. For example, since we discretize a line  $qq_1$  into two sample points as shown in Figure 4, the continuous offset line  $pp_1$  is also discretized as two boundary points,  $p$  and  $p_1$ . In Step (5), we only check the cells that the “stick” intersects. So it is possible that line  $pp_1$  intersects a cell while two sticks related to  $p$  and  $p_1$  will miss the cell. The error region related to the case is also shown in the figure. Any boundary points within the region can be mistakenly classified if we only use  $qp$  as the “stick”. Therefore, the “stick” needs to be elongated to  $p'$  instead. The maximum length of  $qp'$  such that  $qp'$  and  $q_1p_1'$  will not intersect a cell while line segment  $pp_1$  will intersect it is  $\frac{SS_{Point}}{2}$  in the 2D case and  $\frac{\sqrt{2}SS_{Point}}{2} \approx 0.8 * SS_{Point}$  in the 3D case.

In Step (11), a boundary point  $p$  is used to judge if another candidate boundary point  $p_{test}$  is within the distance  $r$  from sample point  $q$  related to  $p$ . For all the grids that  $pq$  intersects, if a grid is entirely within the distance  $r$  from  $q$ , we can mark the grid and delete all boundary points within the grid later. If the grid is not entirely inside, we need to iterate each boundary point within the grid to judge if it is within the distance  $r$  from  $q$ . We use a hash-map data structure to associate points with grids.

In order to minimize the approximation error due to discrete effects, we use different judging approaches for different types of points  $q$  (face, edge, or vertex).

- (i)  $q$  is a vertex sample point: We use a sphere to judge  $p_{test}$ . That is, (a) if  $\|qp_{test}\| < r$ ,  $p_{test}$  is inside the offset distance; (b) otherwise,  $p_{test}$  is outside the offset distance. There is no approximate error related to this approach since no discretization is needed for a vertex.
- (ii)  $q$  is a edge sample point related to edge  $e$ : Because of the discretization of a continuous edge into sample points, each point  $q$  actually represents an edge segment  $[q - \frac{SS_{Point}}{2}, q + \frac{SS_{Point}}{2}]$ . Therefore, suppose point  $q'$  is the projection of  $p_{test}$  onto the edge  $e$ . (ii.a) if  $\|q'q\| \leq \frac{SS_{Point}}{2}$ , we use a cylinder to judge if  $p_{test}$  is inside or outside the offset distance; (ii.b) if  $\|q'q\| > \frac{SS_{Point}}{2}$ , we use a sphere to judge if  $p_{test}$  is inside or outside the offset distance. The upper bound on the error of our approach is the difference between a sphere and a

cylinder, both with radius  $r$  and center  $q$  within the range  $[q - \frac{SS_{Point}}{2}, q + \frac{SS_{Point}}{2}]$ , which is

$$\left(\frac{r}{SS_{Point}} - \sqrt{\left(\frac{r}{SS_{Point}}\right)^2 - \frac{1}{4}}\right) \cdot SS_{Point}$$

(iii)  $q$  is a face sample point related to face  $f$ . Because of the discretization of a continuous face into sample points, each point  $q$  actually represents a face region centered at  $q$  with radius  $\frac{SS_{Point}}{2}$ . Therefore, suppose point  $q'$  is the projection of  $p_{test}$  onto the face  $f$ . (iii.a) if  $\|q'q\| \leq \frac{SS_{Point}}{\sqrt{2}}$ , we use an offset plane to judge if  $p_{test}$  is inside or outside the offset distance; (iii.b) if  $\|q'q\| > \frac{SS_{Point}}{2}$ , we use a sphere to judge if  $p_{test}$  is inside or outside the offset distance. The upper bound on the error of our approach is the difference between a sphere with radius  $r$  and center  $q$  and a tangent plane at  $p$  within the distance range of  $\frac{SS_{Point}}{\sqrt{2}}$  from  $p$ , which is

$$\left(\frac{r}{SS_{Point}} - \sqrt{\left(\frac{r}{SS_{Point}}\right)^2 - \frac{1}{2}}\right) \cdot SS_{Point}$$

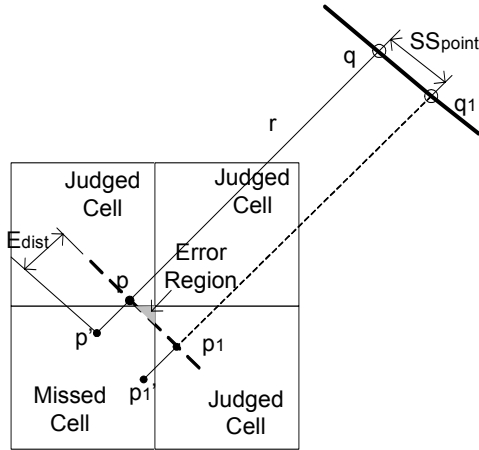


Figure 4. Elongating the “Stick” of a Boundary Point due to Discrete Effects.

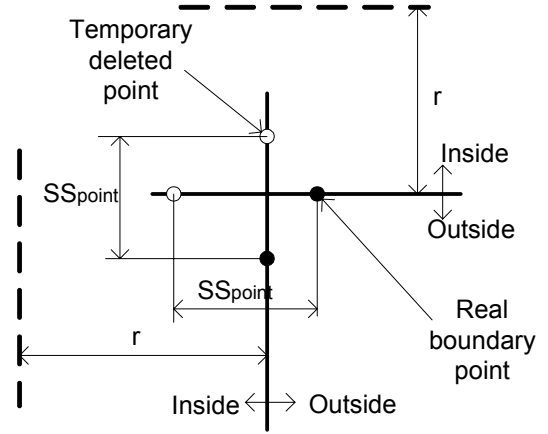


Figure 5. An Intersection Point and related Temporary Deleted Points.

## 6. Constructing Boundary from Boundary Points

The *real boundary points* generated in Section 5 are part of set  $SP_C(\partial(S \uparrow^* r))$ , which we will use to construct  $\partial(S \uparrow^* r)$ . However, it should be noticed that points related to sharp features of the offset geometry (vertices and edges) are still missing (refer to View A in Figure 3). These points will not be generated even if we use very small sample point size, although the approximation error will be smaller. In our method, we explicitly calculate a set of points for edges and vertices of the offsetting meshes and add them to  $SP_C(\partial(S \uparrow^* r))$  for reconstructing  $\partial(S \uparrow^* r)$ .

A sharp feature (edge or vertex) of the offset geometry means two or more geometry elements intersect at the feature. Therefore, there must be a special set of *inside boundary points* around the intersection points whose distances from  $\partial S$  are within range  $(r - SS_{Point}, r)$ . We called these points *temporary deleted points*. As shown in Figure 5, two *temporary deleted points* must be within the distance range  $(r - SS_{Point}, r)$  from  $\partial S$  since the distance between a *real boundary point* and an *inside boundary point* is  $SS_{Point}$ . This property can be used to distinguish *temporary deleted points* from *inside boundary points*. In the algorithm presented in Section 5, we use  $r - SS_{Point}$  instead of  $r$  as inside/outside criteria in Step (8), and mark a point as *temporary*

*deleted point* if the distance from it to the checked boundary point is  $(r-SS_{Point}, r)$  in Step (11); finally in step (12), we use all the points marked as *temporary deleted points* to form a new point list after removing them from the boundary point list.

For each *temporary deleted point*, we search its neighboring *temporary deleted points* within distance  $\sqrt{3}SS_{point}$ . Notice since the point list is actually a hash-map, we can easily search the neighboring points based on its cell index. After finding all neighboring *temporary deleted points* with different normals, we use an approach based on quadratic error function (QEF) (Garland 1999) to calculate the intersection point. That is, a vertex  $v$  is an intersection point if it minimizes a quadratic error function defined as  $E[v] = \sum_i (n_i \bullet (v - p_i))^2$ , where the pairs  $p_i, n_i$  are the position and offset normal of a *temporary deleted point*. The point dimension of  $v$  is the number of non-zero eigen-values in the QEF calculation (1=plane, 2=edge, 3=vertex). Before adding  $v$  to the boundary point list, we need to verify that (1)  $v$  is not a duplicated point which is added before; (2)  $v$  is not inside the “stick” of any *temporary deleted points*. We define these calculated intersection points as *added boundary points*, which form a complete set  $SP_c(\partial(S \uparrow^* r))$  with *real boundary points*.

The generated points can be displayed directly by using point-based displaying approaches (Pfister, Zwicker et al. 2000). Techniques were also developed for point-sampled geometry, such as up-sampling and simplification of points (Pauly 2003). All the approaches can be used to directly process points in  $SP_c(\partial(S \uparrow^* r))$ .

The approach to reconstruct a surface model from a set of points was studied intensively in 3D scanning applications. Grid based reconstruction methods, such as the Dual-Contour method (Ju, Losasso et al. 2002), use a set of uniform grids marked as *inside/outside* to generate quads. A well known problem for grid based reconstructing method is all the features that are smaller than a grid size will disappear in the constructed models. A possible improvement is to use some non-uniform grids based on octree to recursively refine the grids for small features. Some grid-less reconstruction method, such as Ball-Pivoting algorithm (Bernardini, Mittleman et al. 1999), can construct triangles based on rolling a certain size ball along the outside of points. However, it is difficult to assign a good size of the ball for our application and several iterations may be required. In this study, we used the Dual-Contour method in reconstructing polygonal meshes.

In many cases, the generated polygonal meshes may be too dense especially for small sample point size. Research on polygonal mesh simplification (Garland 1999) can also be used to reduce the mesh size of the generated models.

After discussing the major steps of our method, we present the theoretical analysis on error, performance and memory requirements in the next section.

## 7. Point-based Method and Its Analysis

By putting all pieces together, the steps of our point-based offsetting method are listed as follows.

- (1) Determine sample point size  $SS_{Point}$  based on error requirement  $\epsilon$  and the smallest feature size of the input solid model;
- (2) Generate a set of sample points  $SP_{circle}$  and  $SP_{sphere}$  as the sampling points of a circle and a ball with radius  $r$ ;

- (3) Load in meshes of the input solid model and generate a set of sample points  $SP_{model}$  based on  $SS_{Point}$ ;
- (4) Iterate each point in  $SP_{model}$ , generate a set of boundary point  $BP$  based on the sample point type (vertex, edge or face),  $SP_{circle}$  and  $SP_{sphere}$ ;
- (5) Iterate each point in  $BP$ , delete all *inside boundary points* from  $BP$  and construct a *temporary deleted point list*;
- (6) Calculate a set of *added boundary points* for edges and vertices of the offset model based on the *temporary deleted points*;
- (7) If boundary representation is required, construct a surface model from boundary points  $BP$ ;
- (8) If necessary, simplify the generated surface model;

The whole processes are also illustrated in Figure 1. Our theoretical analyses of the method are presented in the remaining of the section.

### 7.1 Error Analysis

Our method of generating an offset model for an input surface model is an approximation scheme. There are four approximation steps in the process; their corresponding error analysis follows.

- (i) Input Surface Model  $\rightarrow$  Sample Points (Step 3): Each sample point is taken from the boundary of the input surface model and represented in floats. The discontinuity of the surfaces is also captured since edges and vertices are sampled individually. The approximation error in this step is very small and can be omitted.

- (ii) Sample Points  $\rightarrow$  Boundary Points (Step 4): Each boundary point is taken from an arc or a sphere with radius  $r$  and represented in floats. So the approximation error in this step is very small and can be omitted.

- (iii) Deleting Boundary Points which are within offsetting distance  $r$  (Step 5): The approximation error of a vertex sample point by a sphere is accurate; the approximation error of an edge sample point by a cylinder is less than  $(\frac{r}{SS_{Point}} - \sqrt{(\frac{r}{SS_{Point}})^2 - \frac{1}{4}}) \cdot SS_{Point}$ ; the approximation error of a face sample point by a

segment plane is less than  $(\frac{r}{SS_{Point}} - \sqrt{(\frac{r}{SS_{Point}})^2 - \frac{1}{2}}) \cdot SS_{Point}$ . So if  $\frac{r}{SS_{Point}} = 4$ , the maximum error we will

misjudge a boundary point inside or outside the  $r$  distance is less than  $0.06 \cdot SS_{Point}$ .

- (iv) Boundary Points  $\rightarrow$  Surface Model (Step 7): In the offset models, only three kinds of geometries exist. They are faces related to  $F(S)$ , cylinders related to  $E(S)$  and spheres related to  $V(S)$ . The error generated in this step is geometry-dependent. We used a grid-based method which converts multiple boundary points into a single grid point. Therefore, any features that are smaller than a grid size will disappear. However, if all features are comparably bigger than the grid size and grid points are selected from boundary points to make the offset meshes uniform, the reconstructed meshes will be accurate for offset faces; the maximum error for cylinders in the offset meshes is

$(\frac{r}{SS_{Grid}} - \sqrt{(\frac{r}{SS_{Grid}})^2 - \frac{1}{4}}) \cdot SS_{Grid}$  by using the linear interpolation of polygonal meshes; the maximum error

for spheres in the offset meshes is  $(\frac{r}{SS_{Grid}} - \sqrt{(\frac{r}{SS_{Grid}})^2 - \frac{1}{2}}) \cdot SS_{Grid}$ . So if  $SS_{Grid} = 2 \cdot SS_{Point}$  and  $\frac{r}{SS_{Point}} = 4$ ,

the maximum error in this approximation is  $0.26 \cdot SS_{Point}$ .

By considering all the steps, the error bound of our approach is

$(\frac{r}{SS_{Point}} - \sqrt{(\frac{r}{SS_{Point}})^2 - \frac{1}{2}}) \cdot SS_{Point} + (\frac{r}{SS_{Grid}} - \sqrt{(\frac{r}{SS_{Grid}})^2 - \frac{1}{2}}) \cdot SS_{Grid}$ . Obviously the approximation error will be smaller

if we reduce grid size. Also notice a feature smaller than  $SS_{Grid}$  in the offset results will disappear since we use a grid-based surface reconstruction method. A grid-less reconstruction method is being explored to remove this limitation.

## 7.2 Performance Analysis

Suppose  $S$  is the maximum extent of the input surface model. Our method has the following computational complexity:

Steps (1), (2), (3) (Sample Point Generation): The scan conversion algorithm goes through each triangle to check if it intersects scan lines. Since we only need to calculate the scan lines within the bounding box of a triangle, the computational time is  $O(N_{Triangle} \frac{S^2}{SS_{Point}})$ . The computational time to sample points and edges are  $O(N_{Vertex})$  and  $O(N_{Edge} \frac{S}{SS_{Point}})$  respectively, which are generally much smaller than the time of sampling triangles.

Step (4), (5), (6) (Boundary Point Generation and Processing): The time to process a sample point is constant for constant  $\frac{r}{SS_{Point}}$ , although different constant factors are related to various types of offset points (vertices, edges or faces). Therefore the computational time is proportional to the number of sample points, which is  $O(N_{Triangle} \frac{S^2}{SS_{Point}} + N_{Edge} \frac{S}{SS_{Point}} + N_{Vertex})$ ;

Step (7), (8) (Isosurface Extraction and post-processing): The computational time is proportional to the number of grids, which is  $O(\frac{S^3}{SS_{Point}^3})$ .

Therefore the total Complexity of our method is  $O(\frac{S^3}{SS_{Point}^3} + N_{Triangle} \frac{S^2}{SS_{Point}})$ .

## 7.3 Memory Analysis

The main memory requirements of our algorithms are an array of grid cells, a list of sample points and a list of boundary points: (i) The size of the cell array is  $(\frac{S_x S_y S_z}{SS_{Point}^3}) \times \frac{1}{4}$  bytes since each cell only needs two bits; (ii)

The size of the sample point list depends on the surface area of the input model. It is well known that the surface area of a model is proportional to  $Size^2$  while its volume is proportional to  $Size^3$ . Therefore the size of the sample list is estimated as  $\frac{Area}{SS_{Point}^2} \times 17$  bytes since each sample point takes 17 bytes in our

implementation; (iii) similarly, the size of boundary point list is  $\frac{Area}{SS_{Point}^2} \times 30$  bytes since each boundary point takes 30 bytes in our implementation.

After theoretically analyzing the accuracy and performance, we present the experimental results for various test cases in the next section.

## 8. Test Results

We used C++ programming language with *Microsoft Visual C++* compiler to implement the presented algorithms. In this section, we present our test results by highlighting the accuracy of the results and the performance of the algorithms.

### 8.1 Experimental Accuracy

As analyzed in Section 4, the accuracy of our approach depends on the sample point size  $SS_{Point}$  and radius  $r$ . To test the accuracy of our method, we used a sphere, a cube, a cylinder and a pyramid since their offset results are analytically known and can be constructed easily. We took the size of all the input models as

25x25x25 mm and the offset distance as -2.5 mm (offset inside). Obviously the offset results should be a sphere, a cube, and a cylinder with the size of 20x20x20 mm, and a pyramid with the size of 16.91x16.91x16.91 mm respectively. Three different sample point sizes were tested from 0.156 to 0.625 mm. We used commercial software (*Geomagic Qualify 7* from *Raindrop Geomagic Inc.*) to compare the offset results generated by our algorithms and the ideal offset results constructed in CAD software. The offset accuracy based on orthogonal distance between two comparing models is given in Table 1. Obviously, the error of the offset results decreases when the sample point size  $SS_{point}$  decreases. Also since we explicitly calculated a set of boundary points for sharp features of the offsetting meshes, the approximation errors are very small for a model without any small features (e.g. a cube).

**Table 1. Offset Accuracy Test Results.**

Models	Offset (mm)	$SS_{point}$ (mm)	Polygonal Mesh Error (mm)		
			Maximum Distance	Average Distance	Standard Deviation
Sphere	-2.5	0.625	0.00068	0.00030	0.00015
	-2.5	0.313	0.00040	0.000075	0.000075
	-2.5	0.156	0.00025	0.000075	0.000050
Cube	-2.5	0.625	0.0	0.0	0.0
	-2.5	0.313	0.0	0.0	0.0
	-2.5	0.156	0.0	0.0	0.0
Cylinder	-2.5	0.625	0.00040	0.00015	0.00010
	-2.5	0.313	0.00060	0.000050	0.000050
	-2.5	0.156	0.00038	0.000050	0.000050
Pyramid	-2.5	0.625	0.0	0.0	0.0
	-2.5	0.313	0.00085	0.000025	0.00013
	-2.5	0.156	0.0017	0.000025	0.00018

## 8.2 Algorithm Performance

Beside accuracy tests, we also performed a number of tests on different geometries by using various offset distances and sample point sizes. All the tests were done in a PC with a 1.7 GHz *Pentium Xeon* processor and 2GB *DRAM* running *Windows XP*. As analyzed in Section 7, the main memory requirements of our approach are two point lists and a grid cell array. The three main steps of our approach are (a) sample point generation (Step 1, 2, 3); (b) boundary point generation and processing (Step 4, 5, 6); and (c) offset surface model generation (Step 7). Correspondingly we present the test results for three solid models on memory size and running time for negative offsets in Table 2. The test results for positive offsets are similar to them and therefore omitted. The screen captures of the generated offset results for Stanford bunny, dragon, and an engine block are shown in Figure 6, Figure 7, and **Error! Reference source not found.** respectively. Accordance to the analysis in Section 7.3, the required memory for cell array increase significantly when point sample size decreases. As discussed in Section 5, using octree data structure to replace the cell array can greatly improve the memory requirement. Offset surface model generation from boundary points takes a big portion of the running time. A grid-less reconstruction approach is being explored for further improvement on algorithm performance.

Table 2. Algorithm Performance of our Approach for Different Cases.

Models	Size	Offset (mm)	Point Sample Size (mm)	Memory (MB)		Running Time (Second)			
				Point Lists	Cell Array	Sample Point Generation	Boundary Point Generation and Processing	Offset Surface Model Generation	Total
Bunny	X: 0.934 Y: 0.926	-0.5	0.13	27.6	13.8	5	25	25	55
		-1.0	0.20	18.3	4.9	3	35	11	49

(Tri#: 69,670)	Z: 0.724	-1.5	0.25	16.3	2.5	3	51	9	63
Dragon (Tri#: 712,073)	X: 1.229	-0.5	0.13	128	12.7	25	484	109	618
	Y: 0.867	-1.0	0.20	100	4.7	31	440	39	510
	Z: 0.550	-1.5	0.25	99	2.5	27	711	50	788
4CylBlock (Tri#: 163,374)	X: 4.242	-0.25	0.15	492	613	80	293	1373	1746
	Y: 3.486 Z: 2.341	-0.50	0.20	348	189	48	223	447	718

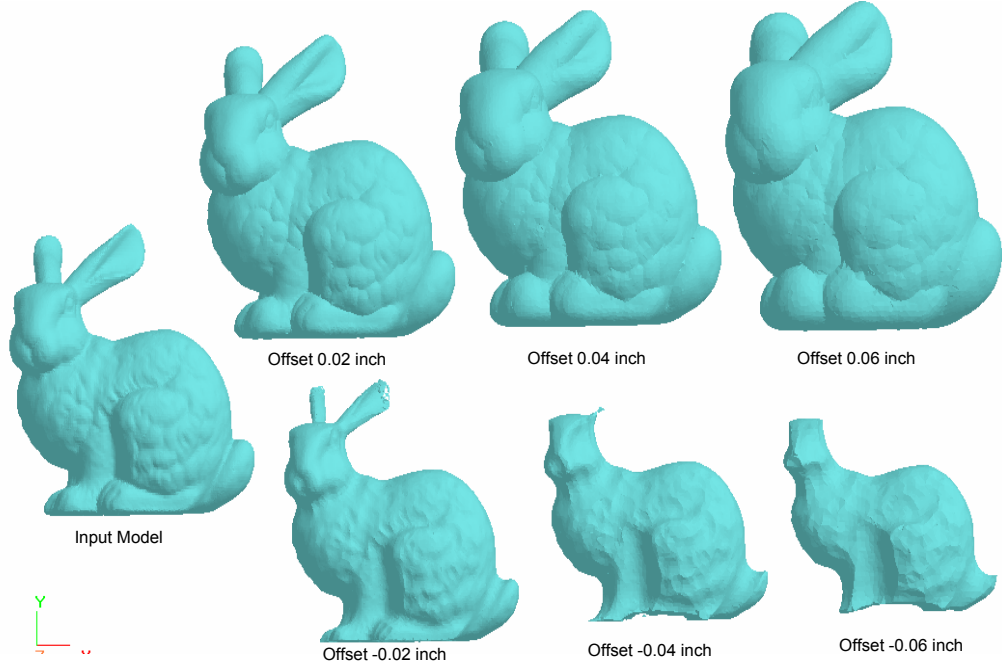


Figure 6. Screen Capture of the Offset Results for Bunny.

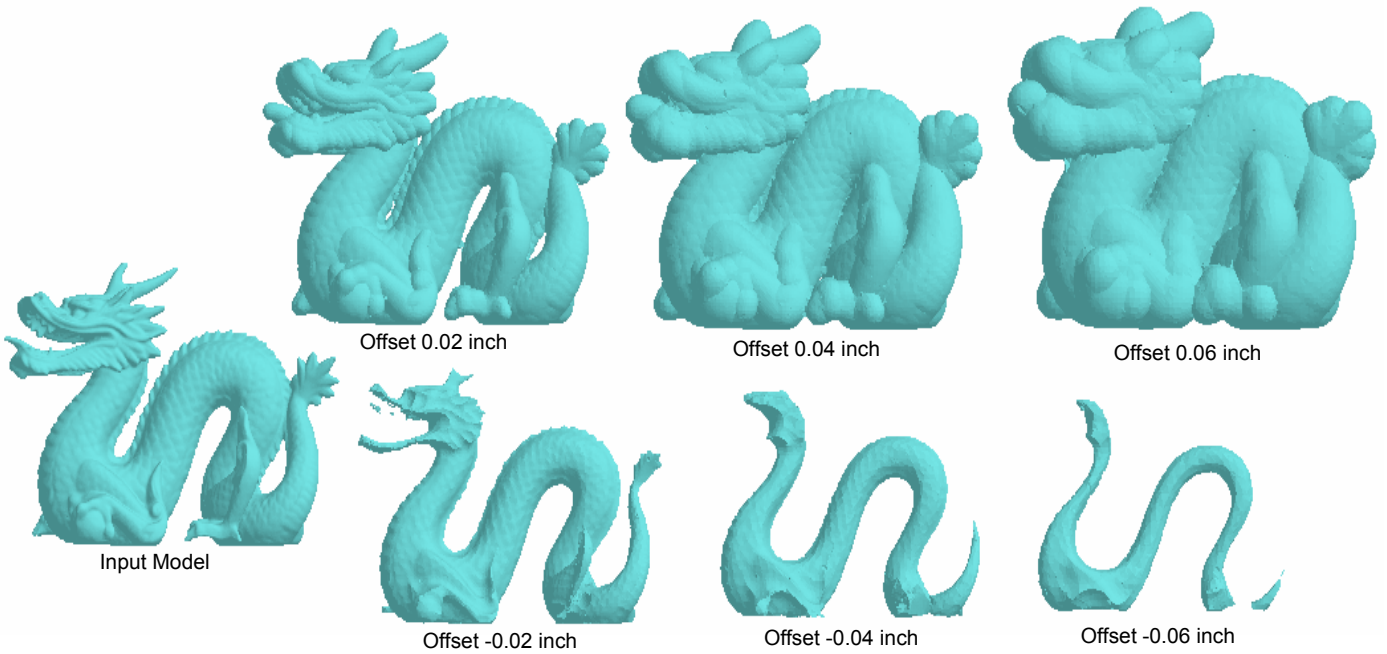


Figure 7. Screen Capture of the Offset Results for Dragon.

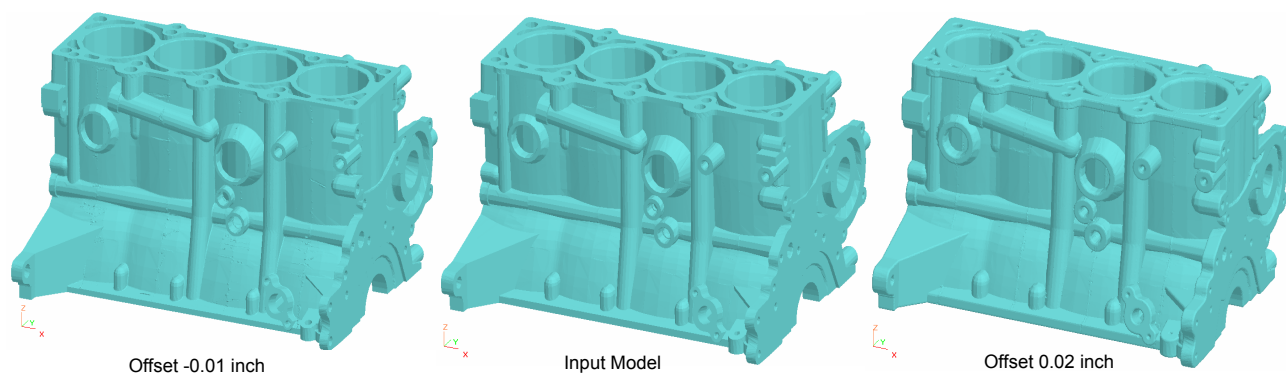


Figure 8. Screen Capture of the Offset Results for an Engine Block.

The generated offset results can be directly used in generating slicing contours for a rapid prototyping machine or tool paths for a CNC machine. For additive processes, negative offsets are used based on a laser spot size; while positive offsets based on a cutting tool size are used for subtractive processes. An enlarged engine block model can be used in the casting process. This ensures the following machining processes can have enough allowances for final part geometry.

## 9. Conclusion and Future Work

We propose a hybrid point-based offsetting method for 2D/3D solid models. It samples the boundary, generates surfels in the vicinity of the model, uses a spatial grid to eliminate candidate samples that are too close to the original surface and restores the offset boundary through interpolation. The offset approach is general for both inside and outside offset operations and can be used for any offset distance. The input geometries have no limitations on their complexity. We presented algorithms that are straightforward to implement. A relation between the offset accuracy and sample point size is presented to ensure a tight error bound of the offset results. We tested different geometries with various values of sample point size and offset distances. Experimental results show that the accuracy and performance of our algorithms are satisfactory.

Different geometry representations (voxels, points and surfaces) have their unique advantages. Linking them together into a hybrid representation provides a promising direction to solve difficult geometry problems. There are many areas for future work. We would like to further improve the accuracy and performance of our algorithms. We would also like to investigate some grid-less techniques for surface reconstructing. Finally, we would like to extend our method to non-uniform offsetting and general Minkowski operations.

## References

- Adams, B. and P. Dutre (2003). Interactive Boolean Operations on Surfel-Bounded Solids. Proceedings of ACM SIGGRAPH, San Diego, CA.
- Allen, S. and D. Dutta (1998). "Wall Thickness Control in Layered Manufacturing for Surfaces with Closed Slices." Computational Geometry: Theory and Application **10**: 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.
- Du, Q., M. D. Gunzburger, et al. (2003). "Voronoi-based Finite Volume Methods, Optimal Voronoi Meshes, and PDEs on the Sphere." Computer Methods in Applied Mechanics and Engineering **192**: 3933-3957.
- Farouki, R. T. (1985). "Exact Offset Procedures for Simple Solids." Computer-Aided Design **2**(4): 257-279.
- Foley, J. D., A. v. Dam, et al. (1995). Computer Graphics: Principles and Practice in C, Addison-Wesley.

Friskens, S. F., R. N. Perry, et al. (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. Ph.D. Dissertation, Department of Computer Science. Carnegie-Mellon University, Pittsburgh, PA.

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

Gurbuz, A. Z. and I. Zeid (1995). "Offsetting Operations via Closed Ball Approximation." Computer-Aided Design **27**(11): 805-810.

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

Kaufman, A., D. Cohen, et al. (1993). "Volume Graphics." IEEE Computer **26**(7): 51-64.

Kim, Y. J., G. Varadhan, M. C. Lin, D. Manocha (2003). "Fast Swept Volume Approximation of Complex Polyhedral Models." ACM Symposium on Solid Modeling and Applications 2003, 11-22.

Lam, T. W., K. M. Yu, et al. (1997). "Octree Reinforced Thin-Shell Rapid Prototyping." Journal of Materials Processing Technology **63**: 784-787.

Maekawa, T. (1999). "An Overview of Offset Curves and Surfaces." Computer-Aided Design **31**(3): 165-173.

McMains, S., J. Smith, et al. (2000). Layered Manufacturing of Thin-Walled Parts. ASME Design Engineering Technical Conference, Baltimore, Maryland.

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

O'Rourke, J. (1998). Polygon Triangulation. Computational Geometry in C, Cambridge University Press.

Pauly, M., R. Keiser, et al. (2003). Shape Modeling with Point-Sampled Geometry. Proceeding of ACM Siggraph, San Diego, CA.

Pauly, M. (2003). Point Primitives for Interactive Modeling and Processing of 3D Geometry. Ph.D. Dissertation, Department of Computer Science. Swiss Federal Institute of Technology at Zurich.

Pfister, H., M. Zwicker, et al. (2000). Surfels: Surface Elements as Rendering Primitives. ACM-SIGGRAPH, New Orleans, Louisiana.

Pham, B. (1992). "Offset Curves and Surfaces: A Brief Survey." Computer-Aided Design **24**(4): 223-229.

Qu, X. and B. Stucker (2003). "A 3D Surface Offset Method for STL-format Models." Rapid Prototyping Journal **9**(3): 133-141.

Ritter, G. W., Joseph (2001). Handbook of Computer Vision Algorithms in Image Algebra, CRC Press.

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

Sethian, J. A. (1996). Level Set Methods and Fast Marching Methods. Cambridge University Press.

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.

Szeliski, R. (1990). Real-Time Octree Generation from Rotating Objects. Cambridge, MA, DEC-Cambridge Research Laboratory.

Watt, A. (2000). 3D Computer Graphics, Addison-Wesley.