Proceedings of the ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference
August 3-6, 2008, Brooklyn, New York, USA

DETC2008/CIE-49432

LAYER DEPTH-NORMAL IMAGES FOR COMPLEX GEOMETRIES - PART ONE: ACCURATE MODELING AND ADPATIVE SAMPLING

Yong Chen
University of Southern California
Los Angeles, CA 90089
yongchen@usc.edu

Charlie C. L. Wang
The Chinese University of Hong Kong
Shatin, N. T., Hong Kong, China
cwang@mae.cuhk.edu.hk

ABSTRACT

The layered depth-normal images (LDNIs) is an implicit representation of solid models that sparsely encodes the shape boundary in three orthogonal directions. We present a LDNI-based geometric modeling method for applications with high accuracy requirements. In our method, we first construct LDNIs models from input polygonal models. The accuracy of the generated LDNIs models can be controlled by setting pixel width during the construction process. Even for very complex geometries and high accuracy requirements, the construction process is fast with the aid of graphics hardware. Based on the LDNIs models, we then perform geometric modeling operations. Two types of operations are presented including regularizing and Boolean operations. The geometric modeling operations are straightforward and easy to be implemented robustly. From the processed LDNIs model, an adaptive sampling method is presented to construct a cell representation that includes both uniform and octree cells. Finally 2-manifold polygonal mesh surfaces are constructed from the cell representation. For high accuracy requirements that are typical in CAD/CAM applications, we present a volume tiling technique and a parallel implementation to accelerate the computation. Our method achieves a good balance between the accuracy and computational resources. We report experimental results on a variety of CAD models. The results demonstrate the effectiveness and efficiency of our approach especially for modeling complex geometries.

KEYWORDS: geometric modeling, implicit representation, Boolean operation, adaptive sampling, cell contouring, parallel computation.

1 Introduction

The boundary representation (B-rep) is the most popular representation of 3-dimensional geometry for CAD/CAM applications. Commercial geometric kernels, such as *ACIS* and *Parasolid*, are widely used in CAD/CAM applications. Both of them are based on the B-rep. The geometric operations based on the boundary representation have been extensively studied. For example, Hoffmann [1] studied the Boolean operations based on calculating exact surface intersections and classifying geometric elements into Booleaned combinations. Rossignac and Requicha [2] presented the offsetting operations based on trimming offset geometric elements to construct offset solids. While the approach based on intersection calculation and directly manipulating the boundary representations is accurate, it lacks in simplicity and is prone to robustness problems [3]. Especially for complex geometries that are the focus of the paper, the geometric operations based on the B-rep are even more challenging. With thousands or even millions of polygons in a model, it would be rather difficult to detect and control the degenerated cases during the geometric operations.

In recent years, using the volumetric representation to approximate geometry has been widely studied in computer graphics [4-7]. Various volumetric representations have been proposed such as voxel [8], distance field [9], surfel [10], and ray-rep [11]. The geometric operations based on the volumetric representations are easy to implement and robust. However, geometric modeling based on the volumetric representations, such as voxels, is generally viewed as inaccurate approach. Therefore it is not widely accepted in CAD/CAM applications. The limited accuracy of the volumetric representations may not be a main issue for computer graphics applications whose main goal is to interactively display geometries. However, it is a major concern for most engineering applications, whose accuracy requirement is much higher.

In this paper, we try to achieve a balance between the required memory or computation time, and high accuracy or big model size. Our method makes significant improvements over our previous work [12, 13]. In [12], we presented a novel implicit representation named layered depth-normal images (LDNIs).

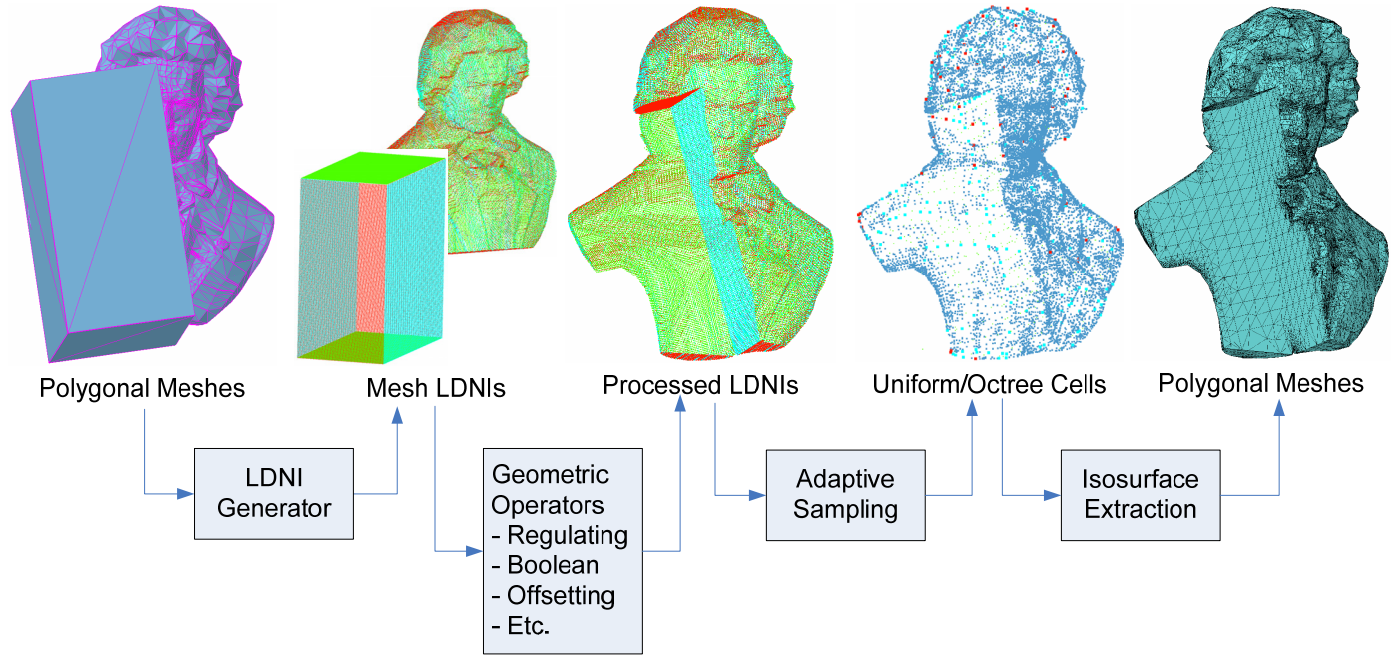

Figure 1: The framework of our LDNIs based geometric modeling method.

We also discussed the conversion between the LDNIs and B-rep models. However, the conversion technique presented in [12] works on uniform grids. Hence for high accuracy requirements, it generates an excessively large number of polygons and is very time-consuming. In [13], a sampling-based method is proposed for accurately approximating geometries defined by various geometric operations. The sampling points generated in the adaptive sampling process are dynamically calculated from the input B-rep models. Therefore, it can be quite time-consuming for complex geometries.

We present a geometric modeling method based on the LDNIs representation for high accuracy applications. The framework of our method is shown in Figure 1. By integrating the key ideas of [12] and [13], our method achieves a good balance between the accuracy and computational resources. Therefore, it is especially suitable for CAD/CAM applications.

The most important properties of our method are:

(1) **Accurate**: We construct the LDNIs based on a minimum feature size. We then adaptively subdivide cells based on a given tolerance. Therefore the reconstructed model is topologically equivalent to the exact surface, and the approximation error is bounded by the resolution of the LDNIs models.

(2) **Efficient**: Adaptive sampling enables us to use a higher resolution to refine only the cells that have complex geometry inside. Therefore we can use a limited number of polygons to achieve a rather high resolution.

(3) **Fast:** Several steps in our method can be easily parallelized. Therefore the computing time can be significantly reduced by using graphics hardware and a PC cluster.

(4) **Scalable**: We sub-divide an input model into multiple tiles. This allows us to process virtually an unlimited resolution since we can further sub-divide a model into more tiles and process them separately.

(5) **Simple**: The geometric operations based on the LDNIs representation are straightforward and easy to implement.

(6) **Capturing sharp features**: our contouring approach can capture sharp corners and edges in the geometry, which is important for engineering applications.

2 Related Work

Our work utilizes several types of implicit representations. The implicit representations become popular because of their simplicity and versatility in performing a wide variety of geometric operations. Many operations such as Boolean, offsetting, blending, and warping can be expressed elegantly using the implicit representations [14-16]. Consequently, the modeling techniques based on the implicit representations have been used in a large number of applications [17, 18].

Several implicit representations have the similar structures as the LDNIs. Menon and Voelcker [11] presented a ray-rep representation by sampling the solid models into parallel rays tagged with h-tag (i.e., the information of half-space at the endpoints of rays). Ray-rep can be used in offsets, sweeps, and Minkowski operations [19]. The ray-rep stores depth values without surface normals in one ray direction. Other similar representations are dexel [20] and Layered Depth Images (LDI) [21]. Heidelberger, *et al* [21] presented LDI in a fast collision detection approach for solid models. They also demonstrated the LDI decomposition can be accelerated in graphics hardware using OpenGL.

Our work converts the processed implicit models to polygonal models. This is because many applications, such as graphics displaying, rapid prototyping and CNC machining, require a parameterized patch representation of geometry. Extensive research has been done in this area. The marching cubes algorithm, proposed by Lorense and Cline [4], is a standard

approach to extract an isosurface from a volume raster of scalar values. Many extensions to the original marching cubes algorithm have been proposed to resolve ambiguities of certain cell configurations and generate topologically consistent iso-surfaces [22]. The original marching cubes algorithm is unable to extract high quality triangle meshes with sharp features. Several extensions have also been proposed to reconstruct sharp features and reduce aliasing artifacts in the reconstructed model [5, 6, 18]. To overcome an excessively large number of triangles required to represent the iso-surface, many methods have also been developed for performing iso-surface extracting adaptively using hierarchies such as octrees and k-D trees [6, 23, 24]. Most algorithms do not provide guarantees on the topology of the reconstructed surface. Some recent work addresses topology-preserving reconstruction by an enhanced cell representation [25] or by additional tests [13, 24].

The remainder of the paper is organized as follows. The accurate sampling of a polygonal model for constructing a LDNIs model is presented in Section 3. The geometric modeling operations based on the LDNIs representation are discussed in Section 4. The adaptive sampling and contouring for constructing polygonal mesh surfaces are presented in Section 5. A parallel computing framework for our geometric modeling method is presented in section 6. A brief discussion of the algorithm performance is also given in the section. The results of four test cases are presented in Section 7. Finally, conclusions and future work are given in Section 8.

3 Accurate Sampling with LDNIs

3.1 LDNIs: An Implicit Representation of Solids

The Layered Depth-Normal Images (LDNIs) is an extension of the Layered Depth Images (LDIs) to sparsely encode the shape of solid models [12]. A structural set of Layered Depth-Normal Images (LDNIs) consists of x-LDNI, y-LDNI and z-LDNI along *X*, *Y*, and *Z* axis respectively (referred to as *axis 1*). 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 *axis 1* is a two-dimensional image with $w_{axis2} \times w_{axis3}$ pixels. For each pixel (*i*, *j*), we shoot a ray from its center along *axis 1* and calculate the intersections of the ray and the surfaces under sampling. Consequently for each pixel (*i*, *j*), we can build a sequence of four-components nodes (d, n_x, n_y, n_z), where *d* specifies the depth from an intersection point *P* to the viewing plane, and $N_P(n_x, n_y, n_z)$ is the surface normal at *P*.

The main difference of the LDNI from the LDI is the recording of surface normal $N_P(n_x, n_y, n_z)$. The surface normal is important not only for adaptive sampling and isosurface extraction (refer to Section 5), but also for the inside/outside judgment along the ray (refer to Section 4.1).

The basic definitions of the LDNIs are given in [12]. Some additional definitions and propositions are given as follows.

Definition 1 A normal index number I_{Norm} is an accumulated integer value along a ray such that: (1) I_{Norm}= 0 at the starting point if it is outside the model; (2) I_{Norm}= 1 at the starting point if it is inside the model; (3) from the starting point, for any intersection point *P* along the ray with unit normal $\boldsymbol{N_{ray}}$, increasing I_{Norm} by 1 if $N_p \cdot N_{ray} < 0$ and decreasing I_{Norm} by 1 if $N_p \cdot N_{ray} > 0$.

For two-manifold solid models, we know:

Proposition 1 For a Layered Depth-Normal Image sampled from a two-manifold solid model, a point *P* on a ray of a pixel is inside the model if $I_{Norm}(P) > 0$; otherwise, it is outside the model.

Proposition 2 For a Layered Depth-Normal Image sampled from a two-manifold solid model, the number of nodes on a pixel should be even; in addition, I_{Norm} should be 0 at both d_{min} and d_{max} of a ray (i.e. outside the model).

The proofs of the above two propositions are straightforward.

Therefore, an input solid model can be implicitly defined by a LDNIs model, which captures all the boundary information of the solid along pre-defined uniform grids. In addition, for any point along the grids, we can quickly judge whether it is inside/outside the model.

3.2 Constructing LDNIs with Graphics Hardware

For any given polygonal model, a LDNIs model can be constructed rather quickly by using graphics hardware [12]. The construction of a LDNI model is similar to the well-

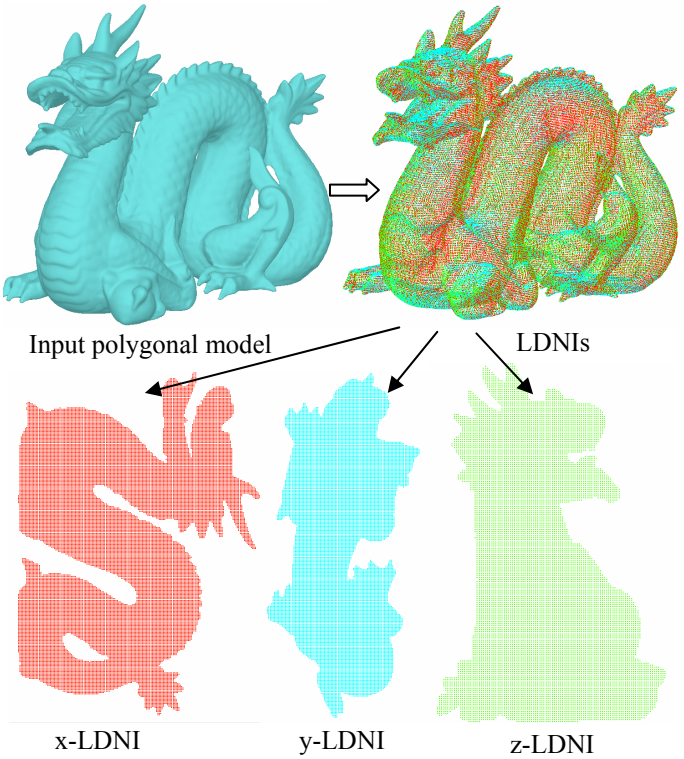

Figure 2: An example of the constructed LDNIs model from a polygonal model.

known scan-conversion algorithm. The viewing parameters are determined by the working envelope, which is slightly larger than the bounding box of the model. An orthogonal projection is conducted for rendering so that the intersection points from parallel rays can be generated. In order to get an accurate surface normal, we encode a unique ID of every polygonal face into a RGB-color. After rendering all the faces by the encoded colors, we can easily identify a face and accordingly retrieve its surface normal from the input model.

Modern graphics hardware is very efficient at manipulating and displaying polygonal models. Similar to the sampling of the LDIs [21], the model under sampling has to be rendered multiple times in order to generate a LDNI with the help of graphics hardware. Based on highly parallel structures, graphics hardware can construct a LDNI very quickly even for a rather complex model. For example, by using *NVIDIA* GeForce 8800 GT, we can construct a LDNIs model (256×256) from a polygonal model as shown in Figure 2 (triangle number = 723,708) within 3.7 seconds (x-LDNI: constructed in 1.3 sec with 58,466 nodes generated; y-LDNI: constructed in 1.2 sec with 49,835 nodes generated; z-LDNI: constructed in 1.2 sec with 46,628 nodes generated).

3.3 Pixel Width and Accuracy of LDNIs

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 we use the graphics hardware to construct the corresponding LDNIs model, the minimum pixel width $\delta = \frac{Ext_{\max} - Ext_{\min}}{w-1}$, where w is the maximum image resolution available in the rendering (e.g. w is 1024 for the graphics hardware with a resolution of 1280×1024).

Remark 1 Using pixel width δ to sample a solid model, a gap or thin-shell on the solid model whose thickness is less than δ may be missed in the Layered Depth-Normal Image that is perpendicular to the gap or the thin-shell.

Therefore, we need to have three orthogonal Layer Depth-Normal Images to ensure the other two LDNIs can capture the features.

Remark 2 Using pixel width δ to sample a solid model, a feature whose overall size is less than δ may be missed in all three orthogonal Layered Depth-Normal Images.

Therefore, for a given model, if the minimum feature size that we are interested in is θ, we must set $\delta < \theta$ to ensure the LDNIs model can capture all the features that are bigger than θ in the given model.

In CAD/CAM applications, the minimum feature size θ is determined by the capability of a manufacturing process that will be used to fabricate the CAD model. A manufacturing process has a limited capability on the minimum feature size due to various limitations. For example, the rapid prototyping processes built physical models layer by layer. Therefore, the smallest feature size in Z direction is limited by the layer thickness. Consequently even infinitely small features can be represented in a CAD model, these features are meaningless in the physical world since they can not be fabricated. Therefore, we can set δ based on the capability of a manufacturing process.

Remark 3 For a manufacturing process whose minimum feature size is θ, we can set pixel width $\delta < \theta$ in constructing a LDNIs model to ensure the accurate sampling of a CAD model for the manufacturing process.

3.4 Volume Tiling of LDNIs

Suppose θ = 0.1 mm and the image resolution w = 1024. The maximum model size that can be processed is $Ext_{\max} - Ext_{\min} \le \delta \times (w-1)$ = 102mm. For a bigger size model or a higher accuracy requirement, we can use a technique called volume tiling. That is, we split the bounding box of the model into smaller tiles. We then process each tile independently (either sequentially or in parallel) and construct their LDNIs models respectively.

The changes required in the LDNIs representation for volume tiling include: (1) we record Ext_{LDNI_min} and Ext_{LDNI_max} in each LDNIs model. They are different from the minimum and maximum extent of a given model. (2) In each pixel (i, j) of a LDNI, we record the normal index number I_{Norm} at the starting point defined by Ext_{LDNI_min}. For volume tiling, the same approach as described in Section 3.2 can be used in constructing the LDNIs model of each tile. The only difference in the OpenGL display is that we zoom in the input polygonal model to a target Ext_{LDNI_min} and Ext_{LDNI_max}. We also save I_{Norm} at Ext_{LDNI_min} for each pixel. Note the viewing plane is still set by Ext_{min} since we know I_{Norm}= 0 at Ext_{min}. We calculate all the sampling points from the rendering process and only record the nodes that are inside the extent

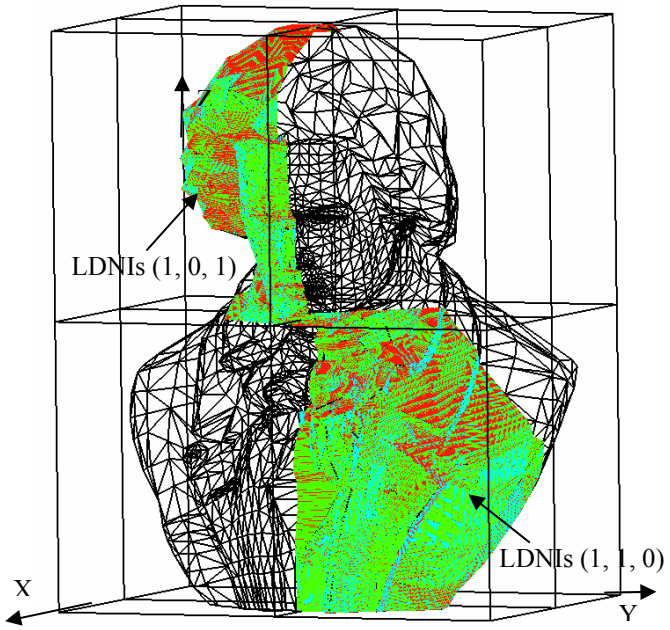

Figure 3: A volume tiling example.

(Ext_{LDNI_min}, Ext_{LDNI_max}] in the constructed LDNI.

An example of volume tiling is shown in Figure 3. The size of the model is 7.3×5.8×4.2 and the pixel width δ is 0.005. For such a case, the required pixel resolution is 1460×1160×840. Suppose the graphics hardware has an image resolution of 800×800. We can divide the model into 2×2×2= 8 tiles. The LDNIs of tile (1, 0, 1) and tile (1, 1, 0) are shown in Figure 3 (with 805,426 and 855,906 nodes generated respectively).

The volume tiling also solves the memory problem in constructing LDNIs models, which is discussed as follows.

Proposition 3 The memory complexity of a LDNI is $O(w^2)$, where w is the image resolution.

Therefore, the number of nodes in a LDNI increases by four times if the accuracy requirement is doubled. Hence, the required memory of a LDNIs model can quickly grow above the memory limitation of a commodity PC. The volume tiling can significantly reduce the amount of information that requires being stored simultaneously.

In addition, as shown in Section 6, the generated tiles can be processed separately and in parallel by using a cluster of PCs.

4 Accurate Modeling on LDNIs

The LDNIs is an implicit representation which has intuitive mathematical definitions. Its data is structurally defined on uniform grids. Therefore, the solid modeling operations developed on the LDNIs representation are easy to implement.

4.1 Eliminating Self-Intersections

Two-manifold polygonal models may have self-intersections generated by some geometric operations. For example, polygonal offsetting by simply offsetting vertices may lead to self-intersections [2]. Three types of self-intersections are common, which include: (1) external loop overlapping; (2) internal loop overlapping; and (3) small loop twisting (refer to *A*, *B* and *C* in Figure 4). The self-intersections in a CAD model may bring problems to a computer-aided manufacturing system. Various approaches have been proposed for removing the self-intersections. For example, an approach based on the B-rep needs to calculate the intersections of all the geometric elements and then judge the validity (inside/outside) of each element.

Based on the LDNIs representation, we can eliminate self-intersections according to the calculated normal index number I_{Norm} (refer to Definition 1). That is, for a LDNIs model, we process its x-LDNI, y-LDNI and z-LDNI separately (sequential or in parallel). For each LDNI, we go through each pixel (i, j) and calculate the I_{Norm} along the ray. We then use the I_{Norm} value of each node to delete all the nodes that correspond to the self-intersections, and store only the nodes that correspond to the real boundary.

Proposition 4 A node P along a ray is valid if and only if the values of I_{Norm} before and after P are 0 and 1, or 1 and 0.

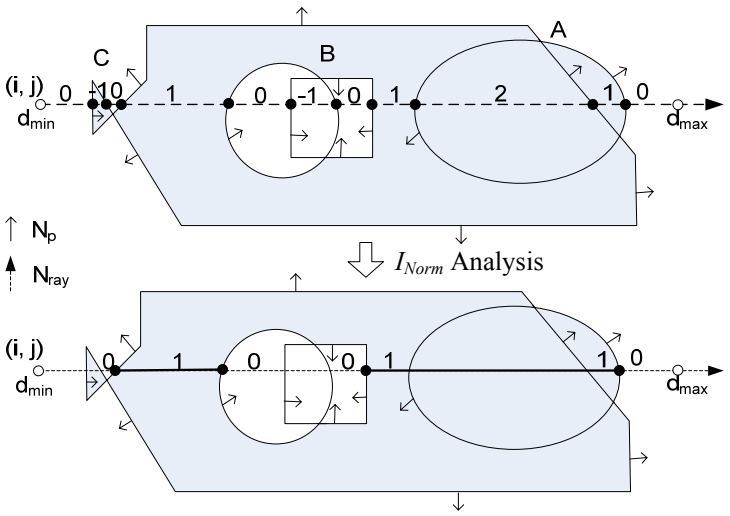

Figure 4: An illustration of self-intersections and our approach based on I_{Norm} to remove them in a LDNI.

An illustrative example of self-intersections is shown in Figure 4 with the three types of self-intersections: (1) external loop overlapping (refer to *A*); (2) internal loop overlapping (refer to *B*); and (3) small loop twisting (refer to *C*). The normals N_{ray} and N_p are also shown in the figure for a ray. Based on Definition 1, we can calculate I_{Norm} along the ray. The calculated I_{Norm} values along the ray are also shown in Figure 4-top. The nodes whose two neighboring I_{Norm} values are 0 and 1, or 1 and 0 define the boundary that separates the inside and outside of the model (refer to Figure 4-bottom). Therefore, we can construct a regularized LDNIs model with only the valid nodes. From such as a LDNIs model, as shown in Section 5, we can then construct a polygonal model that contains no self-intersections. A more complicated test example is given in Section 7 (Test 2).

4.2 Boolean Operations

The Boolean operations are the fundamental modeling operations. They are widely used in various CAD/CAM applications. A Boolean operation Π, such as union, intersection or difference, is a well-defined set operation. The Boolean operations based on the LDNIs representation are straightforward and easy to implement [12].

Suppose the two input LDNIs models are $LDNIs_1$ = {x-$LDNI_1$, y-$LDNI_1$, z-$LDNI_1$} and $LDNIs_2$ = {x-$LDNI_2$, y-$LDNI_2$, z-$LDNI_2$}. $LDNIs_1$ and $LDNIs_2$ must have the same origin, pixel width and bounding box size. Their Boolean operations can be defined as:

$LDNIs_1$ Π $LDNIs_2$ = {x-$LDNI_1$ Π x-$LDNI_2$, y-$LDNI_1$ Π y-$LDNI_2$, z-$LDNI_1$ Π z-$LDNI_2$}; and

x/y/z-$LDNI_1$ Π x/y/z-$LDNI_2$ = $\sum_{i,j}(ray_1(i,j)\Pi ray_2(i,j))$.

Ray_1 and ray_2 correspond to the same pixel (i, j) and locate at the same position. Along the ray, we can calculate $I_{Norm1}(P)$ and $I_{Norm2}(P)$ for any given point P. Accordingly we can determine if $P \in (LDNIs_1$ Π $LDNIs_2)$. Hence the Boolean

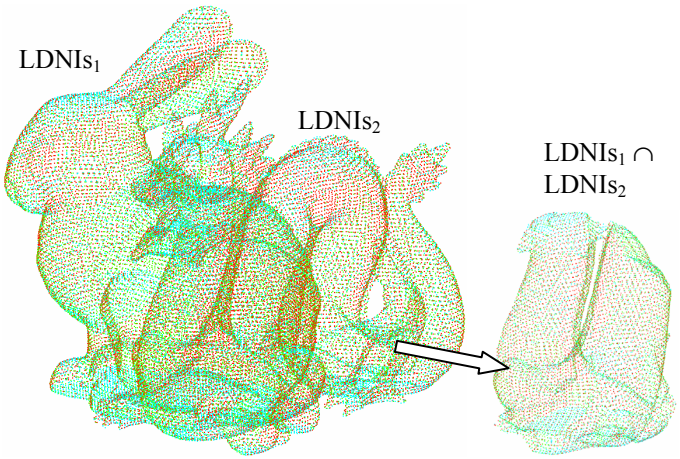

Figure 5: An intersection example between two LDNIs models.

operations on the LDNIs are converted into the Boolean operations on 1-Dimensional segments. Based on the definition of Π, we can select appropriate nodes from ray_1 or ray_2 to construct a Booleaned LDNIs model.

The Boolean operations on the LDNIs models can be performed rather quickly since the judgment on each pixel is quite simple. An intersection example of a bunny and a dragon is shown in Figure 5, which is used in Test 1 (refer to Figure 10). Both LDNIs models have the same resolution (144×142×172). It takes less than 0.1 second to calculate the Booleaned LDNIs model from them.

4.3 Accuracy of Generated LDNIs

The LDNIs representation is based on uniform grids. It has well structured data representation. In the process of eliminating self-intersections and Boolean operations, we make no changes to its structures. So the generated LDNIs model has the same structure as the input LDNIs models. In the operations, we judge the properties of each node and accordingly keep or remove it in the result. Therefore, we know:

Proposition 5 The LDNIs model generated by the aforementioned geometric operations has an approximation error that is smaller than pixel width δ.

5 Adaptive Sampling for Boundary Representation

The LDNIs is an implicit representation that can accurately capture a solid defined by solid modeling operations. However, most computer-aided manufacturing systems, such as rapid prototyping machines, require polygonal meshes as the input. Furthermore, the input model needs to be water-tight without mesh defects such as gaps, holes, or self-intersections. So we need to convert a LDNIs model into a polygonal model for the manufacturing purpose.

In [12], a contouring algorithm to construct two-manifold surfaces directly from a LDNIs model was presented. However, limited image resolutions were used since the triangle number of a reconstructed polygonal model increases in $O(w^2)$, where w is the image resolution of the input LDNIs model. For a big w, the constructed polygonal model has a large number of triangles, most of which are extremely small. Therefore, it is generally not practical to directly construct contours from a highly accurate LDNIs model.

In this section, we present an adaptive sampling method and related contouring techniques for converting a highly accurate LDNIs model into a water-tight polygonal model. The constructed polygonal model is efficient by allocating more triangles in the regions with higher curvatures.

5.1 An Adaptive Cell Representation

In this research, we first construct another type of implicit representation, cell representation [26], from a LDNIs representation. We then construct a water-tight polygonal model from the cell representation. As shown in Section 3, we use a very small sampling size δ in constructing a LDNIs model. We determine δ by the smallest feature size of a solid in order to capture all the features. However, most features are much bigger than δ. Hence the sampling size δ used in the LDNIs model is too small for them. In our method, we use an adaptive cell representation, which includes two types of cells, uniform cells and octree cells [26]. The uniform cells are used for rough sampling; based on it, we then use octree cells to refine the uniform cells which have complex geometry inside.

A detail description of the cell representation is presented in [26]. Note the cells need to be aligned with the LDNIs model. Suppose the pixel width of a LDNIs model is δ. We set the uniform cell size $\gamma = 2^k \times \delta$, where k is the maximum subdivision number of an octree cell. Based on γ and the extents Ext_{LDNI_min} and Ext_{LDNI_max}, the uniform cells can then be calculated. A 2-dimensional illustration of a LDNIs model and the two types of cells is shown in Figure 6. In the figure, we set $k = 3$ so $\gamma = 8 \cdot \delta$. Obviously, we reach the sampling resolution of the LDNIs model when a uniform cell is subdivided by k times. Further subdivision is meaningless

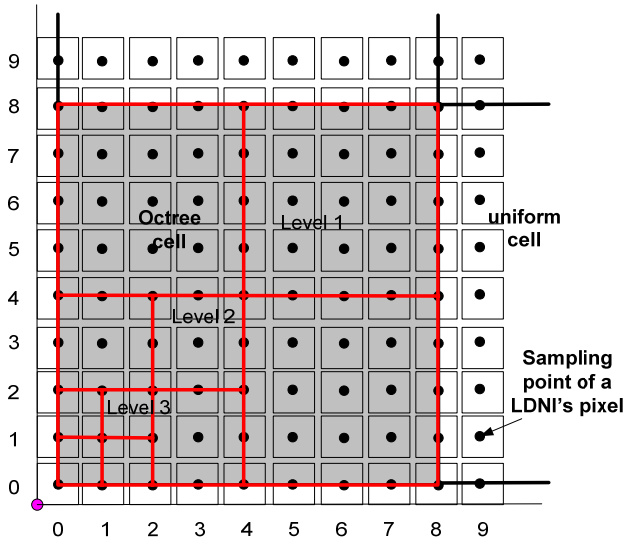

Figure 6: An illustration of the alignment between a LDNIs model and the related uniform and octree cells. In the example, k=3. Each uniform cell is 8 times of pixel width.

since there will be no samples stored in the input LDNIs model.

5.2 Adaptive Sampling Criteria

To construct a cell representation from a LDNIs representation, we need an adaptive sampling approach. In this research, we extend an accurate sampling-based method [13] from a polygonal model to a LDNIs model.

A boundary cell C_k is a cell which has some boundary points inside it. If a cell has no sampling points inside it, it is either an inside or outside cell. There is no need to further subdivide an inside/outside cell. Therefore we only need to consider a boundary cell C_k. Suppose ε is a given approximation error tolerance. Our approach to judge if C_k needs to be subdivided is based on an adaptive sampling test given as follows.

Adaptive sampling test:

(1) Calculate an *error-minimizing* point $\boldsymbol{v_c}$ based on the quadric error function (QEF) of all the points $\boldsymbol{v_i}$ (i = 1,…, k) in C_k. If no $\boldsymbol{v_c}$ is found, return *failed*;

(2) Topological test of $\boldsymbol{v_c}$ to ensure the iso-surface within each cell is topologically equal to a simple disk:
   a. If $\boldsymbol{v_c}$ is outside C_k, return *failed*;
   b. Check normals of $\boldsymbol{v_i}$ (i = 1,…, k) and normal of $\boldsymbol{v_c}$, if normal $\boldsymbol{N_{vc}}$ is invalid or there is a normal flip ($\boldsymbol{N_{vi}} \bullet \boldsymbol{N_{vc}} < 0$), return *failed*;

(3) Geometric tests of $\boldsymbol{v_c}$ based on the geometric error between an approximation and the exact surfaces: Check the distance d_i from $\boldsymbol{v_c}$ to a plane defined by the position and normal of $\boldsymbol{v_i}$(i = 1,…, k), if $d_i > \varepsilon$, return *failed*.

(4) Save $\boldsymbol{v_c}$ as an *error-minimizing* point for C_k and return *succeeded*.

If the adaptive sampling test fails, the cell has complex geometry inside it. Since the current sampling resolution is not sufficient to capture the geometry, we need to subdivide C_k and apply the test recursively.

An adaptive subdivision algorithm based on the adaptive sampling test is presented in [13]. In the algorithm, there are two major cell queries, *Corner_Sign_Query* and *Get_Sampling_points*. *Corner_Sign_Query* determines the *inside*/*outside* sign of a cell corner. *Get_Sampling_Points* returns a set of sampling points $\boldsymbol{v_i}$ (both position and normal) inside a cell. Both of them can be easily implemented on the LDNIs representation. That is, since a cell corner must lie on a ray (refer to Figure 6), we can easily calculate its I_{Norm} to determine its inside/outside sign. Similarly we can identify nodes that are inside a cell (including its boundary) and return them as the sampling points.

Therefore, an adaptive cell representation can be constructed from a given LDNIs model.

5.3 Mesh Tiling for Multiple LDNIs

After a cell representation is constructed, we use a modified dual contouring method for reconstructing polygons [26, 27]. Unlike the marching cube algorithm, the dual contouring algorithm will not generate cracks for an adaptive grid with different grid sizes. Further, two strategies to generate manifold-preserved mesh surfaces are presented in [27] for overcoming the topology ambiguity that may occur inside the finest octree cells after the maximum subdivision.

As shown in Section 3.4, for big size models or high accuracy requirements, we split the bounding box of a model into smaller tiles and construct a LDNIs model for each tile (refer to Figure 3). We then construct a cell representation from the LDNIs model of each tile. Finally we use the contouring method to construct a piece of polygonal meshes in the tile.

It is critical to ensure that the independently generated meshes can be merged into a proper manifold model. We achieve this by adding a buffer region in the right, back and bottom sides of each tile. The width of the buffer region is one uniform cell size. Therefore, two neighboring tiles will overlap over the buffer region. A 2-dimensional illustration of a tile (i, j) and its buffer region is shown in Figure 7. The two neighboring tiles, (i+1, j) and (i, j+1), are also shown in the figure. They overlap ((i, j) by one layer of uniform cells.

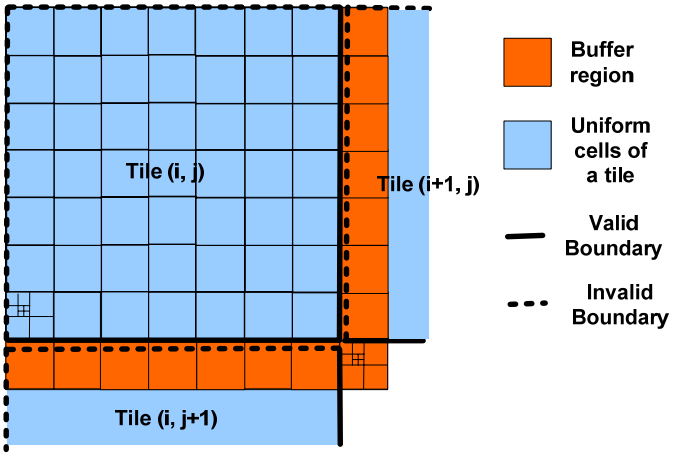

Figure 7: An illustration of the buffer region of a tile to ensure the mesh boundaries of neighboring tiles will match.

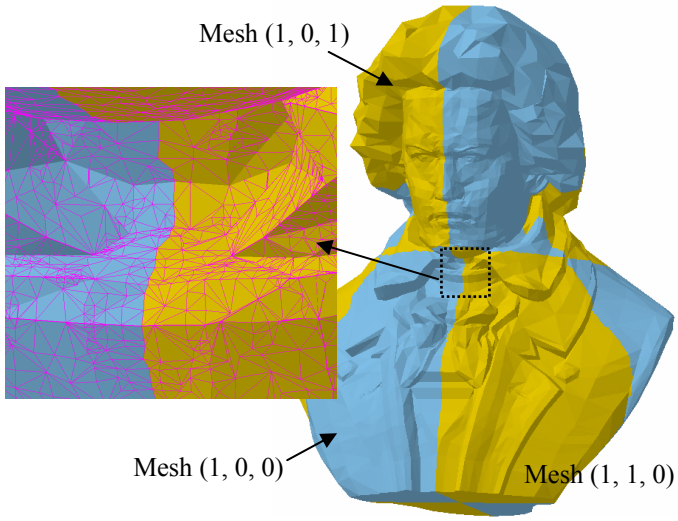

Figure 8: A mesh tiling example for the LDNIs in Figure 3.

We also define the boundary of a tile as (Ext_{LDNI_min}, Ext_{LDNI_max}]. So in the modified dual contouring method, we test each active edge of a cell before constructing a quad for it. If the edge is inside (Ext_{LDNI_min}, Ext_{LDNI_max}], it is a valid edge and we generate a corresponding quad or triangle for it; otherwise, it is an invalid edge and we simply do nothing. Consequently we can ensure the polygonal meshes of two neighboring tiles have no duplicate triangles.

After the polygonal meshes in all the tiles are generated, we can simply merge them together. Since the buffer region overlaps two neighboring tiles and the calculated error-minimizing points in the buffer region are used in reconstructing polygons of both tiles, the meshes in the two tiles have a common boundary. Hence the merged polygonal model is watertight. As an example, the constructed polygonal meshes for the LDNIs given in Figure 3 are shown in Figure 8. A magnified view of a portion of two neighboring meshes is also given in the figure. The merged polygonal model is valid with no holes or duplicate triangles.

5.4 Accuracy of Generated Meshes

In the adaptive sampling test, we calculate an error-minimizing point of a cell from all the sampling points in the cell from a LDNIs model. We also explicitly compare the approximation error with a given tolerance ε. If the approximation error is smaller than ε, we will use the calculated error-minimizing point in the contouring process; otherwise, we subdivide the cell until it reaches the finest level. At the finest level, a cell size is the same as the pixel width δ. Therefore, if we use the cell center to approximate all the features inside the cell, the approximation error in the contouring process will be less than δ. So if we set tolerance ε << δ, we know:

Proposition 6 The polygonal model generated by the aforementioned adaptive sampling and contouring process has an approximation error that is less than pixel width δ.

6 Parallel Computing Framework and Performance

A significant advantage of our method is the simplicity of parallelizing it. We develop a parallel computing framework for modeling complex geometries based on the LDNIs representation (refer to Figure 9). We discuss the computation in each step as follows. In addition, we briefly analyze their theoretical performance.

(1) In order to construct a LDNIs model from a polygonal model, the uniform sampling process goes through each polygon and use scan conversion algorithm to generate intersection points in three axes. Therefore the time complexity of this step is $O(N_{Tri})$ where N_{Tri} is the number of polygons. We use the graphics hardware in constructing LDNIs models. Based on highly parallel structure, the graphics hardware can construct a LDNIs model very quickly. In our current implementation, we use a PC with high-end graphics hardware (*NVIDIA* GeForce 8800 GT) to generate LDNIs models for input polygonal models. All the generated LDNIs models are saved in a network-connected hard disk,

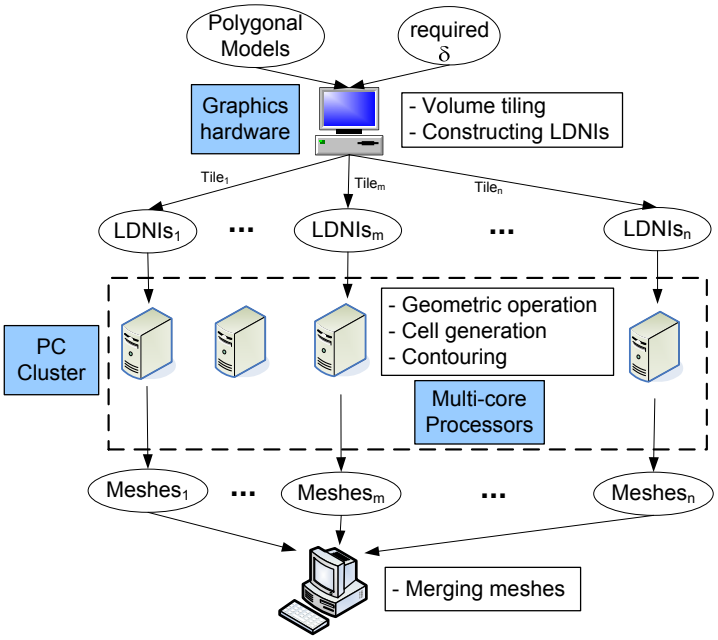

Figure 9: A parallel computing framework for modeling complex geometries using our LDNIs based method.

which can be accessed from a cluster of PCs. The performance of constructing the LDNIs models is very satisfactory, usually within seconds.

(2) The LDNIs model of each tile can be processed separately without other tiles' information. Therefore we can use a PC cluster to process all the tiles in parallel. The total running time is then determined by that of the most complex tile if a PC cluster has a sufficient number of machines. In our current implementation, we use command line to specify a tile number and the parameter values. This can be easily extended to a server-client based architecture.

(3) The geometric operations on the LDNIs representation go through each pixel of a LDNI to determine keeping or discarding the nodes. Hence the time complexity of this step is $O(w^2)$ where w is the pixel resolution. The contouring algorithm spends $O(1)$ time on each cell of the grid. Therefore, the time complexity of the contouring is $O(N_{Cell})$ where N_{Cell} is the number of cells (including both uniform and octree cells). Both the geometric operations and the contouring from the cell representation are rather fast (usually within a second on a PC with Pentinum 4 CPU 3.2 GHz).

(4) The time complexity of constructing a cell representation from a LDNIs representation is $O(N_{Node} \bullet k)$ where N_{Node} is the number of nodes in a LDNIs model and k is the maximum subdivision number. It is the most time-consuming step in our method and usually takes more than 60% of the total running time. Since the adaptive sampling test in constructing octree cells only requires the information related to the cell in the test, we can use a PC cluster to parallelize the program of cell generation. In addition, for each tile we can use multiple threads to process the octree cells stored in a stack. Therefore, the step can be performed much faster in a PC with multi-core processors. In our experience based on a PC with an *Intel*

Core2 Quad CPU Q6600 2.4GHz, the multi-thread implementation is general 2~3 times faster than a single thread implementation.

(5) The polygonal meshes generated separately for each tile can be merged into a polygonal model rather quickly (usually within a second). The time complexity of this step is $O(N_{Tri})$ where N_{Tri} is the triangle number of all the meshes.

We present some examples in Section 7 with our experimental results on the computing time.

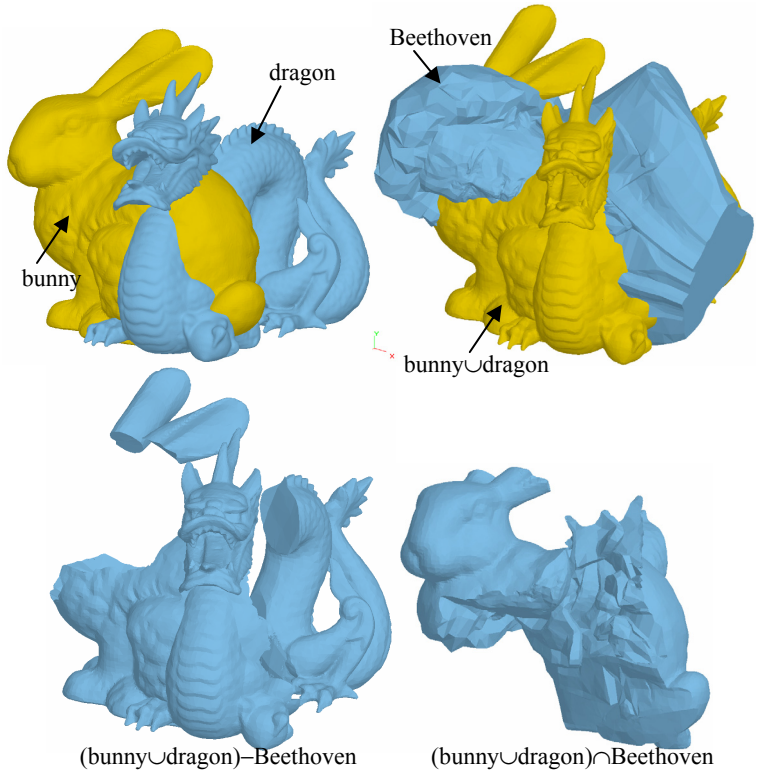

Figure 10: Results of test 1. The three input models and their relative positions are shown in the top; screen captures of the generated results are shown in the bottom.

Table 1: Running Results of Test 1.

Model	Input Tri #	Construct LDNIs (sec)	Geometric Operation (sec)	Adaptive Sampling & Contouring (sec)	Output Tri #
Bunny	69,664	5.3	0.2 (∪)	7.7	143,696
Dragon	723,708	8.6			
Union	143,696	5.4	0.3 (–)	8.6	136,978
Beethoven	5050	3.8			
Union	143,696	5.4	0.3 (∩)	4.6	61,176
Beethoven	5050	3.7			

Note: Pixel width = 0.0032; maximum subdivision number = 4.

Table 2: Computing Time Comparison.

Model	Method presented in [13] (sec)	LDNIs-based method (sec)
Bunny∪Dragon	572	22
(Bunny∪Dragon) –Beethoven	287	18
(Bunny∪Dragon) ∩Beethoven	184	14

7 Results and Discussions

We used C++ programming language with *Microsoft Visual Studio 2005* to implement the proposed method. We have also tested our algorithms using polygonal models with various complexities. Four of the tests are shown in Figure 10-13. Except test 1, all the tests are based on a commodity PC with an *Intel* Core2 Quad CPU Q660 2.4GHz and 4GB DRAM running *Windows Vista*.

The first test is a test case on Boolean operations that was first presented in [13] (refer to Figure 10). The Boolean results on three models (a bunny [28], a dragon [28], and a Beethoven statue) are shown in Figure 10-top. We use the same computer used in [13] (a commodity PC with a 3.2 GHz *Pentium IV* processor and 2GB *DRAM* running *Windows XP*) to do the test. The computing time of the three major steps as discussed in Section 3-5 is given in Table 1. Compared with the results given in [13], the LDNIs based method is significantly faster (15~25 times faster, refer to Table 2). Note the test results are all based on a single thread running in a sequential mode.

The second test is a test case that we take from [29] to test the operation of removing self-intersections (refer to Figure 11). We first design an internal structure based on a microstructure and a given model (a *Beethoven* statue). Based on the structure configurations, we construct a sphere model at each joint and a cylinder model at each strut. We then merge all the sphere and cylinder models without proper Boolean operations.

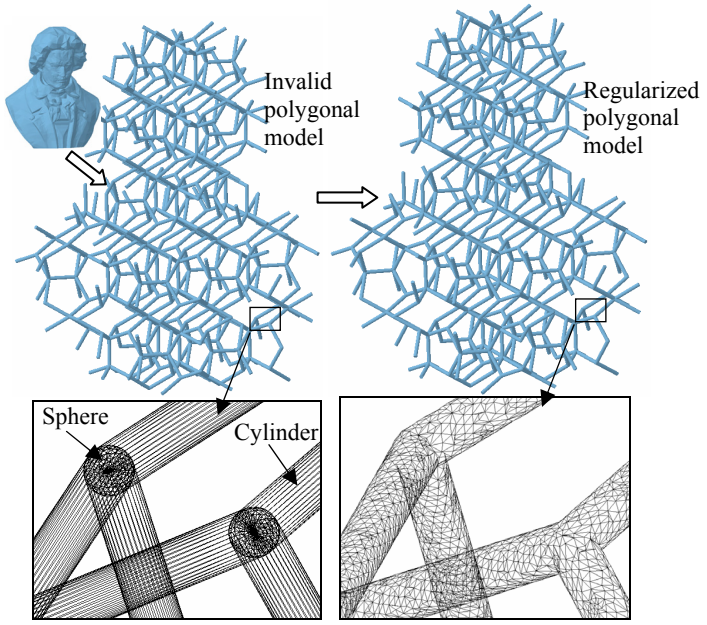

Figure 11: Results of Test 2. The input model is shown in the left; screen capture of the generated model is shown in the right.

Table 3: Running Results of Test 2.

Tile	Construct & Verify LDNIs (sec)	Adaptive Sampling (sec)	Contouring (sec)	Merge meshes (sec)
(0, 0, 0)	7.3	10.3	3.3	1.2
(1, 0, 0)	8.4	13.7	4.8	

Note: Pixel width = 0.005; maximum subdivision number = 3.

The constructed polygonal model is obviously invalid since it has lots of self-intersections (refer to Figure 11.left). We use the geometric operation as discussed in Section 4.1 to eliminate all the self-intersections. The generated model is now valid with all spheres and cylinders are properly Booleaned (refer to Figure 11.right). The size of the input model is 5.0×4.0×2.8. The model has 228,640 triangles. Our algorithm subdivides the model into 2×1×1=2 tiles and run each tile in parallel. The running time of each tile is given in Table 3. Since the tile (1, 0, 0) takes longer, the total running time will depend on it, which takes about 27 seconds. The merged polygonal model has 838,976 triangles.

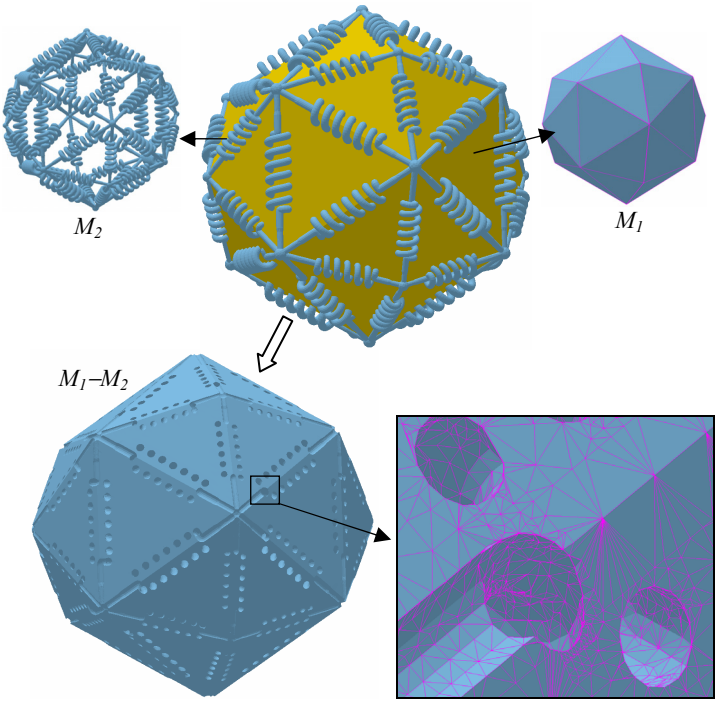

Figure 12: Results of Test 3. The input models are shown in the top; screen capture of the subtraction result is shown in the bottom.

Table 4: Running Results of Test 3.

Tile	Construct LDNIs (sec)	Subtraction (sec)	Adaptive Sampling & Contouring (sec)	Merge meshes (sec)
(0, 0, 0)	3.5	0.2	3.8	1.0
(1, 0, 0)	3.4	0.2	3.7	
(0, 1, 0)	3.3	0.2	3.9	
(1, 1, 0)	3.4	0.2	3.7	
(0, 0, 1)	3.5	0.2	3.9	
(1, 0, 1)	3.4	0.2	3.7	
(0, 1, 1)	3.4	0.2	3.8	
(1, 1, 1)	3.4	0.2	3.8	

Note: Pixel width = 0.0038; maximum subdivision number = 4.

Table 5: Running results of Test 4 (10 tiles with longest time).

Tile	Construct LDNIs (sec)	Subtraction (sec)	Adaptive Sampling & Contouring (sec)	Total (sec)
(1, 0, 2)	6.5	0.3	12.0	18.8
(1, 0, 0)	4.7	0.4	9.5	14.6
(0, 0, 1)	4.6	0.3	6.5	11.4
(2, 1, 0)	4.4	0.3	6.6	11.3
(2, 0, 2)	6.0	0.2	4.1	10.3
(2, 0, 1)	4.4	0.3	5.3	10.0
(2, 1, 2)	4.2	0.3	5.4	9.9
(0, 0, 2)	5.9	0.2	2.8	8.9
(0, 2, 1)	4.3	0.3	4.2	8.8
(1, 2, 2)	4.1	0.3	3.9	8.3

Note: Pixel width = 0.0025; maximum subdivision number = 4.

In the third test, we take a spring model M_2 generated from a given model M_1 based on an approaches presented in [30] (refer to Figure 12). Their relative positions are also shown in the figure. We calculate the subtraction of the two models (M_1- M_2). The size of the input models is 3.2×3.2×3.2 with 328,777 triangles. Our algorithm subdivides the model into 2×2×2=8 tiles and run each tile in parallel. The running time of each tile is given in Table 4. The constructed polygonal model has 617,692 triangles.

In the fourth test, we select 12 test models, many of which are publicly available. We randomly scale, rotate, and position them at each strut of a structure generated from a cube (M_1). We merge the 12 test models into one model (M_2). We then calculate the union result of the structure and the merged model ($M_1 \cup M_2$). The size of the input models is 4.8×4.6×4.5. The two input models have 445,053 triangles. Our algorithm subdivides the model into 3×3×3=27 tiles and run each tile in parallel. The computing time of the 10 tiles that take the longest time is given in Table 5. The merged polygonal model has 1,375,773 triangles.

As shown in the results, the generated triangle meshes are denser around sharp corners and small features due to the adaptive subdivision process. This is highly desirable since these features have higher curvatures. Therefore we can use reasonable amount of triangles to construct an accurate model.

The running time of our method is satisfactory. Even for rather complex geometries as shown in Test 4, we can generate Boolean results within 20 seconds if a sufficient number of PCs are available such that each tile can be processed in parallel. The memory requirements are also satisfactory based on subdividing a model into multiple tiles.

8 Conclusion and Future Work

We have presented a novel LDNIs-based geometric modeling method for accurately modeling complex geometries. Our approach is volumetric and hierarchical, and can achieve the accuracy required in most CAD/CAM applications. We use volume tiling which significantly reduces the memory requirement for processing the model. Hence we can use commodity PCs to handle very large models. Our method can easily be implemented to run in parallel. The computing time can be significantly reduced by using computing devices such as PC cluster, graphics hardware, and multi-core processor. The polygonal model generated by our approach is topologically equivalent to the exact surface and has a two-side Hausdorff error bounded by an error tolerance. Our contouring algorithm can capture all the sharp features in the

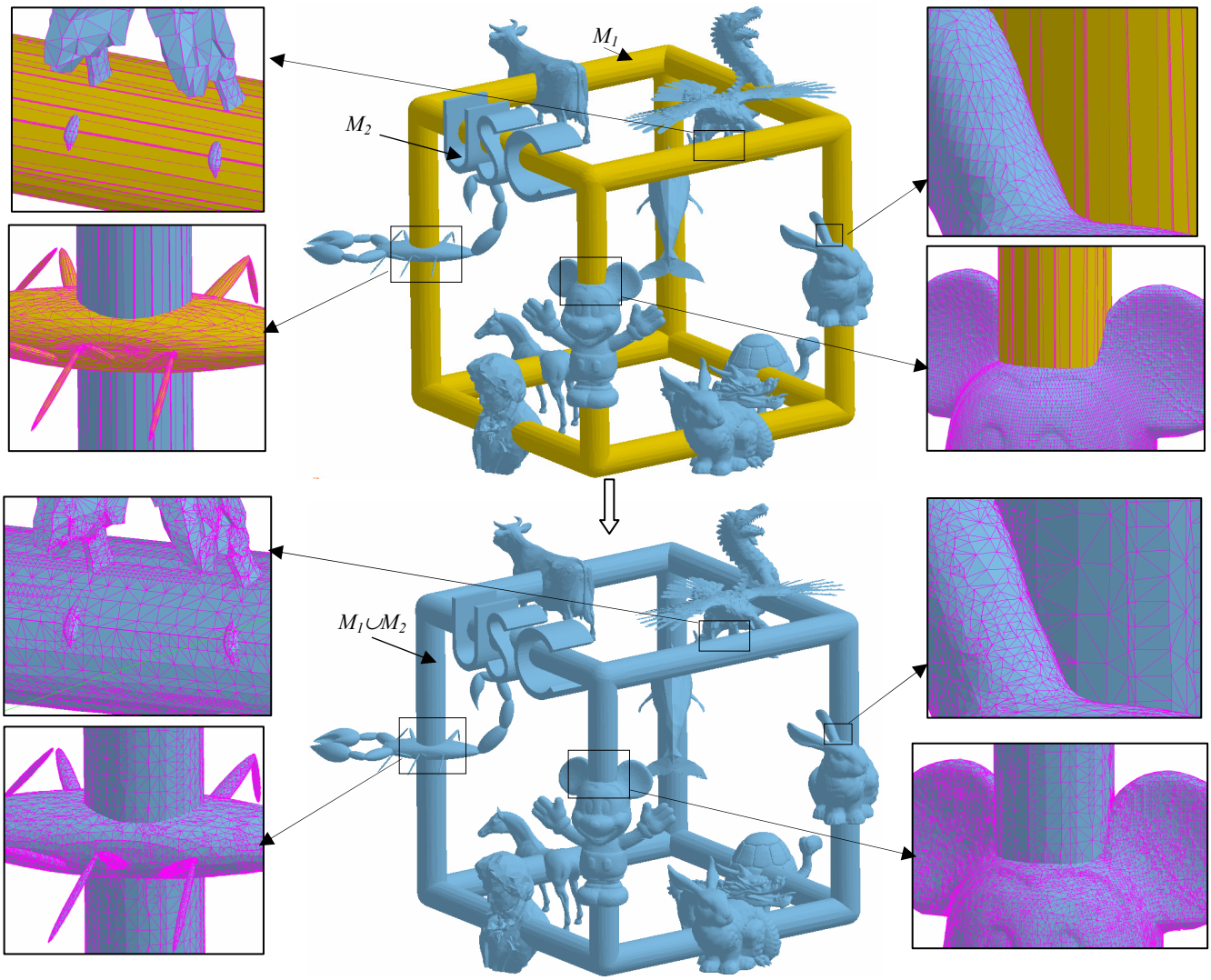

Figure 13: Results of Test 4. The input models are shown in the top; screen captures of the union result are shown in the bottom.

constructed model. In addition, our approach is simple and easy to implement. The experimental results on a variety of CAD models have also verified the effectiveness and efficiency of our method.

Our future work includes: (1) we are investigating the approaches of using graphics processing unit (GPU) to implement some steps of our method to further improve its speed; (2) we are doing robustness tests and exploring theoretically provable method to handle even degenerate data; (3) We also plan to utilize the method presented in this paper in other geometric operations such as offsetting and local geometry modifications.

ACKNOWLEDGEMENTS

The first author would like to acknowledge the support by the *James H Zumberge* Faculty Research and Innovation Fund at the USC. The second author would like to acknowledge the support by CUHK Direct Research Grant CUHK/2050400.

REFERENCES

[1] Hoffmann, C. 1989. Geometric and solid modeling. Morgan Kaufmann, San Mateo, California.

[2] Rossignac, J. and Requicha, A. 1986. Offsetting operations in solid modelling. *Computer Aided Geometric Design*, 129-148.

[3] Hoffmann C. 2001. Robustness in geometric computations. *ASME Journal of computing and information science in engineering*, 1, 143-156.

[4] Lorensen, W. E. and Cline, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of ACM SIGGRAPH 1987*, Computer Graphics Proceedings, Annual Conference Series, 163-169.

[5] Kobbelt, L., Botsch, M., Schwanecke, U., and Seidel, H. P. 2001. Feature-sensitive surface extraction from volume data. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 57-66

[6] Ju, T., Losasso, F., Schaefer, S. and Warren, J. 2002. Dual contouring of hermite data. In *Proceedings of ACM SIGGRAPH 2002*, Computer Graphics Proceedings, Annual Conference Series, 339-346.

[7] Kim, Y., Varadhan, G., Lin, M. and Manocha, D. 2003. Fast swept volume approximation of complex polyhedral models. *Proceedings of the eighth ACM symposium on solid modeling and application*, 11-22.

[8] Kaufman, A., D. Cohen, and R. Yagel. 1993. Volume graphics. *Computer,* 26(7): 51-64.

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

[10] Pauly, M., R. Keiser, L. P. Kobbelt, M. Gross. 2003. Shape modeling with point-sampled geometry. In *Proceedings of ACM SIGGRAPH 2003*, Computer Graphics Proceedings, Annual Conference Series, 641-650.

[11] Menon J.P. and Voelcker H.B. 1995, On the completeness and conversion of ray representations of arbitrary solids, In *Proc. of ACM Symposium on Solid Modeling and Applications 1995*, pp.175-286.

[12] Wang C.C.L. and Y. Chen. 2008, Layered Depth-Normal Images: a sparse implicit representation of solid models. *Computer-Aided Design*, submitted.

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

[14] Blinn, J. F. 1982. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3), 235-256.

[15] Bloomenthal, J. 1997. Introduction to implicit surfaces. Morgan Kaufmann, San Mateo, California.

[16] Pasko, A., Adzhiev, V., Sourin, A. and Savchenko, V. 1995. Function representation in geometric modeling: concepts, implementation and applications. *The Visual Computer*, 11(8), 429-446.

[17] Perry, R. and Frisken, S. 2001. Kizamu: A system for sculpting digital characters. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 47-56.

[18] Ohtake, Y., Belyaev, A. G., and Pasko, A. 2001. Dynamic meshes for accurate polygonization of implicit surfaces with sharp features. *Proceedings of the International Conferences on Shape Modeling & Applications*, 74-158.

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

[20] Huang, Y., J. H. Oliver. 1995. Integrated simulation, error assessment, and tool path correction for five-axis NC milling. *Journal of Manufacturing Systems*, 14(5), 331-344.

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

[22] Cignoni P., Ganovelli F., Montani C., Scopigno R. 2000. Reconstruction of topologically correct and adaptive trilinear isosurfaces. *Computers & Graphics*, 24, 3(2000), 399-418.

[23] Frisken, S. F., Perry, R. N., Rockwood, A. P., and Jones, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 249-254.

[24] Varadhan, G., Krishnan, S., Sriram, T. and Manocha, D. 2004. Topology preserving surface extraction using adaptive subdivision. *Proceedings of Eurographics/ACM SIGGRAPH symposium on geometry processing*, 235-244.

[25] Zhang, N., Hong, W. and Kaufman, A. 2004. Dual contouring with topology-preserving simplification using enhanced cell representation. *Proceedings of IEEE Visualization*, 505-512.

[26] Chen, Y. 2007. Robust and accurate Boolean operations on polygonal models. ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conferences, Las Vegas, Nevada, September 4-7, 2007, DETC2007-35731.

[27] Wang, C. C. and Chen, Y. 2008. Layered Depth-Normal Images for Complex Geometries – Part Two: Manifold-Preserved Adaptive Contouring. ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conferences, New York City, New York, August 3-6, 2008, DETC2008-49576.

[28] Stanford 3D scanning repository (http://www-graphics.stanford.edu/data/3Dscanrep/).

[29] Chen, Y. 2007. 3D texture mapping: a microstructure design method for rapid manufacturing. *Computer-aided Design and Application*, vol. 4, No. 6, pp.761-771.

[30] Chen, Y, S. Wang. 2008. Computer-aided product design with performance-tailored mesostructures. *International CAD Conference and Exhibition*, Orlando, Florida, submitted.