# Feature Conservation and Conversion of Tri-dexel Volumetric Models to Polyhedral Surface Models for Product Prototyping

Yongfu Ren[1], Weihang Zhu[2] and Yuan-Shin Lee[3]

[1]North Carolina State University, ren_yongfu@yahoo.com
[2]Lamar University, weihang.zhu@lamar.edu
[3]North Carolina State University, yslee@ncsu.edu

## ABSTRACT

In this paper, detailed algorithms are presented for eliminating inconsistency in a tri-dexel volumetric models and reconstructing a water-tight polyhedral surface model from the tri-dexel volumetric model. By using the surface normal information saved in each dexel grid node, a refinement process is applied to reconstructing sharp edges and apex vertices in polyhedral surface models. The advantage of tri-dexel model is its balance in modeling accuracy and memory consumption. It is a challenge problem to convert it to a water-tight polyhedral surface model for downstream applications. The proposed techniques can be used as a geometric representation kernel in Computer-Aided Design and Manufacturing (CAD/CAM) and Computer Numerically-Controlled process simulation systems.

## 1. INTRODUCTION

The volumetric representation of geometric models has advantages in 3-Dimensional (3D) scanning data processing, heterogeneous material modeling, and other applications that require simple Boolean operations at interactive speed, e.g., haptic virtual sculpturing and Computer Numerically-Controlled (CNC) process simulation. Several types of volumetric models, such as the voxel model [8] and the dexel model [7], have been proposed in the earlier research. A voxel model represents an object with many small cubes in a regular lattice, as shown in Fig. 1(a). A dexel model represents an object with a grid of long columns compacted together along a certain direction (e.g., Z axis direction), as shown in Fig. 1(b). The tri-ray model proposed by Benouamer has the advantage of accurately representing smooth surface models. Nevertheless, the modeling inconsistency problem existing in tri-ray models has not been elegantly solved in the past research [1]. The extended Marching Cube algorithm [2]., which is applicable to the voxel model only, does not provide an efficient method for Boolean operations, which is very important in time-critical applications such as real-time haptic virtual sculpting [5]. We recently presented a tri-dexel modeling method in [5]. It addresses the limitation that a voxel model or dexel model has in conserving accurate geometrical and topological feature information, especially at sharp edges and corners [6].

Polyhedral surface representation has been widely adopted in the Computer-Aided Design and Manufacturing (CAD/CAM) systems for 3D object modeling, Reverse Engineering, CNC tool path generation and mechanical property analysis. One of the most commonly used polyhedral surface models is the STL (Standard Transformation Language) model, which represents an object with triangular facets. While being ubiquitous in CAD/CAM systems, it is practically difficult to come up with water-tight polyhedral surface models without gaps or holes. The conversion between volumetric representation and surface representation is one of the critical functions in volumetric modeling. The conversion of volumetric model to surface model usually need to meet the following requirements: (1) keeping

accurate positioning information; (2) restoring sensitive geometric features such as sharp edges and vertices; (3) ensuring the water-tight properties of resultant surface models; (4) performing fast Boolean operations to remove or add volumetric shapes. In [3], the condition when there is only one dexel in each grid point was discussed. This condition rarely happens in virtual sculpting or multi-axis NC simulation. In our earlier research work presented in [9,10], a virtual sculpting system with a 5-DOF (degree-of-freedom) haptic interface has been developed for virtual prototyping and machining planning. In our earlier system [9,10], a dexel-based in-process modeling method was adopted and visibility sphere marching algorithms were proposed to convert a dexel model to a polyhedral surface model [11]. To overcome the modeling inaccuracy problem, dexel-based system was later extended into a tri-dexel model based system [5]. The newly invented tri-dexel modeling method provides a better alternative for time-critical applications such as virtual sculpting and CNC process simulation. Meanwhile, it also presents new challenges in model analysis and conversion.

This paper presents our investigation on tri-dexel model analysis and feature conversion of a tri-dexel model to a polyhedral surface model. The remainder of this paper is organized as follows: Section 2 introduces the data structure and basic concepts of the proposed tri-dexel modeling method. In Section 3, the detailed algorithms of regularizing a tri-dexel model to keep data consistency is discussed. The algorithms of the converting a tri-dexel model into a polyhedral surface model are narrated in Section 4. The applications of tri-dexel model in time-critical applications such as haptic virtual sculpting are illustrated in Section 5. Section 6 concludes this paper with future research.
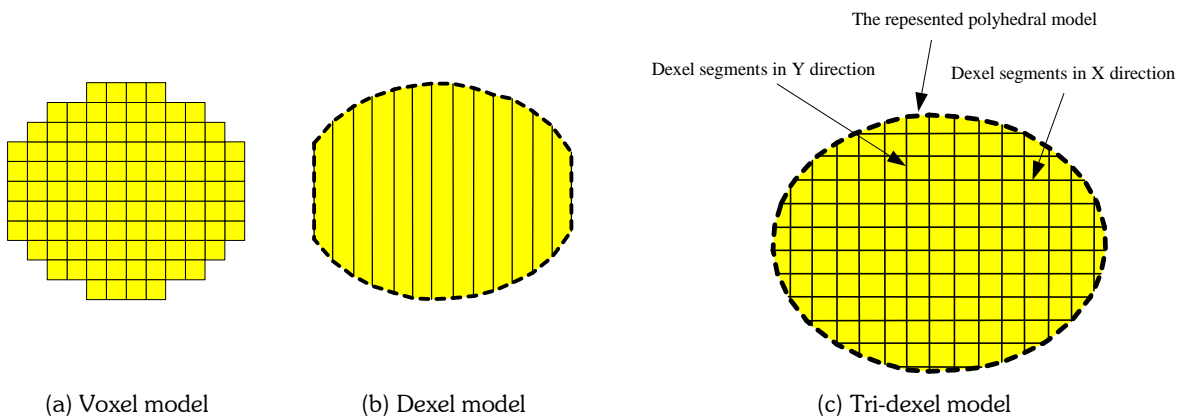


(a) Voxel model      (b) Dexel model      (c) Tri-dexel model

Fig. 1: Comparison of different volumetric modeling methods.

## 2. THE DATA STRUCTURE AND BASIC CONCEPT DEFINITION OF A TRI-DEXEL MODEL

*Data structure of a tri-dexel model*: As shown in Fig. 1(c), a tri-dexel model consists of three overlapping orthogonal dexel models oriented along X, Y and Z axes. Similar to a dexel model, a tri-dexel model is defined relative to an origin point $O$ with a grid interval of $\delta$. As shown in Fig. 2(a), at the grid location $[i, j]$, a set of dexel segments $\{D_{i,j,k}\}$ are organized by linked lists. Each dexel segment $D_{i,j,k}$ consists of two tri-dexel nodes $\xi^l_{i,j,k}$ and $\xi^u_{i,j,k}$ as the lower and the upper ends of a dexel segment. As shown in Fig. 2(b), a tri-dexel node includes information such as the height, color, flag and normal. For specific applications, other information such as material properties can also be included in a tri-dexel node. For a tri-dexel model, the space requirements roughly are $O(N_x \times N_y + N_y \times N_z + N_z \times N_x) \times M_d$, where $N_x$, $N_y$, and $N_z$ are the numbers of grid cells along the three edges of a 3D-grid, and $M_d$ is the maximum number of dexels which are assigned to a vertex of one dexel [4]. $M_d$ can often be considered as a constant. For example, $M_d = 2$ for a convex solid object since one dexel has at most 2 intersections with a convex solid. Thus the memory storage requirement is usually considerably less than the $O(N_x \times N_y \times N_z)$ of a corresponding voxel volume with similar grid intervals.

*Basic concept definitions*: Fig. 3 shows a 2D view of the tri-dexel representation of an ellipsoid. The intersection points of the grid lines are called *grid points*. A dexel segment is subdivided into *grid edges* by grid points, as shown in Fig. 3. Fig. 4 shows a 3D view of a grid cell. For each grid edge, it may be classified as either a *full grid edge* or a *partial grid*

*edge*, depending on the length of the edge. If a grid edge's both ends pass grid points, it is categorized as a full grid edge. Otherwise, it is considered as a partial grid edge.
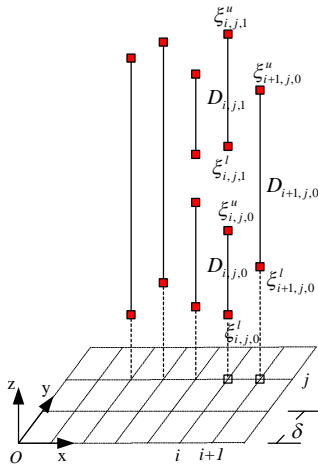


(a) Data structure of a dexel model
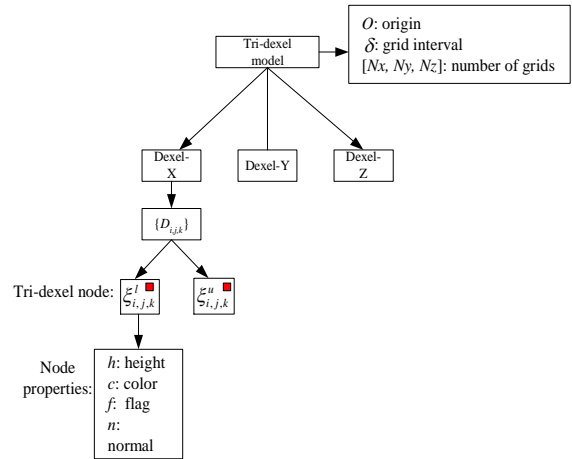
(b) Hierarchical data structure

Fig. 2: Tri-dexel model data structure.

For illustration purpose, each grid point is assigned an ID number, as shown in Fig. 4. Each grid point has three neighboring grid points sequenced counter-clock-wisely. For example, the grid point 0 has three neighbors with IDs of 1, 4, 2, where the points 1, 4, 2 are counter-clock-wise when viewing from point 0. If a grid point is in the middle of a dexel segment, the grid point is flagged as *OCCUPIED*. Otherwise, the grid point is flagged with *NON-OCCUPIED*. As shown in Fig. 3, for each grid cell with grid edges, it is classified as *inner grid cell* if all of its 12 edges are full grid edges. Otherwise, it is classified as *boundary grid cell*.
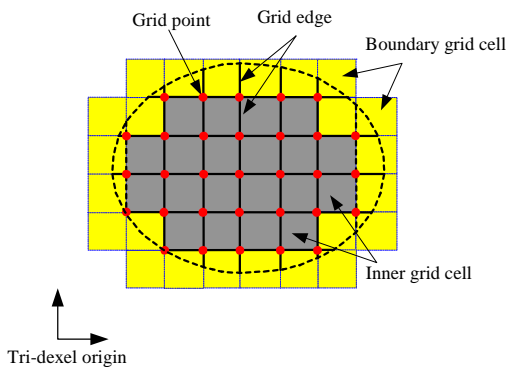


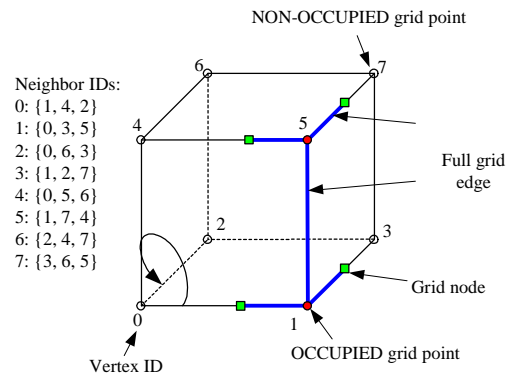Fig. 3: Retrieve the boundary cells of a tri-dexel model.

Fig. 4: Structure of a grid cell of a tri-dexel model.

## 3. DATA REGULARIZATION OF A TRI-DEXEL MODEL

As introduced earlier, a tri-dexel model consists of three overlapping dexel models oriented along X, Y and Z axis directions, respectively. However, the simple combination of three dexel models may result in data inconsistency, due to the numerical errors in the tri-dexelization process or in the dynamic model updating process in applications such as virtual sculpting. Fig. 5(a) shows a type of data inconsistency where both grid points of a grid edge are flagged as *OCCUPIED*, whilst the grid edge is a partial grid edge. In this case, triangulation will be ambiguous. Therefore, a data regularization process needs to be invoked to ensure the data consistency of every grid cell in a tri-dexel model. The rules of data regularization are summarized as follows:

**Rule_I: Data regularization of tri-dexel model**

(1) If both end grid points are flagged as OCCUPIED, a full grid edge is added between two end points. As shown in Fig. 5(a), a full grid edge is added to eliminate the data inconsistency.

(2) If both end grid points are flagged as NON-OCCUPIED, a segment with the length of the grid cell size is deleted from the edge. As shown in Fig. 5(b), the partial grid edge is deleted with a Subtraction Boolean operation.

(3) When a grid point is flagged as OCCUPIED, short segments with a length of $\rho$ are added along the X, Y, and Z directions, respectively. This is used to get rid of the degenerated cases when dexel segments stop right at a grid point, which may result in degenerated triangles by the triangulation method as presented in Section 4. Fig. 5(c) shows the 2D view of two added short segments along the Y direction (solid blue line).

(4) If a grid edge doesn't pass any grid point, it is discarded, as shown in Fig. 5(d). Scenario (2) may be considered as a special case of Scenario (4).



(a) Regularization with two *OCCUPIED* points          (b) Regularization with two *NON-OCCUPIED* points

(c) Regularization with one *OCCUPIED* point          (d) Regularization by deleting disjointed partial segments
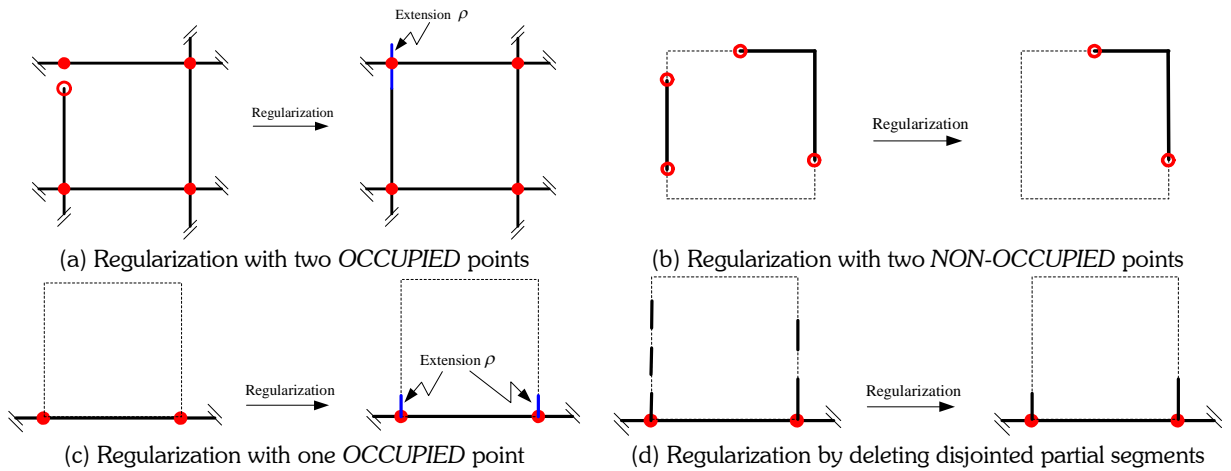
Fig. 5: Data regularization by checking every grid cell to ensure data consistency.

Following the above description, an algorithm of regularizing tri-dexel models is shown as below:

**Algorithm I: Data regularization of a tri-dexel model**

**Input**:

$\Psi_o$ : Initial tri-dexel representation

**Output**:

$\Psi$ : Regularized tri-dexel representation

{ $\phi_w$ }: Boundary grid cells

**Procedure**:

Set $\Psi$ as an empty tri-dexel model;

/* the following codes are used to set the grid points' occupation flags */

**For** (each dexel segment $D_{i,j,k}$ in the X, Y and Z dexel models)

Set all the grid points that the dexel segment $D_{i,j,k}$ passes through as *OCCUPIED*

**End For**

/* the following codes are used to add the full grid edges */

**For** (each grid cell $\phi$ )

**For** (each edge E in grid cell $\phi$ )

**If** (Both endpoints of the edge E are *OCCUPIED*)

Add a full grid edge in $\Psi$ ;

Extend both ends of the full grid edge in $\Psi$ with the length of $\rho$ .

        **End If**

     **End For**

**End For**

/* the following codes are used to add partial grid edges */

**For** (each dexel segment $D_{i,j,k}$ that passes any *OCCUPIED* grid point)

    Add segment $D_{i,j,k}$ in $\Psi$ ;

**End For**

/* the following codes are used to find boundary grid cells */

**For** (each grid cell $\phi$ )

   **If** ( $\phi$ has both *OCCUPIED and NON-OCCUPIED* grid point)

      Save $\phi$ as a boundary grid cell $\phi_w$ ;

   **End If**

  **End For**

 Output $\Psi$ and { $\phi_w$ };

**End.**

The above regularization algorithm leads to a tri-dexel model with the following properties:

(1) On each grid edge (between two grid points), at most one dexel segment exists. As shown in Fig. 5(d), partial grid edges that don't pass through any grid point are not added into the regularized tri-dexel model;
(2) When two neighboring grid points are *OCCUPIED*, the edge between them is a full grid edge.
(3) When two neighboring grid points are *NON-OCCUPIED*, there is no grid edge between them.

The regularized tri-dexel model will be used in the model conversion process to reconstruct a water-tight polyhedral surface model in the following sections.

## 4. CONVERTING A TRI-DEXEL MODEL TO A POLYHEDRAL SURFACE MODEL

The conversion from a volumetric model into a polyhedral surface model is not a trivial task. The problem becomes more difficult when it is required to ensure water-tight property and conserve sensitive features such as sharp edges and apex vertices. The data regularization process discussed earlier is an important step to ensure the water-tight properties of converted polyhedral models. The surface normal information saved in each grid node during the tri-dexelization process as described in [5,6] will be used to reconstruct sensitive features in the model conversion process. The conversion process consists of two major steps: (1) flood-search process to construct initial boundary loops of each boundary grid cell; (2) refine and triangulate the boundary loops to construct a final polyhedral model. The following subsections will discuss the two steps in details.
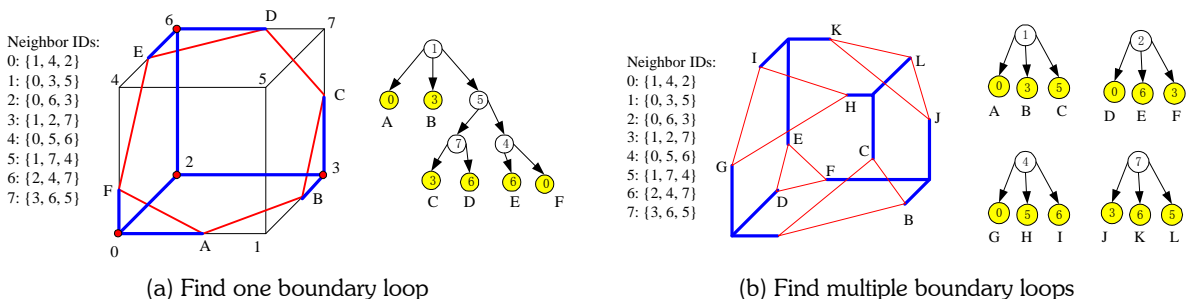


(a) Find one boundary loop                  (b) Find multiple boundary loops

Fig. 6: Construct boundary loops with depth-first search at boundary grid cells.

## 4.1 Constructing Boundary Loops with a Flood-search Process

The rule of thumb for the flood-search process of boundary loops is:

> *Starting from a NON-OCCUPIED grid point, recursively flood-search boundary loops according to the neighboring relationships until all NON-OCCUPIED grid points are visited. Each grid node in a partial grid edge is saved in the boundary node list.*

For example, for the grid cell shown in Fig. 6(a), the flood-search process starts at a *NON-OCCUPIED* grid point 1. Its three neighbors 0, 3, 5 are listed as the children of grid point 1 and a depth-first search process (as if it is organized as a *tree* structure) is used in the flood-search process. Point 0 is visited and it is found to be an OCCUPIED point. Therefore, the grid node A between grid points 1 and 0 is saved as the first node in the boundary loop. Then the point 3 is visited as the second child of point 1 and grid node B is found. Since point 5 is a NON-OCCUPIED grid point, its children are further searched. Notice point 1 has been visited. Then point 7 is visited as the second child of point 5. The flood-search process continues until all NON-OCCUPIED grid points {1, 5, 7, 4} are visited. The flood-search process leads to a boundary node list of {A, B, C, D, E, F}, as shown in Fig. 6(a). Fig. 6(b) shows another example of the flood-search process. Since the NON-OCCUPIED grid points are disjointed, multiple boundary loops are constructed in Fig. 6(b). Based on the above description, the flood-search algorithm for constructing boundary loops at boundary grid cells is presented as below:

### Algorithm II: Constructing the initial boundary loops of each boundary grid cell

**Input**:    $\Psi$ : Regularized tri-dexel representation

   { $\phi_w$ }: Boundary grid cells of $\Psi$

**Output**:   { $\varpi_w$ }: Boundary loops represented by grid node lists

**Procedure**:

   **For** (each boundary grid cell $\phi_w$ )

      **For** (each of its eight grid points $V$ in $\phi_w$ )

      **If** ($V$ is not *OCCUPIED* and not visited)
         /* the flood-search subroutine */
         Set $V$ as visited;

         **For** (each of V's 3 neighboring grid point $V_n$ in its neighboring list)

         **If** ( $V_n$ is *OCCUPIED*)

            Store the grid node on the edge from V to $V_n$ into $\varpi_w$ ;

         **Else** ( $V_n$ is not visited)

            Call the flood-search subroutine with the grid point $V_n$ ;

         **End If**
         **End For**
      **End If**
      **End For**
   **End For**
**End.**

As the result of **Algorithm II**, a set of boundary loops are generated. They will be further refined and triangulated as described in Section 4.2.

## 4.2 Refining and Triangulating of Boundary Loops

In the tri-dexelization process, it is unlikely for dexel segments to exactly pass through sharp edges and apex vertices. Consequently, sensitive features may be lost if the triangulation process is applied to the initial boundary nodes generated by **Algorithm II**. Fortunately, with the normal information stored in each grid node, sensitive features can be restored by a refinement process.

The objective of the refinement process is to find sharp edges and apex vertices from the initial boundary loops retrieved by **Algorithm II**. As shown in Fig. 7(a), an initial boundary loop consists of four grid nodes A, B, C and D, with the corresponding normal vectors $n_A$, $n_B$, $n_C$ and $n_D$ respectively. Intermediate points are computed based on the positions and normals of neighboring grid nodes. For example, the intermediate point E is computed as the intersection of three planes: the 1$^{st}$ plane is the bottom plane passing grid points 0, 1, 2, 3; the 2$^{nd}$ plane passes the location of node A with the normal of $n_A$; the 3$^{rd}$ plane passes the location of node B with the normal of $n_B$. Similar process can be applied to compute the points F, G, and H. If the intermediate point E is within the grid cell and the distance from E to the edge AB is larger than a given tolerance $\varepsilon$, it is considered as a *valid intermediate point* and is saved in an intermediate point list. Otherwise, it is considered as an *invalid intermediate point* and discarded. For the example as shown in Fig. 7(a), points G and E are invalid and are discarded because the distances to the correspondent edge AB and CD are smaller than the given tolerance $\varepsilon$. On the other hand, points F and H are saved as valid intermediate points.



(a) Find 2 intermediate points        (b) Find 3 intermediate points and 1 apex vertex



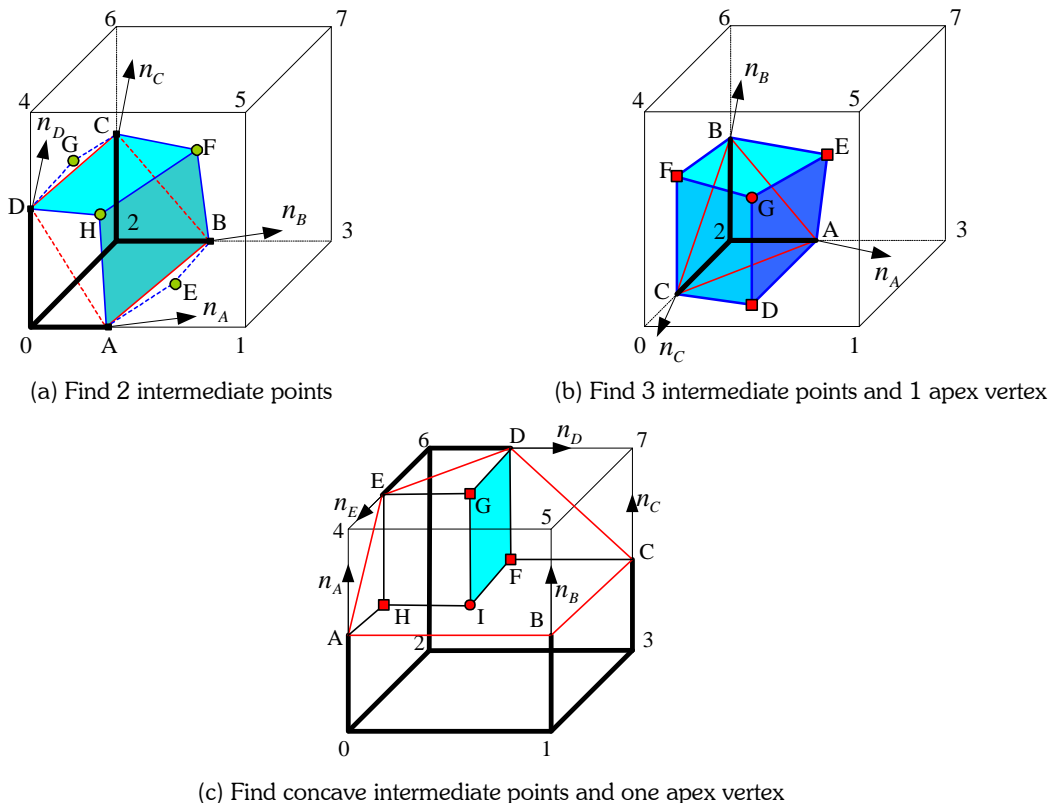(c) Find concave intermediate points and one apex vertex

Fig. 7: Refinement of the boundary loops generated by Algorithm II, before triangulation process.

If there are at least 3 intermediate points generated from the above refinement process, an apex vertex may be calculated. As shown in Fig. 7(b), three intermediate points D, E, F have been generated by the refinement process. The first three non-duplicated grid nodes A, B, C are recorded during the process of finding the intermediate points D, E, F. Then the apex vertex G is computed as the intersection of three planes: the 1$^{st}$ plane passes through A with the normal of $n_A$, the 2$^{nd}$ plane passes through B with the normal of $n_B$, the 3$^{rd}$ plane passes through C with the normal of $n_C$. If the apex vertex G is within the grid cell, it is accepted as a *valid apex vertex*. Otherwise, it is discarded as an *invalid apex vertex* and. Fig. 7(c) shows another example of refinement, where concave shapes are generated. In Fig. 7(c), points F, G and H are valid intermediate points. With the first 3 grid nodes C, D and E which were involved in the computation of F, G, H, the apex vertex I is computed as the intersection of 3 planes passing C, D and E. Note that the grid nodes A and B don't contribute to the intermediate points and they are not used in calculating the apex vertex G.

The examples show that the proposed refinement process can be applied to restore both convex and concave sensitive features. After the refinement process, the boundary loops can be triangulated into a mesh.

## 5. APPLICATIONS AND ILLUSTRATIVE EXAMPLES

The presented techniques and algorithms were implemented using Visual C++ programming language and OpenGL functions on a 2.6GHz personal computer with 1GB memory at North Carolina State University. Fig. 8(a) shows the reconstructed polyhedral models from tri-dexel models for a cube, a cylinder and a sphere, respectively. Fig. 8(b) shows the comparison of the Boolean operation results with and without refinement process. The resolution of the tri-dexel model is 26x26x43. As can be seen in Fig. 8, compared with the model without refinement, the refined model shows much better accuracy with sharp edges and apex vertices. Fig. 9 shows the refinement comparison of a joystick base model. The joystick tri-dexel model has the resolution of 67x67x168. The computation results also indicate that the reconstructed polyhedral model is water-tight and keeps the important features such as sharp edges and vertices.
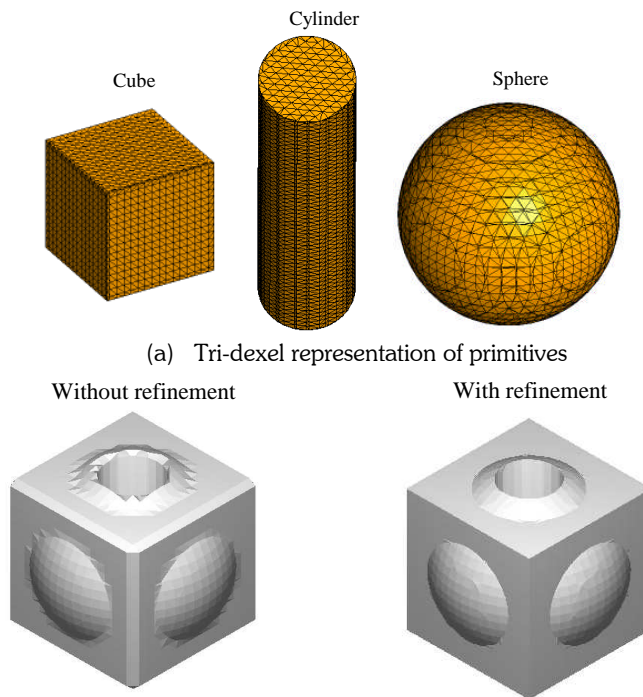


(a)   Tri-dexel representation of primitives



(b)   Boolean operation: Cube+Sphere-Cylinder, Step = 0.58, Number of grids = [26, 26, 43]
Fig. 8: Tri-dexel representation of primitives and Boolean operations.

The presented tri-dexel structure is applied in a haptic aided design system shown in Fig. 10. A PHANTOM desktop haptic device is linked with the developed tri-dexel based design system to provide real-time input and force feedback. As shown in Fig. 11, a virtual ball-endmill tool is used as the sculpting tool to modify a jaw model. Graphics hardware has been used to speed up the tri-dexelization process for Boolean operation [5]. The modification results are shown in Fig. 12. Compared with the original model shown in Fig.s 12(a) and (c), the modified model in Fig.s 12(b) and (d) shows better smoothness in the modified region. Besides, the intact region in the reconstructed model is very close to that in the original model. The computation results show that the developed tri-dexel model represents complex geometry with good accuracy.

## 6. CONCLUSIONS AND FUTURE RESEARCH

This paper presents tri-dexel model analysis and its feature conservation conversion to polyhedral surface model. The detailed techniques of regularizing tri-dexel models are discussed with the classification of grid points in boundary grid cells. The reconstruction of water-tight polyhedral surface models from a tri-dexel model is realized by flood-searching of grid nodes in boundary grid cells. Sensitive features such as sharp edges and apex vertices are reconstructed accordingly using the normal information of grid nodes. The computation results show that the developed tri-dexel

data structure can preserve sharp features of accurate water-tight models. The presented tri-dexel data structure can be used in CAD/CAM systems and CNC simulation/verification systems. In the future, more applications of the proposed tri-dexel modeling method will be pursued.
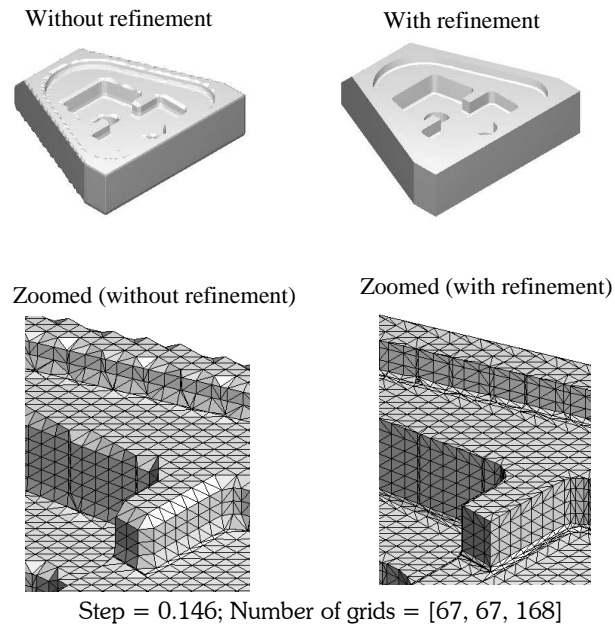
Without refinement

With refinement



Zoomed (without refinement)

Zoomed (with refinement)



Step = 0.146; Number of grids = [67, 67, 168]

Fig. 9: Tri-dexel representation of a joystick base model.
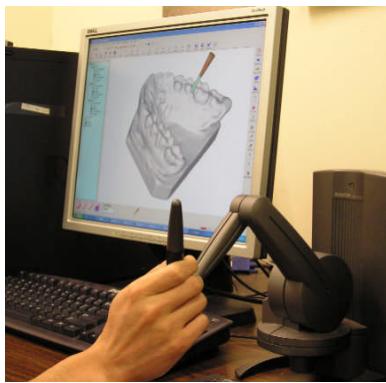


Fig. 10: A tri-dexel-based haptic virtual design system.



Tri-dexelized Jaw model

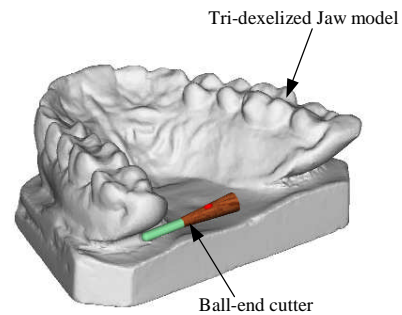Ball-end cutter

Fig. 11: Haptic virtual sculpting of a tri-dexel model.

## 7. ACKNOWLEDGEMENTS

## 8. REFERENCES

[1]    Benouamer, M. O.; Michelucci, D.: Bridging the gap between CSG and Brep via a triple ray representation, ACM Symposium on Solid Modeling and Applications, 1997, 68-79.

[2]     Kobbelt, L. P.; Botsch, M.; Schwanecke, U.; Seidel, H.-P.: Feature Sensitive Surface Extraction from Volume Data, Computer Graphics (SIGGRAPH 01), 57-66, 2001.

[3]     König, A. H.; Groller, E.: Real Time Simulation and Visualization of NC milling Processes for Inhomogeneous Materials on Low-End Graphics Hardware, In F.-E. Wolters, N. M. Patrikalakis (eds.), Proceedings of CGI'98 (Computer Graphics International), IEEE Computer Society, Hannover, Germany, June 1998, 338-349.

[4]     Muller, H.; Surmann, T.; Stautner, M.; Albersmann, F.; Weinert, W.: Online Sculpting and Visualization of Multi-Dexel Volumes, ACM Symposium on Solid Modeling and Applications, 2003, 258 - 261.

[5]     Ren, Y.; Lai-Yuen, S. K.; Lee, Y.-S.: Virtual prototyping and manufacturing planning by using tri-dexel models and haptic force feedback, Virtual and Physical Prototyping, 1(1), 2006, 3-18.

[6]     Ren, Y.; Zhu, W.; Lee Y.-S.: A Tri-dexel Model for Virtual Prototyping, Accepted for publications in the Transactions of SME/NAMRI, 2008.

[7]     Van Hook, T.: Real-Time Shaded NC Milling Display, Computer Graphics (Proc. SIGGRAPH '86), 20(4), August 1986, 15-20.

[8]     Wang, W. S.; Kaufman, A. E.: Volume sculpting, ACM Symposium on Interactive 3D Graphics, Monterey CA USA, 1995, 151-156.

[9]     Zhu, W.; Lee, Y.-S.: Dexel-based force-torque rendering and volume updating for 5-DOF haptic product prototyping and virtual sculpting, Computers in Industry, 55, 2004, 125-145.

[10]    Zhu, W.; Lee, Y.-S.: Five-axis pencil-cut planning and virtual prototyping with 5-DOF haptic interface, Computer-Aided Design, 36, 2004, 1295-1307.

[11]    Zhu, W.; Lee, Y.-S.: A marching algorithm of constructing polyhedral models from Dexel models for haptic virtual sculpting, Robotics and Computer-Integrated Manufacturing, 21(1), 2005, 19-36.

(a) The shaded original polyhedral model

(b) The shaded polyhedral model converted from the tri-dexel model

(c) The original polyhedral model

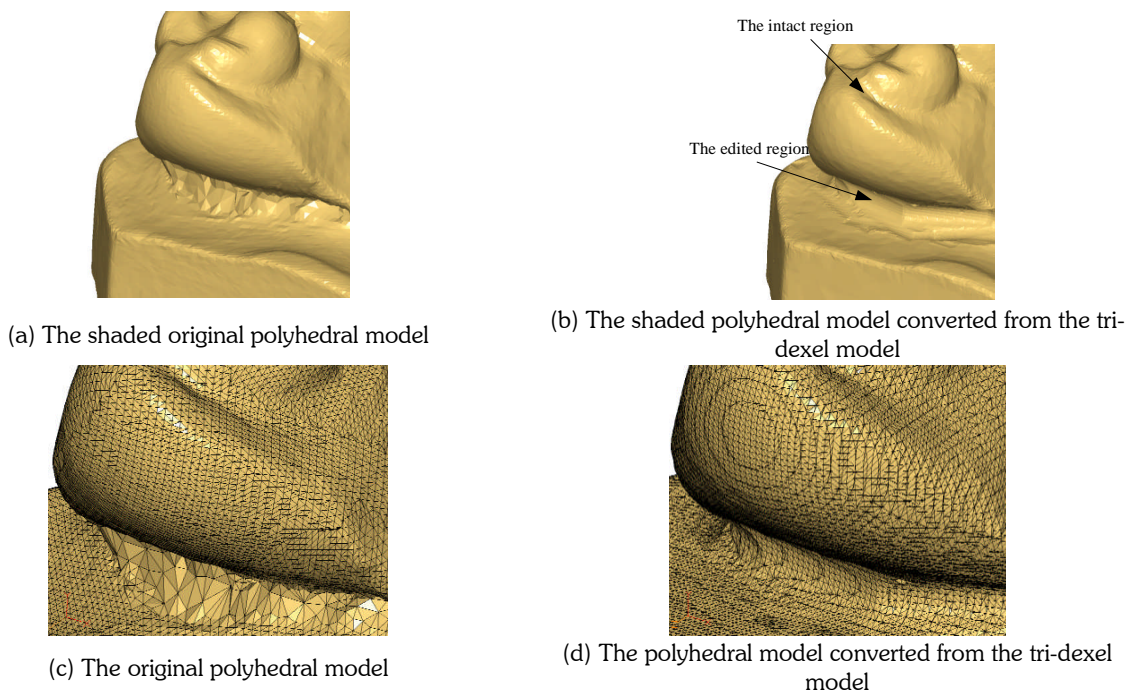(d) The polyhedral model converted from the tri-dexel model

Fig. 12: Comparison of the converted polyhedral surface model with the original model for an un-carved region.