Title:
Special Cases in Object Slicing 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, Anomalies

DOI: 10.14733/cadconfP.2018.71-75

Introduction:

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 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. The method described herein is based on our point-based slicer algorithm [1,2] performed on NURBS objects [3].

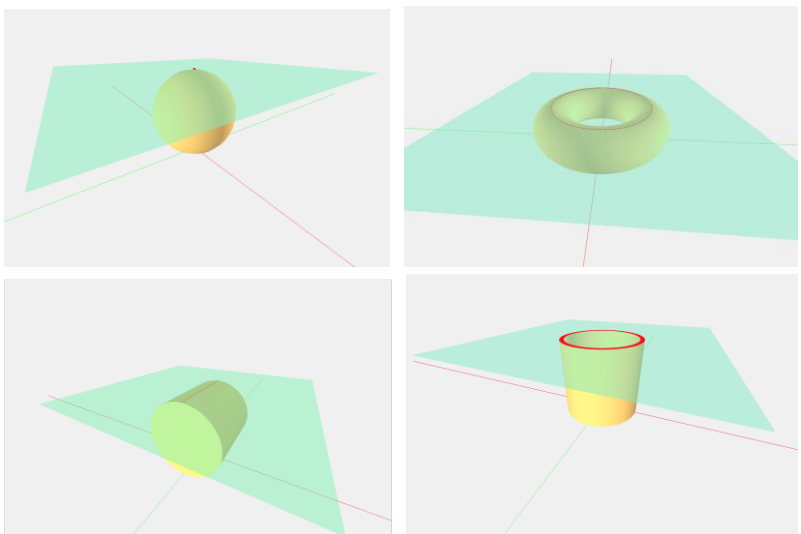

Fig. 1: Point, curve, line and planar touch cases.

Overview of point-based slicing algorithm:
The point-based slicing algorithm has the following main components [1, 2], 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.

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.

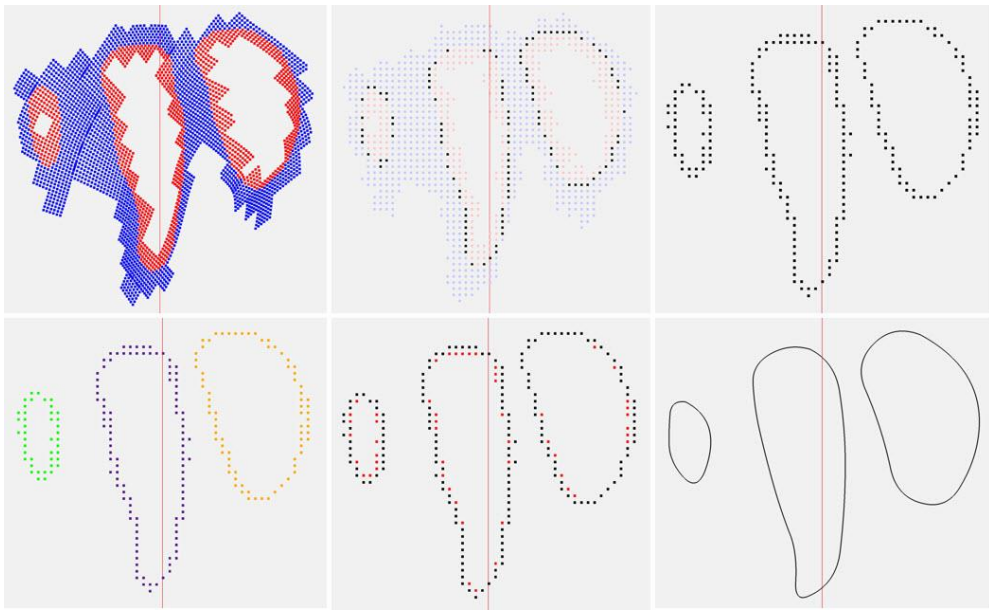

Fig. 2: The point-based intersection process.

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.

General algorithm with anomaly detection:
The algorithm proceeds exactly as described in the previous section 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 classifications 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 [1]
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

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 a few touch cases to shed some light on the algorithm above.

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.

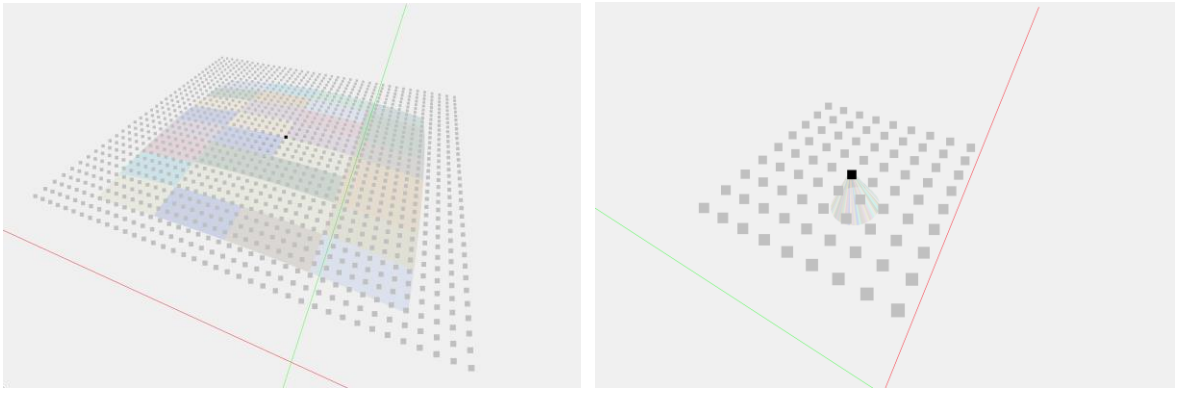

Fig. 3: Sphere touch (left), cone touch (right).

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.

The line and curve touch 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.

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 in 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.

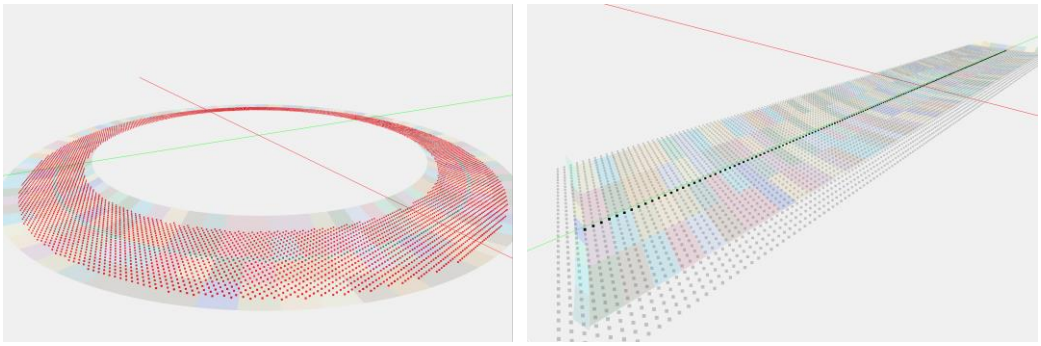

Fig. 4: Torus touch (left) cylinder line touch (right).

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 will also provide 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.

References:

[1] 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

[2] 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

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