A B S T R A C T

A Fast Slicing Algorithm of the 3D-Scanned Point Clouds Object for Rapid Prototyping

ByungJin Moon

Mechanical Engineering Major
Department of Mechanical & Aerospace Engineering
The Graduate School
Seoul National University

Reverse Engineering(RP) technology and *Rapid Prototyping* technology have brought a great change in the field of design and production industries. For the *Reverse Engineering*, it is necessary to remove the uncertainty lies in the 3D-scanned point clouds data. In general, the conversion into a surface model is adopted to represent the scanned data. In the same manner, in order to prototype point clouds, the *Surface Reconstruction* process must be performed first and then the *Slicing* operation follows. However, this conventional workflow is inefficient because it requires too much manually operated computations in the preprocessing step. Even after the conversion, additional *Mesh Healing* operations must be conducted to correct non-manifoldness in the converted surface model.

Thus, in this paper, we propose a fast slicing algorithm to calculate the layer-by-layer cross-sections of point clouds for RP process. The proposed algorithm adopts a 2.5-dimensional *Ray-Representation(R-Rep.)* model to

slice the object, which stores the information of intersections between rays and the model. First, we define the *Moving Least Square (MLS)* Surface from the points and search for *k-Nearest Neighbors* of each ray to project them on the ray. The projected points become *Query Points* with high probabilities that an intersection exists around them. Then, the MLS value computed at each query is used to the *Regula-Falsi* method to find the intersections between two consecutive queries with opposite signs.

In the proposed algorithm, the input data is directly converted into the R-Rep. model without surface reconstruction process or mesh restoration process. In addition, the errors in the model geometry can be quickly detected and corrected by examining the *Non-Zero Winding Rule* from each ray. Finally, we can directly obtain black/white cross-sectional images from determining the inside/outside of the model using the *Even-Odd Rule* from all rays.

As a result, it shows that the point clouds can be efficiently prototyped by converting them directly into the R-Rep. model, instead of the surface model.

Keyword : Reverse Engineering, 3D-Scanning, Point Clouds, Rapid Prototyping, R-Rep. Model, Moving Least Squares Surface

Student# : 2017-27031

INDEX

LIST OF FIGURES & TABLES

I. INTRODUCTION

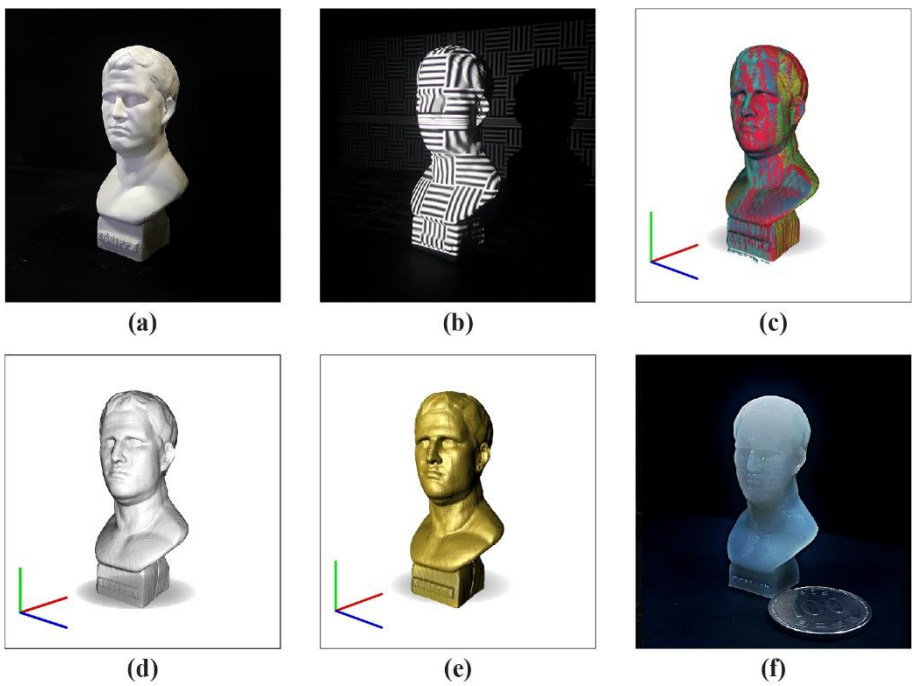


Figure 1. *3D-Scanning process during Reverse Engineering, using DAVID SLS-2 Structured-light 3D Scanner.* **(a)** *A physical gypsum Agrippa model.* **(b)** *Projection of structured light on the model.* **(c)** *Captured partially-scanned data.* **(d)** *The reconstructed surface model from scanning data.* **(e)** *Advanced rendering techniques applied on the surface model.* **(f)** *Rapidly-Prototyped model.*

In the field of design and production engineering, *Reverse engineering* refers to a sequential workflow for 3D-Scanning a physical object and converting it into a digitalized CAD model for complicated 3D modeling or engineering analysis **(Figure 1)**. This enables designers and engineers to solve real-world

problems aided by the explosive computational power of a computer.

The raw data obtained from the 3D-scanning process is a *Point Cloud*, which refers to a set of points scattered in the three-dimensional space. In some scanning devices, normal vector and color information of a point can also be obtained during the capturing process, by using the local surface construction method [1] and a color camera, respectively. Else, there have been several studies to define normal vector data of point clouds manually [2], [3]. The point clouds contain a large number of uncertainties within them, such as noisy points scanned from background objects or missing areas obscured by the model part. Thus, in order to conduct various computer-aided operations on the scanned data, it is necessary to refine them and obtain a definite model [4]. Conventionally, the conversion into a *Boundary Representation(B-Rep.)* surface model is commonly adopted to describe the point clouds data [5]. *Surface Reconstruction* refers to the conversion process of point clouds into a surface model by creating facets from points. However, the converted surface model usually contains non-manifold geometries, such as self-intersections or *Holes*, which cause errors during the computational process. *Mesh Healing* refers to the process of calculating a water-tight model by detecting and correcting the defective geometries contained in a surface model.

Rapid Prototyping(RP), also called as *3D Printing* technology is the process of building a 3D sculpture by printing and stacking 2D cross-sections

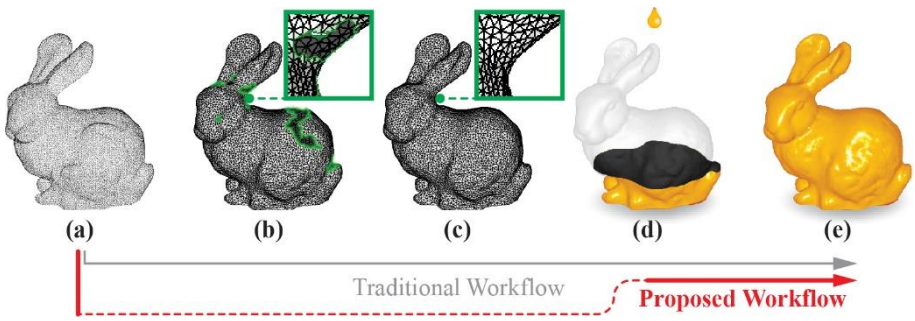


Figure 2. *The comparison between conventional workflow with the proposed workflow.* **(a)** *The raw point clouds* **(b)** *A reconstructed surface model with few non-manifold geometries* **(c)** *A repaired surface model* **(d)** *Sliced images of model from each layer* **(e)** *An Additive-manufactured model.*

of the model. The *Slicing* is the operation for computing the cross-sections of a 3D CAD model at specific layer height during the RP process. Due to the compatibility of a surface model, slicing algorithms are generally developed for surface models. Therefore, in the traditional workflow of the point clouds RP process, the *Surface Reconstruction* must precede before the *Slicing* operation. This leads to an increase of overall work time, and sometimes even the algorithm does not guarantee the successful result.

In order to improve such limitations of the traditional workflow, we introduce a fast slicing algorithm for point clouds as shown in **(**

Figure 2**)**. The proposed algorithm converts normal-estimated point clouds into the 2.5-dimensional ray-rep. model, which is a much proper format for the RP process. Then we slice the model to export a series of 2-BIT B/W images. These set of sliced images are sent to a high-resolution photo-curing

3D printer to build a three-dimensional sculpture by stacking photo-curing resins hardened with light sources. The *Support* structure for the RP process is automatically generated by our own algorithm using the sequential slice images. We implement the proposed algorithm with *Visual C++* IDE and use *OpenMP* parallel computing to boost up iterative operations.

II. RELATED WORKS

With the advancement of reverse engineering technology and rapid prototyping technology, there have been several studies on integrating two technologies by directly slicing point clouds.

The Tangent Plane(TP) approach to directly slice scanned point clouds data was proposed by P. Pal [6]. It uses the concept of a *TP* to approximate locally-defined tangential planes from point clouds. However, this slicing algorithm is not suitable for additive manufacturing process because it focused on the application of creating the toolpath for CNC machining.

The Subdivision Error-based Segmentation method for point clouds slicing algorithm was proposed by G. H. Liu et al. [7], [8]. It constructs a *Curve Model* by segmenting and subdividing the point clouds until the subdivision error goes below the predefined tolerance. Then the cross-sectional contours of point clouds in each layer can be computed from the constructed model.

The Moving Least Squares(MLS) method has been widely adopted in existing studies for point clouds processing, which was firstly introduced by D. Levin [9]. The MLS method is quite robust to noisy data in 3D-scanned point clouds since it always defines a closed signed distance field from a set

of points. P. Yang et al. [10] used the MLS method to improve conventional projection-based points slicing algorithm. It adaptively defines the range of each layer for computing contours within proper curvature from point clouds. T. Yuan et al. [11] also used the MLS method to calculate a set of polygon-approximated cross-sections in each layer thickness. It solves the energy function to find an MLS surface point, which becomes a corner of the polygon, along with locally defined *Moving Direction*. P. Yang et al. [12] and Y. Chen et al. [13] proposed algorithms to find critical points in the model by applying the *Morse-Smale Complex* to MLS method. Then a *Reeb Graph* can be obtained from linking critical points in the model. Finally, the model can be sliced using the constructed graph structure. The algorithm proposed by Y. Chen et al. also creates proper anchor structures using the contour-based support generation method.

Previous methods for direct slicing & fabricating 3D-scanned point clouds are focused on generating a series of contour-based slices by minimizing computational works from other steps in the workflow. It is because most of the existing studies have concerned about the most popular method of 3D-printing, called *Fused Deposition Method(FDM)*. Due to the development of optical devices, a *Stereolithographic(SLA)/ Digital Light Processing(DLP)* type 3D-printer has become increasingly popular in recent years. In general, a SLA/DLP type printer builds a model faster than an FDM type printer, because it does not require complex tool paths data of nozzle to print each sliced cross-

section. Instead, it usually adopts a 2-bit black/white cross-sectional image to distinguish between curing and non-curing areas. Thus, in contrast to conventional studies, the algorithm proposed in this paper exports sequential monochromatic images as results of the slicing operation, which is more suitable for the technology that meets the trend.

III. METHODOLOGY

–

III-1. MODEL REPRESENTATION

–

III-1. a | 2.5-Dimensional R-Rep. Model

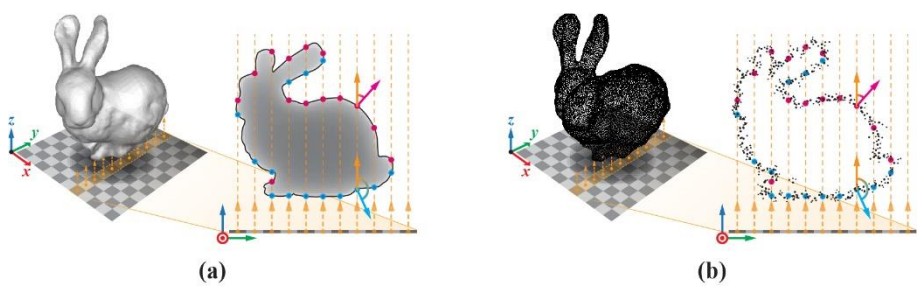


Figure 3. *The concept of a Ray-Representation model.* **(a)** *The R-Rep. conversion from a surface model* **(b)** *The conversion from point clouds.*

The *Ray-Representation*, or the *R-Rep.* refers to a solid model representation method that simplifies the model geometry by using only information about intersections between rays and a model [14] as shown in **(Figure 3)**. In the proposed algorithm, we use an uniformly distributed discrete 2D grid casting rays on the model, while the ray-model intersections are stored with values along a continuous 1D-ray coordinate system. This is often called a *2.5-Dimensional Representation* due to its method for describing the three-

dimensional model with mixed lower-dimensional information.

In the R-Rep. model, non-manifold geometries of the model can be quickly detected by a simple inspection using ray properties. For a water-tight manifold model, there must be pairs of intersections, that is, entering and exiting the model, resulting in an even number of total intersections stored in a ray. This simple intersections-counting rule is called the *Even-Odd Rule*. Furthermore, the normal vector information can be additionally used to estimation for more complex, which is called *the Non-Zero Winding Rule*.

As a *Layer-based Manufacturing* process, which accumulates 2D cross-sectional layer images to construct a 3D sculpture, adoption of the 2.5D R-Rep. model for the RP process guarantees several advantages. P. Huang et al. introduced 2D image-based model processing techniques for the layered manufacturing [15], using a semi-implicit ray-representation model named *LDNI(Layered Depth-Normal Images)*, which is proposed by C. C. L. Wang and Y. Chen et al. [16][17]. We adopt a similar ray-based model representation inspired by the LDNI. The adopted model stores a ray intersection using a pair of integer indices$(i,j) \in \mathbb{Z}^2$, $\{\ i,j \mid i = 0,\cdots,cols,\ j = 0,\cdots,rows\ \}$ from discrete 2D-grid coordinates and the depth value $z = \{\ z_i \in \mathbb{R} \mid i = 1,\cdots,n_{inter}\ \}$ from the 1D-ray coordinate. Each intersection also contains the approximated normal vector information of the MLS surface. However, unlike LDNI, which defines ray grids in 3 orthogonal directions from the 3D Cartesian coordinate, the introduced algorithm only construct one ray grid

oriented the manufacturing direction.

In the introduced R-Rep. model, two coordinates perpendicular to the prototyping direction are replaced with a two-dimensional discrete grid, $\boldsymbol{G} \in \mathbb{Z}^{\boldsymbol{2}}$ for purpose of simplification. In other words, the resolution of the $\boldsymbol{G}$ is the most important factor in the algorithm that determines the approximation accuracy of the model compared to the original data. If the grid resolution is too low, the memories are used efficiently while the original model is not well-represented. If the grid resolution is too high, the original data is finely approximated by the model but running times and memories for algorithm increase dramatically.

If the resolution of the ray grid is specified, then computation for the grid width can be skipped. Nevertheless, we introduce a simple method for calculating the appropriate grid width criterion for the input point clouds data in the following chapter.

III-2. PREPROCESSING

III-2. a | Multi-dimensionally Linked k-d Tree Construction

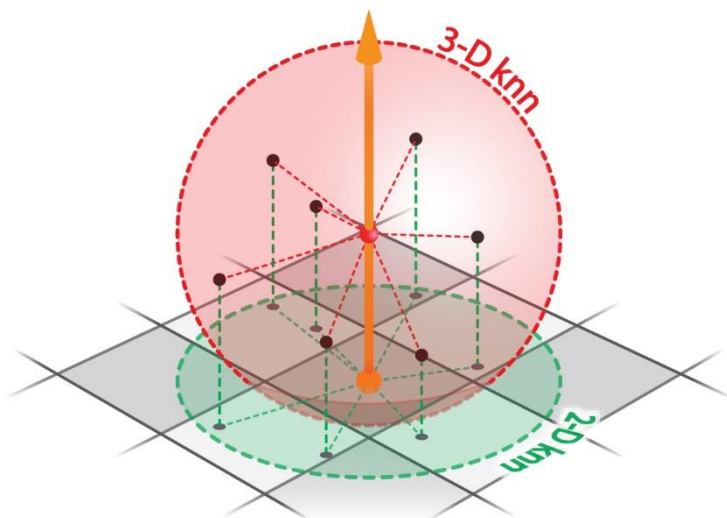


Figure 4. *The nearest neighbors in k-D spatial tree structure with cross-dimensional linkage between 2D and 3D.*

A *k-d Tree* refers to the binary-partitioned tree structure in the k-dimensional space. By using the k-d Tree, several quick query search methods are available for a set of points data in the specific k-dimensional space [19], e.g. *kNN(k-Nearest Neighbors) or ANN(Approximate Nearest Neighbors)*. However, since the R-Rep. model for RP process shows the 2.5-dimensional nature,

sometimes it is convenient to conduct the searching operations in two different dimensions, respectively. To do this, we simultaneously construct a 2-D tree for projected points on grid plane while a 3-D tree for original point clouds is constructed in the three-dimensional space**(Figure 4)**. The constructed multidimensional tree simply links each point stored in a 2-D tree with the original 3D data by managing data using the index of each point. The structure has the maximum size of 10 leaf-nodes in both 2-D and 3-D tree, implemented by a C++ library for *nanoflann(nano-Fast Library for Approximate Nearest Neighbors)*.

III-2. b | Density & Orientation Based Points Clustering

A point cloud data is not randomly distributed in the three-dimensional space but tends to be densely positioned around features near the model surface. With such a characteristic, the 3D-scanned point cloud data is suitable for a density-based spatial clustering algorithm named *DBSCAN(Density-Based Clustering of Applications with Noise)* proposed by M. Ester et al. [20].

The density-based clustering algorithm for a point cloud $\boldsymbol{P} = \{ \boldsymbol{p}_i \in \mathbb{R}^3 \mid i = 1, \cdots, n_p \}$ generally needs 2 user-input variables: $\varepsilon \in \mathbb{R}$, the radius for neighborhood, and $minPts \in \mathbb{Z}$, the minimum number of neighbors within ε of $\boldsymbol{p}_i$, $\boldsymbol{Q}_{\boldsymbol{\varepsilon},\boldsymbol{i}} = \{ \boldsymbol{q}_{ij} \in \mathbb{R}^3 \mid i = 1, \cdots, n_p, j = 1, \cdots, n_q \}$ for to be set as a core point of a cluster. We reduce the number

of variables used in the proposed algorithm by computing appropriate $\boldsymbol{\varepsilon}$ for a given ***minPts*** by taking the median value from the set of distances from a point $\boldsymbol{k_1}$ *th nearest neighbor* $D_{k_1}^3 = \{ d_{i,k_1} \in \mathbb{R} \mid i = 1, \cdots, n_p, \ \boldsymbol{p}_i \in \mathbb{R}^3 \}$, computed in the original three-dimensional space for entire members in the given point cloud P as following :

$$\varepsilon = d_{\text{median}} = \frac{d_{\lfloor 0.5 n_p \rfloor, 2minPts} + d_{\lfloor 0.5(n_p - 1) \rfloor, 2minPts}}{2}$$

Here, n_p is the number of points in the point clouds, and d_{i,k_1} refers to the three-dimensional Euclidean distance to 3D k_1^{th}NN of $\boldsymbol{p}_i$. In the proposed algorithm, we use $\boldsymbol{minPts = 30}$ for k_1. We take the twice value of the number of minimum points, 2*minPts* for *k* and use the median of $D_{k_1}^3$ to set $\boldsymbol{\varepsilon}$ rather than the average value, since it is the much more robust method when database contains lots of outliers. Our idea is to filter noisy but densely-positioned points, by setting a higher number of *minPts* as k_1 to compute median of $D_{k_1}^3$. That is, since there are sufficient neighbors around a non-noise point, finding the epsilon for a bigger value of k may not make a big difference. Based on the original algorithm, we additionally use the information of normal vectors $\boldsymbol{N} = \{ \vec{\boldsymbol{n}}_i \in \mathbb{R}^3 \mid i = 1, \cdots, n_p \}$ stored in each point. In the conventional density-based clustering algorithm, the distance between two points is defined as the simple three-dimensional Euclidean distance between two points.

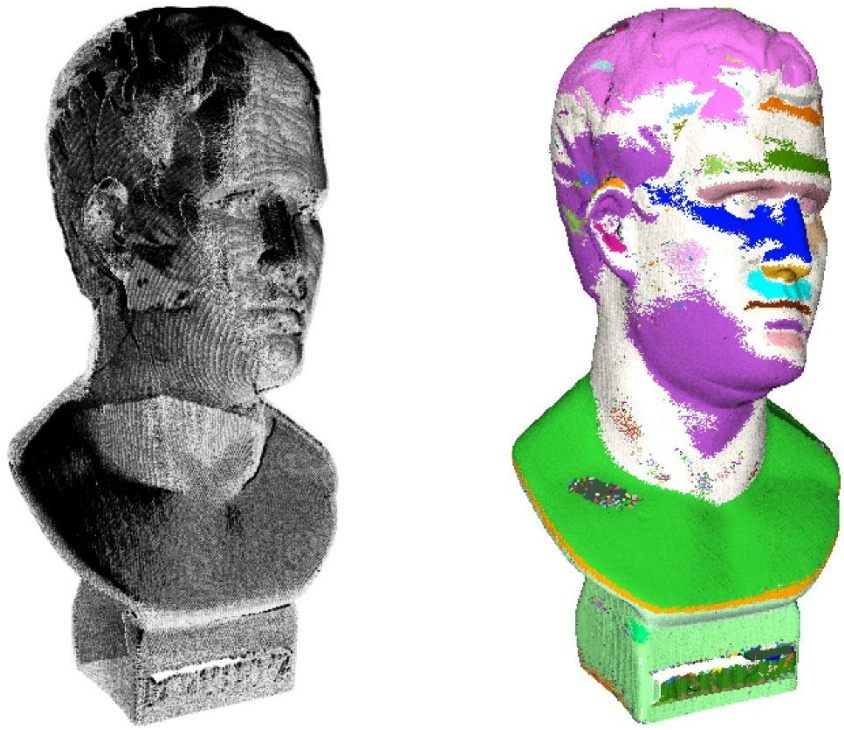
$$\boldsymbol{dist}_{conv}(\boldsymbol{q_{ij}}) = \left\|\boldsymbol{p}_i - \boldsymbol{p}_j\right\|$$

However, we use an additional fourth-dimensional information to define a new distance metric between a point member $\boldsymbol{p}_i$ and its neighbor $\boldsymbol{q}_{ij}$, by multiplying the inverse of inner product value of the two normal vectors, $\langle \vec{\boldsymbol{n}}_i, \vec{\boldsymbol{n}}_j \rangle$, which refers to the similarity between the two normal vectors.

$$\boldsymbol{dist}_{new}(\boldsymbol{q_{ij}}) = \frac{\left\|\boldsymbol{p}_i - \boldsymbol{p}_j\right\|}{\langle \vec{\boldsymbol{n}}_i, \vec{\boldsymbol{n}}_j \rangle}$$

Thus, it is possible to filter not only a noisy point that are scanned from the background object spatially far from the target model, but also a point with the normal vector whose value erroneously changes around the edge of each partially-scanned data.

In the 2.5D R-Rep. model, there are two types of intersections: an intersection that is ***entering*** and one that is ***exiting*** the model. The two different types of intersections are defined respectively from the adjacent points with normal vectors opposite or parallel to the *manufacturing direction* $\overrightarrow{\boldsymbol{m}}$. We use this property to confine the type of intersection expected from each cluster, by separating the clustering process into two groups by its orientation O, the positive or negative sign of inner product value between the normal vector and the manufacturing direction vector, $\langle \vec{\boldsymbol{n}}_{\boldsymbol{i}}, \overrightarrow{\boldsymbol{m}} \rangle$. Considering the estimation error of the normal vector obtained from the 3D-scanning process, we cluster the point $\boldsymbol{p}_{\boldsymbol{i}}$ whose normal vector is almost perpendicular to the



(a) (b)

Figure 5. *The points clouds clustered based on the Density & Orientation-based Clustering algorithm.* **(a)** *Points in the raw point clouds data.* **(b)** *The clustered point cloud based on the algorithm. The random color is assigned for each cluster.*

manufacturing direction, that is $\langle \vec{n}_i, \vec{m} \rangle \approx \mathbf{0}$, in a different group. We define a point is ***neutral***, when the angle between $\vec{n}_i$ and $\vec{m}$ φ satisfies $|\varphi - 90°| < 5°$. As a result, all the clustered points in the point clouds can be classified into three groups during the clustering process: ***entering***, ***exiting*** and ***neutral***.

For every clustered point member, it is labeled as the *Core Point* of the cluster, if the number of points within the ε-neighbor, regardless of the type,

is bigger than *minPts* while there are more points within the ε-neighbor than 75% of same type. Else, if a point contains the fewer number of points than *minPts* within its ε-neighbor, it is simply labeled as a ***noise***. This enables the original point clouds data to be used more efficiently during the ray intersections calculation, by eliminating noises and subdividing the point clouds. After the clustering process, every member in the point clouds belongs to one of the following four groups: ***entering*** cluster, ***exiting*** cluster, ***neutral*** cluster or ***noises*** **(Figure 5)**. Here we call clusters of the type *entering* or *exiting* as the ***practical*** clusters. The following table shows how the original point cloud data is segmented after the clustering algorithm.

<table>
<tr><th colspan="4">Original Point Clouds</th></tr>
<tr><th colspan="3">Clustered Points</th><td rowspan="3">noises</td></tr>
<tr><td colspan="2">Practical Clusters</td><td rowspan="2">neutral</td></tr>
<tr><td>entering</td><td>exiting</td></tr>
</table>

Table 1. *Segmentation result of the input point clouds data after clustering process*

III-2. c | Ray Grid Determination

In the proposed algorithm, we determine the proper resolution for the ray grid so that a reasonable number of ***practical*** points are located near each ray. Thus, we project only the points in ***practical*** clusters, that is, ***entering*** clusters $\boldsymbol{p}_i \in$

$\mathbb{C}_{enter}$ and points in the ***exiting*** clusters $\boldsymbol{p}_i \in \mathbb{C}_{exit}$ onto the grid plane, except for ***noises*** and the ***neutral*** type points. For the projected points $\boldsymbol{P'} = \{\ \boldsymbol{p}'_i \in \mathbb{R}^2 \mid i = 1, \cdots, n_{enter}, \cdots, (n_{enter} + n_{exit}),\ \}$, a set of the two-dimensional $\boldsymbol{k_2}$*th nearest neighbors* distances for the *entering* or *exiting* type of clusters, $D^2_{k_2,type} = \left\{\ d'_{i,k_2,type} \in \mathbb{R} \mid i = 1, \cdots, n_{\mathbb{C}_{type}},\ \boldsymbol{p}'_i \in \mathbb{R}^2 \right\}$ are computed in each cluster type. We use $\mathrm{k}_2 = 10$ in the proposed algorithm. In the same manner as in the calculation of ε, the grid width $w \in \mathbb{R}$ can be simply determined by using an average value here. This is because the *neutral* clusters and outliers –too close or too far apart when projected onto a plane– are excluded in $D^2_{k_2,type}$ so the average value gives a fine representation for the distance from each point to its 2D k_2^{th}NN. Since the average value is computed from the *kNN Search*, which has the radial search area, we conduct the simple arithmetical correction to guarantee an equivalent geometric probability of containing *k* points within square grid cell with width w.

$$w^2 = \pi * {d'_{average}}^2$$

$$w = \sqrt{\pi} * d'_{average} = \sqrt{\pi} * \frac{\sum_i^{n_{\mathbb{C}_{type}}} d'_{i,k}}{n_{\mathbb{C}_{type}}}$$

Also, for the case that some partially scanned data in specific directions are sparse, we take a smaller average value to preserve features for gird width w, as following:

$$w = \sqrt{\pi} * \min\left(\frac{\sum_i^{n_{\mathbb{C}_{enter}}} d'_{i,k,enter}}{n_{\mathbb{C}_{enter}}}, \frac{\sum_i^{n_{\mathbb{C}_{exit}}} d'_{i,k,exit}}{n_{\mathbb{C}_{exit}}}\right)$$

III-3. MOVING LEAST SQUARES SURFACE

III-3. a | MLS Query Candidates

Since the Moving Least Squares method discriminates the interior and the exterior of a model to define a surface, *Query Points* are required for computing values of the MLS implicit function. Our idea is that, based on the discrete connectivity of the R-Rep. model, the intersections of each ray is likely to be positioned near the nearest neighbors of the ray. We collect 2D kNN of each ray in projected points belong to each *practical cluster* to obtain *good* query points, which have a high possibility to find an intersection nearby them. Then, we move to a 3D point linked from each collected 2D k-NN and project it onto the ray. This projected point become a *Query Candidate* for a query point **(Figure 6)**.

However, calculating the MLS values for all candidates takes a long time and is inefficient, so we cut the number of query points by merging points within the distance δ. In the proposed algorithm, we take grid width w for δ. We firstly sort the candidates by their depth values in the one-dimensional ray coordinate system. Then, the candidate with the lowest depth value in each ray is converted into a query point immediately. From the second candidate,

we check whether the candidate is within the δ-distance from the previously defined query point. In this way, we merge all candidates in the δ-distance from the last query point into a new query point away from the previous query point by distance δ. This *merging* operation continues until there is no candidate within the δ distance from the last query point, and if there is a point that has not yet been examined, it becomes the new query point. Otherwise, the algorithm is terminated.

Through this process, the number of query candidates is reduced and the duplicate calculation of MLS values for the densely-located query candidates can be skipped.

III-3. b | Ray-MLS Surface Intersections Solving

For sorted query points stored in a ray, we compute the MLS values from all query points to find pairs of two consecutive queries with different signs. We use an improved version of the *Regula-Falsi* approach, named *Illinois* method [21], to numerically find a root between the two queries to define an intersection using the numerically found depth value $z \in \mathbb{R}$ and pairs of ray indices $(i, j) \in \mathbb{Z}^2$ information. We set value for maximum iteration steps $\boldsymbol{iter_{max} = 10}$ of *Regula-Falsi* method, and the one-tenth grid width for the threshold of the interval between the two points, that is $\boldsymbol{\gamma = \frac{1}{10} w}$.

.

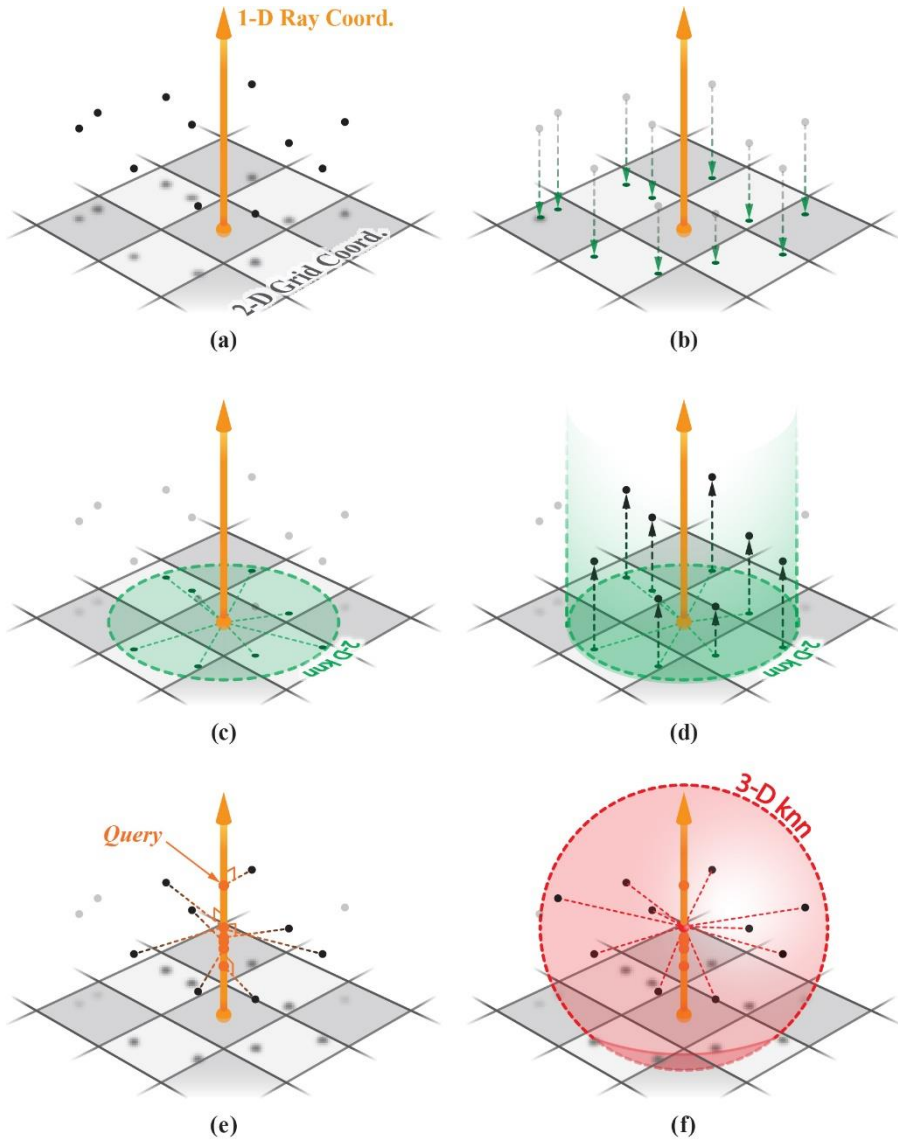


Figure 6. *Query points determination process.* **(a)** *The raw point clouds data in 3D space* **(b)** *Points projected on 2D-Ray grid plane* **(c)** *Radius Search algorithm in 2-D tree, using the ray origin as a searching query* **(d)** *Radius neighbors in 2D plane linked(unprojected) with its original 3D point clouds* **(e)** *MLS Query Points projected on 1D-Ray Coordinate.* **(f)** *k-Nearest Neighbors Search algorithm in 3-D tree using the MLS query as a searching query.*

III-4. SLICING

III-4. a | Ray Perfectness Estimation

In order to compute a slice image in the *n*-th layer in a R-Rep. model, the pixels are painted in white when the number of ray intersections below the layer is odd, which indicates interior of the model, and the pixels are painted in black when the number of intersections is even, which indicates exterior of the model, based on the *Even-Odd Rule*. By determining pixel colors from all rays at the layer height, we can write a slice image with size of the ray grid layer-by-layer. However, there is a precondition for using this rule. That is, each ray must have its intersecting points in sequential order for pairs of an *entering* intersection followed by an *exiting* one, and therefore the number of total intersections of a ray must always be an even number.

The *Nonzero-Winding Rule* is the generalization of this property. In this rule, the ray value is incremented when the ray passes through the *entering* intersection, and the value is decremented when it passes the *exiting* intersection. As a result, the value of the ray must be the zero after passing the model, or it can be inferred that there may be missing intersections in the ray. Using this rule, we define a ray that satisfies the rule as a *perfect* ray.

We confine the range of rays for the *Perfectness Estimation*. In order to define such an i*nterested* ray, we search for the nearest radial neighbors within radius of $\sqrt{2}w$ for *clustered points*, except for *noises*. If there is at least one radial neighbor for a ray, the ray is marked as white in a two-dimensional image, which means the ray passes near the model. This kind of ray is named a *passing* ray. For this two-dimensional image, you can define *interested* rays that are likely to pass the model by performing series of erosion and dilatation operations and finding and closing their contours. Our final goal is to turn all *interested* rays into *perfect* rays. Here, we add new query points that are uniformly distributed over the entire range of a ray for the *imperfect* rays and calculate the MLS value at the point to find the missing ray.

III-4. b | Export Slice Images

Even after correction, noisy pixels may still remain in the sliced layer image. However, since the noise at this step is not clumped and does not have a specific shape, very simple two-dimensional image processing can eliminate noise. By using a simple method similar to defining the ray of interest, the slice image obtained from each layer goes under some sequential dilation and erosion process to denoise data, called *Opening* or *Closing* process. Finally, the result of the slicing operation is exported as sequential 2-bit B & W PNG format image files.

IV. RESULTS & DISCUSSION

Results

The sequential sliced images were obtained from a series of operations manually conducted by the users, to estimate the performance of the proposed algorithm. We converted the point clouds into the reconstructed surface model with *Poisson Surface Reconstruction* method then we filled the holes in the converted model to get a water-tight surface model. Finally, we conducted the *Ray Casting* operation on the all facets in the reconstructed surface model to generate 2.5D R-Rep. volumetric model. We use three gypsum models for 3D scanning process in the proposed algorithm: *Agrippa* **(Figure 7)**, *Niobe* **(Figure 8)**, and *Venus* **(Figure 9)**.

Limitation and Future Works

However, in the proposed algorithm the images from some layers show erroneous topology within it, due to misaligned normal vectors from some partially scanned local point clouds. Thus, the algorithm needs a more robust method to classify not only the spatially noisy points, but also points with mal-oriented normal vectors within it.

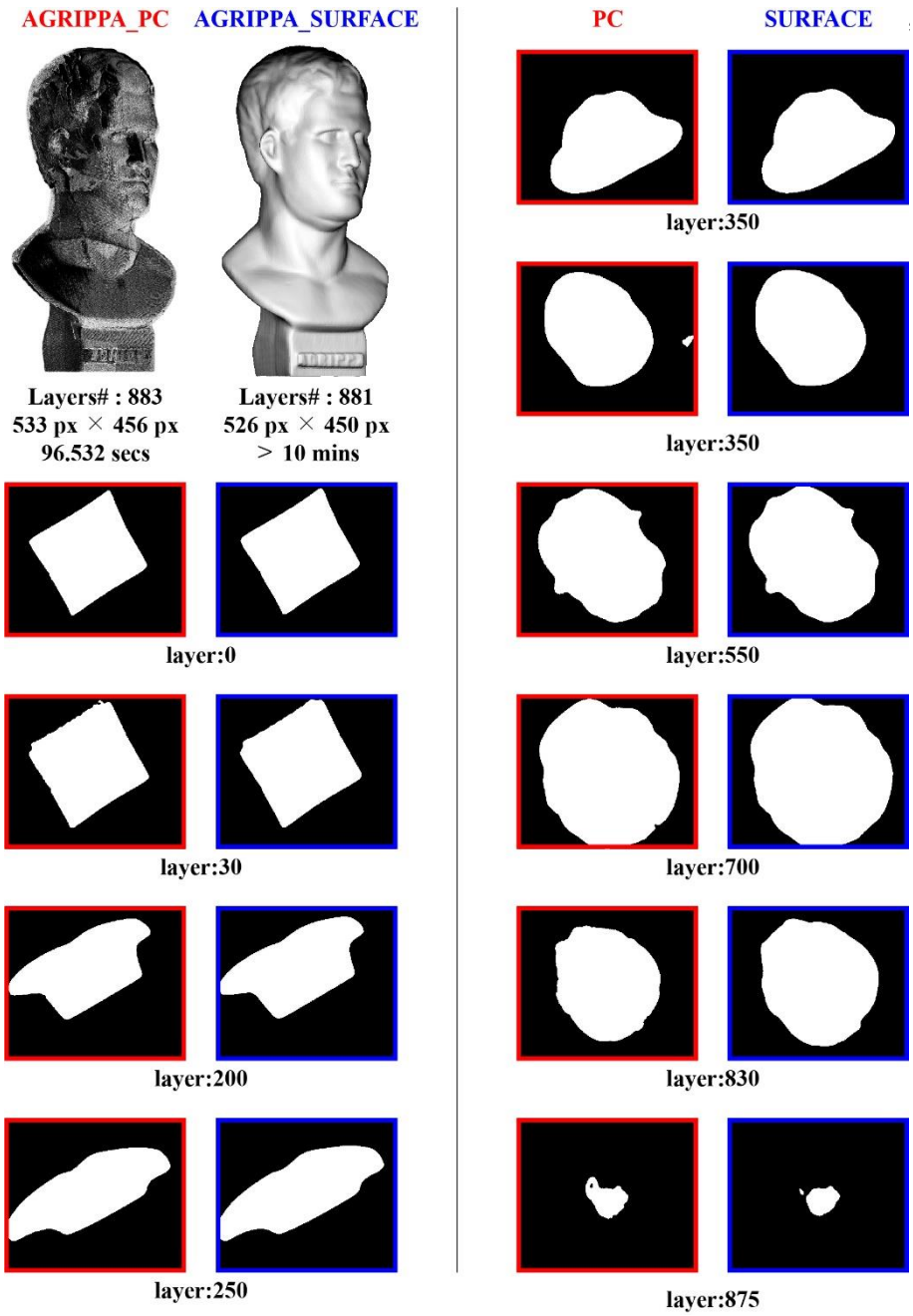


Figure 7. *The layer-by-layer comparison of sliced images of the "Agrippa" gypsum model between directly sliced from Point Cloud(Left) and sliced after Poisson reconstructed and hole-closed Surface model(Right).*

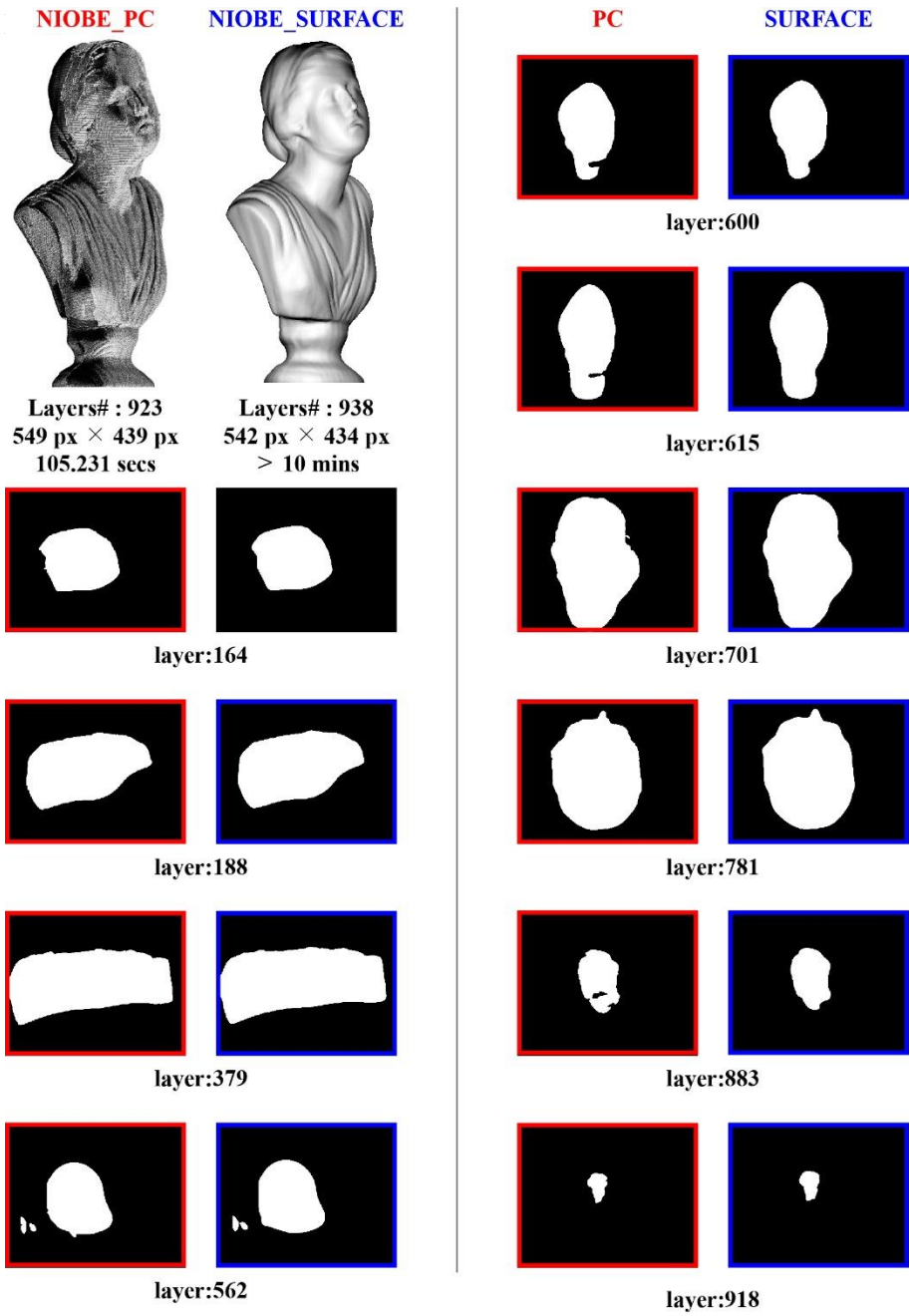


Figure 8. *The layer-by-layer comparison of sliced images of the "Niobe" gypsum model between directly sliced from Point Cloud(Left) and sliced after Poisson reconstructed and hole-closed Surface model(Right).*

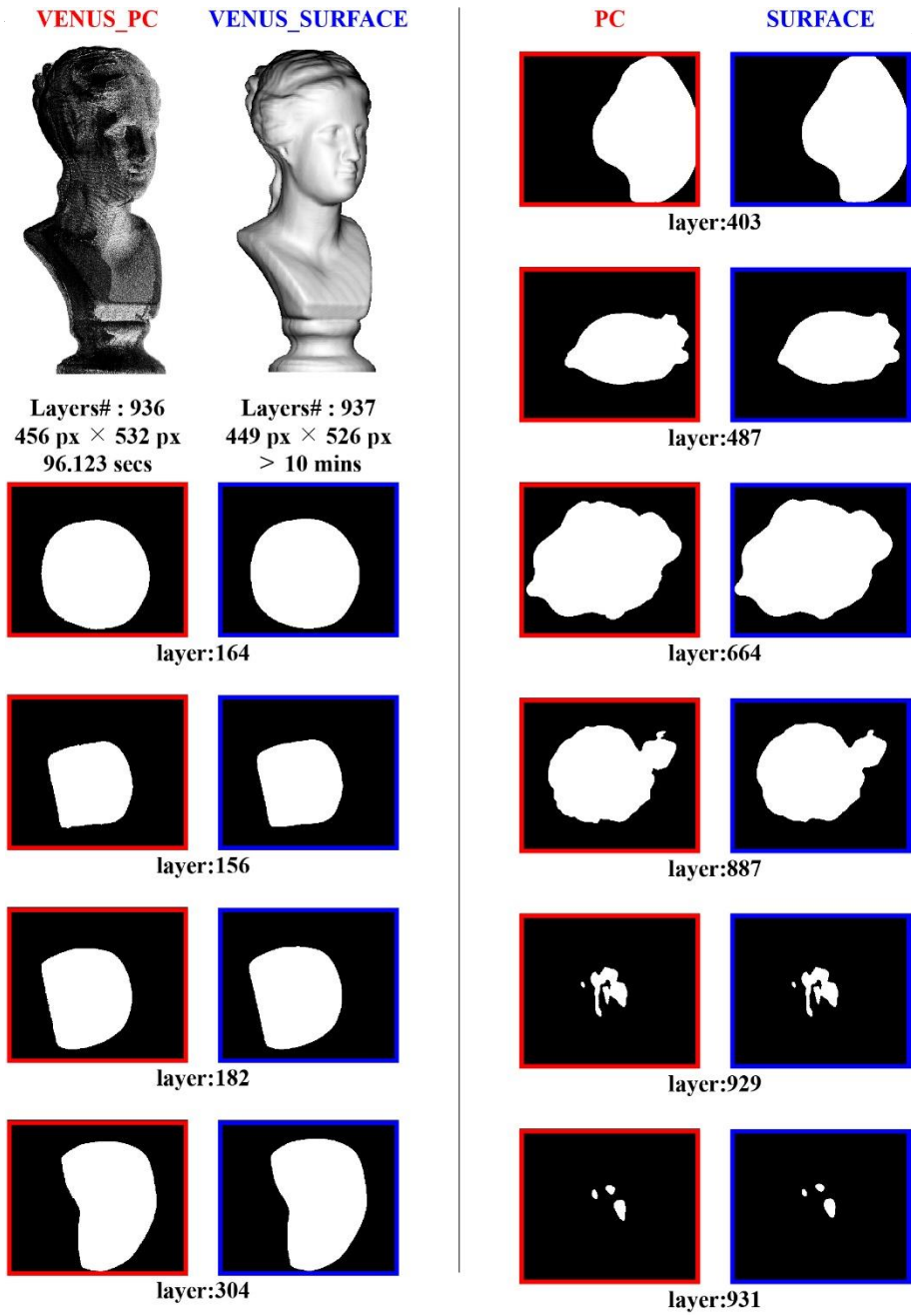


Figure 9. *The layer-by-layer comparison of sliced images of the "Venus" gypsum model between directly sliced from Point Cloud(Left) and sliced after Poisson reconstructed and hole-closed Surface model(Right).*

V. CONCLUSION

In this paper, we calculated a series of slice images from the point clouds for RP process, without going through the conversion into a surface model. Instead, we adopt a 2.5-dimensional R-Rep. model with intersections defined from the MLS surface based on the discrete connectivity of the ray grid. In particular, we cluster the original point cloud data using the information of density and normal vector. Thus, a series of following operations for computing the ray intersections become much efficient. Then we use the general property of *Nonzero Winding Rule* to detect missing intersections and export a series of black and white images by discriminating the interior and the exterior of the model *by the Even-Odd Rule*.

The algorithm directly generates sliced images of a model by computing the ray-MLS surface intersections from the raw point clouds. The average time elapsed on slicing operations for several gypsum models, including writing and exporting the images, is less than 2minutes. The resultant sequential images obtained from the proposed algorithm show quite similar topology of the given model compared to the reconstructed surface model.

REFERENCES

[1] N. J. Mitra, A. Nguyen and L. Guibas, “Estimating surface normals in noisy point cloud data”, International Journal of Computational Geometry and Applications(2004), Vol.14, No.4-5, Pages 261-276

[2] D. O. Yang and H-Y. Feng, “On the normal vector estimation for point cloud data from smooth surfaces”, Computer-Aided Design(2005), Vol.37, Pages 1071-1079

[3] K. Klasing, D. Althoff, D. Wollherr and M. Buss, “Comparison of Surface Normal Estimation Methods for Range Sensing Applications”, IEEE International Conference on Robotics and Automation(2009), Kobe, Japan, May 12-17, 2009

[4] M. Pauly, M. Gross and L. P. Kobbelt, “Efficient Simplification of Point-Sampled Surfaces”, IEEE Visualization(2002), Boston, MA, USA, 27 Oct.-1 Nov., 2002

[5] R. Fabio, "From point cloud to surface: the modeling and visualization problem", International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences(2003), XXXIV-5/W10

[6] P. Pal, “An easy rapid prototyping technique with point cloud data”, Rapid Prototyping Journal(2001), Vol.7, No.2, Pages 82-90

[7] G.H. Liu, Y.S. Wong, Y.F. Zhang and H.T. Loh, “Error-based segmentation of cloud data for direct rapid prototyping”, Computer-Aided Design(2002), Vol.35, Pages 633–645

[8] G.H. Liu, Y.S. Wong, Y.F. Zhang and H.T. Loh, “Modelling cloud data for prototype manufacturing”, Journal of Materials Processing Technology(2003), Vol.138, Pages 53–57

[9] D. Levin, “The approximation power of Moving Least-Squares”, Mathematics of Computation(1998), Vol.67, No.224, Pages 1517-1531

[10] P. Yang and Xiao. Qian, “Adaptive Slicing of Moving Least Squares Surfaces : Toward Direct Manufacturing of Point Set Surfaces”, Journal of Computing and Information Science in Engineering(2008), Vol.8, No.3

[11] T. Yuan, Xiao. Peng and D. Zhang, “Direct rapid prototyping from point cloud data without surface reconstruction”, Computer-Aided Design and Applications(2018), Vol.15, No.3, Pages 390-398

[12] P. Yang, K. Li and Xiao. Qian, “Topologically Enhanced Slicing of MLS Surfaces”, Journal of Computing and Information Science in Engineering(2011), Vol.11, No.3

[13] Y. Chen, K. Li and Xiao. Qian, “Direct geometry processing for telefabrication”, Journal of Computing and Information Science in Engineering(2013), Vol. 13, No.4

[14] J. Menon, R. J. Marisa and J. Zagajac, “More Powerful Solid Modeling through Ray Representation”, IEEE Computer Graphics and Applications(1994), Vol.14, No.3, Pages 22-35

[15] P. Huang, C. C. L. Wang and Y. Chen, “Algorithms for layered manufacturing in image space”, Advances in Computers and Information in Engineering Research, Vol.1, Chapter.15

[16] Y. Chen, “Layer Depth-Normal Images for complex geometries-part one : Accurate modeling and adaptive sampling”, International Design Engineering Technical Conferences and Computers and Information in Engineering Conference(2008), Brooklyn, New York, USA, August 3–6, 2008

[17] C. C. L. Wang and Y. Chen, “Layered Depth-Normal Images: a sparse implicit representation for solid models”, arXiv:1009.0794(2010)

[18] C. C. L. Wang, Y-S. Leung and Y. Chen, “Solid modeling of polyhedral objects by Layered Depth-Normal Images on the GPU”, Computer-Aided Design(2010), Vol.42, No.6, Pages 535-544

[19] J. L. Bentley, “Multidimensional Binary Search Trees Used for Associative Searching”, Communications of the Association for

Computing Machinery(1975), Vol.18, No.9, Pages 509-517

[20] M. Ester, H-P. Kriegel, J. Sander and Xiao. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”, Knowledge Discovery and Data Mining(1996), Proceedings, Pages 226-231

[21] M. Dowell and P. Jarratt, “A Modified Regula Falsi Method for Computing the Root of an Equation”, BIT Numerical Mathematics(1971), Vol.11, No.2, Pages 168-174

초 록

3차원 스캔 점군 모델의 쾌속 조형을 위한 빠른 슬라이싱 알고리즘

문 병 진

서울대학교 대학원
기계항공공학부 기계공학전공

역공학(Reverse Engineering)과 쾌속 조형(Rapid Prototyping, 3D Printing) 기술은 설계 및 생산 분야의 산업 전반에 큰 변화를 불러왔다. 역공학을 수행하기 위해서는 3차원 스캐닝 작업에서 얻는 점군 데이터에 포함된 불확정성을 제거하고 데이터를 변환하는 작업이 선행되어야 하는데, 이를 위해서 표면 모델로의 변환이 일반적이었다. 마찬가지로, 기존에는 3차원 스캔된 물체의 점군 데이터를 쾌속 조형하기 위해, 단면 정보를 계산하는 슬라이싱(Slicing) 작업에 앞서 점군을 표면 모델로 변환하는 표면 재구성(Surface Reconstruction) 작업이 선행되어야 했다. 하지만 이러한 변환 작업은 선행 과정임에도 불구하고 많은 수동적인 연산을 필요로 하고, 변환된 이후에도 표면 모델의 오류를 교정하는 메쉬 복원(Mesh Healing) 작업을 수반하므로 매우 비효율적이다.

따라서, 본 논문에서는 표면 모델로의 변환을 거치지 않고

점군 데이터를 곧바로 슬라이싱 하는 알고리즘을 제안한다. 제안된 알고리즘은 점군 데이터를 입력 받은 뒤, 광선과 모델의 교차점으로써 형상을 묘사하는 2.5차원 광선 표현(R-Rep.) 모델로 이를 변환한다. 먼저, 점군 데이터로부터 이동 최소 자승(Moving Least Square, MLS) 표면을 정의하고 각 광선의 최근접 k 이웃(k-Nearest Neighbors)을 광선 위에 투사함으로써 교차점을 찾을 확률이 높은 쿼리(Query) 지점을 얻는다. 이후 각 쿼리에서 MLS값을 계산하여 가위치법(Regula-Falsi Method)을 이용해 값의 부호가 서로 다른 연속된 두 쿼리 사이에서 수치해석적으로 교차점을 찾음으로써 광선 표현 모델을 얻는다.

제안된 알고리즘에서는 표면 재구성 및 복원 작업을 거치지 않고 점군 데이터를 광선 표현 모델로 곧바로 변환한다. 또한, 각 광선에서 감기 횟수 규칙(Non-zero Winding Rule)과 홀-짝 규칙(Even-Odd Rule)을 이용해 모델의 내·외부를 판별하여 최종적으로 흑백 이미지 형식의 단면 정보를 출력할 수 있었다.

결과적으로, 기존의 과정에서 표면 모델로의 변환 과정을 생략하고 곧바로 점군 데이터를 광선 표현 모델로 변환함으로써 빠르고 효율적으로 점군 모델을 쾌속 조형할 수 있음을 보였다.

주 요 어 : 역공학, 3차원 스캐닝, 점군 데이터, 쾌속 조형, 3차원 프린팅, 광선 표현 모델, 이동 최소 자승 표면

학 번 : 2017-27031