Title:
Slicing Point Clouds for 3-D Printing

Authors:
William Oropallo, woropall@mail.usf.edu, University of South Florida
Les A. Piegl, lespiegl@mail.usf.edu, University of South Florida
Paul Rosen, prosen@usf.edu, University of South Florida
Khairan Rajab, khairanr@gmail.com, Najran University

Keywords:
3-D printing, NURBS, Point cloud, Object slicing

DOI: 10.14733/cadconfP.2017.5-9

Introduction:

Due to the complexity and the instability of numerical methods, the common practice of slicing has been to decompose the free-form object, designed with NURBS [4], into simple facets (triangles) and slice the faceted (STL) model. This seems like a doable approach, however, decades of experience has shown that this approach, while avoids complex mathematical issues, creates its own numerical problems. To begin with, tessellating a complex NURBS model is a difficult task and the authors have yet to see a robust tessellator. But never mind the tessellator, the fundamental problem flies in the face of CAD companies that work collaboratively, yet their systems are largely incompatible. When the part travels down the design pipeline, it is converted from one data format to the other. By the time it reaches the printer, it suffers from gaps, overlapping surfaces, dangling edges, just to name a few. Put a tessellator to it and slice it, and you have a disaster.

This paper argues that it is time to ditch everything that is complex, error prone and inaccurate, and organize the rest into a simple system that is easy to maintain. In this research we eliminated the STL conversion and all error–prone numerical algorithms, and converted the NURBS object to the simplest entity: a set of points, Figure 1. The slicing is done on the point cloud model employing practically no numerical procedure.

The organization of the paper is as follows. First we introduce some printer notations along with a summary of point cloud generation. Then we process points into layers, separate multiple contours, find boundaries, purge some cells and fit a B-spline curve to get the final section.

Printer setup and point cloud generation:

We take two parameters from the 3-D printer: (1) λ - the layer thickness. We assume this to be a constant throughout the printing process, and (2) ε - accuracy, i.e. the addressability of the printer, the printer head can move from position to position with at least that much distance. No other assumptions are made and no special characteristic of the printer is needed.

Given a collection of B-spline surfaces $S_i(u,v)$, $i=0,\ldots,\mathrm{N}$, covering the complex part (see Figure 1), we want to turn this set of surfaces into a point cloud Q_j, $\mathrm{j}=0,\ldots,\mathrm{M}$, so that for each query point Q_k there is a one-ring neighborhood in which there is at least one point Q_l so that $\left|\mathrm{Q}_k-\mathrm{Q}_l\right|<\delta \quad \delta<\varepsilon$. In other words, we are processing the NURBS model into a quasi-uniform point set so that any circle with a ε radius contains at least one point. This may seem like a simple task, however, to do it

economically without oversampling and to do it efficiently, it requires a sophisticated algorithm whose details are given in [3]. For reasons of space saving we refer the reader to this paper for further details.

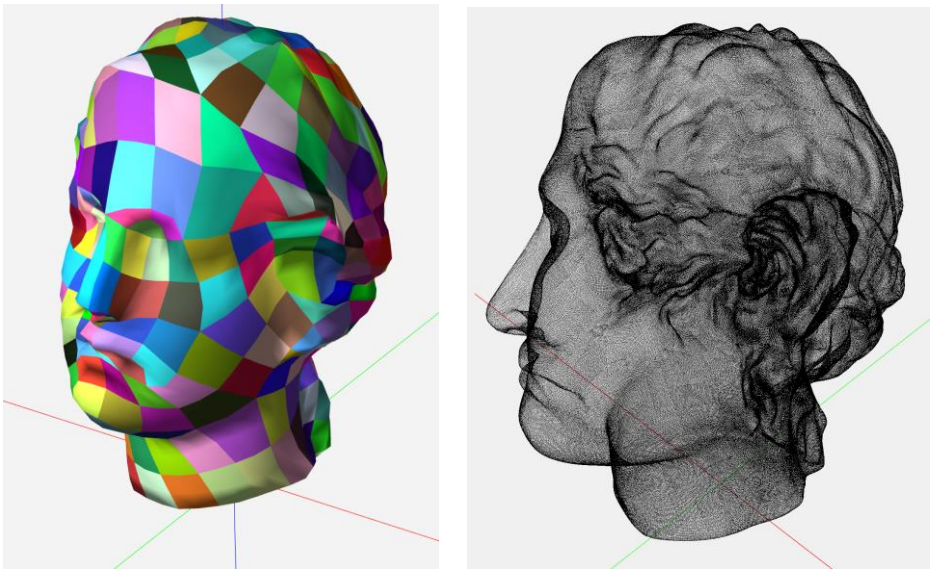

Fig. 1: Head modeled by a set of NURBS surfaces (left), point cloud model (right, model courtesy of Direct Dimensions, Inc.).

Overview of the algorithm:

Given the point cloud, we are ready to intersect it with a slicing plane. First, we lay a grid of size ε on that plane. The vertices of the grid are the addressable locations of the printer head. Second, we add voxels of ε size on each grid cell above and below the plane. Each cell is then marked as WHITE and the slicing plane is considered preprocessed for intersection. The next step is to find all sub-surfaces that intersect the slicing plane. Then for all points on the sub-surfaces we do the following:

- If the point is more than ε distance away from the plane, it is discarded.
- If the point is within ε distance, we find the voxel the point is in and mark the corresponding cell as GRAY.
- For all GRAY cells do:
    - If there are points in the voxels above and below, we mark the cell BLACK.
    - If there are points in the voxel above, we mark the cell RED.
    - If there are points in the voxel below, we mark the cell BLUE.

Figure 2 shows the RED, the BLUE as well as the BLACK cells for layer No. 3 in the head model above. Please note that at this stage only a partial intersection is found in terms of BLACK cells where the surfaces cross the plane. Other intersections occur from one cell to the next, not within the same cell. In order to find these, we need to look for transitions from RED to BLUE or BLUE to RED, Figure 3.

As Figures 3 shows, there are more BLACK cells representing the intersection curve than we need, so we are going to discard some of them to create a one cell thick point set. Since the curve is used in printing, the outer-most cells are retained whereas the inner ones will be eliminated. A simple way to do that is to use a seed fill algorithm to fill the area outside the closed curve [2]. To prepare, first compute the bounding box of all BLACK cells and offset the box by one cell in each direction. Mark all cells in the box WHITE.

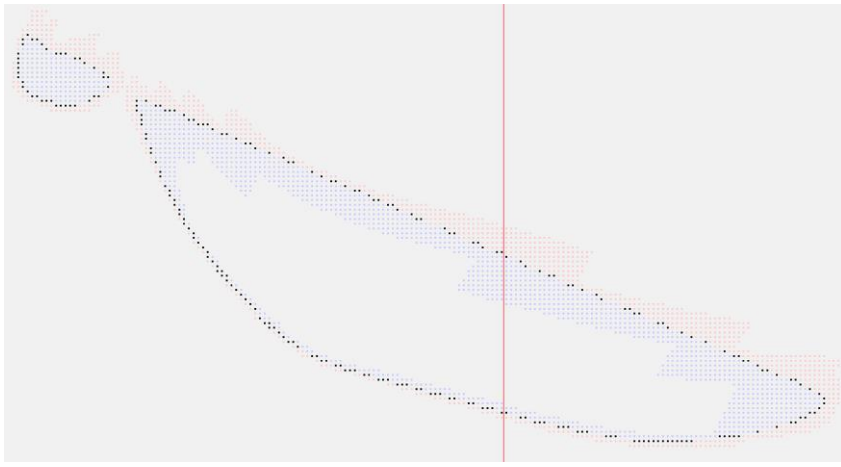

Fig. 2: RED, BLUE and BLACK cells for layer No. 3.

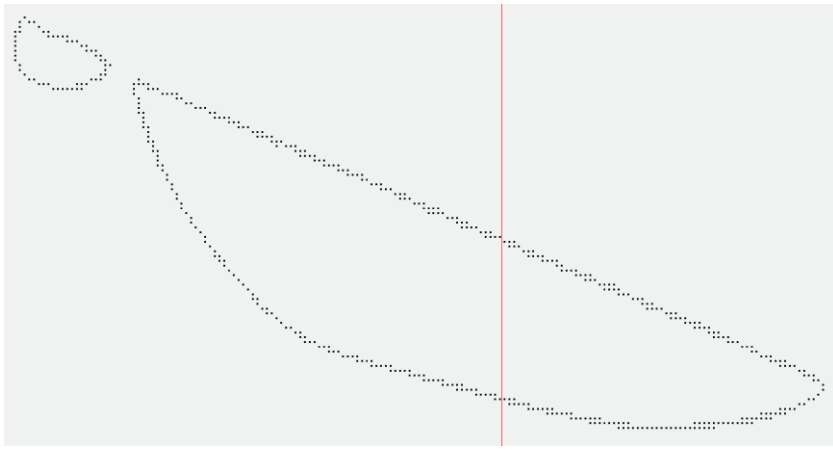

Fig. 3: A superset of BLACK cells covering the intersection of layer No. 3.

The algorithm starts with the lower left cell (the artificially created one that is known to be empty) and uses a 4-connected pattern to flood the area outside the closed curve. The 4-connected pattern has four cells for each cell, up and down and left and right of the cell.

Once the recursion stops, all boundary cells are collected and the rest are eliminated. Figure 4 shows the result of the flood fill algorithm: the red cells are eliminated whereas the black ones are kept. The result is a one cell thin discrete set of curves that will now be turned into a smooth B-spline curve. To do that, the points need to be ordered and to make that easier, one final beautification needs to be done: the removal of corner configurations. A corner configuration is three neighboring cells forming a right angle, e.g. the current cell has a right and a top neighbor, a right and a bottom neighbor, a left and a bottom neighbor, and a left and a top neighbor. The cell that is at the corner is eliminated.

The one cell thin discrete curve is approximated by a B-spline curve to within a given tolerance that is at most ε. The algorithm was designed for medical data that is very similar to our cell-based data, however, it tends to be much more complicated than CAD data. The accuracy of the fitting is so good that it never misses a cell (MRI pixel) while conveying the shape of the data. Here we only outline the method, the details of which are found in [1].

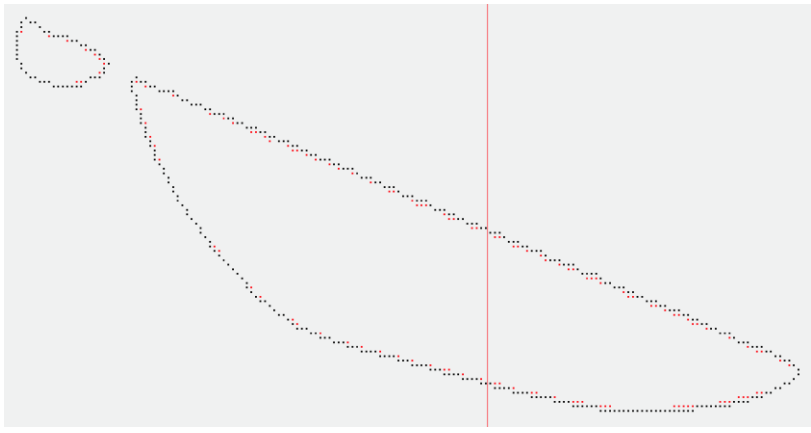

Fig. 4: Finding boundary cells for layer No. 3.

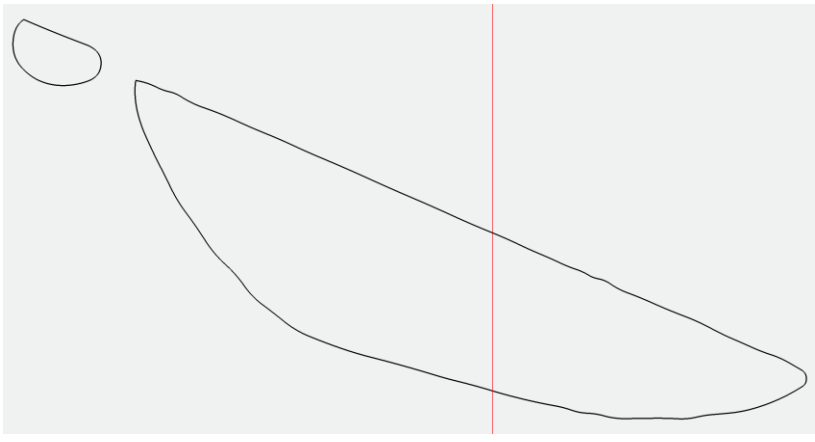

Fig. 5: B-spline curve fitted to the points in layer No. 3.

Let us represent the cells by their centers. This gives us a set of ordered points Q_i, $\mathrm{j}=0,\ldots,\mathrm{M}$, that must be approximated by a B-spline curve of degree p [4]:

$$C(u)=\sum_{i=0}^{n} N_{i,p}(u)P_i$$

After the approximation the curve should not deviate from the point set more than the tolerance. That is, the following must hold:

$$\max_i \left|Q_i - C(\mathrm{u}_i)\right| < \varepsilon \quad Q_i \text{ is assumed at } \mathrm{u}_i$$

The outline of the algorithm is as follows:

- De-noise the point set if necessary. In most cases this is not necessary for 3D printing.
- Decompose the point set into regions of similar complexity.

- The number of decomposed segments are used to determine the number of control points needed to achieve the required accuracy.
- Use the decomposition points to compute a knot vector for approximation with or without end derivatives. Note that the choice of the knot vector is critical both for efficiency and for numerical stability.
- Fit a B-spline curve with the computed degrees of freedom and the knot vector and check the error.
- If the error satisfies the required tolerance, we are done. Otherwise, a better decomposition is computed and the fitting is repeated.

Figure 5 shows the final result for layer No. 3. Figure 6 shows the final results: intersections for layers 3, 260 and 452 on the left, and all intersections for every ten slices on the right.

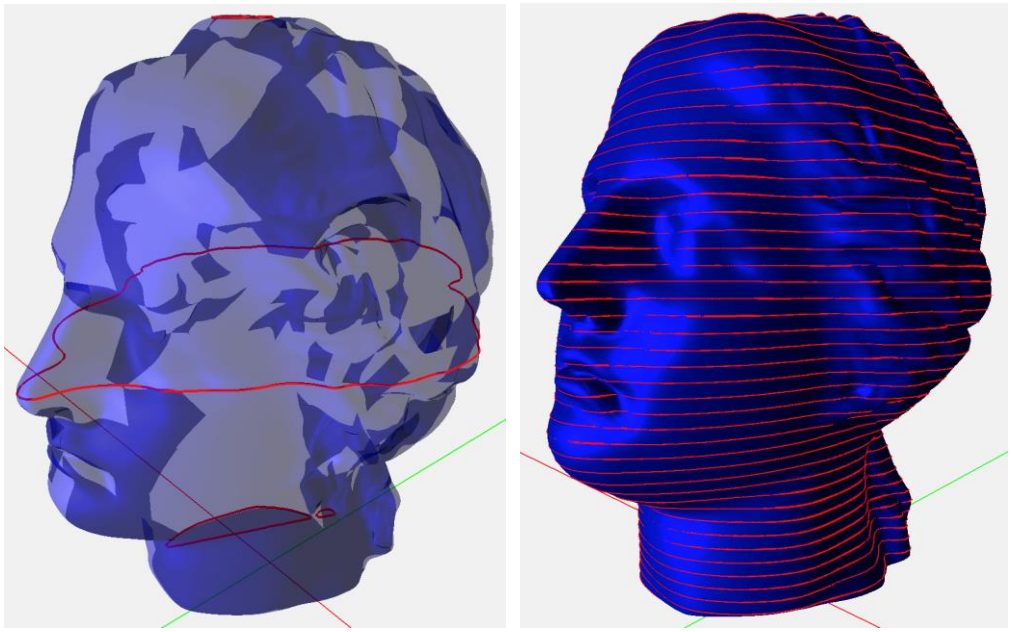

Fig. 6: Layers 3, 260 and 452 (left) and every ten slices from top to bottom (right).

References:

[1] Grove, O.; Rajab, K.; Piegl, L. A.; Lai-Yuen, S.: From CT to NURBS: bio-modeling with B-spline curves, Computer-Aided Design & Applications, 8(1), 2011, 3-21. https://doi.org/10.3722/cadaps.2011.3-21

[2] Heckbert, P. S.: A seed fill algorithm, in Glassner, A. S. (ed.) Graphics Gems, Academic Press, New York, 1990.

[3] Oropallo, W.; Piegl, L. A.; Rosen, P.; Rajab, K.: Generating point clouds for slicing free-form objects for 3-D printing, Computer Aided Design & Applications, 14(2), 2017, 242-249. http://dx.doi.org/10.1080/16864360.2016.1223443

[4] Piegl, L.; Tiller, W.: The NURBS Book, Springer-Verlag, New York, NY, 1997. http://dx.doi.org/10.1007/978-3-642-59223-2