Point cloud slicing for 3-D printing

William Oropallo, Les A. Piegl , Paul Rosen & Khairan Rajab

COMPUTER-AIDED DESIGN & APPLICATIONS, 2017
https://doi.org/10.1080/16864360.2017.1353732

Point cloud slicing for 3-D printing

William Oropallo[a], Les A. Piegl[a], Paul Rosen[a] and Khairan Rajab[b]

[a]University of South Florida, USA; [b]Najran University, Saudi Arabia

ABSTRACT
This paper revisits a more than half a century old problem: slice a free-form object into layers for manufacturing. A point based approach is taken that would have been prohibitive even a decade ago. Due to modern hardware, plenty of storage and a plethora of software packages, the time has come to ditch complicated and error prone numerical code and deploy a simple point based method to achieve robustness and accuracy that have been lacking for a very long time.

KEYWORDS
3-D printing; NURBS; point cloud; object slicing

1. Introduction

Object slicing has been around since the advent of CAD techniques in design and manufacturing. In the early days complex objects, such as a ship hull or an airplane fuselage, have been sliced into cross sections, with some distance apart, and a skin was pulled over the sections to complete the design. The method was termed lofting [15] because it needed so much space that it was done in the loft. As time went on, lofting became a powerful tool to model incredibly complex objects; it became the scan-line method of CAD, borrowed from the scan-coherence principle of computer graphics. It has been rediscovered from time to time [17] to aid in design and fabrication of virtually any objects from household items to entire buildings. 3D printing is, effectively, the consequence of decades of cross-sectional practices where the slices are stacked on top of each other instead of keeping them apart, and eliminating the need to generate the skin.

Unless the design has been generated in layers, as in cross sectional design using planar sections, 3D printing needs to decompose the object into slices, which requires sectioning with planes parallel to the device's moving tray. This may seem like a simple task, however, there are still numerical as well as algorithmic challenges that indicate that either more work or a complete paradigm change is needed [11]. Figure 1 illustrates the point. The left image shows the model we use in this paper, whereas the right picture illustrates the result of slicing the object with Slic3r after STL conversion from Rhino 3D. Apparently, something is still terribly wrong!

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, 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 as in Figure 1.

This paper argues that it is time to clean house; 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 2. The slicing is done on the point cloud model employing practically no numerical procedure. A few decades ago such an approach would have been prohibitive because of the need to store and process millions of points, however, due to advancements in technology, our approach is not only viable, it is now mainstream computing.

Object slicing has a long history in the literature and we give proper credit to the prior art. These techniques either rely on the precise NURBS model or compute the

CONTACT William Oropallo woropall@mail.usf.edu; Les A. Piegl lespiegl@mail.usf.edu; Paul Rosen prosen@usf.edu; Khairan Rajab khairanr@gmail.com

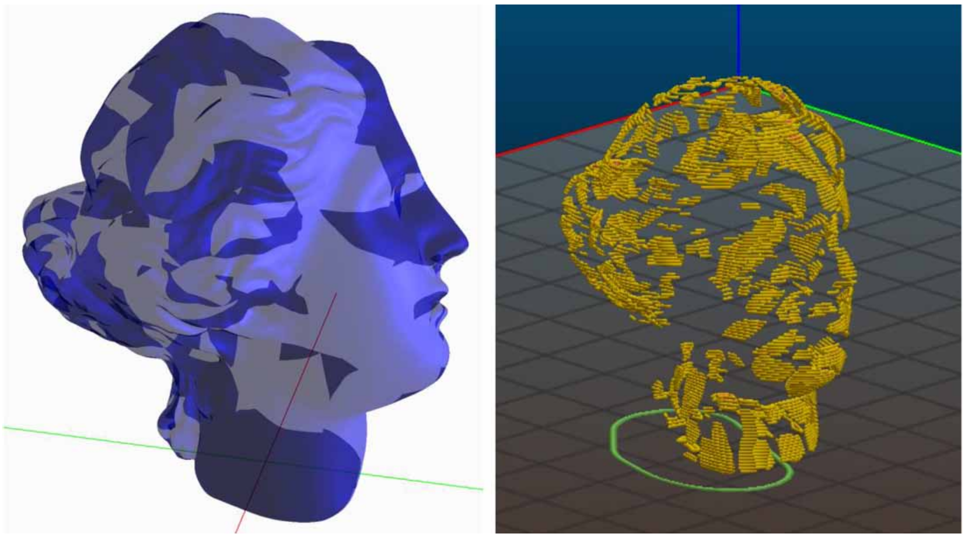

Figure 1. Test model (left), STL-based slices (right).

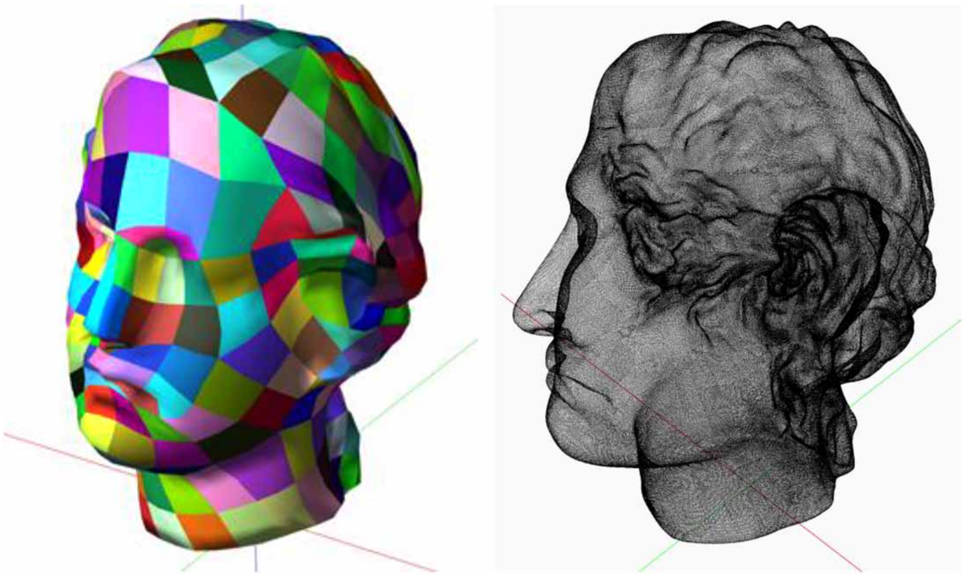

Figure 2. Head modeled by a set of NURBS surfaces (left), point cloud model (right).

slices from the STL conversion. None of them has been relied upon in this work [1–2, 5–7, 8–10, 13–14, 16, 18–25].

The organization of the paper is as follows. First, we introduce some NURBS 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. At the end we offer some conclusions.

2. B-spline notation

To better comprehend the method presented herein, some B-spline notations are in order. A B-spline surface of degree p in u-direction and q in v-direction is a tensor-product surface in the following form [15]:

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)P_{i,j}^{w}$$

where $P_{i,j}^{w}$ are the weighted control points, $N_{i,p}(u)$ and $N_{j,q}(v)$ are the normalized B-splines defined over the knot vectors

$$U = \{\underbrace{u_0 = \cdots = u_p}_{p+1}, u_{p+1}, \ldots, u_r, \underbrace{u_{r-p} = \cdots = u_r}_{p+1}\}$$

$$V = \{\underbrace{v_0 = \cdots = v_q}_{q+1}, v_{q+1}, \ldots, v_s, \underbrace{v_{s-q} = \cdots = v_s}_{q+1}\}$$

We assume that the knot vectors are always clamped, i.e. the end knots are repeated with the multiplicities above. If the B-spline surface has no internal knots, it degenerates to a Bezier surface, which is used for point cloud generation in our previous paper [12].

3. Printer setup and point cloud generation

We take two parameters from the 3-D printer:

- λ - the layer thickness. We assume this to be a constant throughout the printing process.
- 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 $S_i(u, v), i = 0, \ldots, N$, covering the complex part (see Figure 2), we want to turn this set of surfaces into a point cloud $Q_j, j = 0, \ldots, M$, 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 $|Q_k - Q_l| < \delta \; \delta < \varepsilon$. 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 [12]. For reasons of space saving we refer the reader to this paper for further details.

4. Point processing into layers

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. Two examples are shown in Figure 3, one for layer No. 3 (top) and one for layer No. 260 (bottom). It is quite evident that the angles of intersection make a big difference; on the top the surfaces are nearly parallel to the plane, whereas on the bottom they are almost perpendicular. More details on how to process the sub-surfaces can be found in [12].

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.

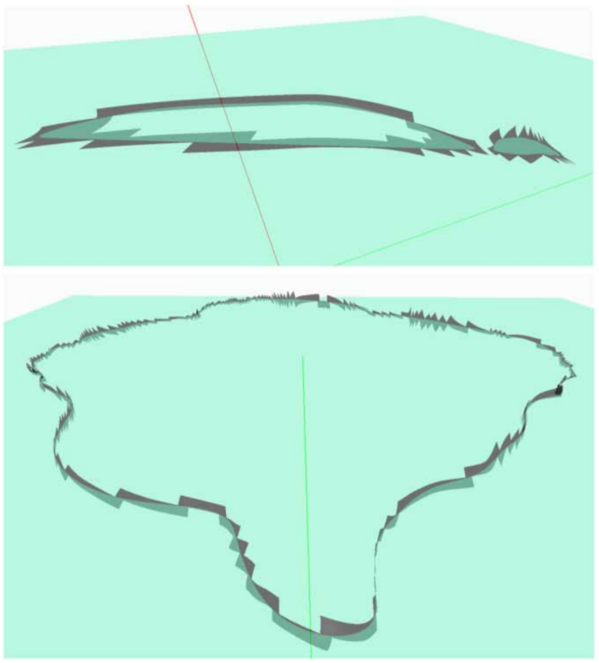
Figure 3. Sub-patches intersecting the slicing plane.

Figure 4 shows the RED and BLUE cells for layers 3 and 260. Note how well the intersection is delineated on the border between the sea of RED and BLUE cells. Figure 5 shows BLACK cells for layer 3, embedded into the RED and BLUE cells.

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. That is:

- For all RED cells find transitions from RED to BLUE, i.e. if (x,y) is RED and (x + 1,y) is BLUE, mark both cells as BLACK.
- For all BLUE cells look for transitions and mark the BLUE and the RED in the transition as BLACK.
- Mark all BLACK cells as NONVISITED and NOCENTER for later purposes.

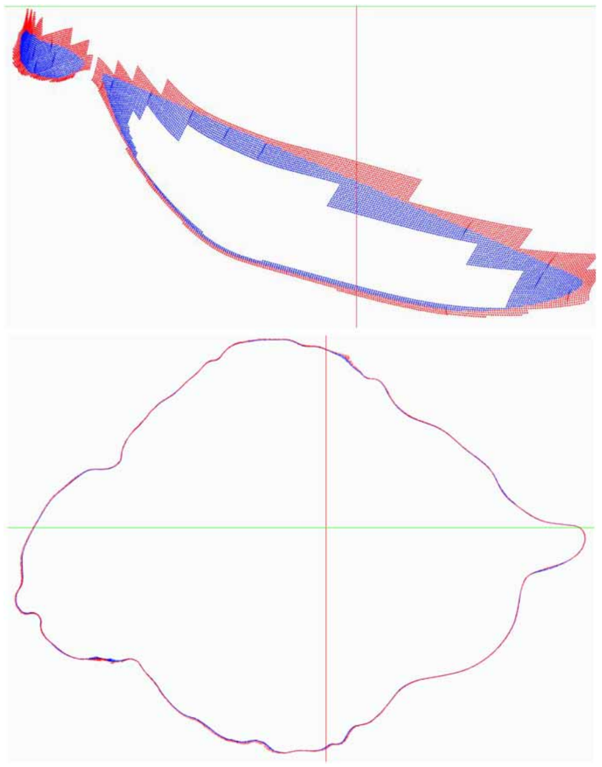
Figure 4. RED and BLUE points for layers 3 (top) and 260 (bottom).

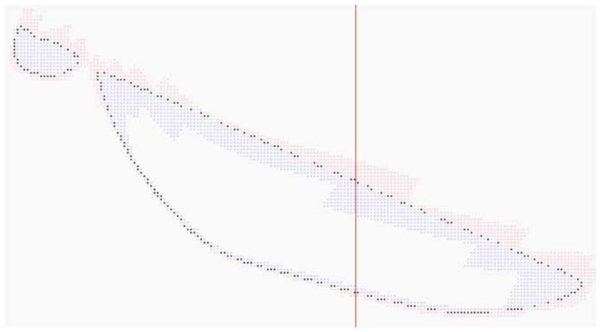
Figure 5. RED, BLUE and BLACK cells for layer 3.

After this step we have a gap free, maximum two pixels thick set of BLACK cells, Figure 6.

5. Separating the intersection curves

After all BLACK cells have been processed, it is time to see how many intersection curves there are and separate them. Because the sea of BLACK cells delineate a discrete curve that is at most two pixels thick (for each cell the minimum number of neighboring cells in x- and y-directions is at most two), we employ a 3×3 mask that is moved along the digital curves to collect all BLACK cells that belong to one closed intersection curve. The algorithm is as follows:

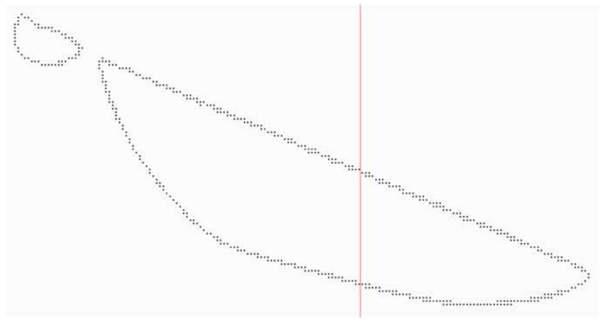
Figure 6. A superset of BLACK cells covering the intersection of layer 3.

- All BLACK cells have been marked as NONVISITED and NOCENTER in the previous step.
- Pick any BLACK cell, center the mask at this cell and mark it as CENTER.
- Mark all BLACK cells inside the mask as VISITED and place them to the output stream.
- Move the center to any one of the VISITED BLACK cells that is not marked as CENTER and collect all NONVISITED BLACK cells.
- Repeat the process for all BLACK cells that are marked as NOCENTER and collect all BLACK cells that are NONVISITED, and place them to the output.
- One curve segment is found when there are no NOCENTER and NONVISITED BLACK cells left.

One can visualize the process by imagining a rectangular vacuum head of 9 holes (3×3) used to collect small balls placed along a closed curve. Before the head can move forward, all balls need to be sucked up, i.e. all cells that are in the mask must be marked VISITED. The vacuuming stops when no balls are left (all BLACK cells are marked VISITED and CENTER) Figure 7 shows the results of intersection curve segment separation for layer 3.

Please note that this algorithm assumes that the intersection curves are at least one cell (ε) apart, i.e. as the

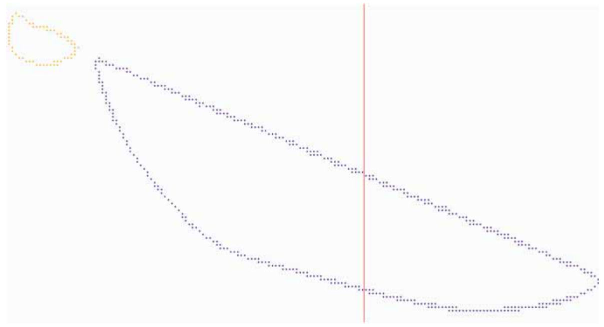
Figure 7. Two intersection curves are marked with orange and blue for layer 3.

3×3 mask, centered on one BLACK cell, moves along the curve, it does not hit another segment. If it does, the curves are touching or form a loop and the algorithm considers them as one curve.

6. Finding boundary cells

As Figure 6 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 [4]. 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.

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. The best way to describe the algorithm is by using recursion. Calling the function FILL4, the algorithm is as follow:

```
• Get the current cell (x,y)
• If not a BOUNDARY or GRAY or a BLACK cell
    ○ Mark it GRAY (the fill color)
    ○ FILL4(x + 1,y)
    ○ FILL4(x − 1,y)
    ○ FILL4(x,y + 1)
    ○ FILL4(x,y − 1)
• Else if it is a BLACK cell, set it to be BOUNDARY cell
```

Once the recursion stops, all boundary cells are collected and the rest are eliminated. Figure 8 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.

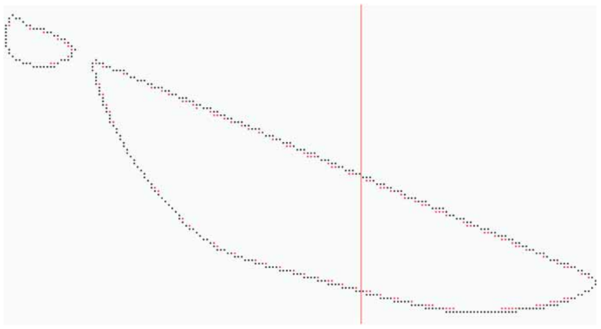
Figure 8. Finding boundary cells for layer 3.

7. Fitting B-spline curves

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 [3].

Let us represent the cells by their centers. This gives us a set of ordered points $Q_i, j = 0, \dots, M$, that must be approximated by a B-spline curve of degree p [15]:

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

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

$$\max_i |Q_i - C(u_i)| < \varepsilon \quad Q_i \text{ is assumed at } u_i$$

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 9 shows the final result for layer No. 3.

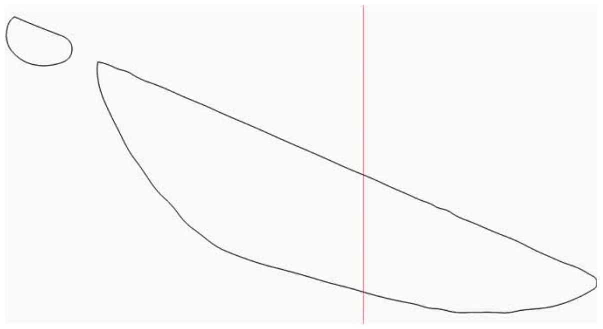

Figure 9. B-spline curve fitted to the points in layer 3.

8. Examples and tests

Figure 10 walks the reader through the process of intersection using slice No. 452. Due to the small number of cells involved, the inner workings of the algorithm are better visualized. The top left image shows the RED and BLUE cells, the top right adds BLACK cells to the RED and the BLUE. After the transition has been computed between all RED and BLUE cells, new BLACK cells are added as shown in the middle left. Three curve segments are identified in the image in the middle right.

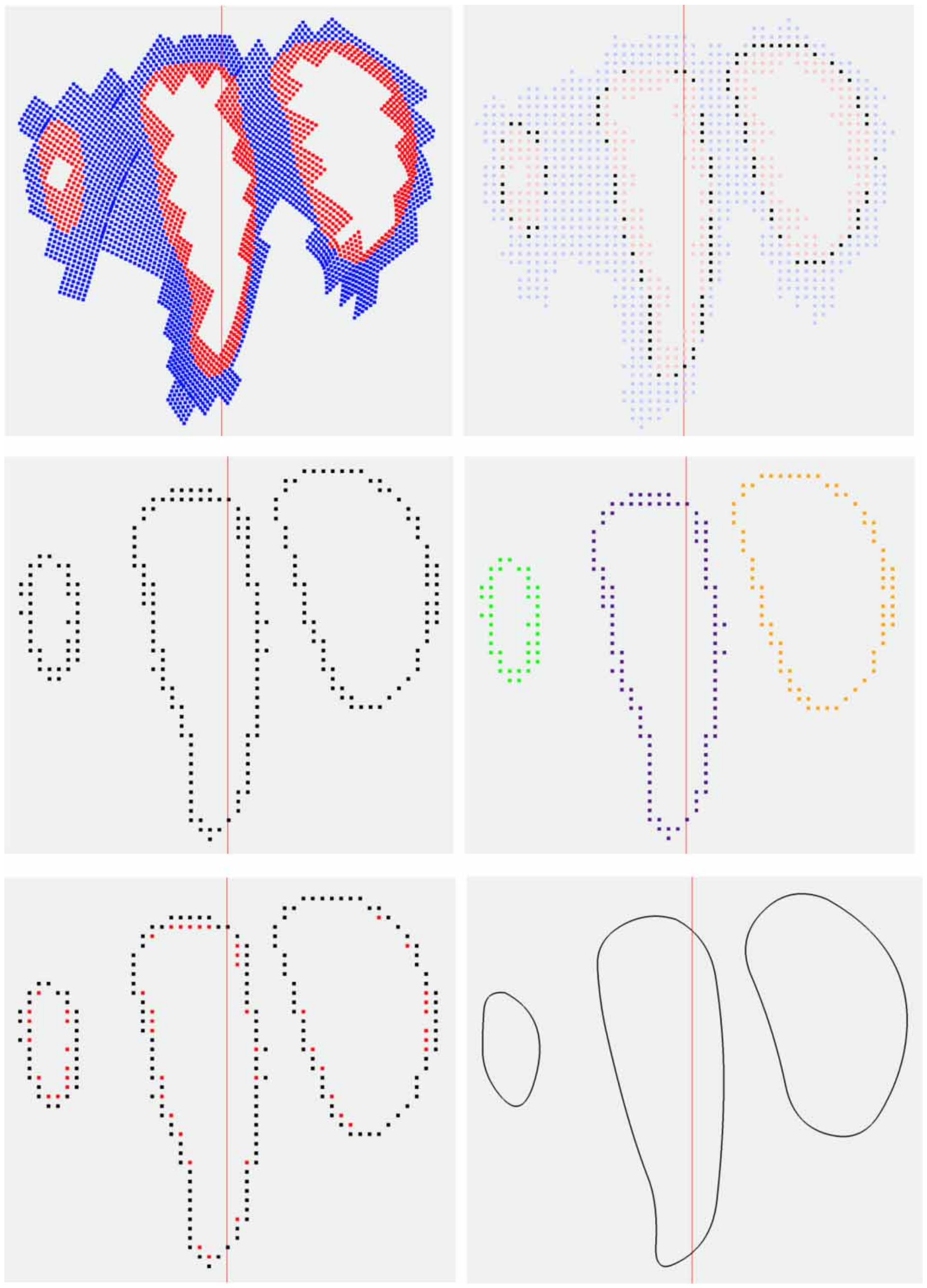

Figure 10. Steps of the slicing algorithm for layer 452.

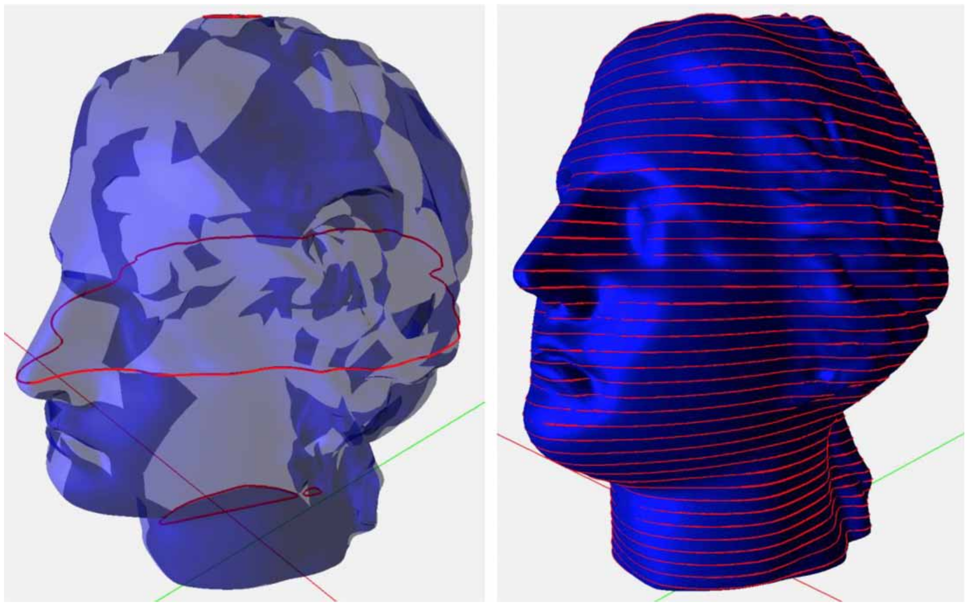

Figure 11. Layers 3, 260 and 452 (left) and every ten slices from top to bottom (right).

The boundary BLACK cells are computed on the bottom left, followed by curve fitting on the right.

The three slices, belonging to layers 3, 260 and 452, discussed above, positioned on the head model, along with every ten slices from top to bottom, are illustrated in Figure 11.

An important question to be answered is: how good is the slicer in terms of accuracy? To answer this question, we sliced the head 0.1 millimeter apart, generated the intersection curves and computed their distances from the original model using discrete sampling and point projection. The results are as follows:

- Total number of sampling points: 855,670
- Average sampling points per slice: 1,812
- Average minimum error: 0.0005 ± 0.000007
- Average maximum error: 0.050689 ± 0.001435
- Average overall error: 0.024674 ± 0.000027

That is, the method is very accurate, even the average maximum error is about one-half of the allowed tolerance with 2.8% standard deviation.

9. Conclusions

A slicing algorithm is presented that is based on discrete sampling of the original model. We believe that the time has come for point-based methods to become viable technologies for geometry processing of free-form objects. The algorithm presented herein is robust, very accurate and requires acceptable amount of storage. It is not real-time yet, however, very close with only a few seconds needed to process each slice on a simple laptop with no hardware accelerator. Since the 3D printer is orders of magnitude slower, allowing a second or two for the slicer will not slow down the printing process.

ORCID

Les A. Piegl http://orcid.org/0000-0003-0629-8496
Paul Rosen http://orcid.org/0000-0002-0873-9518
Khairan Rajab http://orcid.org/0000-0002-1260-5854

References

[1] Debapriya, C.: Asimava, *R. C.: A semi-analytic approach for direct slicing of free form surfaces for layered manufacturing, Rapid Prototyping Journal*, 13(4), 2007, 256–264. https://doi.org/10.1108/13552540710776205

[2] Dolenc, A.; Makela, I.: Slicing procedure for layered manufacturing techniques, *Computer-Aided Design*, 26(2), 1994, 119–126. https://doi.org/10.1016/0010-4485(94)90032-9

[3] 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

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

[5] Jastin, T.; Jan Helge, B.: Local adaptive slicing, *Rapid Prototyping Journal*, 4(3), 1998, 118–127. https://doi.org/10.1108/13552549810222993

[6] Jamieson, R.; Hacker, H.: Direct slicing of CAD models for rapid prototyping, *Rapid Prototyping Journal*, 1(2), 1995, 4–12. https://doi.org/10.1108/13552549510086826

[7] Jin, G. Q.; Li, W. D.; Gao, L.: An adaptive process planning approach of rapid prototyping and manufacturing, *Robotics and Computer-Integrated Manufacturing*, 29, 2013, 23–38. https://doi.org/10.1016/j.rcim.2012.07.001

[8] Kulkarni, P.; Dutta, D.: An accurate slicing procedure for layered manufacturing, *Computer-Aided Design*, 28(9), 1996, 683–697. https://doi.org/10.1016/0010-4485(95)00083-6

[9] Ma, W.; But, W.-C.; He, P.: NURBS-based adaptive slicing for efficient rapid prototyping, *Computer-Aided Design*, 36, 2004, 1309–1325. https://doi.org/10.1016/j.cad.2004.02.001

[10] Mani, K.; Kulkarni, P.; Dutta, D.: Region-based adaptive slicing, *Computer-Aided Design*, 31(5), 1999, 317–333. https://doi.org/10.1016/S0010-4485(99)00033-0

[11] Oropallo, W.; Piegl, L. A.: Ten challenges in 3D printing, *Engineering with Computers*, 32(1), 2016, 135–148. https://doi.org/10.1007/s00366-015-0407-0

[12] 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. https://doi.org/10.1080/16864360.2016.1223443

[13] Pandey, P. M.; Reddy, V.; Dhande, S. G.: Slicing procedures in layered manufacturing: a review, *Rapid Prototyping Journal*, 9(5), 2003, 274–288. https://doi.org/10.1108/13552540310502185

[14] Pandey, P.; Reddy, N. V.; Dhande, S. G.: Real time adaptive slicing for fused deposition modeling, *International Journal of Machine Tools and Manufacture*, 43(1), 2003, 61–71. https://doi.org/10.1016/S0890-6955(02)00164-5

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

[16] Sabourin, E.; Houser, S. A.; Bohn, J. H.: Adaptive slicing using stepwise uniform refinement, *Rapid Prototyping Journal*, 2(4), 1996, 20–26. https://doi.org/10.1108/13552549610153370

[17] Sass, L.; Chen, L.; Sung, W. K.: Embodied prototyping: exploration of a design-fabrication framework for large-scale model manufacturing, *Computer-Aided Design & Applications*, 13(1), 2016, 124–137. https://doi.org/10.1080/16864360.2015.1059202

[18] Sikder, S.; Barari, A.; Kishawy, H.: Effect of adaptive slicing on surface integrity in additive manufacturing, Proc. ASME International Design Engineering Technical Conference, DETC2014-35559, 2014. https://doi.org/10.1115/detc2014-35559

[19] Starly, B.; Lau, A.; Sun, W.; Lau, W.; Bradbury, T.: Direct slicing of STEP based NURBS models for layered manufacturing, *Computer-Aided Design*, 37, 2005, 387–397. https://doi.org/10.1016/j.cad.2004.06.014

[20] Sun, S.; Chiang, H.; Lee, M.: Adaptive direct slicing of a commercial CAD model for use in rapid prototyping, *International Journal of Advanced Manufacturing Technology*, 34, 2007, 689–701. https://doi.org/10.1007/s00170-006-0651-y

[21] Topcu, O.; Tascioglu, Y.; Unver, H.: A method for slicing CAD models in binary STL format, Sixth International Advanced Technologies Symposium, Elazig, Turkey, 141–145, 2011.

[22] Wong, K.; Hernandez, A.: A review of additive manufacturing, International Scholarly Research Network, ISRN Mechanical Engineering, 2012, ID 208760.

[23] Yau, H.-T.; Kuo, C.-C.; Yeh, C.-H.: Extension of the surface reconstruction algorithm to the global stitching and repairing of STL models, *Computer-Aided Design*, 35, 2003, 477–486. https://doi.org/10.1016/S0010-4485(02)00078-7

[24] Zhang, L.-C.; Han, M.; Huang, S.-H.: An effective error-tolerance slicing algorithm for STL files, *International Journal of Advanced Manufacturing Technology*, 20, 2002, 363–367. https://doi.org/10.1007/s001700200164

[25] Zhao, Z.; Laperriere, L.; Adaptive direct slicing of the solid model for rapid prototyping, *International Journal of Production Research*, 38(1), 2000, 69–83. https://doi.org/10.1080/002075400189581