Geometric Analysis and Computation Using Layered Depth-Normal Images for Three-Dimensional Microfabrication

Tsz-Ho Kwok, Yong Chen
Department of Industrial and Systems Engineering
University of Southern California

Charlie C. L. Wang
Department of Mechanical and Automation Engineering
The Chinese University of Hong Kong

1. Introduction

Additive manufacturing (AM), also know as three-dimensional (3D) printing, is a fabrication method using the principle of material accumulation, usually in layers. AM is a direct manufacturing process that can fabricate parts directly from digital models without part-specific tools or fixtures. The digital models used in the AM processes are usually created by Computer-Aided Design (CAD) tools or 3D scanners (see Figure 1). In order to transfer information among different technology platforms, a *de facto* standard for such digital models is the STL file format. For examples, most CAD systems can export their native CAD formats to STL, and the boundary representation (B-Rep) of implicit representations can be sampled into STL. In the reverse engineering using various types of 3D scanners, the scanned sampling points or the medical imaging data can also be triangulated into the STL format.

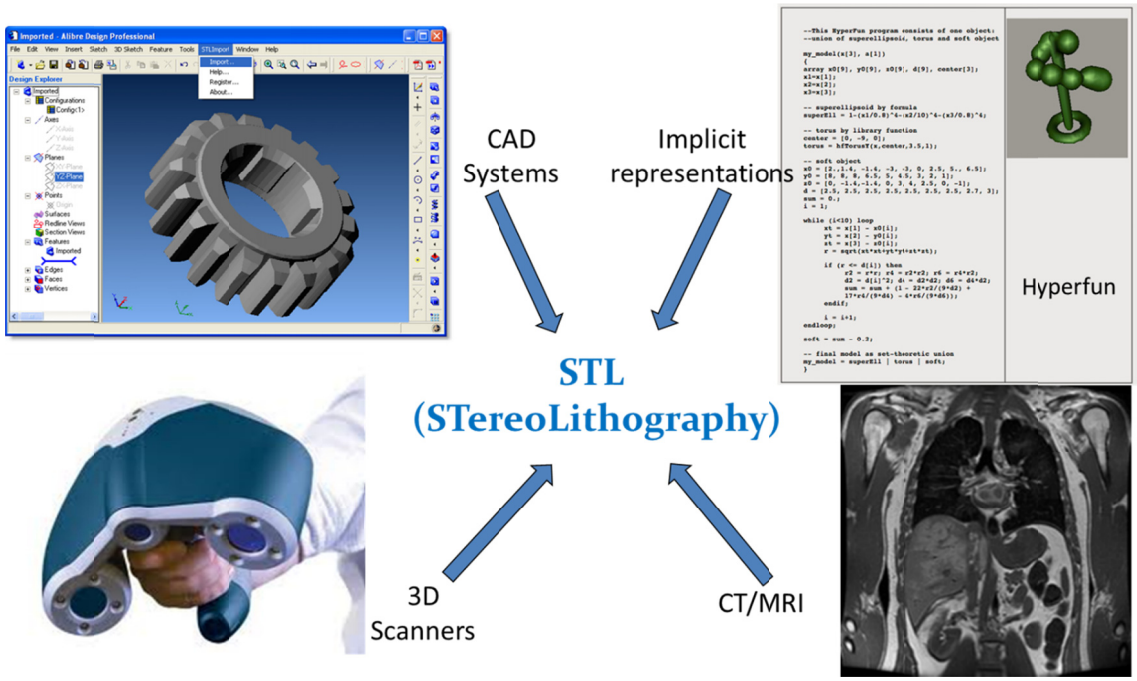


Figure 1. Digital data created by different technologies use the STL file format for additive manufacturing.

STL (STereoLithography), also known as Standard Tessellation Language, is a file format that was originally developed for 3D Systems's stereolithography systems. In a STL

file, a set of triangular facets are described to define the shape of a digital model. Each triangular facet contains three vertices as boundaries and a unit normal (in the order of the right-hand rule). A facet example is shown in Figure 2. The STL file format is simple and easy to generate. Currently most AM machines and 3D printers can accept it as the input of digital models.

facet normal n_x n_y n_z
  outer loop
    vertex $v1_x$ $v1_y$ $v1_z$
    vertex $v2_x$ $v2_y$ $v2_z$
    vertex $v3_x$ $v3_y$ $v3_z$
  endloop
endfacet

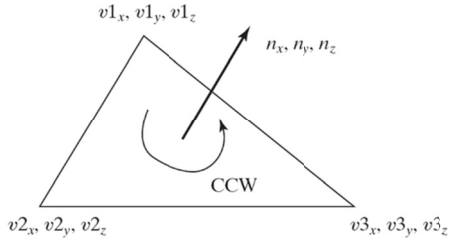


Figure 2. A triangular facet defined in a STL file.

To fabricate a STL model, a sequence of process planning steps is required, which may vary from different AM processes (e.g. powder-based, liquid-based, filament-based, or sheet-based fabrication processes). For liquid-based microfabrication processes such as two-photon polymerization (TPP), the process planning steps typically include changing model's orientation and position, slicing, generating support structures, and layer tool path planning (refer to Figure 3). First, the input STL model needs to be positioned inside the available working envelope, and be oriented properly for purposes such as reducing the needed support structures or minimizing fabrication time. The support structures are generated to enable the building of shapes with overhangs or cantilevered sections, where there is no previous layer for accumulating new materials on. Accordingly, slicing is performed by intersecting the model with a set of parallel planes along the *Z*-axis to obtain the contour information of each layer. Based on the sliced contours, the layer tool paths, i.e., the sequence of laser drawing, are computed for goals such as minimizing the shrinkage-related deformation and reducing the laser drawing time, etc. After the tool paths have been planned, numerical control (NC) machine code can be generated and eventually interpreted by the machine controller during the physical fabrication process.

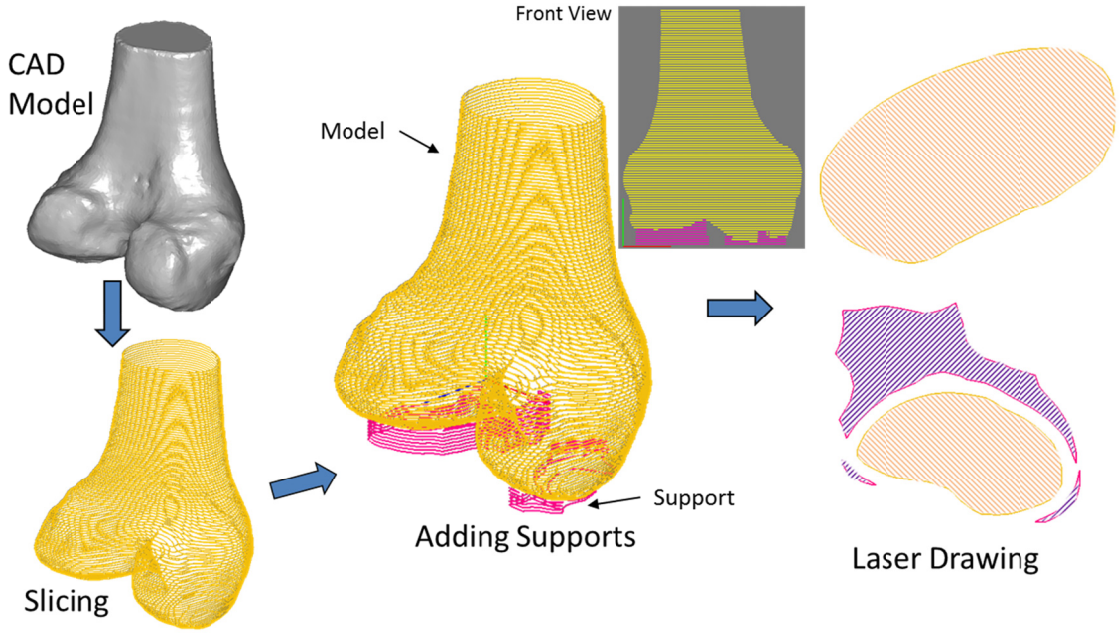


Figure 3. Main process planning steps in the liquid-based micro-stereolithography processes.

Geometric analysis and computation are critical for the microfabrication processes during such digital model preparation and process planning steps. For example, an input STL file can be successfully fabricated only if the defined geometry is valid, i.e. manifold, water-tight, and no self-intersections. If the input model is invalid, the built object will have undesired defects, and the building process may fail in many cases. Hence a geometric operator to perform mesh verification and repairing is needed to convert an input polygonal model into a valid STL file. As another example, a fabrication tool (e.g., a laser beam in TPP) has certain shape and size. Hence an offset CAD model is needed in the tool path planning to compensate the given tool size. Offsetting a solid S by a distance r into a grown or shrunken version of S has been well defined for point sets in *Euclidean* space $\boldsymbol{E}^2$ or $\boldsymbol{E}^3$ [1]. Figure 4 shows an offsetting example. The offsetting results of an octa-flower model based on different tool sizes are shown in the figure. Two pairs of the offset models are shown together to illustrate the uniform distance that can be achieved. The related shelled parts can also be used to reduce the material usage in the microfabrication process.

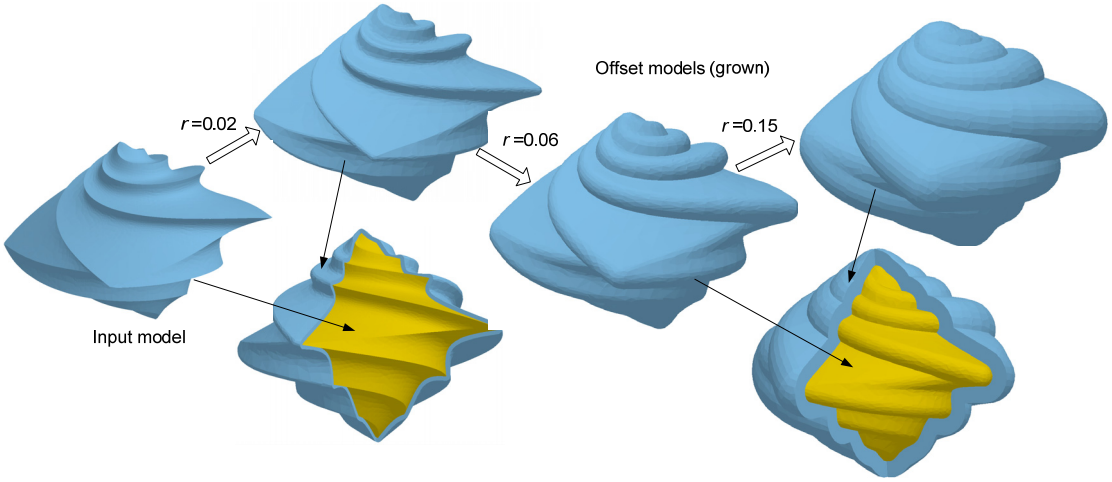


Figure 4. An example of multiple offsetting operations.

Geometric operators such as mesh regulation and uniform offsetting belong to a class of geometric analysis and computation problems that are important for three-dimensional microfabrication and other computer-aided design and manufacturing applications. To better enable future TPP process development, this chapter introduces a novel geometric computation framework based on a new geometric representation named *Layered Depth Normal Images* (LDNIs). The LDNIs is an extension of the *ray representation* (ray-rep) in solid modeling [2]. Based on a well-structured discrete sampling approach, a structural set of LDNIs consists of x-, y-, and z-LDNI along X, Y, and Z axis respectively. The three images are located to let the intersections of their rays form the $w_X \times w_Y \times w_Z$ nodes of uniform grids in $\Re^3$. A LDNI in each axis is a two-dimensional image with each pixel represented by a sequence of four-components nodes $P(d, n_x, n_y, n_z)$, where d specifies the depth from an intersection point P to the viewing plane, and $N_P(n_x, n_y, n_z)$ is the surface normal at P. Therefore, the shape of a solid model can be sparsely encoded into a set of sampling points, which can achieve a balance of required memory and computing time. Benefited from the compact and intuitive representation, the solid modeling operations developed for LDNI are robust and easy to implement. Instead of processing the *continuous* geometric boundary in $\Re^3$, all the geometric operations are based on a set of *discrete* sampling points.

The remainder of the chapter is organized as follows. Section 2 briefly reviews the background of geometry representations and related works. The details of LDNI and the

LDNIs-based computational framework are presented in Section 3. The conversion between LDNIs and the boundary representation such as STL is discussed in Section 4. Several LDNIs-based geometric operations are presented in Section 5. Several applications that can benefit from the LDNIs-based computational framework are illustrated in Section 6. Finally the chapter summary is given in Section 7.

2. Background and Related Work

Geometric analysis and computation are essential for the computer aided design and manufacturing (CAD/CAM). Currently, the Boundary Representation (B-rep) is the most popular representation of three-dimensional geometry for CAD/CAM applications. Commercial geometric kernels, such as *ACIS* and *Parasolid*, are all based on the B-rep. The geometric operations based on the B-rep have also been extensively studied. While being accurate, the B-rep based approaches lack in simplicity and are prone to robustness problems. In a survey paper, *Piegl* [3] identified the top ten research challenges in the field of computer-aided design. Among them, the top two challenges are *robustness* and *geometric uncertainties* (e.g. handling special cases). The B-rep based computational techniques have inherent difficulties in handling both challenges [4, 5].

The main reason for the robustness challenges in the B-rep based geometric operations is the non-trivial problem of preserving topology consistency considering the round-off errors in the floating-point based calculations. That is, due to the floating-point arithmetic used in digital computers, there is a gap between geometry in theory that is based on the assumption of precise-computation and geometry in practice that is based on actual computations in finite precision. In geometric programs, numerical errors can lead to misjudgments of the combinatorial and topological relations of geometric objects and, consequently, inconsistency of algorithms. Some existing approaches that have been proposed for robust geometric operations are discussed as follows [4, 5].

(1) Exact arithmetic approach: An intuitive way to avoid inconsistency in geometric computation is to avoid numerical errors. That is, if the input data is strictly correct and the algorithm employs only basic numeric operations, the results can be represented by rational numbers precisely. Since containing no errors, there are no misjudgments. The approach is used in software libraries such as *LEDA* and *CGAL*. However, the approach is too restrictive on input. In addition, algorithms can be rather slow especially after multiple operations. Therefore, the exact computation approach with algebraic numbers is currently only efficient for applications that contain a small number of geometries.

(2) Reliable calculation approach: Topological relations are usually judged by the signs of computed numbers. So if the signs can be recognized correctly, consistent judgments can be achieved even if the computed numbers contain errors. Therefore, numerical computations are delayed and accumulated until the last judgments. Interval arithmetic can also be used to enclose the result with a floating-point interval. However, similar to the exact arithmetic approach, expensive cost has to be paid for computation and hence, it can be rather slow. The approach also requires careful algorithm design. Consequently, the implementation can be extraordinary complicated. Currently no general-purpose techniques have been emerged.

(3) Symbolic reasoning approach: The main idea of the approach is to avoid the inconsistency in the topological judgments in algorithms. Consequently, it requires an error analysis for all the judgments based on inexact arithmetic. The computed signs are classified into “reliable” and “unreliable” according to the error analysis, and only “reliable” results are used. However, an algorithm that is developed based on the approach is unnecessarily

complicated. Big efforts are required in classifying all the topological tests in an algorithm, which can be a daunting task in industrial practice.

(4) Hybrid approaches: Some researchers proposed to classify topological tests done in the algorithm into two groups: mutually independent tests and the remaining tests. The mutually independent tests mean that the result of the test does not affect the results of any other tests. So for them, numerical computation based on inexact arithmetic can be used. For the other group of tests, exact approaches are used to ensure the logical consequences of the results.

In addition to numerical errors, the input geometric elements may be degenerated such as two surfaces overlap or tangent along a curve. Approaches such as numerical perturbation have been developed for eliminating degenerate configurations in geometric modeling processes [6].

The robustness problems in the geometric computation based on B-Rep can be severe and hard to solve, especially for complex geometries that are suitable for layer-based AM processes. To avoid such difficulties in the direct manipulation of boundary representation, previous work based on volumetric approaches [7, 8, 9] and sampling point approaches [10] have been presented. These approaches first generate volumetric grids and sampling points to approximate the model. Accordingly, the computation of geometric operations, e.g. mesh regulation and offsetting, can be performed based on volumetric representations, which is more robust, compact and easy to implement. The geometric computational framework introduced in this chapter is also based on a volumetric representation; however, the representation of LDNIs is sparser (like a sparse matrix), and can achieve a good balance between robustness, accuracy and efficiency.

The simplest volumetric representation of a solid model is voxel-based (ref. [11]). However, the binary voxels cannot give a good representation of smooth surface and sharp features. The methods based on distance-fields are soon employed to replace binary voxels. A survey of 3D distance-field techniques can be found in [12]. Sharp edges and corners are still removed during the sampling of uniform distance-fields. Over-sampling could somewhat reduce the aliasing error by taking the cost of increasing storage memory in uniform sampling or by taking the cost of more computing time in adaptive sampling. Furthermore, as being observed by Kobbelt et al. in [13], even if an over-sampling is applied, the associated aliasing error will not be absolutely eliminated since the surface normals in the reconstructed model usually do not converge to the normal field of the original model. Based on this reason, recently developed volumetric approaches always encode both the distance from a grid node to the surface under sampling and the normal vector at the nearest surface point to the grid node (see [14]) – which is called Hermite data.

The layered depth-normal images (LDNIs) presented in the chapter encode Hermite data in points during the sampling procedure. Like [13, 15], LDNIs do not encode Hermite data on grid nodes but on surface intersection points of ray casting. A method is developed based on [16] to accelerate the encoding of Hermite data on LDNIs by the graphics hardware. The LDNIs representation is also somewhat similar to the Ray-rep in the solid modeling literatures [17, 18]. Menon and Voelcker [17] sampled the solid models into parallel rays tagged with h-tag (i.e., the information of half-space at the endpoints of rays), so that the completeness of Ray-rep can be generated. The conversion algorithm between Ray-rep and B-rep or CSG is given in [17]. As mentioned in [18], Ray-rep can make problem easy in the applications involving offsets, sweeps, and Minkowski operations. However, different from the LDNIs, the Ray-rep only stores depth values without surface normals in one ray direction. Furthermore, the algorithm presented in [17] to convert models from Ray-rep to B-rep does not take the advantage of structurally stored information so that it involves a lot of global search and could be rather time-consuming.

3. Layered Depth-Normal Images and Related Computational Framework

Geometric operations based on volumetric approaches are robust and easy to be implemented. While being well accepted in computer graphics applications, volumetric representations have not been widely used in CAD/CAM applications mainly due to the general concerns of its accuracy and efficiency. Different from computer graphics applications, most engineering applications have much higher accuracy requirements. This section introduces a novel volumetric representation – *Layered Depth Normal Image* (LDNI), which can achieve a good balance between the requirements on robustness, simplicity, and accuracy. The LDNI representation (ref. [19, 20, 21]) is first discussed as follows.

3.1. Layered Depth-Normal Image (LDNI)

Layered Depth-Normal Image is a new representation to implicitly encode the shape of a solid model as a structured collection of points with Hermite data.

Definition 3.1.1 A single Layered Depth Image (LDI) with a specified viewing direction is a two-dimensional image with $w \times w$ pixels, where each pixel contains a sequence of numbers that specify the depths from the intersections (between a ray passing through the center of pixel along the viewing direction and the surface to be sampled) to the viewing plane, and the depths are sorted in the ascending order. Note that the intersections here exclude the case that a ray is parallel to the intersected faces.

Definition 3.1.2 A single Layered Depth-Normal Image (LDNI) is an extension of LDI where each depth is coupled with the unit normal vector of the sampled surface at the intersection point: *x*-LDNI is a Layered Depth-Normal Image viewed along the inversed direction of *x*-axis (i.e., the LDNI is perpendicular to *x*-axis), and *y*-LDNI and *z*-LDNI are perpendicular to *y*- and *z*-axis respectively.

Remark 3.1.1 An edge is defined as *silhouette-edge* if only one of its adjacent polygonal faces is along the current viewing direction. When a ray intersects an edge shared by two faces, no intersection will be counted if this edge is a *silhouette-edge* and one intersection will be sampled for the *non-silhouette-edges*. For a *non-silhouette-edge*, the normal vector at either of its two adjacent faces will be selected and encoded randomly.

Definition 3.1.3 A structured set of Layered Depth-Normal Images (LDNIs) consists of x-LDNI, y-LDNI and z-LDNI with the same resolution $w \times w$, and the images are located such that the intersections of their rays intersect at the $w \times w \times w$ nodes of uniform grids in $\Re^3$.

Figure 5 gives a two-dimensional illustration of LDNIs, where the red dots and arrows indicates the Hermite data points recorded on the x-LDNI and the blue ones illustrate the Hermite data points on the y-LDNI. The example information stored in one pixel on the *x*-LDNI (linked by the red dash line) and one pixel on the *y*-LDNI (linked by the blue dash line) is also illustrated in Figure 5. The slots with blue background present the depth values and the yellow slots denote unit normal vectors. From Definition 3.1.3, it can be found that the information stored in LDNIs is different from other uniformly sampled implicit representation – here only the set of Hermite data points on the surface of a model are sparsely sampled and stored. The stored sample data likes the elements in sparse matrices. Consequently, LDNIs is considered as a sparse implicit representation.

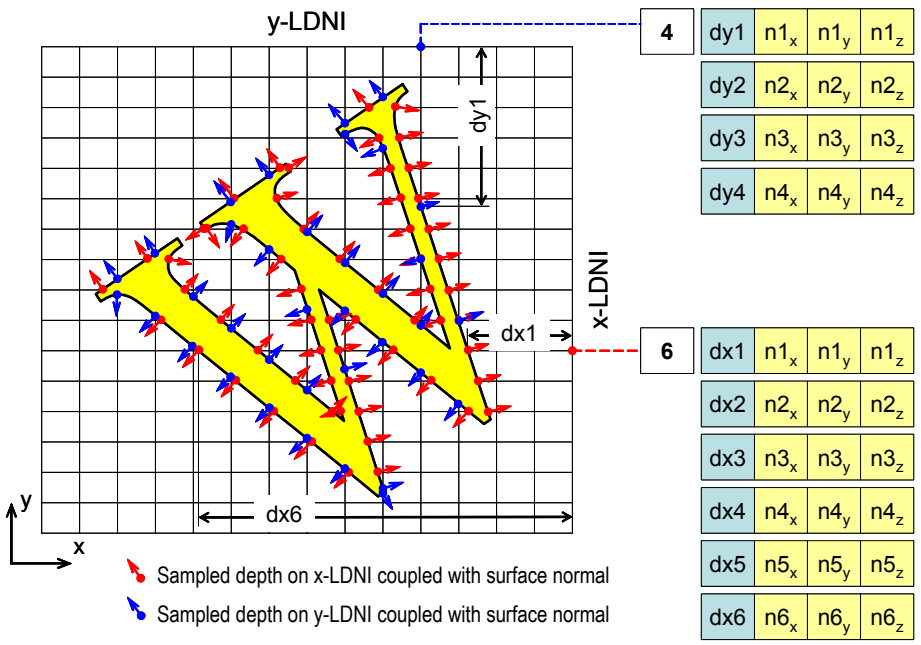


Figure 5. A two-dimensional illustration of Layered Depth-Normal Images (LDNIs), where the dots represents the location of sampled depth and the arrow denotes the unit surface normal vector at this point. Red color is employed for the *x*-LDNI that is perpendicular to x-axis, and blue is for *y*-LDNI.

Remark 3.1.2 The boundary surface of a solid model will not self-intersect.

Definition 3.1.4 For a correctly sampled solid model represented by Layered Depth-Normal Images, the number of sampled depths on a pixel must be even.

Note that when using graphics hardware accelerated method to obtain LDNIs, the guarantee of this property is based on the implementation of rasterization on the hardware. According to the experimental tests, even number of intersections is always reported when the mesh surface of input solid modes is closed. Moreover, the self-intersections on closed mesh surfaces can be eliminated by the method in [22]. For the Layered Depth-Normal Images with pixel width d, a gap or thin-shell on the solid model whose thickness is less than d may be missed on images that are perpendicular to the gap or the thin-shell. This is the reason why three orthogonal LDNIs are needed to record the solid models with thin features.

The information stored in a pixel with the size ranges from $O(1)$ to $O(k)$ where k is the maximal number of layers of the model from this viewing direction. On most practical models, k is a constant number that satisfies $k << w$; in the worst case, $k \to w$ on all pixels, the upper bound of LDNI's memory complexity, $O(w^3)$, is reached. Therefore, the memory complexity of LDNI is $O(w^2)$ on most practical models, and with $O(w^3)$ in the worst case.

In summary, the Layered Depth-Normal Image (LDNI) is a point representation which sparsely encodes the shape of solid models in three orthogonal directions [22, 23]. 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 $\Re^3$. A LDNI in an axis is a two-dimensional image with $w_i \times w_j$ pixels, where *axes i, j, k* are orthogonal to each other. Each pixel of a LDNI contains a sequence of numbers 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. That is, a

sequence of four-tuples (d, n_x, n_y, n_z) can be built, where d specifies the depth from an intersection point P to the viewing plane, and $N_P(n_x, n_y, n_z)$ is the surface normal at P. A LDNIs example for a dragon model is shown in Figure 6.

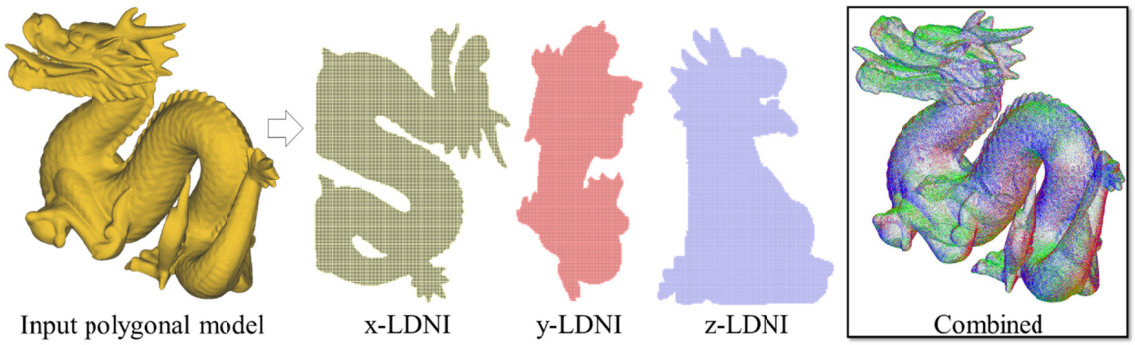


Figure 6. An example of the constructed LDNIs model from a polygonal model.

Therefore, the shape of a solid model can be effectively represented by three LDNIs that are perpendicular to three orthogonal axes, respectively. Some good properties of the LDNIs representation include:

(a) Sampling points are well-structured;
(b) For a given sampling rate w, the memory complexity is $O(w^2)$ instead of $O(w^3)$ required by a voxel representation. It is similar to the memory complexity of adaptively sampled implicit representations but the LDNI is more compact and better structured;
(c) Points can be adaptively down-sampled based on a given tolerance;
(d) High accuracy can be achieved by volume tiling;
(e) Operations based on LDNIs are easy to be implemented and parallelized.

3.2. A LDNIs-based Geometric Computational Framework

The framework of the LDNIs-based geometric computation method is shown in Figure 7.

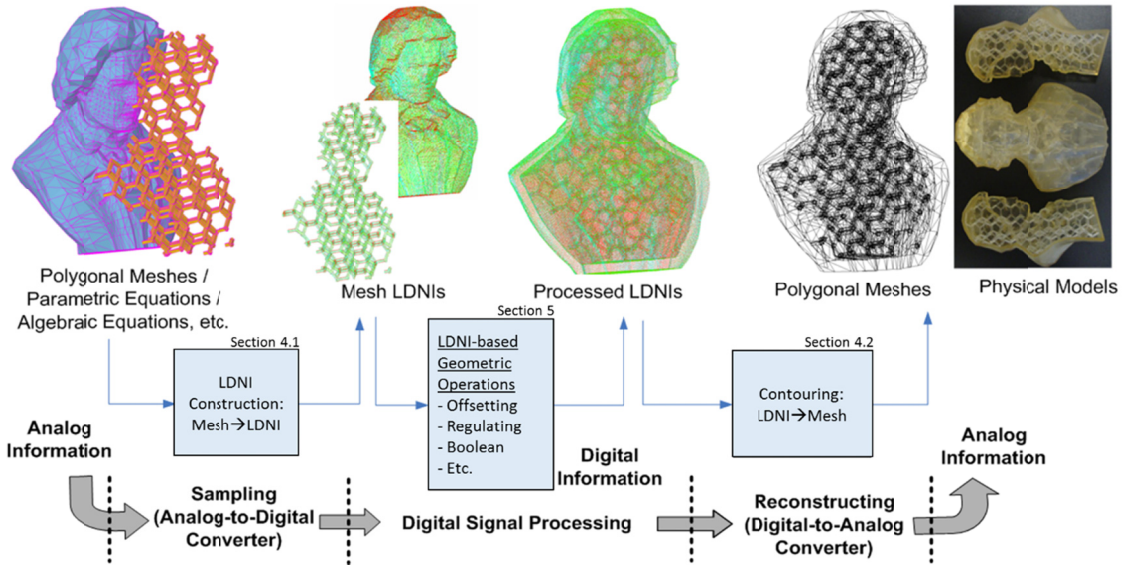


Figure 7. The framework of the LDNIs-based geometric computation method.

From the exact geometry such as closed 2-manifold polygonal meshes defined in a STL file, a LDNIs model can be efficiently constructed by a rasterization technique that can

be implemented using graphics hardware. Based on such well-structured sampling points, various solid modeling operations can be performed quickly and robustly. For solid modeling operations such as *Minkowski* sum or sweeping, multiple operations such as union can be repeatedly performed based on the computed LDNIs model. Point-based rendering techniques [24] can also be used to directly display the LDNIs models. For additive manufacturing systems that require boundary representations as the input, the processed LDNIs model can be converted into a polygonal model based on a contouring method. For example, as shown in Figure 7, in order to add an internal truss structure inside a shelled model, both polygonal models are first converted into related LDNIs models. A LDNIs-based Boolean operator is then used to compute a processed LDNIs model based on them. Finally a polygonal model can be reconstructed from the processed LDNIs model. Note that the newly constructed polygonal model is now valid without defects such as self-intersections. Such a polygonal model can be built by a microfabrication process such as TPP.

The presented framework as shown in Figure 7 has certain similarity to the well-known digital communication and signal processing processes. That is, continuous geometric information is first converted into discrete digital information; various operations can then be performed based on such digital information; finally the processed digital data is converted back to continuous geometric information. Similar to the digital signal processing (DSP) technology, an analog-to-digital converter (ADC), digital signal processing methods, and a digital-to-analog converter (DAC) have been developed in this framework. They are discussed in more detail in the following sections: the conversion between polygonal meshes and LDNIs models will be presented in Section 4, and the LDNIs-based geometric operations will be discussed in Section 5, followed by some applications in Section 6.

4. Conversion between LDNIs and Polygonal Meshes

4.1. Construction of LDNIs: from B-rep to LDNIs

To construct LDNIs from a solid model H, an approach similar to the well-known scan-conversion algorithm can be used with the aid of graphics hardware. Similar to the sampling of LDI in [25], the surface meshes of H have to be rendered multiple times. The viewing parameters are determined by the working envelope, which is slightly larger than the bounding box of the model. Orthogonal projection is adopted for rendering such that the intersection points from parallel rays can be generated.

The repeated times of rendering are determined by the depth complexity n_{max} of the model H with the given direction (e.g., the model in Figure 5 is with $n_{\max} = 8$ and $n_{\max} = 6$ for x-LDNI and y-LDNI, respectively). The depth complexity value n_p at every pixel can be read from the stencil buffer after the first rendering, in which the stencil test configuration allows only the first fragment to pass per pixel but still increment the stencil buffer in the later fragment pass. After that, $n_{max} = \max(n_p)$ can be determined by searching n_p on all pixels and the depth values of the first pass fragments are stored in the depth buffer. If $n_{max} > 1$, additional rendering passes $n = 2$ to n_{max} will generate the remaining layers, and the stencil test configuration allows only the n-th fragment to pass. For the pixels with $n_p < n_{max}$, layers from (n_p+1) to n_{max} do not contain valid depth values and are neglected.

The depth values stored in the depth buffer are floating-point for most graphics cards. The above algorithm generates an unsorted layered depth image; hence a post-sorting step is required for the computed depths at each pixel. In order to avoid repeatedly sending the geometry and connectivity data from the main memory to the graphics hardware during the sampling – such data communication takes a lot of time for complex models, a glList was compiled onto the graphics card, which can then be called for rendering the models repeatedly (refer to [26]). By this way, the model to be sampled was only sent through the

data communication bottleneck once, which greatly speeds up the sampling procedure. Figure 8 gives an illustration of the constructed *z*-LDNI.

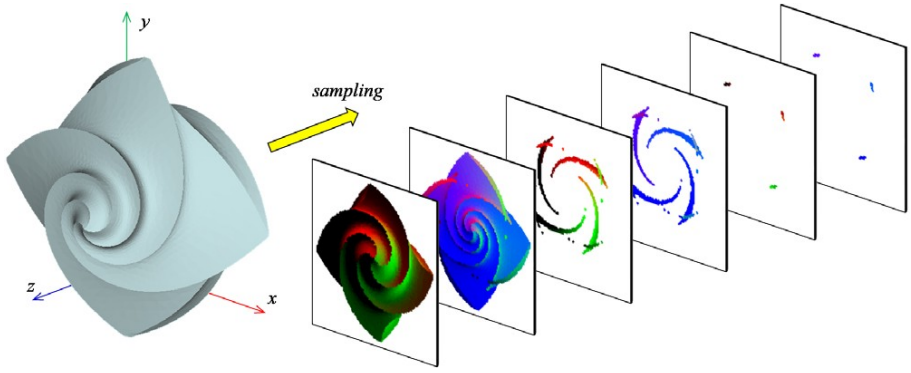


Figure 8. A solid model represented by LDNIs can be stored as a list of 2D textures in graphics memory – an illustration without encoding.

In order to get an accurate surface normal, a unique ID was first encoded to every polygonal face. The number of ID for each face is then mapped into a RGB-color. Therefore, after rendering all faces by the colors according to their IDs, it can be easily identified which face contributes to a sample fragment by the RGB-color. Accordingly the surface normal on the face can be retrieved from the input model and then encoded at the sample. As each color component is with 8 bits, up to 2^{24} distinguishable triangles can be rendered into the frame buffer, which is usually much more than the required number for practical use.

Based on highly parallel architecture, modern graphics hardware is very efficient on displaying polygonal models. With the aid of graphics hardware, a LDNIs model can be constructed rather quickly (usually less than 1 second) from a polygonal model even if the model has complex geometries.

The accuracy of a LDNIs model depends on the pixel width δ used in the rendering process. Suppose the bounding box of an input model is given as Ext_{min} and Ext_{max}. If the graphics hardware is used in constructing the corresponding LDNIs model, the minimum pixel width $\delta = \frac{Ext_{\max} - Ext_{\min}}{w-1}$, where $\boldsymbol{w}$ is the maximum image resolution available in the rendering (e.g. $\boldsymbol{w}$ is 1024 for the graphics hardware with a resolution of 1280×1024). Depending on the available memory, the maximum image resolution can be much larger if the rendering is performed by a simulated frame buffer. However, the rendering speed would be slower.

For a model with large size and/or high accuracy requirement, a technique called *volume tiling* can be used [27]. That is, the bounding box of a model is first split into smaller tiles. Each tile is then processed independently (either sequentially or in parallel) and their LDNIs models are constructed separately. The changes required in the LDNIs representation for volume tiling include: (1) Recording Ext_{LDNI_min} and Ext_{LDNI_max} in each LDNI model. They are different from the minimum and maximum extent of a given model; (2) in each pixel (i, j) of a LDNI, recording the normal index number I_{Norm} at the starting point defined by Ext_{LDNI_min}. Hence, in constructing the LDNI model of each tile, the input polygonal model is zoomed into a target Ext_{LDNI_min} and Ext_{LDNI_max}. Note that the viewing plane is still set by Ext_{min} since I_{Norm}= 0 at Ext_{min}. All the sampling points are calculated from the rendering process and only record the nodes that are inside the extent (Ext_{LDNI_min}, Ext_{LDNI_max}] in the constructed LDNI.

4.2. Contouring Algorithm: from LDNIs to Two-Manifold Polygonal Meshes

A LDNIs model is an implicit representation of a solid defined by geometric operations. However, most AM systems and 3D printers require polygonal meshes as input CAD models.

It is generally challenging to faithfully reconstruct a B-rep model from an implicit representation especially for features whose sizes are close to the resolution of the sampling points. This section presents a novel contouring method for reconstructing polygons from a LDNIs solid [28]. The method can handle an arbitrary number of intersection points and edges in a cell edge and cell face, respectively. It can also handle multiple shells within a cell. In addition, the method can better capture small features that have similar size to the resolution of sampling points. In the constructed contour of a LDNIs solid, it is ensured that no artifacts such as self-intersections exist. Further, two strategies to generate manifold-preserved mesh surfaces [23] can be used to overcome the topology ambiguity that may occur inside the finest octree cells after the maximum subdivision. As a result, the constructed polygonal model is manifold with no gaps or overlapping surfaces. The contouring algorithm contains four steps:

Step 1: Grid node construction

The grid nodes are the intersections of the rays defined by three LDNIs. For a grid node which is an intersection of 3 rays on *x*-LDNI, *y*-LDNI and *z*-LDNI respectively, its *inside*/*outside* status defined by the three LDNIs is expected to be consistent. If inconsistent classification of grid signs exists, a majority voting approach is used, that is, a grid node is classified into *inside* the model if it falls into the volume defined by at least two of the LDNIs.

Step 2: Cell edge construction and regularization

Cell edges are constructed between the grid nodes by the intersections (depth samples) on its corresponding ray of a LDNI. Different types of edges are constructed according to the number of depth samples falling in the interval of the edge. The cell edge with its two end-nodes having different signs and with intersection points on the edge is named as *intersect-edge*, and the one having different sign nodes but with no intersection point is named as *none-intersect-edge*. The *none-intersect-edge* may be generated due to the numerical errors. The cell edges with end-nodes having the same sign and no intersection point are defined as *empty-edges*, and the ones with same sign nodes but with intersection points are denoted as *complex-edges*. Based on the heuristics that there is only one thin structure passing a cell edge, the following rules are employed to regularize the cell edges.

Rule 1 A *complex-edge* with more than two intersections will only keep two intersections that are the closest to the end-nodes of the edge.

Rule 2 An *intersect-edge* with more than one intersection keeps only one sample that is the closest to the middle point of the edge and with the normal vector compatible to the signs on the end-nodes (i.e., the sample with normal's direction consistent with signs on the end-nodes of the edge).

Step 3: Cell construction and vertices positioning

A color flooding algorithm is employed to cluster the grid nodes in one cell whose signs indicate *outside*. In each cell, a grid node with *outside* flag is chosen as a seed to fill with a color *c*, and a flooding algorithm is used to fill the color *c* to all grid nodes in this cell that are linked to this seed by *empty-edges*. If a new *outside* seed node is found on the cell, the flooding is applied again with a new color. This flooding will be repeated until no new seed node is found. The number of vertices to be constructed in this cell depends on the number of colors. After grid nodes with *outside* flag are divided into clusters by different colors, the position of every node cluster is determined by the Hermite data points on the *intersect-edge* and the *complex-edge* linking to this set of grid nodes. The computation is

through minimizing the Quadratic Error Function (QEF) defined by these Hermite data points (ref. [15]). For the *complex-edges* with two Hermite data points, each of the Hermite points is classified to the cluster holding one of the two end-nodes. Few vertices associated with no Hermite data point may be generated on the *none-intersect-edges* because of the numerical error. For them, the final positions are determined through Laplacian smoothing after constructing the mesh connectivity.

Step 4: Mesh construction

For every regularized *intersect-edge*, one quadrilateral face is constructed by linking the vertices in its four neighboring cells. The adopted vertices should be the one associated with the *outside* node on this intersect-edge. For every regularized *complex-edge*, two quadrilateral faces, each for one end-node on the edge, are constructed in the similar way. All faces should be constructed in the orientation to let its normal facing outwards.

The contouring method is illustrated in Figure 9. Similar to the *marching cubes* algorithm, triangle meshes T_i are generated for each cell C individually. In addition, it is ensured that $T_i \subset C$. Hence, no self-intersection can happen between the triangles of different cells. As shown in Figure 9.a and b, the nodes and sampling points are first classified in a LDNIs solid to ensure their consistency. The contour edges on each cell face are then computed (refer to Figure 9.c). Accordingly, contour loops from the contour edges of all six cell faces can be constructed, and triangle meshes for each contour loop are computed (refer to Figure 9.c). A contouring result of a processed LDNIs model is shown in Figure 10.

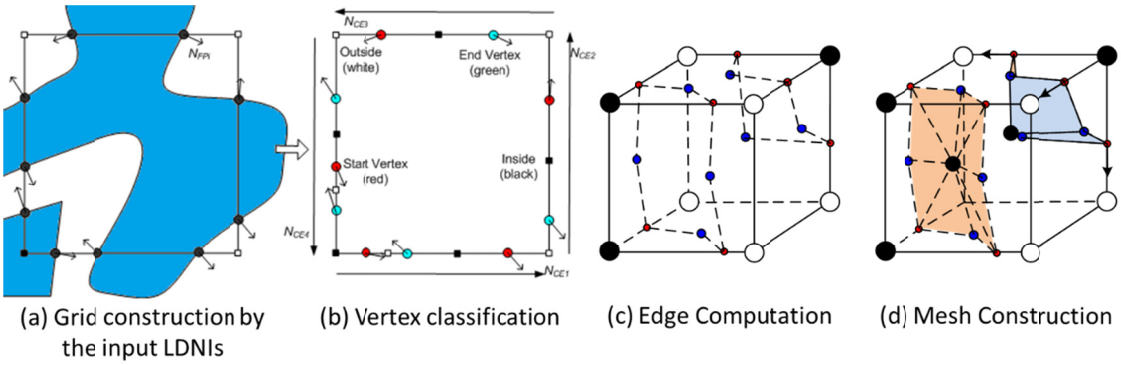


Figure 9. An illustration of the contouring method.

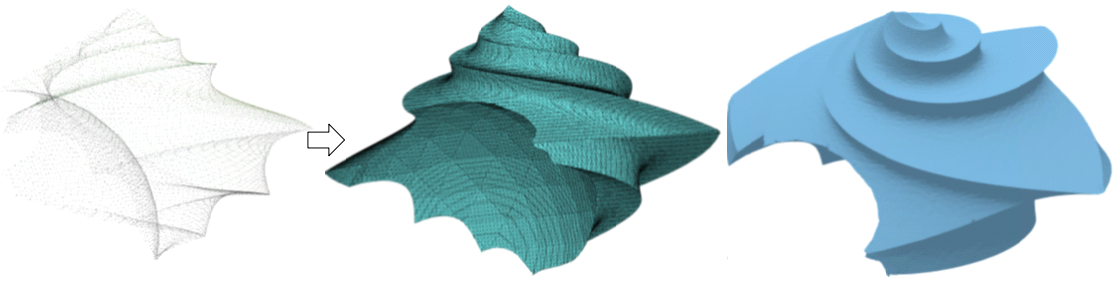


Figure 10. The contouring result of a processed LDNIs model.

In summary, the LDNIs-based approach has the following advantages:

- Less memory is required for storing complex models based on the same approximating error;
- The data structure retains the information of complex-edges such that it can reconstruct sharp-features or thin structures in a relative low sampling rate;

- The data structure and relevant algorithms are well structured; hence it can easily employ the power of the accelerated graphics hardware (e.g., GPU [19]) or other parallel computing devices.

5. LDNIs-based Geometric Operations

Computation algorithms based on the LDNIs representation have been developed for geometric operations including offsetting, regulation, Boolean, and sweeping. A primary strength of the point-based representation is that all the shape changes are implicitly defined by a set of points. Hence, the topological consistency of the final shape is guaranteed in the contouring process. Therefore, the LDNIs representation is especially suitable for geometric operations that require complex topological changes. In comparison, a core robustness issue for the B-rep is the consistency in judging topological relations by considering numerical errors. Due to the radical topological changes required in the geometric operations such as offsetting and regulation, the robust implementation based on the B-rep is well-known challenging.

5.1. LDNIs-based Uniform Offsetting

Although the offsetting operation is mathematically well defined, computing an offset model for a given solid is difficult. Position changes by an offset distance generally lead to self-intersections and consequently topological changes. Consequently, trimming invalid offset surfaces in a 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.

The uniform offsetting of a solid S by a distance r has been precisely defined for point sets in *Euclidean* space $\boldsymbol{E}^2$ or $\boldsymbol{E}^3$. As shown in Figure 11, suppose a ball with radius r is defined as b_r, the two offsetting operations can be defined as: (1) *S grown by r* as $S\uparrow_r = S \oplus b_r$, and (2) *S shrunk by r* as $S\downarrow_r = S \otimes b_r$, where a special case of the *Minkowski* sum of A and B, denoted $A \oplus B$, is defined as $C = A \oplus B = \{a+b \mid a \in A, b \in B\}$, and a special case of the *Minkowski* difference, denoted $A \otimes B$, is $\overline{\overline{A} \oplus B}$.

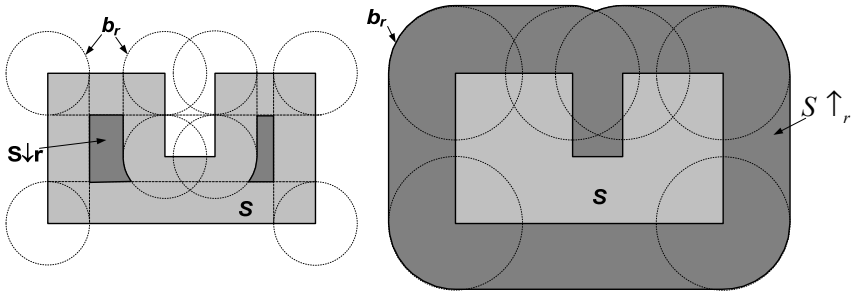


Figure 11. Offsetting a solid S by a distance r.

The principle of the LDNIs-based offsetting method is to first compute the offset surfaces of input surfaces by displacing each surface point $\boldsymbol{q}$ by a distance r along the unit normal $\boldsymbol{n}$, i.e., let $\boldsymbol{p} = \boldsymbol{q} + r\boldsymbol{n}$. However, some of the displaced points (e.g., $q_i\|_{\mathrm{r}}^{+}$) may be at a smaller distance to other points of ∂S (e.g., $q_k\|_{\mathrm{r}}^{+}$). When p is an invalid point, $q_i\|_{\mathrm{r}}^{+}$ and $q_k\|_{\mathrm{r}}^{+}$ will intersect each other. Such a self-intersection is a core challenge to be addressed in the offsetting operation. The displaced sampling points, whose minimum distance to $\partial(S)$ is less than the offset distance r, is defined as **inner points**, and **boundary points** are all the

displaced sampling points that are on $\partial(S \uparrow^* r)$. Therefore, the goal is to remove the inner points such that the boundary points can be computed to approximate the boundary of $S \uparrow^* r$.

It is challenging to robustly implement the direct trimming of self-intersections and overlapping surfaces based on the boundary representation. A novel LDNIs-based offsetting method [29] is based on the steps of directly computing offset boundary, converting the boundary into structurally sampled points, and accordingly filtering the sampling points in order to reconstruct offset contours. An illustration of the method for a given solid model is shown in Figure 12. First, a set of offset surfaces is computed directly from the vertices, edges and triangles of the input model. The offset surfaces will form a continuous boundary (refer to Figure 12.a). However, the generated offset surfaces may have complex self-intersections with multiple surfaces that are closer to the original model than the distance *r*. To trim the invalid offset meshes, a LDNIs model is constructed to sample the offset meshes (refer to Figure 12.b). The accuracy of the generated LDNIs models can be controlled during the construction process. All the sampling points in the offset LDNIs are then processed using a set of point filters. Based on them, all the invalid points are identified and discarded from the offset LDNIs model. The remaining points after the filtering process are shown in Figure 12.c. Final, an offset contour can be reconstructed from the processed LDNIs. The computed uniform offsetting model is shown in Figure 12.d.

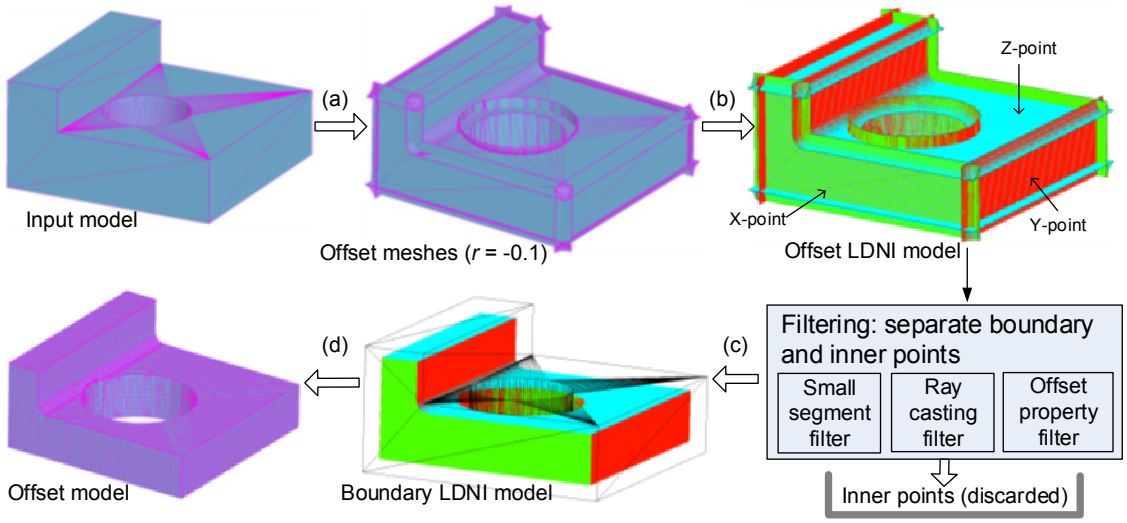


Figure 12. An overview of the LDNIs-based uniform offsetting method. First, a continuous offset boundary is constructed; then the boundary is sampled from three axes to compute a point-based model; the sampling points are filtered and inner points are discarded; finally the remaining boundary points are used to construct the contour of offset model.

Similar to the digital signal processing technology, a set of point filters are developed to process all the LDNIs points such that all the inner points can be removed to avoid self-intersections in the computed offset model. The first filter, named **small segment filter**, determines whether a pair of sampling points come from two overlapping surfaces and hence 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 sampling points are generated from invalid edges. The implementation details of the three filters can be found in [29]. A 2D case to illustrate the aforementioned filters in removing inner points is shown in Figure 13.

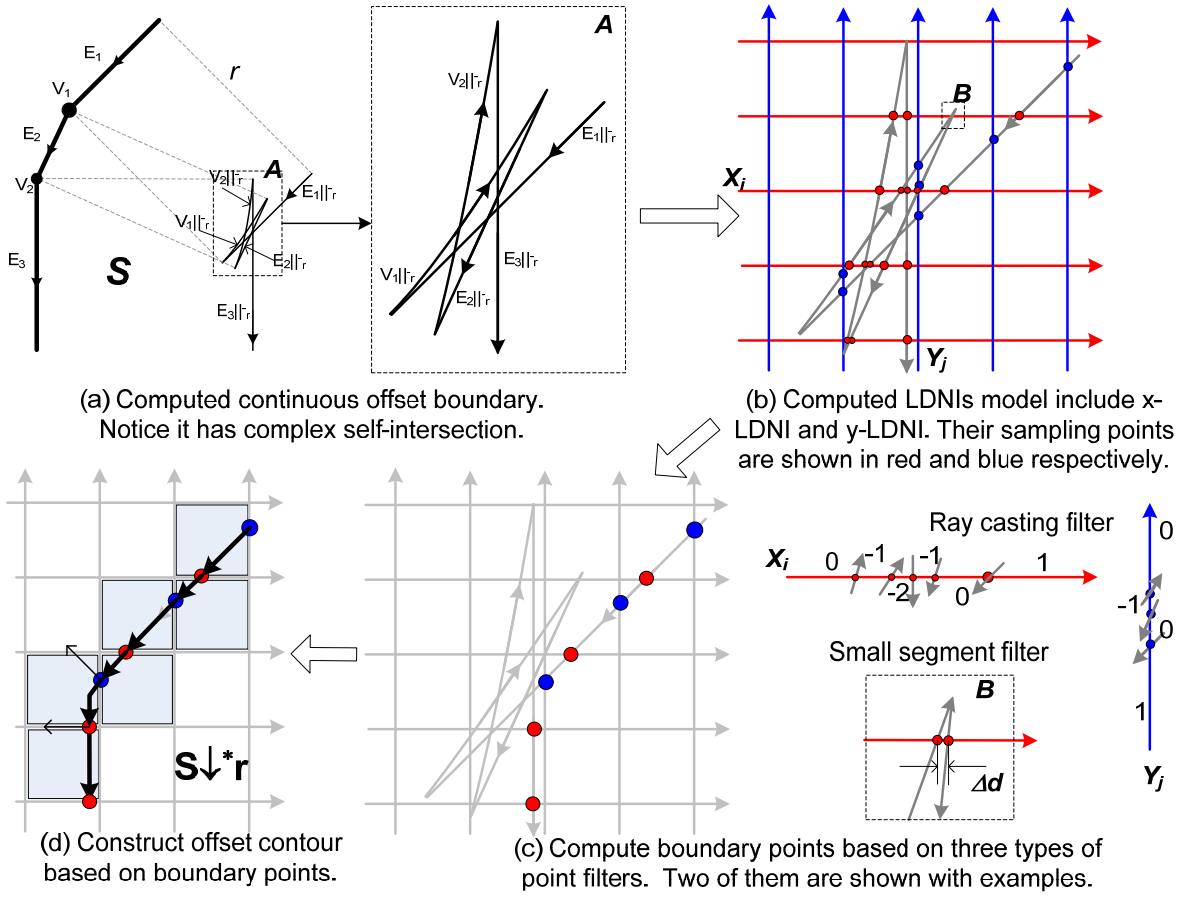


Figure 13. A 2D illustration example of *S* shrunk by a distance *r*.

5.2. LDNIs-based Regulation Operator

Most layer-based AM systems use polygonal models in the STL format as input. In addition, the input polygonal models need to be manifold, water-tight, and no self-intersections. However, it is difficult to enforce such a requirement since the syntax of a STL file only requires a set of triangles with their normal vectors. A 2D illustrative example is shown in Figure 14. The input boundary as shown in Figure 14.a is obviously invalid since the input meshes have self-intersections that can be classified as: (1) loop twisting, (2) external loop overlapping, and (3) internal loop overlapping. For an arbitrary ray, its I_{Norm} value along the ray can be computed (refer to Figure 14.a). Accordingly, the nodes whose two neighboring I_{Norm} values are 0 and 1, or 1 and 0 can be identified (in black dots as shown in Figure 14.b). All the other sampling points are shown as red dots in the figure. After removing all the red dots, the I_{Norm} values along the ray are only 0 or 1. Figure 14.c shows the portions of the ray with $I_{Norm} = 1$ and $I_{Norm} = 0$ in red and black respectively. Accordingly, the processed ray can be used in defining the boundary of a regulated model in which all the self-intersections have been removed (refer to Figure 14.d). This kind of geometric operation is called *mesh regulation*, in which an arbitrary input polygonal model with defined surface orientation (i.e. normal) is converted into a valid STL file [27].

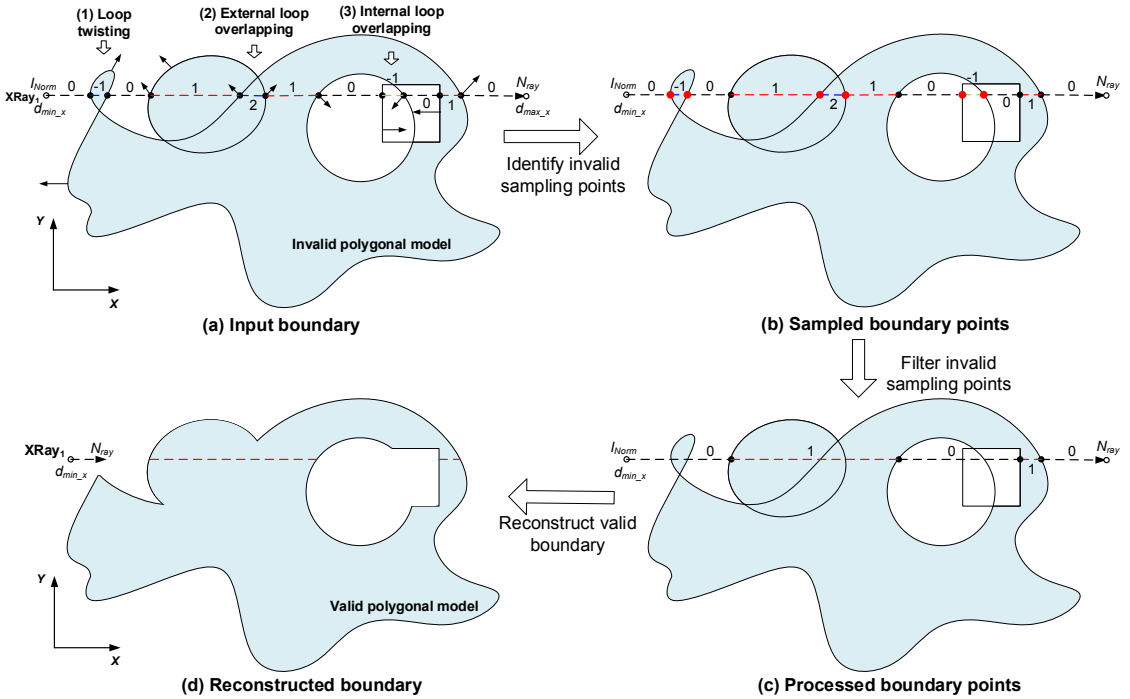


Figure 14. An illustration of removing self-intersections based on I_{Norm} values along the ray: (a) input boundary, (b) sampled boundary points, (c) processed boundary point, and (d) reconstructed boundary.

For a solid defined in the LDNIs representation, the regulation operation is straightforward and easy to implement. A LDNIs model consists of a set of well-organized 1D volumes defined on uniform grids with accurate depth and normal information. Accordingly, the aforementioned ray casting filter is used in removing invalid points. For a given LDNIs model, its *x*-LDNI, *y*-LDNI and *z*-LDNI are processed separately (sequential or in parallel). For each x/y/z-LDNI, it goes through each pixel (i, j) and sort all the points P_1~ P_n based on their depths. Then, I_{Norm} is calculated for each line segment along the ray. Finally, the calculated I_{Norm} can be used to delete all the inner points whose two neighboring I_{Norm1} or I_{Norm2} are not (0, 1) or (1, 0) from the LDNIs model. Hence, all the nodes that correspond to the self-intersections will be deleted and only the nodes that correspond to the boundary of a regulated solid will be stored in the processed LDNIs model. The ray casting filter removes an even number of nodes along a ray. This is because the difference of $I_{Norm}(P^-)$ and $I_{Norm2}(P^+)$ for any given point P along the ray is 1 or -1. Hence, the points that are neighboring to the line segments with $I_{Norm} > 1$ or $I_{Norm} < 0$ must be even.

An input STL model may define 3D geometry that has far more complex self-intersections than the 2D example as shown in Figure 14. An illustration example of such STL models is shown in Figure 15. In the test, a structure configuration based on a given model of a *Beethoven* statue was generated using a microstructure. A sphere and a cylinder are added at each joint and strut of the structure, respectively (refer to Figure 15.a). Since all the spheres and cylinders are simply placed together, the constructed STL model is invalid due to complex self-intersections. The regularized polygonal model based on the LDNIs-based mesh regulation method is shown in Figure 15.b. The newly constructed STL model is now valid as all the self-intersections have been removed. The validity of the regulated model is further tested by building the model using a SLA machine. The fabricated object is shown in Figure 15.c. A split version of the statue with the designed complex internal structures is shown in the figure. Note that a significant benefit of the layer-based AM processes is their capability of fabricating truly complex shapes that were impossible by other

approaches. A general and robust regulation method based on LDNIs can ensure an arbitrarily complex polygonal model to be built by TPP systems.

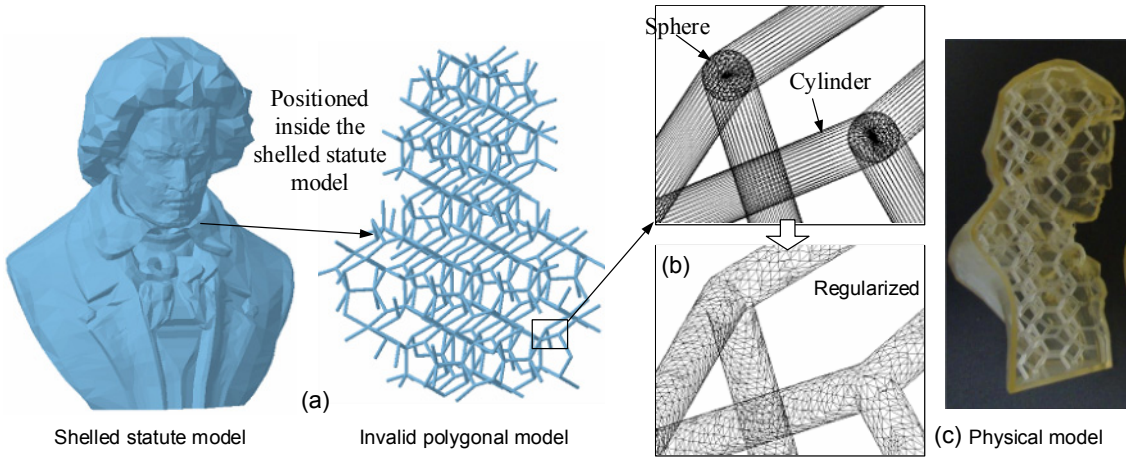


Figure 15. Test result of an internal structure: (a) an input model with self-intersections, (b) the regularized polygonal model, and (c) the physical model fabricated by SLA.

5.3. LDNIs-based Boolean Operation

A Boolean operation, such as union, intersection or difference, is one of the most important geometric operations. For solid models in the LDNIs-based representation, the Boolean operations are straightforward and easy to implement. As discussed in Section 3.1, a LDNIs model is actually a set of well-organized 1D volumes. It inherits the good property of simplicity on Boolean operations from ray-rep. When computing the Boolean operation of two LDNIs models H_A and H_B, the Boolean operations can simply be conducted by the depth-normal samples on each ray as long as the rays of H_A and H_B are overlapped. This request can be easily satisfied during sampling. More specifically, when sampling H_B into a LDNIs solid H_B, the origin of sampling envelope is carefully chosen to ensure that the rays of H_B overlap the rays of H_A. In addition, the two input models are identically oriented with the same sampling frequency.

The algorithm of Boolean operations on rays is briefly described as follows. On two overlapped rays $R_A \in H_A$ and $R_B \in H_B$, if either R_A or R_B is empty (i.e., with $n_A = 0$ or $n_B = 0$, where n_A and n_B are the numbers of samples on R_A and R_B), the point processing will be very simple. For the case neither R_A nor R_B is empty, one can perform a 1D Boolean operation easily by moving the samples of R_A and R_B according to their depth values. During the movement, the resultant samples are generated and stored on a new ray R_{res}, where the resultant samples are those leading to a change of inside/outside status. The analysis of current sample can be performed with the help of a logic operator, $OPRT(A, B)$. The operator $OPRT(A, B)$ for different types of Boolean operation is defined in Table 1.

Table 1. Logic operator $OPRT(A, B)$

Status of A and B		Type of Operations		
Inside A	Inside B	∪ (union)	∩ (intersection)	\ (difference)
True	False	True	False	True
False	True	True	False	False
True	True	True	True	False
False	False	False	False	False

In short, whether a depth value d is inside or outside a 1D volume can be checked by detecting whether there are odd (or even) number of samples whose depth values are less than d. Then, the inside or outside status on a resultant 1D volume is determined by $OPRT(A,B)$. After scanning all samples on R_A and R_B, the resultant samples can be outputted into a list of sample, R_{res}. After small segments and self-intersections being removed, a LDNIs model only consists of a set of well-organized one-dimensional (1D) volumes in each pixel. Therefore, the Boolean operations on two solid models are converted into the Boolean operations on a set of 1D segments. An illustration is shown in Figure 16. For a better performance, the segments of H_A for a pixel are first sliced into more segments by the samples from H_B that fall in the interval segments of H_A. An example of the LDNIs-based union operator for two input 3D models is shown in Figure 17.

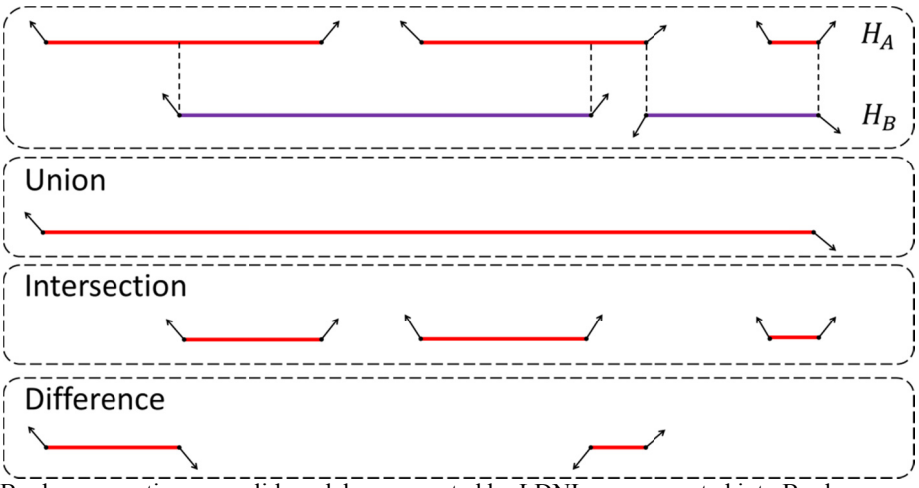


Figure 16. Boolean operations on solid models represented by LDNIs are converted into Boolean operations on 1D segments.

Figure 17. An example of Boolean operation: ‘‘Buddha ∪ Filigree’’.

5.4. Robustness Enhancement

Two common robustness problems that need to be considered are numerical error and degenerate data. Numerical error occurs due to the use of floating-point arithmetic in geometric computations. Degenerate data such as tangential contact between various geometric elements can be produced by geometric operators even if the input models are manifold. Such robustness problems have been extensively studied in computational geometry. Their handling can be challenging for geometric operations based on B-rep.

In the LDNIs-based geometric computation methods, polygonal meshes that define the *continuous* boundary are first converted into a set of *discrete* sampling points. Hence, a geometric operation will be performed only on these points, which dramatically simplifies the robustness problem. In addition, the methods for enhancing the robustness of the LNDIs-based geometric operations are described as follows.

(1) **Numerical error**: The LDNIs representation uses a quantization resolution δ to convert the depth of a sampling point into an integer representation. For a maximum extent Γ, a *N*-bit integer is needed where $N = log_2(\Gamma / \delta)$ to represent any floating-point number within Γ. Therefore, an input geometry is embedded inside a fine integer lattice with size 2^N in each dimension and each intersection point can be clamped to the nearest lattice point. By choosing ω as $m \times \delta$ where *m* is an integer, it can also clamp the uniform grid to their nearest lattice points. Therefore, most computations on 1D volume can be performed based on exact integers.

(2) **Degenerate data**: The aforementioned point filter, *small segment filter*, can be used to remove tangential contacts between geometric elements. With the help of the LDNIs representation, the degenerated data can be easily removed by the filter.

6. Applications in 3D Microfabrication and Others

The LDNIs-based geometric operations can be used in various applications that require high-level robustness and efficiency. Five applications that have been investigated by us are presented in the section as some examples.

6.1. Complex Truss Structure Design and Fabrication

In the design of complex structures, a large number of individual structures can be put together independently. While being straightforward, the designed structures have complex self-intersections that cannot be fabricated. For example, Figure 18.a shows a truss structure design on the surface of a spherical ball. Based on the designed truss structure configuration, a cylinder is added at each joint and a cube is added at each strut. All the cylinders and cubes are merged together. Note that multiple faces from the added cylinders and cubes overlap with each other. Such test case would be challenging for the B-rep based Boolean operations since exact surface intersections are prone to numerical errors. Instead, the aforementioned LDNIs-based mesh regulation method is used in converting the invalid polygonal models into a valid one. Such overlapping surfaces can be handled by the aforementioned small segment filter. The regulated polygonal model is shown in Figure 18.b. The validity of the generated STL model was verified by successfully building it using an AM machine. Figure 18.c shows the built spherical ball with the designed truss structures.

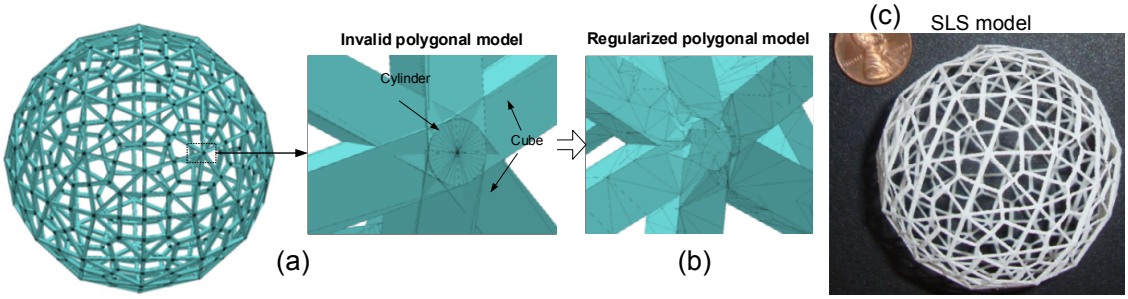


Figure 18. Test result of a ball with complex truss structure: (a) an input model that is designed by putting cylinders and bars together, (b) the regularized polygonal model, and (c) the physical model that was fabricated by an AM process.

6.2. 3D Model Shelling and Shrinkage Compensation

To fabricate a CAD model with less material, one may perform the model shelling operation to create a hollowed object design. The 3D model shelling can be done by offsetting the outer surface inward by a given distance. The outer and inner surfaces can be merged to define the volumetric information of the created shells. Figure 19 shows an example of building a hollowed solid model with interior truss structures. As shown in the figure, the input model is subtracted by its inward offset model to get the hollowed object. Accordingly, the union of the hollowed object and the truss structures can be computed to get the final 3D model to be fabricated. The overall process with the help of graphics processing unit (GPU) is done within 2 seconds [19].

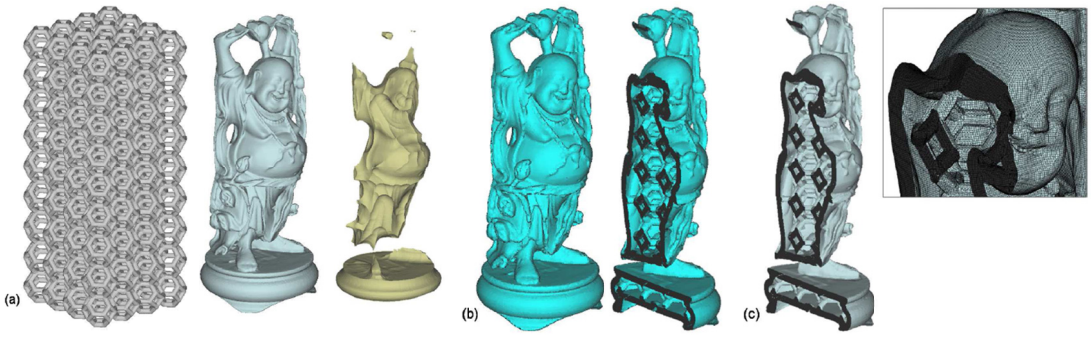


Figure 19. An example of using Boolean operations to build the interior structure of a hollowed solid model. (a) Input models: (left) Truss with 941.9k triangles; (middle) Buddha with 497.7k triangles; (right) Offset of Buddha with 213.3k triangles. (b) The resultant LDNIs solid is obtained after Boolean operations. (c) The final mesh surface with 804.6k quadrangles generated from the LDNIs solid.

Another related application is the shrinkage compensation for AM processes. In the fabrication process, accumulated material undergoes volumetric shrinkage; hence the final fabricated shape may be smaller than the desired shape. A 2D illustration is shown in Figure 20.a. If the model is fabricated directly as the desired shape, the obtained shape after shrinkage will be smaller. If the amount that the shape will shrink is r, the shrinkage compensation can be done by offsetting the model outward to get an enlarged shape. Consequently, fabricating the compensated shape will result in a shape that is closer to the desired shape. An example of growing a 3D model by a distance r is shown in Figure 20.b. For such a test case, most commercial software systems including CAD software systems such as *SolidWorks* and *Pro/E*, and graphics/industrial modeling software systems such as *Maya* and *Rhino* 3D will have difficulties. The test results show that the LDNIs-based offsetting method is general, robust, and efficient.

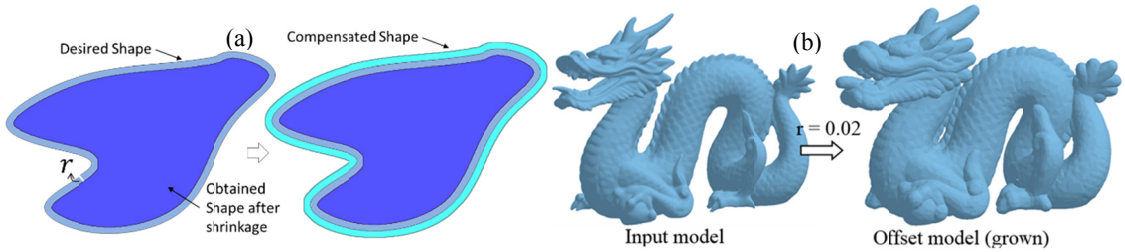


Figure 20. Screen capture of the offsetting results for a 2D contour and a dragon model.

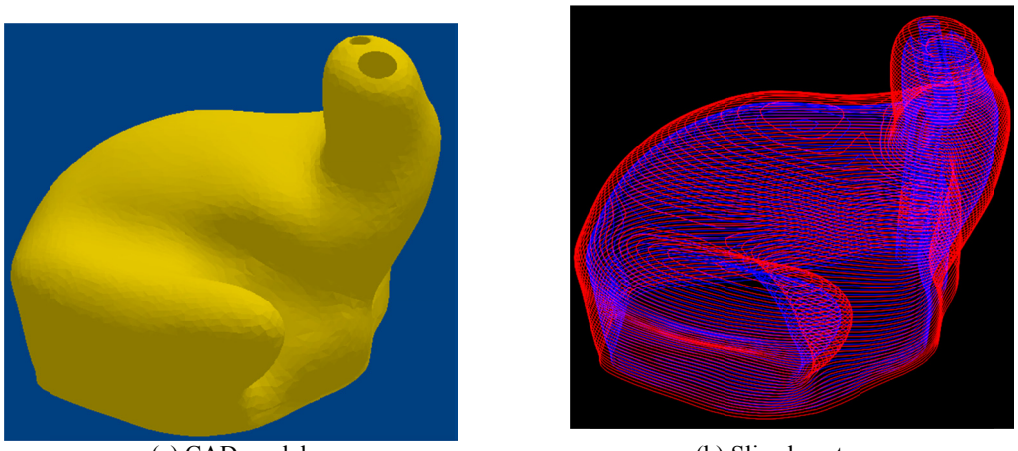
6.3. Tool Path Planning – 2D Slicing and XY Compensation

In order to fabricate 3D objects in 2D layers, a STL model is sliced layer by layer. Each layer can be converted into loops, which are the intersection of the 3D object with a set of 2D horizontal planes. As the plane moves up, successive layers are defined based on the cross

sections of the plane. Analytic geometry could be used to determine the intersections, which are then stored in a dynamic array. Each surface that intersects the plane forms a direct line segment on the planar slice. All these intersection lines will define the contour of the layer. The intersections are illustrated in Figure 21 and Figure 22. The segment L is defined by two vertices $P_1(X_1,Y_1,Z_1)$ and $P_2(X_1,Y_1,Z_1)$. Thus, the following equation can be obtained: $\frac{X-X_1}{X_2-X_1}=\frac{Y-Y_1}{Y_2-Y_1}=\frac{Z-Z_1}{Z_2-Z_1}$, the 2D plane is given by $Z=Z_0$. The intersection point P_0 between the edge L and the plane Z could be calculated as: $X_0=\frac{(Z_0-Z_1)\times X_2-X_1}{Z_2-Z_1}+X_1$ and $Y_0=\frac{(Z_0-Z_1)\times(Y_2-Y_1)}{Z_2-Z_1}+Y_1$. A hearing aid model is shown in Figure 23 to illustrate the slicing result, where the red curve indicates the clockwise contour and the blue curve indicates the counterclockwise contour.

Figure 21. Illustration of slicing process

Figure 22. Intersection plane with object



(a) CAD model

(b) Sliced contours

Figure 23. Hearing aid model.

The sliced contours are basically the target shapes in fabrication of related layers (Figure 24.a). However, since the fabrication tool of TPP (i.e. a laser beam) has certain size. Directly following the path of the contours will result in over-curing (refer to Figure 24.b). In order to incorporate the effect of given tool size, the tool path planning can be performed based on the computed 2D offset of input contours. The compensated tool path for the input 2D contours as shown in Figure 24.a is shown in Figure 24.c.

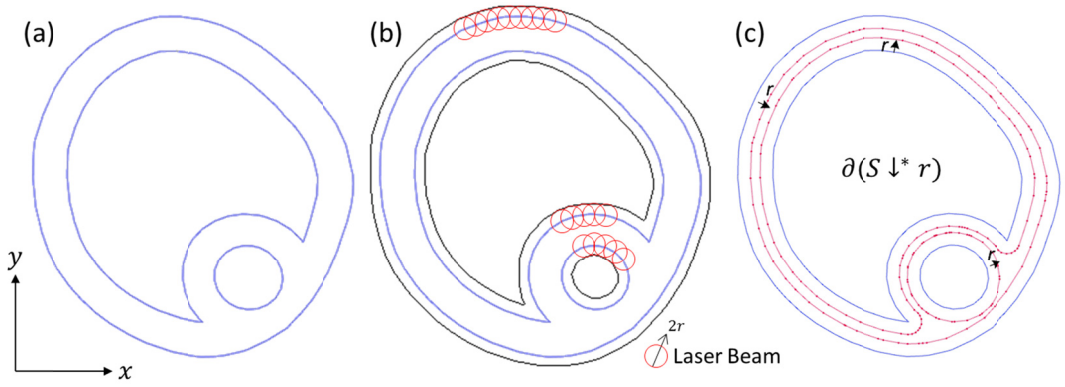


Figure 24. Directly use the contour (a) as tool path results in over-curing (b). The compensated tool path (c) is generated by shrinking *S* by the laser beam radius *r*.

6.4. Tool Path Planning – *Z* Compensation

The accuracy of a stereolithography machine varies from the center to the border of a platform. To achieve a better accuracy, the users are suggested to put their parts in the center of the platform. These areas in the platform with higher accuracy are called ***sweet spots***. There are two reasons for the existing of sweet spots in the SLA machines. (1) The distance from mirror to the resin surface varies from the center to the border (*L* verse *L'* as shown in Figure 25). This may affect laser's focus; (2) the laser enters the resin surface in an angle other than vertical to the surface (α as shown in Figure 25). This will change the cured resin shape related to the incident angle of the laser beam at different positions of the platform.

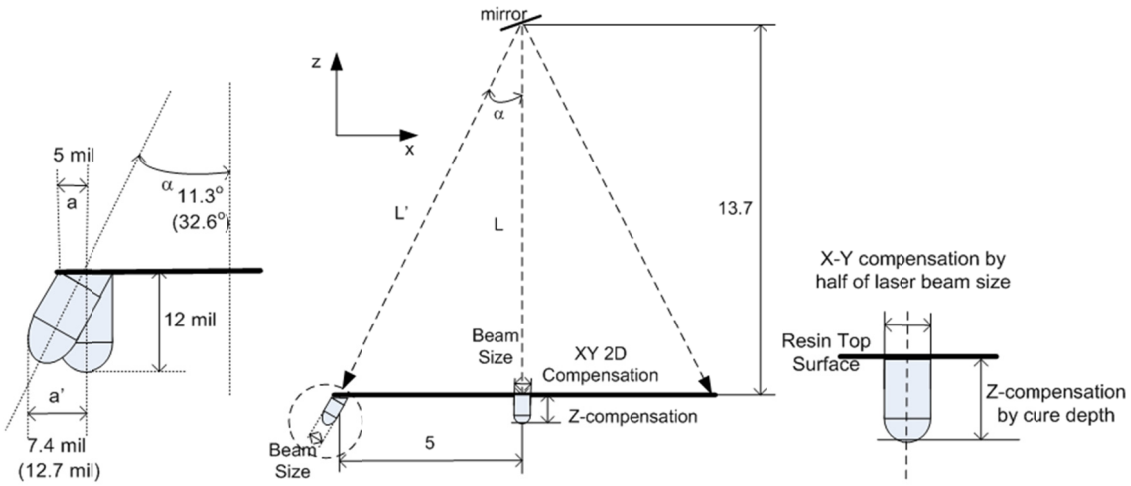


Figure 25. Illustration of laser beams in the platform of a Stereolithography machine.

A *Z* compensation method for the laser-based stereolithography process was presented in [30]. Suppose the laser drawing path is $h(x, y, z)$. For a point $v(x, y, z)$ on $h(x, y, z)$, the corresponding laser shape is defined as $g(x, y, z)$. The laser paths and shapes will form a geometry $f(x, y, z)$ which can be defined as:

$$f(x, y, z) = h(x, y, z) \oplus g(x, y, z)$$

where $A \oplus B = \cup_{b \in B} A_b$, and "∪" denotes set union operation.

However, in the tool path planning problem, the laser drawing path $h(x,y,z)$ has to be compute to fabricate the given geometry $f(x,y,z)$. The laser drawing path can be defined as:

$$h(x,y,z) = f(x,y,z) \oplus (g(x,y,z))^{-1}$$

where $(g(x,y,z))^{-1}$ is the reciprocal set of $g(x,y,z)$.

Let o denote the origin point of the coordinate system in which a convex polytope B is placed. Clearly, $B \oplus \{o\}$ is equal to B, since set $\{\boldsymbol{o}\}$ is a singleton point set. Therefore, $B \oplus B^{-1} = \{o\}$. That is, $(g(x,y,z))^{-1}$ is actually the symmetrical set of $g(x,y,z)$ with respect to the origin point. For every point $v \in g(x,y,z)$, there exist a point $v' \in (g(x,y,z))^{-1}$ such that $v + v' = 0$, where "+" denotes vector addition of two points, and 0 denotes zero vector that is equivalent to the origin point. In addition, suppose a normal can be used to specify whether the related surface is diverging outward or inward. The object B^{-1} will be an opposite shape of B with reverse normals for corresponding surfaces. The above operations are illustrated in Figure 26 with two different shapes of $g(x,y,z)$. Notice $g(x,y,z)$ and related $(g(x,y,z))^{-1}$ are symmetrical around origin points o_1 and o_2, respectively.

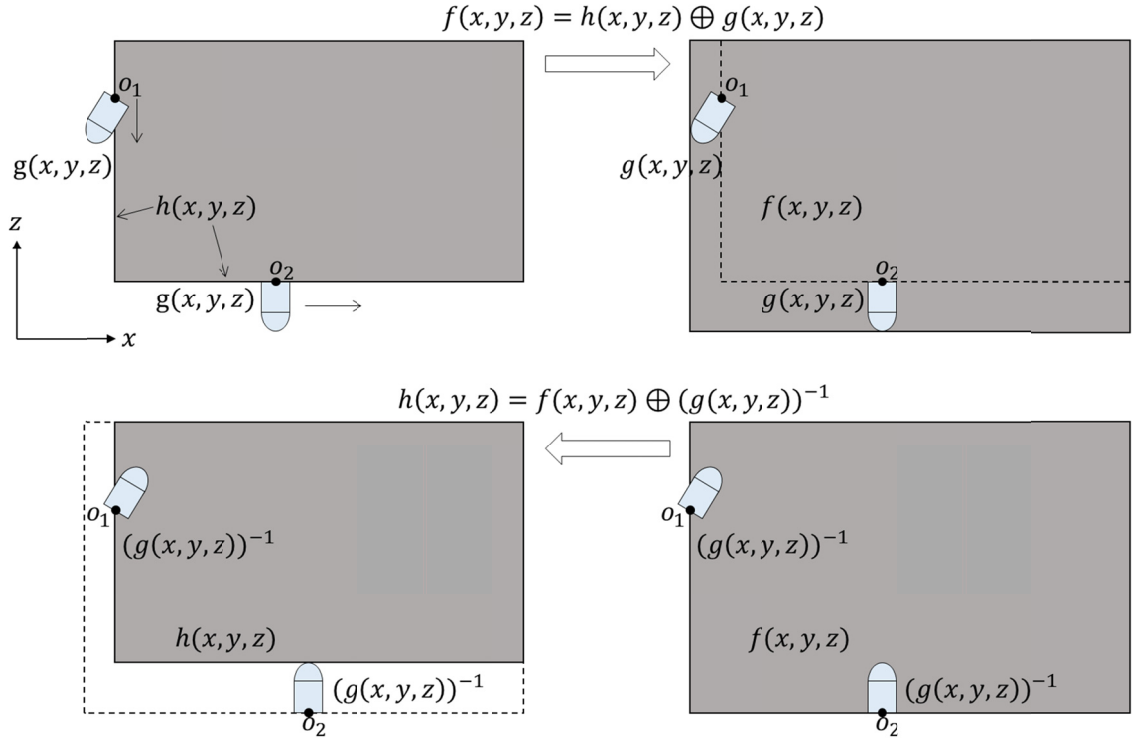


Figure 26. Illustration of reciprocal set for Boolean operations.

6.5. Manufacturability Analysis of 3D Models

Current Rapid Prototyping and Manufacturing (RP/M) service bureaus rely heavily on automatic online quoting systems (refer to examples such as www.zoomrp.com and www.quickparts.com). For such a business model, it is critical to identify any infeasible features that cannot be built by a selected additive manufacturing process. For example, any features whose thickness is smaller than 0.3mm cannot be built by the selective laser

sintering (SLS) process. In addition to infeasible features, it is also important to given a reasonable price based on analyzing the submitted CAD models.

A LDNIs-based manufacturability analysis method has been developed for the purposes [31]. The method can identify small features whose sizes are infeasible for a given AM process. To identify such infeasible features that may locate in any positions with any orientations, the LDNIs-based approach is mainly based on the geometric analysis of the following computations.

- *S infeasible to deposit for size r* as: $\Delta^{+}_{r}(S) = S - S\downarrow_{r}\uparrow_{r}$, and
- *S infeasible to remove for size r* as: $\Delta^{-}_{r}(S) = S\uparrow_{r}\downarrow_{r} - S$,

where '–' is a Boolean operation (subtraction), and '$\uparrow_{r}$' and '$\downarrow_{r}$' are the offsetting operations (growing and shrinking by *r*, respectively).

Note the computed features in $\Delta^{+}_{r}(S)$ and $\Delta^{-}_{r}(S)$ are only determined by the size *r*, with no relations to their positions and orientations. Also the mathematical definitions of $\Delta^{+}_{r}(S)$ and $\Delta^{-}_{r}(S)$ are precise, that is, only the features whose sizes are smaller than 2×*r* will be identified. Features whose sizes are equal or bigger than 2×*r* will be skipped.

A manufacturability analysis example of 2D contours is shown in Figure 27. The computation results of $\Delta^{-}_{r}(S)$ for the machining process are illustrated. As shown in the figure, the input contour is sliced from a 3D model. It has a size of 1.2×0.65. For different tool sizes, the features that cannot be faithfully fabricated are identified as $\Delta^{-}_{r}(S)$, which are visualized in red regions for different sizes *r* in Figure 27.a-d.

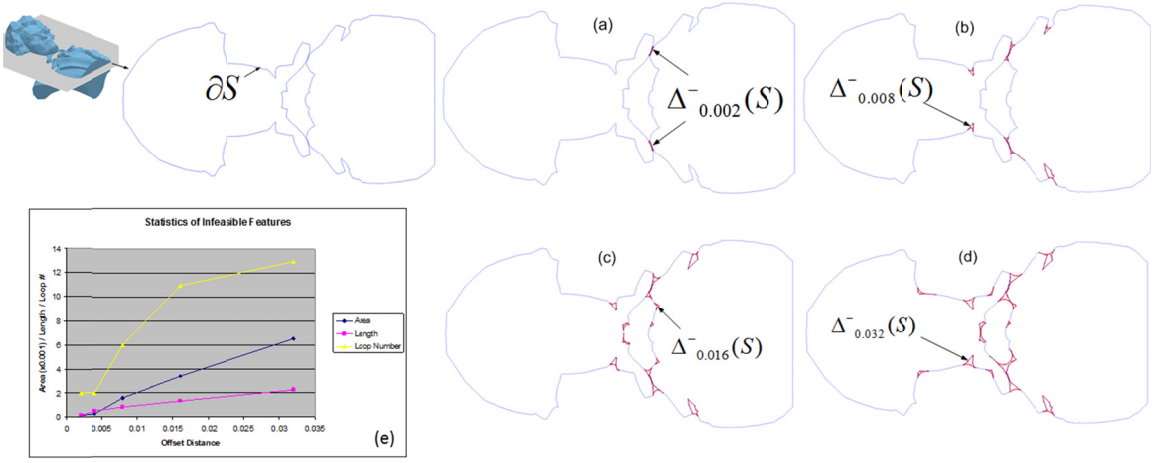


Figure 27. Infeasible features of a 2D polygonal model for different tool sizes and their statistics.

More information of such infeasible features can be computed for cost estimation. Figure 27.e shows a statistics of such infeasible features including loop number, loop lengths and area sizes. An accurate cost estimation of a given CAD model is critical for the web-based rapid prototyping. However, current cost estimation approaches are mainly based on parameters such as the volume and the extent size of a part. Although they are easy to compute, the shape complexity of a given part such as the number of small features has not been considered. However, the information of such small features can be critical for cost estimation since they may significantly affect the building time, and a user may be willing to pay a higher price for a part with more complex features.

7. Summary and Outlook

Robust and efficient geometric analysis and computation is critical for three-dimensional microfabrication and other computer-aided design and manufacturing applications. Current geometric analysis and computation methods based on the boundary representation explicitly define and compute geometry. While being accurate, such approaches lack in simplicity and are prone to robustness problems, especially for complex geometries and geometric operations that require radical topological changes. In this chapter, a geometric analysis and computation method has been presented based on the Layered Depth-Normal Images. To convert between the LDNIs and B-rep models, a set of computation algorithms have been developed. Accordingly, several geometric operations, including offsetting and Boolean, have been implemented based on the LDNIs-based method. The test results illustrate that, although such operations all require radical topological changes, the LDNIs-based computation framework and related algorithms are robust for complex geometry. The efficiency of the LDNIs-based method has also been addressed by developing a parallel computational approach based on technologies such as graphics hardware. Different applications based on the developed computational framework have been discussed for complex component design and manufacturing. Test results have demonstrated the LDNIs-based geometric computation method can be beneficial for 3D microfabrication and other CAD/CAM applications.

Acknowledgement

The work is partially supported by the National Science Foundation grant CMMI-0927397.

References

[1] J. R. Rossignac and A. A. G. Requicha, "Offsetting Operations in Solid Modelling," *Computer Aided Geometric Design,* vol. 3, no. 2, pp. 129-148, Augest 1986.

[2] J. P. Menon and H. B. Voelcker, "On the completeness and conversion of ray representations of arbitrary solids," in *the Third ACM Symposium on Solid Modeling and Applications*, New York, 1995.

[3] L. A. Piegl, "Ten challenges in computer-aided design," *Computer-Aided Design,* vol. 37, no. 4, pp. 461-470, 2005.

[4] J. Winkler and M. Niranjan, Uncertainty in Geometric Computations, Springer, 2002.

[5] S. Schirra, "Precision and Robustness in Geometric Computations," in *Algorithmic Foundations of Geographic Information Systems*, Springer Berlin Heidelberg, 1997, pp. 255-287.

[6] K. Ouchi and J. Keyser, "Handling degeneracies in exact boundary evaluation," in *the ninth ACM symposium on Solid modeling and applications*, Switzerland, 2004.

[7] Y. Chen, H. V. Wang, D. W. Rosen and J. R. Rossignac, "Filleting and Rounding Using a Point-based Method," in *ASME International Design Engineering Technical Conferences*, Long Beach, CA, 2005.

[8] G. Varadhan and D. Manocha, "Accurate Minkowski Sum Approximation of Polyhedral Models," *Graph. Models,* vol. 68, no. 4, pp. 343-355, 2006.

[9] D. Pavic and L. Kobbelt, "High-Resolution Volumetric Computation of Offset Surfaces with Feature Preservation," *EUROGRAPHICS,* vol. 27, no. 2, 2008.

[10] J.-M. Lien, "Point-based Minkowski Sum Boundary," in *Pacific Conference on Computer Graphics and Applications*, Maui, Hawaii, 2007.

[11] U. Tiede, T. Shiemann and K. Hoehne, "High quality rendering of attributed volume data," in *IEEE Visualization*, 1998.

[12] M. Jones, J. Baerentzen and M. Sramek, "3D distance fields: a survey of techniques and applications," *IEEE Transactions on Visualization and Computer Graphics,* vol. 12, no. 4, pp. 581-599, 2006.

[13] L. Kobbelt, M. Botsch, U. Schwanecke and H.-P. Seidel, "Feature sensitive surface extraction from volume data," in *SIGGRAPH*, 2001.

[14] Y. Chen, "An accurate sampling-based method for approximating geometry," *Computer-Aided Design,* vol. 39, no. 11, pp. 975-986, 2007.

[15] T. Ju, F. Losasso, S. Schaefer and J. Warren, "Dual contouring of Hermite data," *ACM Transactions on Graphics,* vol. 21, no. 3, pp. 339-346, 2002.

[16] L. Kobbelt and M. Botsch, "An interactive approach to point cloud triangulation," *Computer Graphics Forum (Eurographics),* vol. 19, no. 3, pp. 479-487, 2000.

[17] J. Menon and H. Voelcker, "On the completeness and conversion of ray representations of arbitrary solids," in *ACM Symposium on Solid Modeling and Applications*, 1995.

[18] E. Hartquist, J. Menon, K. Suresh, H. Voelcker and J. Zagajac, "A computing strategy for applications involving offsets, sweeps, and Minkowski operations," *Computer-Aided Design,* vol. 31, no. 3, pp. 175-183, 1999.

[19] C. C. L. Wang, Y.-S. Leung and Y. Chen, "Solid modeling of polyhedral objects by Layered Depth-Normal Images on the GPU," *Computer-Aided Design,* vol. 42, no. 6, pp. 535-544, 2010.

[20] Y.-S. Leung and C. C. L. Wang, "Conservative sampling of solids in image space," *IEEE Computer Graphics and Applications,* vol. 33, no. 1, pp. 14-25, 2013.

[21] C. C. L. Wang and D. Manocha, "GPU-based offset surface computation using point samples," *Computer-Aided Design,* vol. 45, no. 2, pp. 321-330, 2012.

[22] Y. Chen and C. C. L. Wang, "Layered Depth-Normal Images for complex geometries - part one: accurate sampling and adaptive modeling," in *ASME 28th IDETC/CIE Conference*, New York, 2008.

[23] C. C. L. Wang and Y. Chen, "Layered Depth-Normal Images for complex geometries - part two: manifold-preserved adaptive contouring," in *ASME 28th IDETC/CIE Conference*, New York, 2008.

[24] M. Mortenson, Geometric Modeling (2nd Ed.), New York: Wiley, 1997.

[25] B. Heidelberger, M. Teschner and M. Gross, "Volumetric collision detection for deformable objects," Technical Report No.395, Computer Science Department, ETH Zurich, April 2003.

[26] R. J. Wright and B. Lipchak, OpenGL Superbible, Indianapolis: Ind.:SAMS, 2007.

[27] Y. Chen and C. C. L. Wang, "Regulating complex geometries using Layered Depth-Normal Images for rapid prototyping and manufacturing," *Rapid Prototyping Journal,* vol. 19, no. 4, pp. 253-268, 2013.

[28] Y. Chen and C. C. L. Wang, "Contouring of structured points with small features," in *ASME 30th IDETC/CIE Conference*, Canada, August 15-18, 2010.

[29] Y. Chen and C. C. L. Wang, "Uniform offsetting of polygonal model based on Layered Depth-Normal Images," *Computer-Aided Design,* vol. 43, no. 1, pp. 31-46, 2011.

[30] Y. Chen, "Non-uniform Offsetting for Laser Path Planning of Solid Freeform Fabrication Machines," in *Solid Freeform Fabrication Symposium*, Austin, Texas, August 6 ~ 8, 2007.

[31] Y. Chen and X. Xu, "Manufactruability Analysis of Infeasible Features in Polygonal Models for Web-Based Rapid Prototyping," in *International Conference on Manufacturing Automation (ICMA), pp.120,127, 13-15*, Hong Kong, Dec. 2010.

[32] Y. Li and Y. Chen, "Five-Axis Manufacturing Simulation Based on Normal Arc Mapping and Offset Volume Computation," in *ASME 30th IDETC/CIE Conference*, Montreal, Quebec, Canada, 2010.

[33] S. Kalpakjian and S. Schmid, Manufacturing Engineering & Technology (6th Edition), Prentice Hall, 2009.