www.cadanda.com
Computer-Aided Design AND APPLICATIONS™

Handling Anomalies in Object Slicing for 3-D Printing

William Oropallo[1], Les A. Piegl[2], Paul Rosen[3] and Khairan Rajab[4]

[1]University of South Florida, woropall@mail.usf.edu
[2]University of South Florida, lespiegl@mail.usf.edu
[3]University of South Florida, prosen@usf.edu
[4]Najran University, khairanr@gmail.com

ABSTRACT

This paper introduces and extension to our previous papers [10, 11] to handle anomalies in the point based object slicing method. The anomalies handled are point, line and plane touch cases as well as overlaps. These anomalies can cause major problems in any intersection procedure, yet, they are seldom discussed, let alone handled. It turns out that the point based approach is capable of handling these special cases with minor extensions.

Keywords: 3-D printing, NURBS, point cloud, object slicing, anomalies.
DOI: https://doi.org/10.14733/cadaps.2019.528-538

1 INTRODUCTION

Anomalies have caused concern in computational algorithms since the beginning of CAD/CAD development. Many numerical methods work reasonably well in the general cases, however, when they encounter special cases such as touch or overlap, they tend to fall apart. To account for these cases, special code is normally inserted that deals with the special cases individually. During the eighties, the birth of solid modeling software, the second author would spend years writing special code for all cases that failed with the general purpose code. Although this was a very tedious process, it worked perfectly well simply because each case was well understood and could be handled with ease (and with an awful lot of code). While it was a doable task, given the relatively small number of special cases, it is definitely not scalable and hence cannot be applied to the potentially large variety anomalies. Our point based approach handles these cases with very minor adjustment to the basic algorithm. Once the extension is made, the method becomes general and handles all important cases.

In this paper we investigate how a general purpose point-based slicer can be made more robust by extending its reach to handle two types of anomalies, commonly occurring in object slicing: (1) touch cases, and (2) overlaps. Within the touch case category, we handle point, line, curve as well as planar touch cases, Figure 1. As the slicer moves up from the tray, it encounters these cases and it needs to know how to handle them. The top right of Figure 1 shows an important case. The slicer not only needs to find the circle of touch, it needs to know that this is a

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

touch and the interior of the circle is not to be filled with material. Similarly, the bottom left needs to be identified as well so that the slicer does not look for a closed boundary. The bottom right needs special attention in that the intersection is not only a set of boundary curves but an entire planar domain.

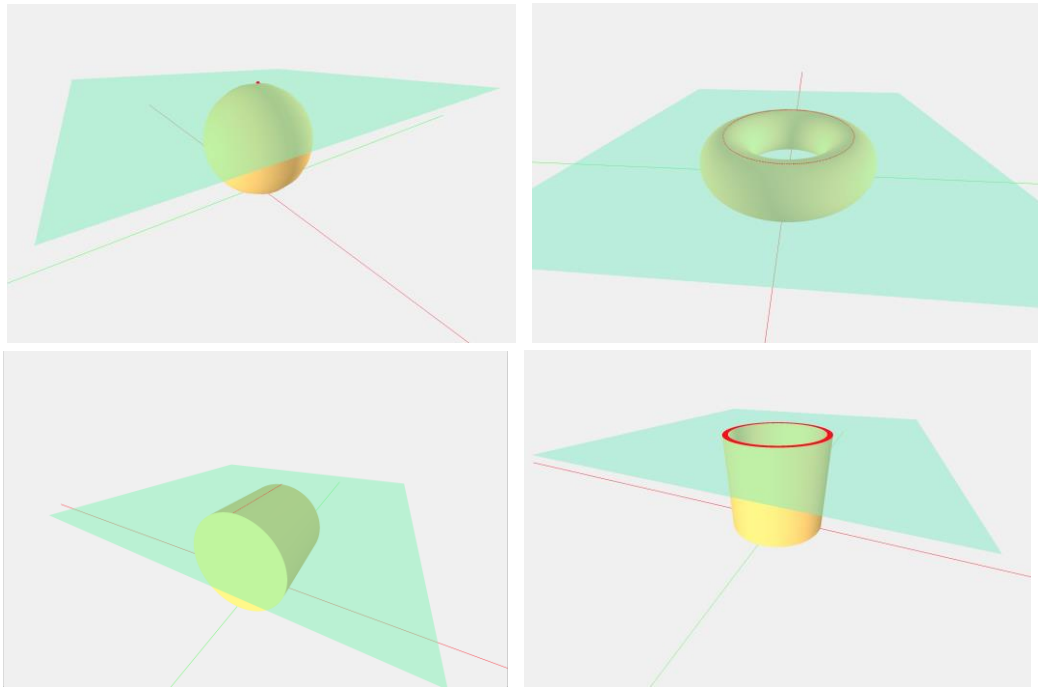

Figure 1: Point, curve, line and planar touch cases.

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 slices from the STL conversion. None of them has been relied upon in this work [1-8, 10-13, 15-23]. The underlying model is assumed to be a NURBS object [14] not an approximation using tessellation.

The organization of the paper is as follows. First, we summarize the point based method illustrating all the important steps. Then the general algorithm is presented. Touch cases as well as overlaps are discussed followed by a comparison with STL based slicing. A conclusions section closes the paper.

2 POINT-BASED SLICING ALGORITHM

The point-based slicing algorithm has the following main components [10, 11], Figure 2. First the NURBS-based model is decomposed into its smaller components, called the Bezier patches.

Then the Bezier patches are further decomposed into smaller surfaces based on the required tolerance and the layer thickness. These tiny surfaces are subsequently binned into a data structure for fast searching as the slicing plane moves up. That is, for each position of the slicing place, there is a list of surfaces that intersect that plane.

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

For each slicing plane points are sampled from the surfaces that are in the local data structure. The sampling is done so that for each point there is a local ring neighborhood where the points are within the required tolerance. The obtained point cloud is ready for slicing.

The slicing begins by laying a grid (of size equals the tolerance) on the plane, and placing voxels (of size equals the tolerance) above and below it. The sampling points are then processed into these voxels and the cells are colored as follows. If there are points in the voxels above and below the cell, it is colored black. If there are points only above, it is marked red, and if there are points only below, it is marked blue. The black cells are intersection cells, whereas the red and blue ones need to be processed. Figure 2 top left and middle show the red, blue and black cells for three intersection loops. Note how well the intersection curve is delineated by the border between the red and blue cells.

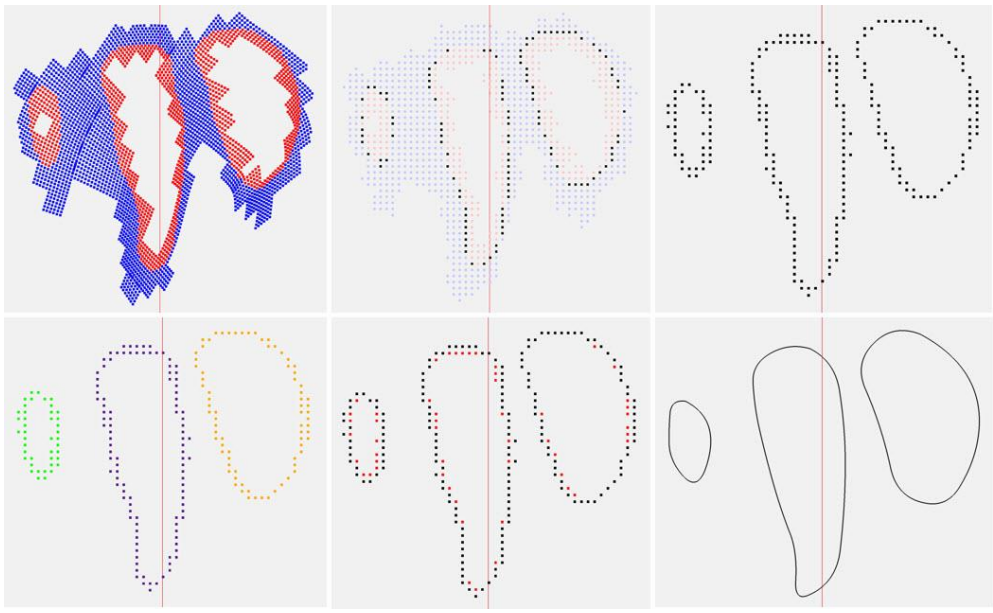

Figure 2: The point-based intersection process.

To fill the gap in the sequence of black cells, the red and blue cells that are involved in the transition in color change are marked black, producing a maximum of two cells wide coverage of the intersection curve, Figure 2 top right.

Using a 3x3 mask the thick intersection curve loops are separated into individual closed curves, Figure 2 bottom left. Some of these curves can degenerate into a line or a point, which requires special attention when filling the region with material.

To thin down the thick array of points that represent the intersection curve, we use a flood fill algorithm. This algorithm, as its name suggests, floods the domain and hits the outermost cells which are then selected to be the intersection points, Figure 2 bottom middle. To store the intersection points for later reuse, and to be able to vary the sampling density, we fit a B-spline curve to the final black points, Figure 2 bottom right. The B-spline curve provides a smooth representation of the intersection curve that can be discretized later on at any level of detail.

In the next section we provide details on how this algorithm can be generalized to handle special cases such as touch cases as well as overlaps.

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

3 GENERAL ALGORITHM WITH ANOMALY DETECTION

The algorithm proceeds exactly as in the previous case up until cell coloring begins. To account for the variety of touch cases, a bit more bookkeeping is necessary with quite a few more flags applied. Because of the large number of flags used, we dropped the coloring scheme and replaced it with named cells. The classification is as follows:

- EMPTY: a cell with empty voxels on it.
- WEAK BELOW: a cell with only a non-empty voxel below the plane (formerly blue).
- WEAK ABOVE: a cell with only a non-empty voxel above the plane (formerly red).
- STRONG: a cell with voxels that have points below and above the plane (formerly black).
- BOUNDARY: cells containing intersection points.
- INTERIOR BOUNDARY: these are intersection contours for the planar touching case that are inside the primary boundary
- PLANAR: used for cells in the planar touching case for the area to be filled
- REPAIRED EXTERIOR: BOUNDARY cells that have been eliminated because they are unnecessary for the boundary creation.
- EXTERIOR: cells that are not BOUNDARY, PLANAR, INTERIOR BOUNDARY or REPAIRED EXTERIOR.

Using these classification for the various cells, the overview of the algorithm is explained below. Please note that handling special cases requires a lot of code and most of it is not very pretty. So the algorithm below may not admit immediate comprehension. However, the examples that follow attempt to clarify many of the complicated steps.

```
Decompose the object into tiny patches as detailed in [10]
For each slicer do
    Sample the surfaces intersecting the slicer
    Generate the grid and the voxels and flag the cells as above
    /* Keep all cells with WEAK ABOVE and BELOW flags for touch classification */
    Separate the cells for multiple contours
    For each individual contours do
        Find the boundary using the fill algorithm
        Mark the cells as BOUNDARY or EXTERIOR
        /* Now find planar areas */
            Search for all remaining EMPTY cells
            Flag them as EXTERIOR
            Flag touching STRONG points as INTERIOR BOUNDARY
        If any boundary cells are touched, mark the INTERIOR BOUNDARY as EXTERIOR
        If no EMPTY cell is left but there are still STRONG cells, mark them as PLANAR
    Order the cells
        Count the number of BOUNDARY cells
        If less than 2, point touch case
        Now order as in the previous algorithm
        Detect if there is an open contour
```


© 2019 CAD Solutions, LLC, http://www.cad-journal.net

Now fit the B-spline curve

Create the contour for all cells with flags INTERIOR BOUNDARY

In the next sections we give more details on the various touch cases to shed some light on the algorithm above.

4 THE POINT TOUCH CASE

The point touch case is handled during the cell ordering phase of the algorithm. Ideally, there should be no more than one point to be ordered, however, due to noise or improper sampling, occasionally there are two points. If it is one point, it is designated as the touch point, otherwise an average is computed of the flagged BOUNDARY points.

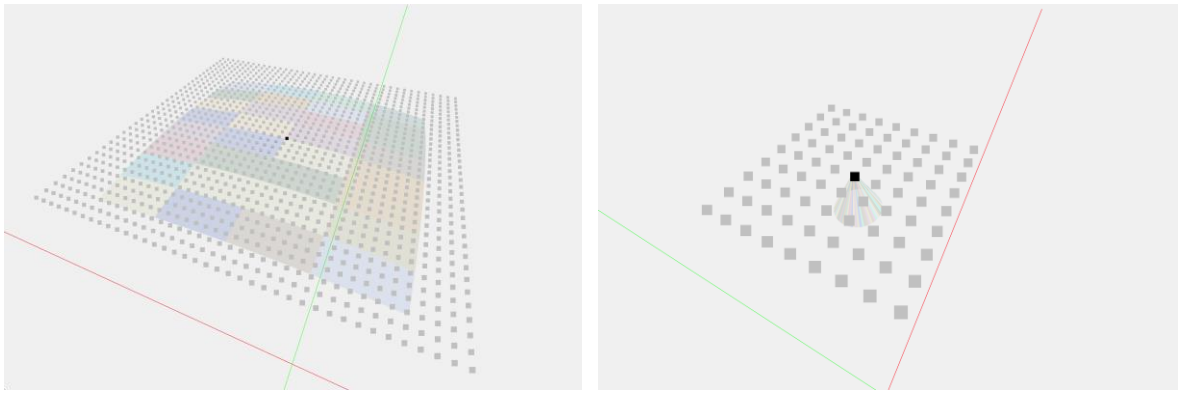

Figure 3: Sphere touch (left), cone touch (right).

Figure 3 shows two examples. On the left there is a spherical surface and a touching plane. The grey cells are EXTERIOR cells, the black is the BOUNDARY, and the colored rectangular surfaces are the tiny patches that are in the local data structure to be processed with respect to the slicing plane. The right image shows the cone case with a similar coloring scheme. Please note that the sampling has to be done with care so that the apex of the cone is hit with at least the accuracy of the manufacturing tolerance. Otherwise it will be missed and the slicing plane is declared non-intersecting.

5 THE LINE AND CURVE TOUCHING CASES

The line and curve touching cases are identified during the separation of the contour loops. In these cases, there are cells with flags WEAK BELOW or WEAK ABOVE only. As the contour forming algorithms proceeds (with the use of the 3x3 mask), only STRONG points are considered. However, to consider the touch case, neighboring points are also examined, i.e. WEAK ABOVE and WEAK BELOW. At the end there are the following cases: (1) there are only WEAK ABOVE or WEAK BELOW cells, in which case we have a touch case, (2) if there are some WEAK ABOVE and some WEAK BELOW cells, then we have an intersection, (3) it can happen that all of the examined cells are WEAK ABOVE and WEAK BELOW, and this points to a (nearly) perpendicular case, i.e. the slicing plane is perpendicular to the surface and the sampling points are within the tolerance below and above.

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

Figure 4 shows a curve and a line touch cases. The curve case on the left is at the bottom of a torus. The red cells are marked as WEAK ABOVE, the light black cells are the STRONG cells and the colored patches are the surfaces that participate in the intersection process. The image on the right shows the case when the cylinder is touching the slicer along one of its rulings. This touch case is also identified during the contour tracing steps, with an added wrinkle: the algorithm finds a dead end, i.e. it does not come around to find the start point. When this happens, the tracing has to resume from the start point in the opposite direction to find the rest of the contour line. The grey cells on the figure show EXTERIOR cells, the black is the contour and the tiny colored patches are the surfaces needed to process the intersection curve. Note that even if it is a cylinder, a lot of tiny surfaces are in the local data structure of the slicer. That is because the system does not know that it is a cylinder. It is processed just like any other NURBS surface. Extra code can be inserted to account for special surface types, such as quadric or planar surfaces. Touch cases for these surfaces can be handled separately without any special consideration. While this is a very practical consideration, something that must be done when implementing the method into commercial systems, it still does not solve the problem of general touch cases and the cases when quadric patches are parts of more complex NURBS objects.

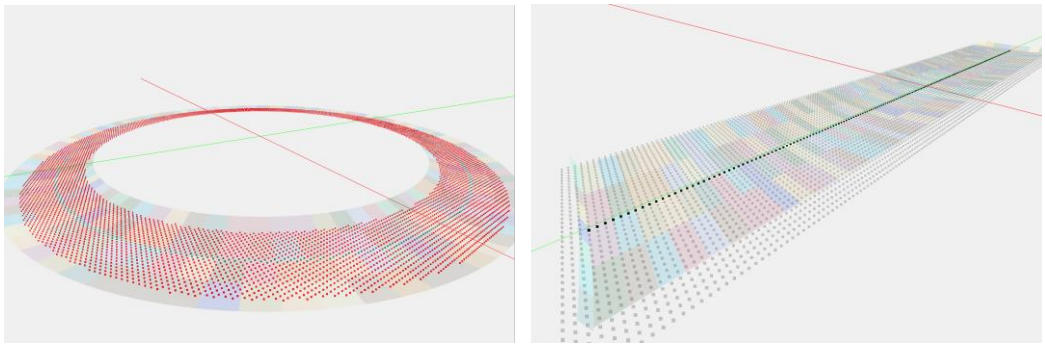

Figure 4: Torus touch (left) cylinder line touch (right).

6 THE PLANAR TOUCH CASE

The planar touch case is identified after the boundaries are found using the flood fill algorithm. The identification needs to be able to separate three cases: (1) the area inside the contour is filled, (2) the area inside is filled, however, has additional interior contours that define holes, and (3) the false positives.

Figure 5 illustrates the process using the bottom of a cylindrical block. The red cells represent STRONG cells, and all cells outside the boundary are EXTERIOR cells, marked grey. During the search inside the boundary we search for STRONG points and flag them as INTERIOR BOUNDARY. Similarly, all EXTERIOR cells with the boundary that are surrounded by STRONG cells are handled the same way.

We also check if we touch any BOUNDARY cells and if so, we need to flag all INTERIOR BOUNDARY cells to EXTERIOR cells. Also, if a BOUNDARY cell touches an EMPTY cell indicates some noise. When there are no more EMPTY cells but there are still STRONG cells that were not flagged as EXTERIOR, they are flagged as PLANAR cells. At this point the PLANAR cells are bounded by one or more curves made up of BOUNDARY cells. If there are PLANAR cells the case is flagged as planar intersection. In Figure 5 the purple cells are PLANAR cells, the black are BOUNDARY, the grey are EXTERIOR and the tiny colored patches form the boundary cap of the cylinder block.

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

As noted above, this case can be, and must be, identified geometrically if the surface type is known. In which case none of the above steps are necessary. However, if the general NURBS object has a planar component, the identification may not be possible and hence the general planar identification process is applicable.

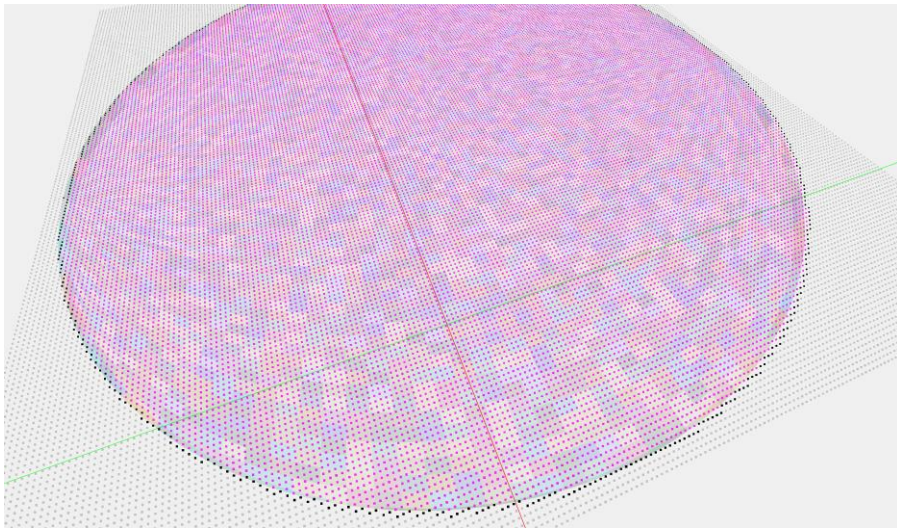

Figure 5: Processing the bottom cap of the cylinder block.

7 THE OVERLAP CASE

Complex objects are designed by stitching individual surface patches together with some level of continuity. Unfortunately, the stitching process is almost never perfect leaving gaps, overlaps and dangling surfaces behind. Figure 6 shows a case of overlapping surfaces in a complex model. Now, this anomaly causes a lot of problems both in the tessellation process and in any numerical code. For example, the iteration can jump from one surface to the next and not converge within the required number of steps. It is always a dangerous maneuver to move across surfaces as they are individually designed with their own parametrizations. Tolerances that work with one surface may not do any good on a neighboring one with vastly different parametrization.

Luckily, the point based approach is not sensitive to these kind anomalies. In fact, the algorithm does not even know if there is an overlap. It just processes the points on all patches in the local data structure and flags the cells according to the locations of the sampling points. When surfaces overlap, the sampling generates points on both surfaces and all the extra points that are not needed are discarded. For example, to set a cell WEAK ABOVE, all we need is one point in the voxel above the cell. If there are two or more, it will not affect the algorithm at all.

Figure 7 shows an example of intersection with the slicer when overlapping surfaces are involved. Note that the blue points are mostly along the intersection curves and are hardly visible due to the size of the image.

8 A COMPARISON WITH STL-BASED SLICING

In this section we give a brief comparison with STL-based slicing. The two models used in the study are shown in Figure 8. The tessellation was done in Rhino and the tessellated model was sliced by Slic3r.

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

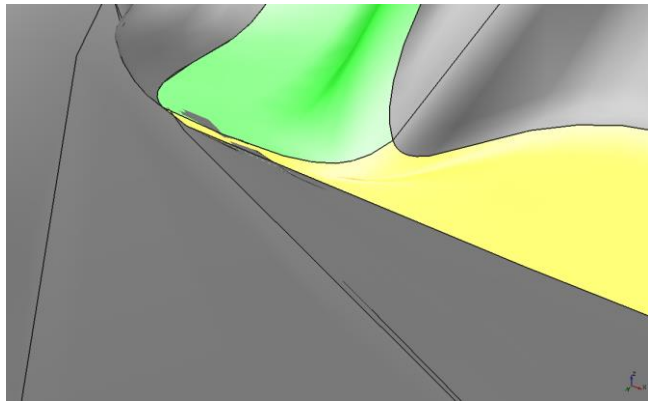

Figure 6: Overlapping surfaces in a complex NURBS model.

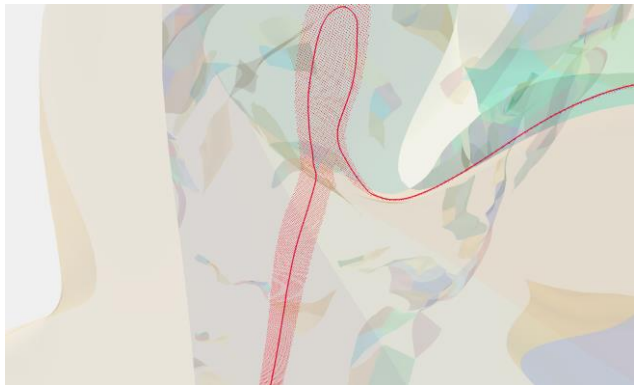

Figure 7: Intersection processing with overlapping surfaces.

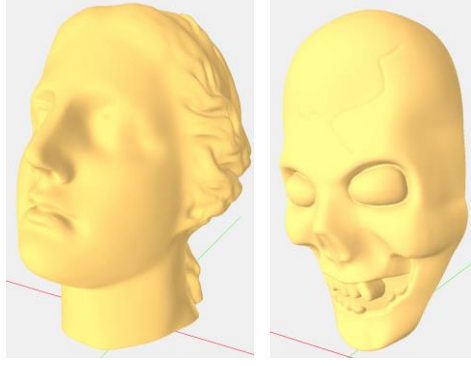

Figure 8: A head and a skull model used in the comparison with point-based slicing.

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

Let us show some examples on the head first. Figure 9 illustrates slice number 100. The first three images show the results of slicing the model based on tessellation to within 0.01, 0.001 and 0.0001 tolerances (left to right). Not what one would expect from a solid slicer. On the right is the point-based slicer producing a very smooth and fairly accurate result (tolerance was set to 0.1 mm).

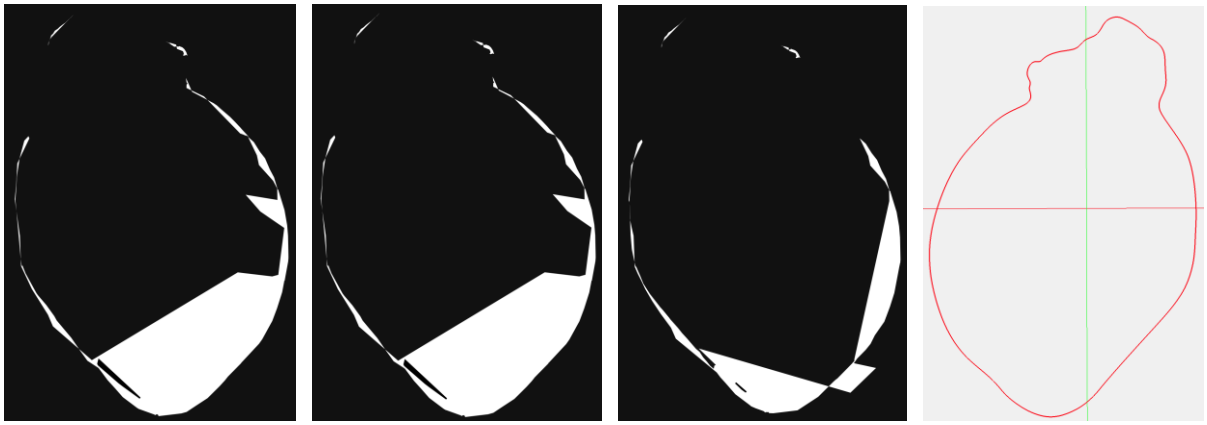

Figure 9: Slicing the tessellated head (left three) and the precise NURBS model (right).

Next, let us slice the skull and show slice number 182, Figure 10. The same tolerances of 0.01 (top left), 0.001 (top right) and 0.0001 (bottom left) have been applied. The pictures are worth a thousand words! Please note that higher tessellation tolerances do not necessarily mean better results, as clearly shown in Figure 9 and 10. Sometimes anomalies can show up for tighter tolerances even though they were not present in cases of lower tolerances,

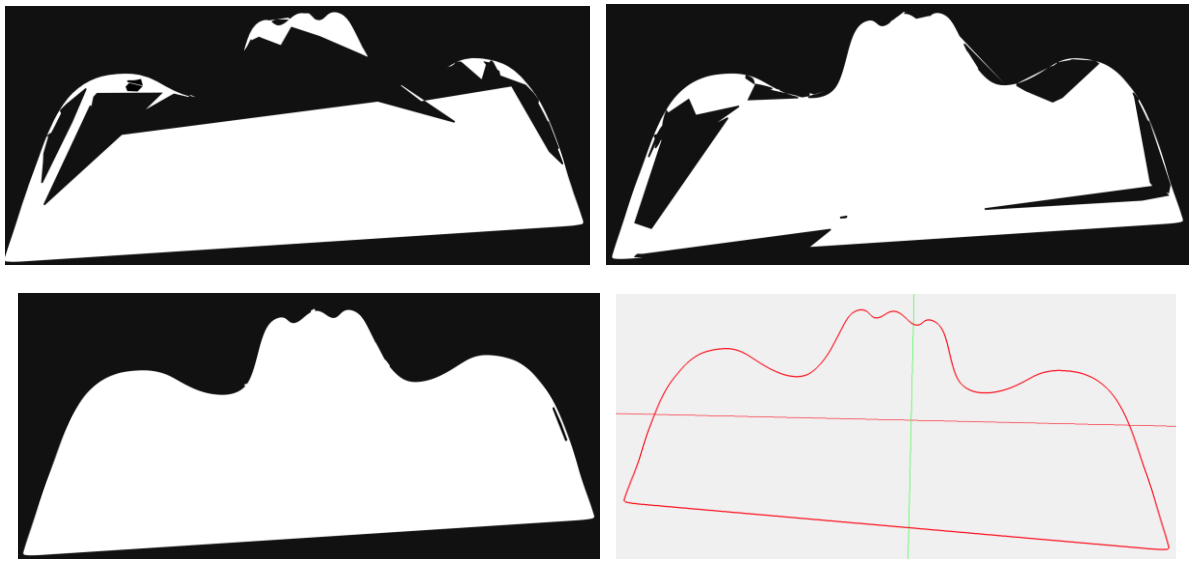

Figure 10: Slicing the tessellated skull and the precise NURBS model.

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

CONCLUSIONS

An extension to our point-based slicing algorithm is presented that handles anomalies that show up during slicing an object for 3-D printing. The extension requires additional bookkeeping, however, it leaves the basic algorithm intact. The results are accurate to within the required manufacturing tolerance. That is, as long as the various touch cases are within the step size, i.e. the cell size, the touch cases are found and handled appropriately. We also provided a comparison to tessellation based methods. As it turns out, depending on the tessellation tolerance used, not only the accuracy but also the topology of the intersection curves change. This is not present in the point-based approach.

After many years of testing the point-based approach, it is our conclusion that it is a very viable alternative to other techniques based on numerical methods or tessellations. It is very robust, accurate to within required manufacturing tolerances, can handle anomalies with minor adjustments, and is reasonably fast to outperform the printer in real time processing.

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] Jastin, T.; Jan Helge, B.: Local adaptive slicing, Rapid Prototyping Journal, 4(3), 1998, 118-127. https://doi.org/10.1108/13552549810222993
[4] 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
[5] 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
[6] 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
[7] Ma, W.; But, W.-C.; He, P.: NURBS-based adaptive slicing for efficient rapid prototyping, Computer-Aided Design, 36, 2004, 1309-1325. http://dx.doi.org/10.1016/j.cad.2004.02.001
[8] 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
[9] 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
[10] 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
[11] Oropallo, W.; Piegl, L. A.; Rosen, P.; Rajab, K.: Point cloud slicing for 3-D printing, Computer Aided Design & Applications, 15(1), 2018, 90-97. https://doi.org/10.1080/16864360.2017.1353732
[12] Pandey, P. M.; Reddy, 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
[13] 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

© 2019 CAD Solutions, LLC, http://www.cad-journal.net

[14] Piegl, L.; Tiller, W.: The NURBS Book, Springer-Verlag, New York, NY, 1997. http://dx.doi.org/10.1007/978-3-642-59223-2
[15] 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
[16] 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. http://dx.doi.org/10.1115/detc2014-35559
[17] 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
[18] 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. http://dx.doi.org/10.1007/s00170-006-0651-y
[19] 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.
[20] Wong, K.; Hernandez, A.: A review of additive manufacturing, International Scholarly Research Network, ISRN Mechanical Engineering, 2012, ID 208760.
[21] 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. http://dx.doi.org/10.1016/S0010-4485(02)00078-7
[22] 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. http://dx.doi.org/10.1007/s001700200164
[23] 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

© 2019 CAD Solutions, LLC, http://www.cad-journal.net