Computer-Aided Design AND APPLICATIONS www.cadanda.com

Robust Slicing Procedure based on *Surfel-Grid*

Di QI[1], Long Zeng[2] and Matthew M. F. Yuen[3]

[1]The Hong Kong University of Science & Technology, qdxaa@ust.hk
[2]The Hong Kong University of Science & Technology, mejackyzeng@ust.hk
[3]The Hong Kong University of Science & Technology, meymf@ust.hk

ABSTRACT

The trade-off between accuracy and efficiency is of major concern in rapid prototyping (RP), where STL is the most commonly used format to present a CAD model. A STL model needs to be sliced into layered contours before it is used to build a RP object. The challenge to achieve the accuracy/efficiency trade-off becomes critical when a STL model contains a large number of triangles. Previous work by Zeng et al. [19] proposed an efficient slicing method that speeded up the layer decomposition of the STL model by adopting a one-dimensional LDNI sampling, which is independent of geometric complexity. However, the accuracy of the generated contours is not entirely satisfactory for the model with sharp features, even by adopting a high sampling resolution. Therefore, the objective of this work focuses on achieving accurate layer contours in a more effective manner without scarifying the high efficiency. In this paper, a grid-structured point representation with neighborhood information embedded, *Surfel-Grid*, is proposed, which consists of a Grid (formed by two in-plane orthogonal rays), and surfels (sampling points with normal vectors) located on the edges of the Grid. After discretization of a STL mesh model into layers of *Surfel-Grid*, loop construction is then applied to each of the *Surfel-Grid* layer. The accuracy of resulting layered contours is improved due to the adoption of two dimensional sampling and the generation of new reconstructed features. Taking advantage of the neighborhood information and *cell validity* of *Surfel-Grid*, each surfel can find and connect with its neighbor surfel directly, guaranteeing high efficiency in the loop construction process. Examples are used to illustrate the higher accuracy and better efficiency obtained using this approach.

Keywords: rapid prototyping, contouring, point model, grid structure.
DOI: 10.3722/cadaps.2013.965-981

1 INTRODUCTION

Rapid prototyping (RP) is an important layered fabrication method because it is simple and efficient. Compared with most conventional prototyping techniques, RP is much faster since the geometric complexity of a CAD model has less impact on its slicing process. A CAD model cannot directly interface with RP machines, which need to be sliced into layered contours before it can be used to build a RP object. The trade-off between accuracy and efficiency is of major concern in rapid prototyping, where STL (STereoLithography) [7], a triangular faceted model, is the most commonly used representation of input CAD model. Traditional slicing algorithms slice the STL model directly by computing the intersection curves between triangular facets and layers of parallel planes, which generally suffer from numerical stability problem [1]. Moreover, when the STL model contains a large number of triangles or is of great geometric complexity, the challenge to achieve the trade-off between accuracy and efficiency becomes more critical. Slicing on a volumetrically-represented model, in contrast, is mathematically compact and robust.

Previous work by Zeng *et al.* [19] (denoted as **Zeng's work** in the sections below) proposed an efficient slicing method that speeded up the layer decomposition of the STL model by adopting a one-dimensional LDNI sampling [18], which is independent of the geometric complexity or mesh density. The loop construction algorithm is then applied to the generated layered point model. For every point on one ray of each layer, in order to find the next point to connect, it needs to traverse all points of the same type on the adjacent ray, which is time-consuming when the point density is high. Besides that, due to the single sampling direction, the accuracy of the slicing contours is not entirely satisfactory for models with sharp features (see Fig. 1(a)), even by adopting a high sampling resolution (grey lines present sampling rays, see Fig. 1(b)). Therefore, the objective of this work focuses on increasing the accuracy of layer contours in a more efficient manner without scarifying the high efficiency, and not that sensitive to the point density at the same time.

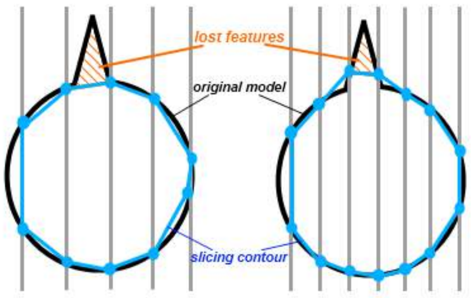

Fig. 1: Sharp feature is cut-off by Zeng's work: (a) Sparse sampling, (b) Dense sampling.

In this paper, a 2D grid structure based slicing algorithm is proposed. Firstly, a STL model is converted into layers of *Surfel-Grids* by adopting two dimensional LDNI sampling. *Surfel-Grid*, a new grid-structured point model, which consists of Grid (formed by two in-plane orthogonal rays), and surfels (sampling points with normal vectors attached) located on the edges of the Grid. The concept of surfel was first proposed by Pfister et.al [14] for rendering purpose. In this structure, surfels are used to preserve sharp features while Grid provides neighborhood information for each individual surfel; Assume the STL model is closed with good connectivity, which should occupy a set of consecutively connected cells, that the loop construction upon the *Surfel-Grid* can be simply decomposed into line-segment connection within each cell. To complete line connection inside one cell, a surfel should only connect with its *neighbor* within the same cell, and the same connecting procedure will 'marching' to their neighboring surfels in the next cell, until the preceding loop is closed. The resulting slicing model is fully compatible with layered fabrication.

Compared with Zeng's work, this work **contributes** in two aspects:

- The accuracy of layered contours is improved significantly, since two-dimensional sampling directions are employed in this work, and sharp features are reconstructed by using normal vectors of surfels;
- Instead of scarifying efficiency for the sake of accuracy, profit from the neighborhood information and *cell validity* criteria of *Surfel-Grid*, the efficiency of loop construction is improved and not much affected by point density: each surfel can access its neighboring surfels effortlessly, so that line-segments inside each cell can be generated directly.

Related work overview - Since this is a slicing algorithm based on a grid-structure point model - *Surfel-Grid*, some discretized representation based contour slicing methods have been reviewed.

Point clouds can be sliced directly into a RP slice file. Lee *et al.* [8] reorganize and reduce the initial scan data to produce the contour data, which can be categorized and fabricated directly as a RP part. Liu *et al.* [9] proposed an error-based segmentation approach from random point cloud, by first subdividing the points into several regions, in each region, feature points will be detected for constructing an intermediate point-based curve model, where the RP layer contours are extracted from. Qiu *et al.* [15] demonstrated a method to extract topological structure from the point cloud, and apply that structure into a MLS (Moving-Lease Squares) surface-based direct slicing process to capture some important features near the positions of topological transitions. Compare with unorganized point cloud, which is lack of layered distribution and topological information, surfels on *Surfel-Grids* are layered distributed and placed orderly on a Grid structure, with neighborhood information attached.

Huang *et al.* [3] extract 2D contours on binary images by employing an MC-like contouring method. The binary image, whose pixels present nodes of 2D cells, is sampled from an implicit solid represented by LDNI [18]. The initial contour line inside each cell can be constructed by connecting the middle point of each edge with different node statuses, in respect to a certain pattern in the MC-like look-up table. However, the middle points are not exact positions of intersections, although two extra operations are employed to smooth the generated contours, sharp feature inside a cell is still not preserved properly. By contrast, the positions of surfels on *Surfel-Grids* are more accurate, and sharp features are reconstructed during the loop construction process as well.

The loop construction method of this work distinguished from the other grid/voxel based contouring approaches. Marching squares, an algorithm to extract contours for a two-dimensional scalar field, is a 2D form of Marching cubes method that shares a similar contouring strategy. Marching cubes (MC) [10] is probably the most popular isosurface algorithm, which performs triangulation cube-by-cube from scalar volumetric data sets. The triangulation pattern in each cube is determined from a pre-defined look-up table, which includes all of topologically unique isosurface-cube-intersection patterns. In order to find a matching pattern, each corner status (in/out) needs to be evaluated for that cube first. After that, to calculate the exact intersection positions for the corresponding pattern, linear interpolation is applied along the boundaries of that cube. The process steps that build the resultant isosurface can be considered as an assembly of triangulation patterns from a sequence of cubes. The original MC method has two limitations. Since the triangulation inside each cube is processed independently, topologically inconsistent decisions may be made for the ambiguous cubes. Many work have been done to resolve this problem [2],[12],[16]. The other drawback is feature-preserving problem. Dual contour methods [5],[16],[17] are presented to achieve a better feature estimation by using Hermite data (i.e., accurate intersection points associated with normal vectors) [6]. In brief, The MC-like loop construction method is pattern based, to identify a particular pattern for one cell, whose node status need to be evaluated first. By contrast, using the characteristics of Surfel-Grid, this contour method is simpler without cell node status pre-evaluation, pattern distinction and matching from the look-up table. Each surfel can access its neighboring surfels effortlessly, and the contour line inside each cell can be connected directly. Meanwhile, the efficiency of loop construction is not sensitive to the point density. Sharp features of the generated contour can be preserved properly during the loop construction process, and topologically consistent decisions will be made for the cells with ambiguous configurations as well. Contours with multiple loops have less impact on the whole efficiency of this work, since only *active* cells [11] are visited during loop construction process.

The remainder of this paper is organized as follows. The algorithm overview is presented in section 2. The detail algorithm which includes two major steps, i.e., *Surfel-Grid* generation includes introduction to *Surfel-Grid* representation, and loop construction are proposed in section 3 and section 4 respectively. Two approximation error criteria, i.e., Maximum Chord Error e_{max} and Chord Area e_{area},, which are used to evaluate the accuracy for the slicing contours, are introduced in section 5. Experiment and comparison results can be found in section 6. Finally, conclusion and future work are given in section 7.

2 ALGORITHM OVERVIEW

There are two procedures for this grid-based slicing algorithm: the *Surfel-Grid* generation and the Loop construction based on *Surfel-Grid*. Fig.2 gives an overview for the whole algorithm. We suppose that the input mesh model is closed with good connectivity.

The *Surfel-Grid* generation process (section 3) converts the STL mesh model (see Fig. 2(a)) into layers of *Surfel-Grids* (see Fig. 2(b)). For a *Surfel-Grid* structure, surfels (see orange and blue points in Fig. 2(c)) preserve sharp features while Grid (grey grid in Fig. 2(c)) provides neighborhood information for each surfel, both plays an important role in guaranteeing the efficiency and topologically consistency for loop construction in the next step.

After *Surfel-Grid* generation, a 2D grid-based loop construction algorithm is performed on the *Surfel-Grid* of each layer (section 4). Since the input mesh model is closed with good connectivity, it should occupy a set of continuously connected cells for each layer. Therefore, the loop construction upon the *Surfel-Grid* can be simply decomposed into line-segment connections within each cell. To improve the accuracy of layered contours, sharp features can be reconstructed by using normal vectors of surfels. Taking advantage of the neighborhood information and *cell validity* of *Surfel-Grid*, those line segments can be joined in sequence efficiently. The resulting slicing model is fully compatible with layered fabrication (see Fig. 2(d)).

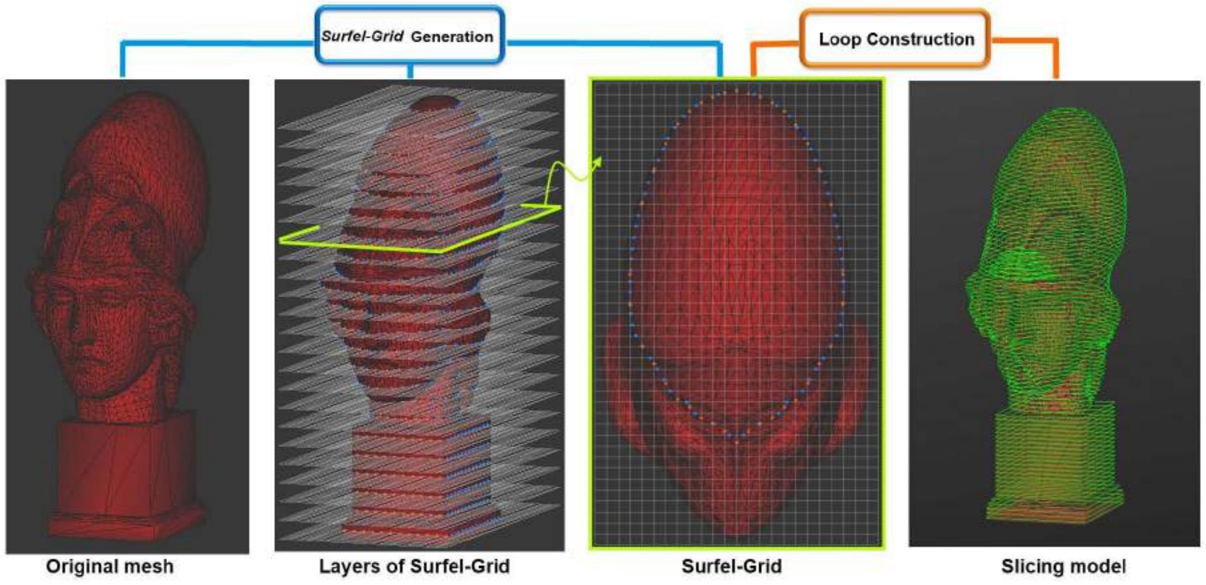

Fig. 2: *Surfel-Grid* based slicing algorithm overview.

3 *SURFEL-GRID* GENERATION

This section defines, *Surfel-Grid* representation, how to discretize and represent a mesh model with it, as well as the criteria to guarantee its validity will be introduced.

3.1 *Surfel-Grid* Representation

Surfel-Grid is a combination of a Grid and surfels (see Fig. 3(a)).

- **Grid** is formed by two orthogonal in-plane rays, made up with nodes, edges and cells. An edge aligns on the boundary of the Grid, denoted as *boundary* edge, and shared by only one cell; the other edges are simply called edge, shared by two *neighboring cells* (see edge $\boldsymbol{e}$ and its neighboring cells $\boldsymbol{C_0}$ and $\boldsymbol{C_1}$ in Fig. 3(b)).

- **Surfel** is a sampling point associated with normal vector, generated by intersecting the boundary of input model and an edge of the Grid.
- **Neighborhood information**

Different from the point-based geometry that presents a 3D model discretely, *Surfel-Grid* preserves original neighborhood information during the sampling procedure. Since the mesh model is closed with good connectivity, it should occupy a set of consecutively connected cells for each layer. As shown in Fig. 3(b), each surfel (say, ***p*** in dark blue) associated on an edge (***e***, thick edge in orange), can find its *neighboring surfels* (***q*** and ***r*** in light blue) effortlessly by the *neighboring cells* (i.e., $\boldsymbol{C_0}$ and $\boldsymbol{C_1}$) of that edge (i.e., ***e***).

- **Validity of a *Surfel-Grid***

Validity of a *Surfel-Grid* is critical to guarantee the efficiency of loop construction and topological correctness of the generated contour in the next step. A *Surfel-Grid* is considered to be valid, when each *non-empty* cell (i.e., cell associated with surfels) is valid, and every surfel can find its *neighboring surfels* by *neighboring cells* of the edge that surfel associated with.

Cell validity - two criteria are derived to identify the validity of a *non-empty* cell: 1) there are even number of surfels inside the cell; 2) there is no more than 1 surfel associated on each edge. Since *Surfel-Grid* is a sampling-based representation, features of the original model, whose size is smaller than the cell size, cannot be precisely presented by this representation. In general, only a *2-surfel cell* (two surfels associated on different edges of a cell, see top of Fig. 3(c)) and a *4-surfel cell* (four surfels associated on the different edges of a cell, see bottom of Fig. 3(c)) are valid by definition. Blue dashed curves indicate the 2D illustration for the boundary of the original model in all of the figures below.

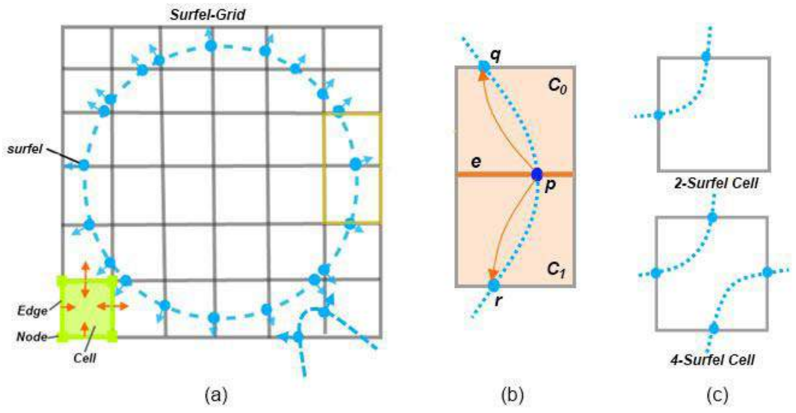

Fig. 3: (a) *Surfel-Grid* representation, (b) *neighboring surfels* ***q, r*** can be accessed by surfel ***p*** through the *neighboring cells* $\boldsymbol{C_0}$, $\boldsymbol{C_1}$ of ***e***, (c) *Cell validity*: only 2-Surfel Cell and 4-Surfel Cell are valid by defination. Blue Dashed curve presents the original boundary of the input model.

Surfel-Grid representation presents the boundary of 3D model with layers of grid-structured surfels, and all of surfels are generated by adopting LDNI discretization technique in this work. However, *Surfel-Grid* is proposed mainly for the slicing purpose, whose two particular characteristics (i.e., neighborhood informaiton and *cell validity* criteria discussed above) are dedicated to improve the slicing performance, which can also be used to make distinction from LDNI representation [18]. Besides presenting the geometric shape discretely like LDNI, the initial neighborhood information of original model can be reserved as well during the process of constructing *Surfel-Grid*, so that each surfel can access its neighboring surfels effortlessly. To speed up and guarantee the topological consistent contours generated during the loop construction process, the *cell validity* of *Surfel-Grid* needed to be fulfilled before performing loop construction.

3.2 *Surfel-Grid* Construction

This section includes LDNI sampling and Surfel-Grid construction.

In order to improve the contour accuracy and speed up the mesh model discretization, the 2-dimensional LDNI technique is employed in this paper. After LDNI sampling, two orthogonal in-plane rays form a Grid for each layer, at the same time, surfels are generated along each ray randomly.

To construct a *Surfel-Grid*, surfels should be mapped onto corresponding edges of Grid properly, according to their depth values. As illustrated in Fig. 4, the yellow line is a 2D illustration of the view plane for each sampling direction. The depth value of each surfel to the view plane can be retrieved from the depth buffer supported by OpenGL. To determine which edge a surfel (circled in green) with depth ***d*** (green arrow) should be mapped to, simply refer to Eqn. (3.1) and Eqn. (3.2).

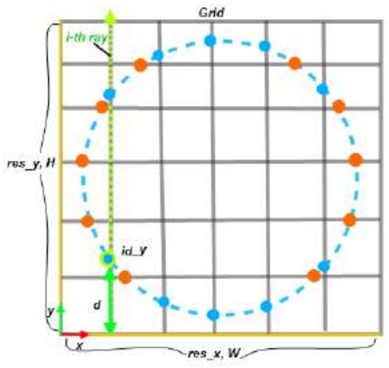

Fig. 4: *Surfel-Grid* construction: mapping each generated surfels to one proper edge according to its depth value ***d***.

As shown in Fig. 4, *res_x* and *res_y* are the number of cells along the x-axis and y-axis respectively; *W* and *H* are the width and height of Grid respectively. An edge of index (*i*, *id_y*) means that it is the *id_y*-th edge on the *i*-th ray (green dashed line with arrow) along the y-axis.

On the *i*-th ray of the y-axis, a surfel (in blue) with depth ***d***, should be mapped to an edge of that ray, whose index is *id_y*, which can be calculated by Eqn. (3.1), *integerized* by means of chopping the decimal value to the nearest integer.

$$id\_y = \text{integerized}\left(\frac{d \times res\_y}{H}\right) \tag{3.1}$$

Similarly, for a surfel (in orange) on the *j*-th ray along the x-axis, the index of the edge (i.e., *id_x*) can be calculated by Eqn. (3.2):

$$id\_\text{x} = \text{integerized}\left(\frac{d \times res\_\text{x}}{W}\right) \tag{3.2}$$

ambiguous Surfels – As illustrated in Fig. 6(a), a surfel (circled in blue) may locate close to a node such that it become an *ambiguous surfel* that could be mapped to either of the edges (vertical edges pointed by arrows) sharing that node. In order to make sure the *Surfel-Grid* is valid, a*mbiguous Surfels* are only mapped after performing the *Harmonize* operation.

3.3 *Surfel-Grid* Processing

Since the validity of the *Surfel-Grid* is essential to the efficiency and correctness of loop construction, the *cell validity* of *non-empty* cells should be checked after the *Surfel-Grid* generation. According to the criteria of *cell validity*, a cell is invalid if it is neither a *2-surfel cell* nor a *4-surfel cell*. There are two operators created to handle the invalid cells: *Merge* and *Harmonize*.

3.3.1 Merge

There may be more than one surfel associated on one edge of a cell, when the size of a feature to be presented is smaller than the cell size. The surfels on one edge need to be merged according to the number of surfels on that edge is even or odd:

- Even number of surfels (see Fig. 5(a)): all of the surfels on that edge should be removed;

- Odd number of surfels (see Fig. 5(b)): only one surfel is kept for that edge.

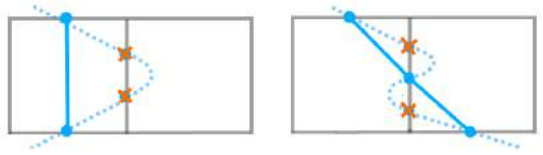

Fig. 5: *Merge* operator for edge with more than one surfels associated with: (a) even number of surefls, (b) odd number of surfels.

3.3.2 Harmonize

The mapping of *ambiguous Surfels* to the grid is suspended during the *Surfel-Grid* construction process, and can be continued after the mapping of the other *non-ambiguous Surfels* has been finished. As illustrated in Fig. 6(a), the *ambiguous Surfel* (circled in blue) locates very close to the node that could be associated to either the upper or lower edge along that vertical ray. The number sign in each of the four related cells states how many surfels are being attached **without** counting *ambiguous Surfel*. After placing that *ambiguous Surfel* onto a proper edge (i.e., upper edge in Fig. 6(b)), every related *non-empty* cell is fulfilled with *cell valid* (i.e., either *2-surfel cell* or *4-surfel cell*).

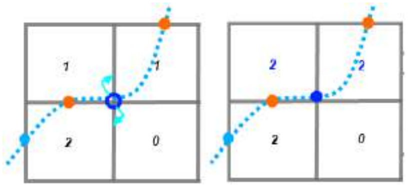

Fig. 6: *Harmonize* operator for *ambiguous surfels*: (a) Before mapping the *ambiguous surfel* (highlighted in blue circle, (b) After performing *Harmonize*, each *non-empty* cell is valid. Blue surfels are associated on the vertical rays, while orange ones are on horizontal rays.

After *Surfel-Grid* processing, the *Surfel-Grid* of each layer is valid, i.e., every *non-empty* cell is either a *2-surfel cell* or *4-surfel cell*, and every surfel can find its *neighboring surfels* by *neighboring cells* of the edge that surfel associated with.

4 LOOP CONSTRUCTION

The original mesh model is converted into layers of valid *Surfel-Grids* after *Surfel-Grid* generation. Based on a valid *Surfel-Grid* of each layer, loop construction can be simply decomposed into line-segment connections within each cell. Take advantage of the neighborhood information extracted from *Surfel-Grid*, those line segments can be joined in sequence efficiently. In order to increase the accuracy of the slicing contour, *feature points* are estimated within the cells that contain sharp features. With Fig.7 as reference, terminologies used in the algorithm are introduced below.

Terminologies:

- *seed surfel* (highlighted in the pink circle) – starting point of an individual loop;
- *inactivated edge* – an edge associated with a surfel that has not been connected to any loop yet;
- *unaccomplished cell* – a cell contains *inactivated edge*;
- *current surfel* (highlighted in the green circle) – a starting point of line-segment to be connected inside its *neighboring cell*, which is also the end point of the line-segment has already been connected inside the other *neighboring cell*, if it is not a *seed surfel*;
- *pair cell* (highlighted in yellow) – a *neighboring cell* of *current surfel*, which is also an *unaccomplished cell*;
- *pair surfel* –the end point of line-segment to be connected in the *pair cell*, which is associated on an *inactivated edge* of *pair cell*;

- *neighbor* –each surfel has and only has two *neighbors* to connect directly in a loop. A *neighbor* can be either a surfel or a *feature point*. For example, a *pair surfel* is a *neighbor* of *current surfel* and vice versa, if there is no *feature point* generated inside the *pair cell* (see Fig. 8(b)); otherwise, the *feature point* is a *neighbor* for both *current surfel* and *pair surfel* (see Fig. 8(a)). Note that *neighbor* is not identical with *neighboring surfel* mentioned above.

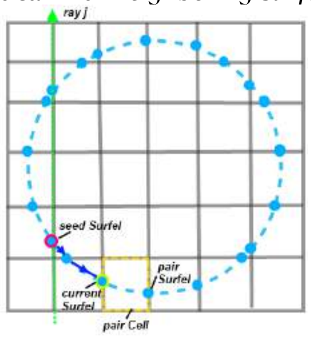

Fig. 7: Illustration of loop construction process.

To prevent self-intersection occurring, each surfel should participate in one loop only. There are two steps for loop construction: finding *seed surfel* and finding *pair surfel*. The pseudo-code for the whole loop construction procedure refers to Algorithm 4.1.

4.1 Finding *seed surfel*

As illustrated in Fig. 7, a *seed surfel* can actually be any surfel on that loop, since the loop is closed. In this work, a *seed surfel* can be found on the first *inactivated edge* when traversing each edge along each vertical ray from left to right. The procedure of finding *seed surfel* is carried out several times for multiple loops. To start a new loop, *seed surfel* is set as *current surfel* and then passed into next step.

4.2 Finding *pair surfel*

As mentioned before, each surfel only connect with its *neighbors* in a loop. For each given *current surfel*, the process of finding *pair surfel* is also a process of finding a *neighbor* to connect and then generate new line-segments inside a *pair cell*. The *pair surfel* is first set as *current surfel*, and the whole procedure is repeated till the loop is finally closed.

To improve the accuracy of generated contours, sharp features are preserved by *Surfel-Grid*, and reconstructed in *2-surfel cells* that contain sharp features in this work. Since the *Surfel-Grid* is valid, the *pair cell* is either *2-surfel cell* or *4-surfel cell*. Pseudo-code for this step can be found in Algorithm 4.2.

4.2.1 2-surfel cell

As shown in Fig. 8, a *pair surfel* (i.e., $\boldsymbol{q}$) for *2-surfel cell* is always the one attached on an *inactivated edge*. Instead of directly connecting the *current surfel* $\boldsymbol{p}$ (circled in green) with the *pair surfel* $\boldsymbol{q}$, a *feature point* (red point) is obtained as suggested by Kobbelt et al. [6] to reduce the approximation error.

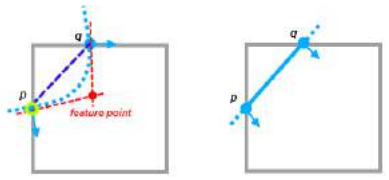

Fig. 8: Finding *pair surfel* in a *2-surfel cell*: (a) cell with sharp feature, (b) cell without sharp feature.

Feature Estimation – as illustrated in Fig. 8(a), if a *2-surfel cell* contains a sharp feature, the intersection of two tangent vectors (dashed lines in red) of those two surfels yields a point inside this cell, denoted as *feature point* (red point), which is closer to the real sharp feature (peak point along the blue dashed curve) than directly connecting those two surfels (purple dashed line). The tangent vector of one surfel can be computed by using the projection of its normal vector on the grid plane. Since this is a contour extraction problem in 2D, the computation process is quite straightforward, i.e., resolving two linear functions with two unknown variables, those two variables are actually 2D coordinates of the *feature point* on that grid plane.

However, *Feature Estimation* calculation is not performed for every *2-surfel cell*. In this work, a *2-surfel cell* is identified as containing a sharp feature only if the angle between two surfel normal vectors is larger than a user defined threshold (e.g., 5 degree in this work).

If there is no *feature point* generated in a *2-surfel cell* (see Fig. 8(b)), then *current surfel* and *pair surfel* are *neighbors* to each other, and there is only one line-segment generated inside this cell; otherwise, their *neighbors* will be the new generated *feature point*, and two line-segments are generated: *current surfel - feature point* and *feature point - pair surfel* (see Fig. 8(a)). Therefore, a *pair surfel* can be a *neighbor* of *current surfel*, only if there is no *feature point* inside that cell (see Fig. 8(b)).

4.2.2 4-surfel cell

Similar to the problem occurs in MC-like methods, the loop topology inside the *4-surfel cell* is ambiguous due to inadequate sampling resolution.

Ambiguous topology in 4-surfel cell - as shown in Fig. 9, any one of those candidate *neighboring surfels* (highlighted in orange) in the *4-surfel cell*, could be a *pair surfel* for the *current surfel* (highlighted in green circle), however, different configurations present different contour topologies inside that cell, one is 'thin-shell' (see Fig. 9(a)), the other is 'gap' (see Fig. 9(b)), In order to generate topologically consistent contours, as suggested by Wang and Chen [17], one configuration should be chosen consistently. The configuration (see in Fig. 9(b)) of producing a 'gap' is selected by this work. This separate-loops preferring choice denoted as '***separate loops preference***'.

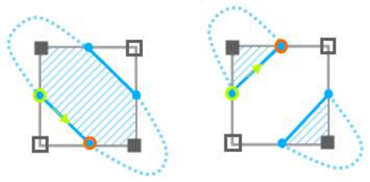

Fig. 9: Ambiguous topologies in *4-surfel cell*: (a) thin-shell, (b) gap.

Separate loops preference - As illustrated in Fig. 10, In order to make sure the two line-segments generated in the *4-surfel cell* belong to two separate loops, the candidate *neighboring surfels* of *current surfel* in the *4-surfel cell* are needed to be classified into IN/OUT states. A surfel will be identified as 'IN' when it locates at a position where the ray enters the boundary of the original model, while as 'OUT' when the ray goes out. No matter what state the *current surfel* is, its *pair surfel* is always the one with 'OUT' state among those two candidate *neighboring surfels* (which are always with opposite states). Therefore, the generated contours are topologically consistent by following this preference. For the *current surfel* of *4-Surfel cell*, its *pair surfel* is always its *neighbor* that connected with directly.

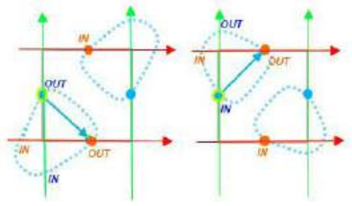

Fig. 10: Separate loops preference. Rules of generating two line-segments which belong to two separate loops: (a) 'OUT' *current surfel* connects with 'OUT' *pair surfel*, (b) 'IN' *current surfel* connects with 'OUT' *pair surfel*.

Algorithm 4.1 – ***Loop Construction***

```
Input: Surfel-Grid
while(1)
1.      Find seed surfel
        if ( cannot find any seed surfel )
            break;
2.      current surfel = seed surfel;
        while (1)
3.        pair surfel = FindPairSurfel(current surfel);
          if (pair surfel is seed surfel)
              break;
        End
End
Output: Loop(s)
```

Algorithm 4.2 – ***FindPairSurfel***

```
Input: current surfel
1.1     if (current surfel is seed surfel)
          pair cell <- one of the neighboring cells of current surfel;
1.2     else
          pair cell <- the neighboring unaccomplished cell;
2.      check sub-case of the pair cell:
2.1       if (2-surfel cell)
3.1         pair surfel <- neighboring surfel on the inactivated edge of pair cell
4.          Feature Detection (2-surfel cell)
4.1           if (contains sharp feature)
5.              feature point <- Feature Estimation (current surfel, pair surfel)
                neighbor of current Surfel <- feature point
                connect current surfel with feature point, add line-segment into Loop(s)
                neighbor of pair Surfel <- feature point
                connect feature point with pair surfel, add line-segment into Loop(s)
4.2           else
                neighbor of current Surfel <- pair surfel
                neighbor of pair Surfel <- current surfel
                connect current surfel with pair surfel, add line-segment into Loop(s)
2.2       else if (4-surfel cell)
3.2         pair surfel <- separate loop preference (i.e., candidate neighboring surfel with 'OUT' state)
6.      Activated the edge where pair surfel is lying
7.      the number of remaining surfels of that pair cell -= 2;
8.      if (account==0)
          Set pair cell to be accomplished;
Output: pair surfel
```

5 LOOP ACCURACY ESTIMATION

To estimate the approximation error and compare the slicing accuracy with other slicing algorithms, two approximation error estimation criteria are used to evaluate the accuracy of the slicing contours: Maximum Chord Error e_{max} and Chord Area e_{area}.

5.1 Maximum Chord Error e_{max}

Maximum chord error is the maximum Hausdorff distance between the boundary of the original model and the slicing contour [13]. The Hausdorff distance for a mesh model and its slicing model cannot be estimated directly. As illustrated in Fig. 11, for each layer, the boundary of the original mesh (green line-segments) can be obtained from calculating the intersection between the mesh boundary and the layer plane (black grid). Green dots are intersection points (e.g., V_n, V_{n+1}) between edges of input mesh and layer plane, denoted as *plane vertices*, intersection line-segments are the green lines that connect adjacent *plane vertices* (e.g., $\overline{V_nV_{n+1}}$). Blue dots are surfels (e.g., *p*, *q*) lying at the intersection position of intersection line-segments of original mesh (e.g., $\overline{V_nV_{n+1}}$) and the edges of cells. Similarly, $\overline{pq}$ is a line-segment on the slicing contour that connects adjacent surfels (i.e., *p* and *q*).

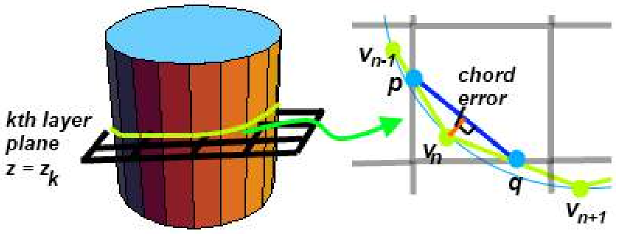

Fig. 11: Chord error estimation for mesh model.

Therefore, the *vertex chord error* for each *plane vertex*, denoted as *chord_error*$_V$, is the distance from the vertex (e.g., V_n (u_V, v_V)) to its corresponding line-segment on the slicing contour $\overline{pq}$ (i.e., $\overline{(x_p,y_p),(x_q,y_q)}$) by using Eqn. (5.1).

$$\begin{cases} chord\_error_V = \sqrt{\dfrac{\left|\left(u_V - x_p\right)\left(x_q - x_p\right) + \left(v_V - y_p\right)\left(y_q - y_p\right)\right|}{\sqrt{\left(x_q - x_p\right)^2 + \left(y_q - y_p\right)^2}}} \\ x_p \le u_V < x_q \quad or \quad y_p \le v_V < y_q \end{cases} \tag{5.1}$$

The maximum *vertex chord error* for each layer can be obtained after comparing *chord_error*$_V$ for every *plane vertex* of that layer. The final maximum chord error for the whole slicing model e_{max}, is the **average value** among all of the maximum *vertex chord error* from each layer.

5.2 Chord Area e_{area}

Besides maximum chord error, another criterion chord area is derived for estimating accuracy, denoted as e_{area}. As illustrated in Fig. 12, for each layer, chord area is the difference (orange areas) between the area of the polygon enclosed by the boundary of mesh model (i.e., green polygon) and the area of the polygon enclosed by the slicing contour (i.e., blue polygon).

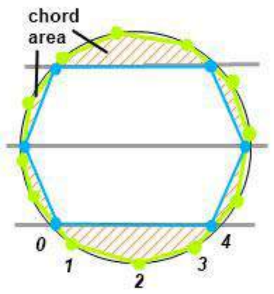

Fig. 12: Chord area for mesh model.

The chord area of each layer, denoted as e_{area_layer}, can be estimated by using Eqn. (5.2), where ***M*** is the number of *plane vertices* (u_i,v_i), ***N*** is the number of surfels (x_j,y_j).

$$e_{area\_layer}=\frac{1}{2}\sum_{i=0}^{M\text{-}1}(u_i v_{i+1}\text{-}u_{i+1}v_i)\text{-}\frac{1}{2}\sum_{\mathrm{j}=0}^{\mathrm{N}\text{-}1}(x_j y_{\mathrm{j}+1}\text{-}x_{\mathrm{j}+1}y_j) \tag{5.2}$$

The final chord error for the whole slicing model e_{area}, is the **average value** of all e_{area_layer} from each layer.

6 EXPERIMENT RESULTS AND COMPARISON

To demonstrate the main characteristic of the proposed algorithm, i.e., achieving more accurate slicing contours in a more efficient manner, as illustrated in Fig. 16, Fig. 17, and Fig. 18, all of three examples contains multiple loops and sharp features, are tested respectively by this and Zeng's work in the same computing environment, i.e., 64 bit Windows 7 PC (Intel® Core™ i5 CPU 750 @ 2.67GHZ, 2.66GHZ, and 4.00GB RAM). The accuracy of slicing contours can be demonstrated from two aspects, the approximation errors (i.e., maximum chord error e_{max} and chord area e_{area}) introduced above, as well as sharp feature preservation. Similarly the slicing efficiency can be presented from two aspects: slicing time cost on achieving certain accuracy and the efficiency of loop construction under different *sampling resolutions*. This section contains two parts: experiment results and comparison between this and Zeng's work.

6.1 Experiment Results

Firstly, a set of approximation errors (e_{max} and e_{area}) of slicing contours and the total slicing time T_t are recorded during increasing the sampling resolution for each example, and total slicing time T_t consists of two steps: LDNI sampling time T_{LDNI} and loop construction time T_{Loop}. In this work, T_{LDNI} is 2 dimensional LDNI sampling, and T_{Loop} is the total time of *Surfel-Grid* construction and loop construction; while in Zeng's work, T_{LDNI} is 1 dimensional LDNI sampling, and T_{Loop} is just for loop construction time.

In order to obtain similar *sampling resolutions* ***w*** (i.e., total number of rays) for both work, the *resolution interval* ***Res*** (i.e., distance between adjacent rays in millimeter) of each direction employed in this work is simply set as **twice** as the one applied in Zeng's work, since this work is based on 2 dimensional LDNI sampling while Zeng's work is 1 dimensional. As the number of layers does not affect the contour accuracy, the layer thickness is fixed as 0.0762 mm for all examples and for both work.

Each example is tested by the same set of *sampling resolutions* in increasing order (i.e., *resolution interval Res* are shown in decreasing order), and the experiment results for both work can be found in Tab. 1.

Example		This work				Zeng's work			
	Res/mm	0.1524	0.1016	0.0508	0.0254	0.0762	0.0508	0.0254	0.0127
Fig. 16	e_{area}/mm2	5.981	3.245	0.382	0.203	14.168	13.299	10.890	8.707
	e_{max}/mm	0.187	0.089	0.068	0.059	0.786	0.770	0.756	0.752
	T_t/ms	353	388	500	711	312	358	470	722
	T_{LDNI} | T_{Loop}	326 | 27	335 | 53	393 | 107	462 | 249	218 | 94	233 | 125	268 | 202	311 | 411
Fig. 17	e_{area}/mm2	1.252	0.8	0.114	0.021	3.301	2.782	0.829	0.395
	e_{max}/mm	0.058	0.055	0.027	0.015	0.083	0.077	0.043	0.022
	T_t/ms	159	179	232	298	148	172	235	340
	T_{LDNI} | T_{Loop}	142 | 17	155 | 24	179 | 53	192 | 106	119 | 29	125 | 47	164 | 71	185 | 155
Fig. 18	e_{area}/mm2	2.722	1.128	0.591	0.472	5.428	3.765	1.181	0.787
	e_{max}/mm	0.189	0.119	0.089	0.071	0.954	0.756	0.438	0.245
	T_t/ms	198	240	325	391	204	243	294	377
	T_{LDNI} | T_{Loop}	183 | 15	218 | 22	280 | 45	305 | 86	171 | 33	196 | 47	203 | 91	221 | 156

Tab. 1: Experiment results of three examples shown in Fig. 16, Fig. 17 and Fig. 18 for this and Zeng's work under **similar** *sampling resolutions* ***w***: In order to obtain similar *sampling resolutions* (i.e., total number of rays) for both work, the *resolution interval Res* (distance between adjacent rays in millimeter) of each direction employed in this work is simply set as **twice** as the one applied in Zeng's work (first row), since this work is based on 2 dimensional LDNI sampling while Zeng's work is 1 dimensional. A same set of *sampling resolutions* in increasing order (i.e., resolution interval *Res* in decreasing order) are applied for each example. Contour accuracy is presented by two approximation errors mentioned above, i.e., chord error e_{area} (second row) and maximum chord error e_{max} (third row). Total slicing time T_t (forth row) is composed of LDNI sampling time T_{LDNI} and loop construction time T_{Loop} in the fifth row.

From the experiment results recorded in the Tab. 1, we can find that, this slicing method can achieve a better contour accuracy overall, since smaller approximation errors (i.e., chord area e_{area} and maximum chord error e_{max}) can be obtained for each example and under each *sampling resolution* by this work. Moreover, the slicing efficiency is not sacrificed by this work for the sake of accuracy, the total slicing time T_t for both work are maintained as similar for each individual *sampling resolution*, even though the LDNI sampling time T_{LDNI} of this work is much longer than Zeng's work (nearly two times), which illustrates higher efficiency of this loop construction. For each example, the loop construction time T_{Loop} of this work is nearly half of Zeng's work under each sampling resolution.

Both of this and Zeng's work share a similar trends which can be obtained from the Tab. 1 as well: the smaller the *resolution interval Res* is, the higher *sampling resolution* is, the longer the slicing time T_t cost, and better the accuracy (lower approximation errors e_{max} and e_{area}) will be achieved.

To demonstrate the trends for both work more clearly, approximation errors e_{area} and e_{max} are plotted w.r.t. to the total slicing time T_t for each example according to the data recorded in Tab. 1, are shown in Fig. 13, Fig. 14 and Fig. 15 respectively.

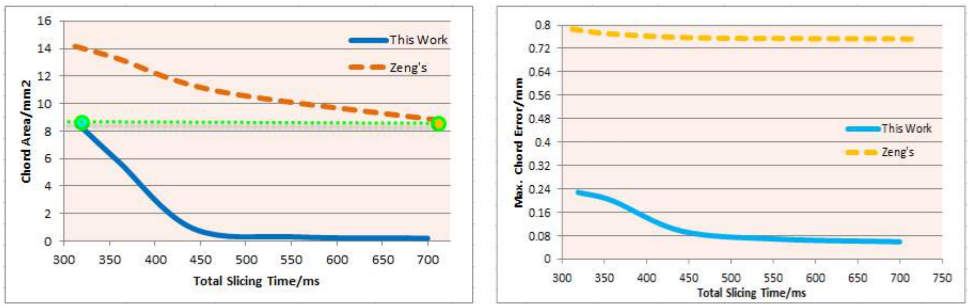

Fig. 13: Example in Fig. 16: the trends of approximation errors (e_{area} and e_{max}) with respect to the total slicing time (T_t) for this and Zeng's work: (a) e_{area}-T_t, (b) e_{max}-T_t.

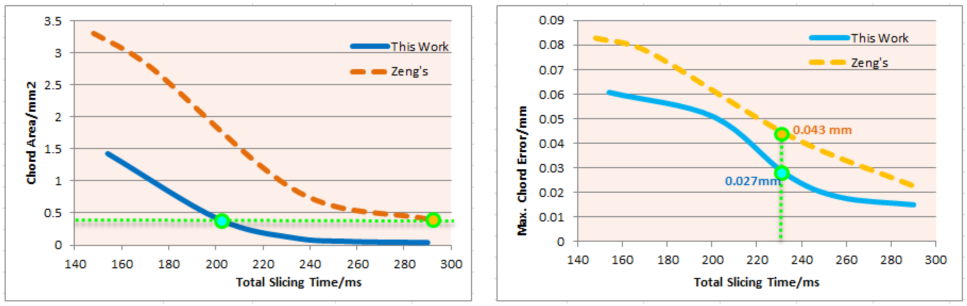

Fig. 14: Example in Fig. 17: the trends of approximation errors (e_{area} and e_{max}) with respect to the total slicing time (T_t) for this and Zeng's work: (a) e_{area}-T_t, (b) e_{max}-T_t.

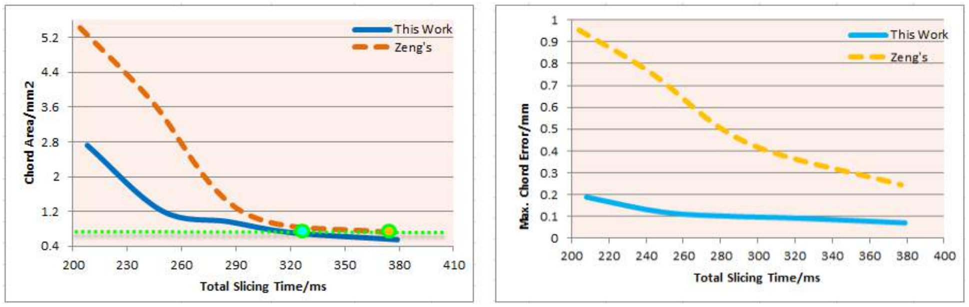

Fig. 15: Example in Fig. 18: the trends of approximation errors (e_{area} and e_{max}) with respect to the total slicing time (T_t) for this and Zeng's work: (a) e_{area}-T_t, (b) e_{max}-T_t.

Secondly, to demonstrate the main characteristic of this work (i.e., achieving more accurate layer contours in a more efficient manner) more specifically, a comparison between this and Zeng's work is applied from two aspects (i.e., accuracy and efficiency) in the section below.

6.2 Comparison

6.2.1 Accuracy

The Accuracy for this and Zeng's work is compared from two different angles: approximation errors e_{area} and e_{max}, and sharp feature preservation.

- **Approximation errors e_{area} and e_{max}** – the slicing contour accuracy can be presented by two approximation errors chord area e_{area} and maximum chord error e_{max} as mentioned above, the smaller the values are, the more accuracy of the slicing contour is. The contour accuracy comparison between this and Zeng's work can be performed by comparing those two errors of slicing contours, with respect to similar total slicing time Tt. Taking example model in Fig. 17 for example, as shown in the e_{max}-T_t of Fig. 14, the e_{max} of this work is ~0.027mm while ~0.043mm for Zeng's work during the same slicing time T_t, i.e., 230ms. Similar accuracy comparison results can be addressed from the other figures (see e_{max}-T_t of Fig. 13 and Fig. 15) as well. Therefore, higher contour accuracy can be achieved by this work than Zeng's work during the same slicing time.
- **Feature preserving** - from the detailed view of example Fig. 16(c) which shows the zoom-in area in the yellow circle in Fig. 16(b), the slicing contours (green curves) of the proposed algorithm fits the original mesh model (red) very well, and all of the sharp features are preserved properly. In contradiction, the deviation between the slicing contour (in blue) and the boundary of the original mesh model done by Zeng's work is quite obvious (see Fig. 16(d). So for example Fig. 17 and example Fig. 18 in a similar way.

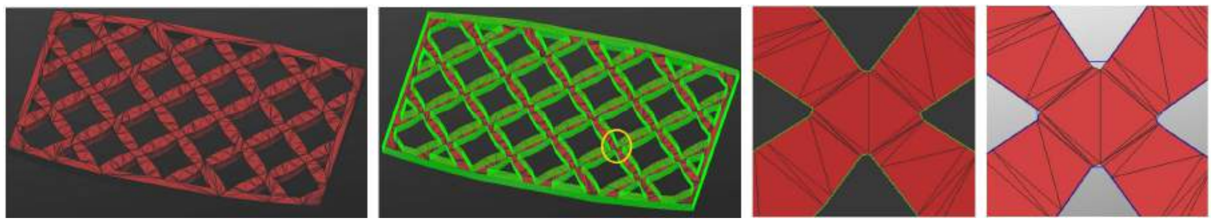

Fig. 16: Example Fig. 13: (a) Original model, (b) Slicing model, (c) detail view of this work, (d) detail view of Zeng's work.

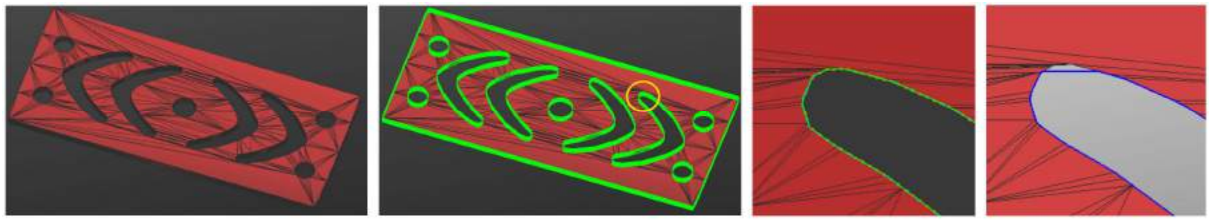

Fig. 17: Example Fig. 14: (a) Original model, (b) Slicing model, (c) detail view of this work, (d) detail view of Zeng's work.

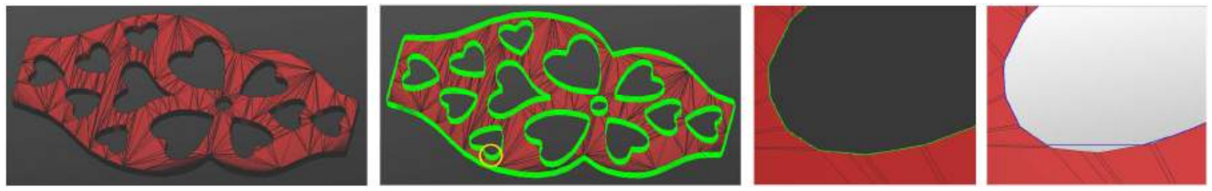

Fig. 18: Example Fig. 15: (a) Original model, (b) Slicing model, (c) detail view of this work, (d) detail view of Zeng's work.

6.2.2 Efficiency

The efficiency comparison for both work are carried out from two parts as well:

- Total Slicing time T_t of achieving the same contour accuracy - the slicing contour accuracy can be presented by approximation errors: maximum chord error e_{max} and chord area e_{area} as mentioned above. Comparison on slicing efficiency between this and Zeng's work, can be executed by comparing which algorithm cost shorter slicing time T_t to achieve the same approximation error e_{max} or e_{area}. As illustrated from e_{area} – T_t in Fig. 16, Fig. 17 and Fig. 18, we can find out that, this work takes obvious shorter time to achieve the same approximation error than Zeng's work.
- Efficiency of loop construction – as mentioned in the the section 6.1, to make sure the expriment can be executed under similar *sampling resolution* ***w*** (i.e., total number of rays), the *resolution interval* ***Res*** (i.e., distance between adjacent rays) employed by this work is simply set as twice as the one applied in Zeng's work, since 2 dimentional LDNI sampling is obtained by this work. Those *sampling resolutions* can be simiply denoted as *w1*, *w2*, *w3* and *w4* in Fig. 19, which are corresponding to the resolution interverals for both work respectively in the first row of Tab. 1. The loop construction time T_{Loop} of example Fig. 18 in Tab. 1, are ploted according each *sampling resolution* w, and shown in Fig. 19. We can find that, the loop contructioin time of this work is nearly half of Zeng's work for each *sampling resolution w*, which demenstrate a better loop construction efficiency of this work.

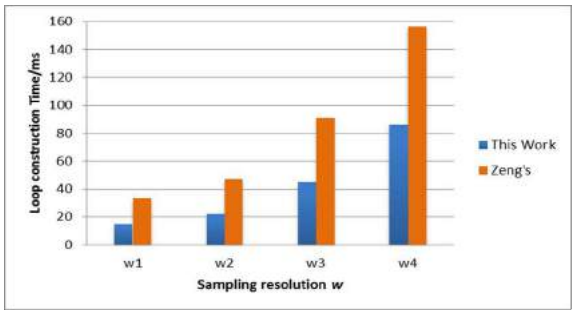

Fig. 19: Loop construction time comparison of example Fig. 18 between this and Zeng' work under the same *sampling resolution* ***w***.

7 CONCLUSION AND FUTURE WORK

As shown in Fig. 13, Fig. 14 and Fig. 15, both e_{area}-T_t and e_{max}-T_t follow a similar trend in this and Zeng's work - slicing accuracy is improved while the computational time becomes longer. However, this work presents an improved accuracy with both the errors e_{area}-T_t and e_{max}-T_t showing better results. Experiment results in Tab. 1 demonstrate that this slicing algorithm can achieve higher accuracy than Zeng's work [19] during the same period of time. Moreover, this work took 15% to 50% less time than Zeng's work to achieve the same accuracy, which means we have indeed achieved higher accuracy and better efficiency.

To increase the accuracy of slicing contours as well as maintain an overall high efficiency in the proposed work, sharp features are estimated in the *2-surfel cells* only. The future work may preserve the sharp features in the *4-surfel cells* as well to further improve the slicing accuracy.

During testing, it was also found that most of the slicing time (near 80%) costs are on LDNI sampling procedure, since LDNI sampling was adopted for each sampling direction separately. Therefore, future work may consider further reducing the sampling time by employing parallel sampling technique.

8 ACKNOWLEDGEMENT

The example model 1, 2 and 3 in Fig. 13, Fig. 14 and Fig. 15 are exported from the data base of JewelCADPro software [4], the first author would like to acknowledge the support by Innovation and Technology Fund UIM/210 and Jewellery CAD/CAM Ltd.

REFERENCES

[1] Hoffmann, C.-M.: Robustness in geometric computations, ASME Journal of Computing and Information Science in Engineering, 2001.
[2] Ho, C.-C.; Wu, F.-C: Cubical Marching Squares: Adaptive Feature Preserving Surface Extraction from Volume Data, Computer Graphics Forum, 24(3), 2005, 537-545. http://dx.doi.org/10.1111/j.1467-8659.2005.00879.x.
[3] Huang, P.; Wang, C.-L.; Chen, Y.: Self-intersection free and topologically faithful slicing of implicit solid, ASME IDETC/CIE 2011 Conference, Washington, DC, USA, 2011.
[4] JewelCAD Pro, http://www.jewelcadpro.com, Jewellery CAD/CAM Ltd.
[5] Ju, T.; Losasso, F.; Schaefer, S.; Warren, J.: Dual contouring of Hermite data, ACM Transactions on Graphics, 21(3), 2002, 339-346. http://dx.doi.org/10.1145/566654.566586.
[6] Kobbelt, L.-P.; Botsch, M.; Schwanecke, U.; Seidel, H.-P.: Feature sensitive surface extraction from volume data, In Proc. of SIGGRAPH 2001, 57-66.
[7] Kumar, V.; Dutta, D.: An assessment of data formats for layered manufacturing, Computer-Aided Design, 28(3), 1997, 151-164.
[8] Lee, K.-H.; Woo, H.: Direct integration of reverse engineering and rapid prototyping, Computers & Industrial Engineering, 38(1), 2000, 21-38. http://dx.doi.org/10.1016/S0360-8352(00)00017-6.
[9] Liu, G.-H.; Wong, Y.-S.; Zhang, Y.-F., Loh, H.-T.: Error-based segmentation of cloud data for direct rapid prototyping, Computer-Aided Design, 35(7), 2003, 633-645. http://dx.doi.org/10.1016/S0010-4485(02)00087-8.
[10] Lorensen, W.-E.; Cline, H.-E.: Marching cubes: A high resolution 3D surface construction algorithm. In SIGGRAPH 1987, ACM Press, New York, USA, 1987, 163-169.
[11] Newman, T.-S.; Yi, H.: A survey of the marching cubes algorithm, Computer & Graphics, 30(5), 2006, 854-879. http://dx.doi.org/10.1016/j.cag.2006.07.021.
[12] Nielson, G.-M.; Hamann, B.: The asymptotic decider: resoving the ambiguity in marching cubes, Visualization, 1991.
[13] Pandey, P.-M.; Reddy, N.-V.; Dhande, S.-G.: Slicing procedures in layered manufacturing: a review, Rapid Prototyping Journal, 9(5), 2003, 274-288. http://dx.doi.org/10.1108/13552540310502185.
[14] Pfister, H.; Zwicker, M.; van Baar, J.; Gross, M.: Surfels: surface elements as rendering primitives, In SIGGRAPH 2000, ACM Press, New Orleans, LA, USA, 2000, 335-342.

[15] Qiu, Y.-J.; Zhou, X.-H., Qian, X.-P.: Direct slicing of cloud data with guaranteed topology for rapid prototyping, The International Journal of Advanced Manufacturing Technology, 53(1), 2011, 255-265. http://dx.doi.org/10.1007/s00170-010-2829-6.

[16] Schaefer, S.; Ju, T.; Warren, J.: Manifold dual contouring, IEEE Transactions on Visualization and Computer Graphics, 13(3), 2007. http://dx.doi.org/10.1109/TVCG.2007.1012.

[17] Wang, C.-L.; Chen, Y.: Layered Depth-Normal Images for Complex Geometries: Part Two --- Manifold-Preserved Adaptive Contouring, ASME Conference 2008, 729-739.

[18] Wang, C.-L.; Leung, Y.-S.; Chen, Y.: Solid modeling of polyhedral objects by Layered Depth-Normal Images on the GPU, Computer-Aided Design, 42(6), 2010, 535-544. http://dx.doi.org/10.1016/j.cad.2010.02.001.

[19] Zeng, L.; Lai, M.-L.; Qi, D.; Lai, Y.-H.; Yuen, M.-F.: Efficient slicing procedure based on adaptive layer depth normal image, Computer-Aided Design, 43(12), 2011, 1577-1586. http://dx.doi.org/10.1016/j.cad.2011.06.007.