Comparing Slicing Technologies for Digital Light Processing Printing

Tsz-Ho Kwok
Department of Mechanical, Industrial and Aerospace Engineering
Concordia University
Montreal, QC H3G 1M8, Canada
Email: tszho.kwok@concordia.ca

The process planning of a manufacturing method is the key to ensure the quality of the fabricated part. In Additive Manufacturing (AM), slicing is a crucial step in process planning to convert a Computer-Aided Design (CAD) model to a machine-specific format. If the slicing results were incorrect, the manufacturing quality would have no way to be assured. Therefore, it is important to understand the performance of different slicing technologies for AM. Digital Light Processing (DLP) printing is an important AM process that has a good surface finish, high accuracy and fabrication speed, and is widely applied in many dental and engineering industries. However, while most other AM processes are toolpath-based, as a process that uses images as the fabrication tool, the DLP printing has its process planning understudied. Therefore, the main goal of this paper is to study, compare and benchmark the slicing technologies for DLP printing. Three slicing technologies are compared: contour, voxelization, and ray-tracing. They are tested with some common defects in slicing, and their usage in computational resources is also reported. The summary and suggestion are given at the end.

1 Introduction

Additive manufacturing (AM) or layered manufacturing is named by the nature of its process, that adds material to build object layer by layer. The most attractive part of the AM is that it can fabricate a three-dimensional (3D) part directly from a computer-aided design (CAD) model without the need of specific tooling or fixture. However, as the object is fabricated layer by layer, the CAD model must be sliced into layers as well, so that the process planning can be done for printing. Slicing acts as the middleman between the CAD model and the 3D printer, and the slicing result is one of the important factors determines the quality of the object. If the slicing is incorrect, the fabrication process is messed up due to the incorrect toolpath planning and the fabricated layer is inaccurate, and thus the shape of the entire piece is ruined. The quality can not be assured, no matter how much tolerance analysis or dimensional control is done on the CAD model. For example, if an object is modeled as an assembly, the mesh may look healthy but there are parts that overlap. If this model is sliced directly to print, the intersection regions are going to have redundant toolpaths. If it is a selective laser sintering (SLS) process, the laser will go over the regions multiple times, resulting in material failures due to different heating effect over the structure. If it is a fused filament fabrication (FFF) process, the nozzle will print in the same place, which will create non-uniform thickness of a layer or even damage the nozzle due to crashing. Therefore, to improve the repeatability and reliability in AM, the capability and limitation of different slicing technologies must be understood and taken into consideration during the design phase.

The first 3D printing technology being commercialized in 1980s is the stereolithography (SLA) process, which is also one of the most important technologies applied in many dental and engineering industries. The Digital Light Processing (DLP) process shares the same principle to cure liquid resin to solid by vat polymerization. While SLA uses a laser tool to draw out the layer line by line, DLP cures the entire layer using a projector [1]. Because of this difference in the processes, their planning is also different, i.e., toolpaths for SLA and images for DLP. Toolpath planning is a common technique for most 3D printing processes like SLS and FFF, so SLA can share the same slicing technologies that create contours from the CAD model. However, DLP is quite a specific process that makes use of images as the media to transfer the shape of designed model into the shape of actually fabricated part using projection, and thus the slicing technology for DLP needs to generate the images. One way is to use the contours generated by the traditional slicing methods as a boundary on an image to separate the regions with/without material. There are also other ways to generate the images directly. Despite the difference, DLP has a remarkable fabrication speed on top of high accuracy and good surface finish in the SLA/DLP process. As one of the most promising technologies for the future of AM, it is important to study and benchmark different slicing technologies for DLP 3D printing, which is the focus of this paper.

The organization of this paper is as follows. The rest of this section will review the related works of slicing. Section 2 will summarize and discuss the slicing technologies for DLP that are selected in this paper. The study consists of two parts: one is the comparison of their performance in handling different defects commonly seen in slicing (Section 3), another one is the comparison of computational resources needed in each technology (Section 4). The paper will be concluded in Section 5.

1.1 Related Works

From CAD model to 3D printing, it requires a number of steps in the process planning [2], including tessellation, orientation, support generation, slicing, and toolpath generation. Among which, slicing is the key step to convert a CAD representation to a machine-specific format, including the computation of profiles and infills for each slice. Slicing was originated in Computer Numeric Control (CNC) machining, and it is the basis to generate 3D cutting toolpaths from CAD model. In 3D printing, the toolpath generation is similar or even simpler as each layer is just a two-dimensional (2D) plane, so the slicing technologies for CNC are applied in 3D printing. The most common slicing algorithm is the contour slicing, which intersects the CAD model with slice planes at different heights and constructs the contours of the surface. There are two ways in contour slicing. One is the closest point method [3, 4] which does the face-plane intersection first, and then retrieves the connectivity for the intersection points using nearest neighbor search. Another one is the topology method [5, 6] which reconstructs the topology of the model first, and then does the intersections with the connectivity. They have different time complexity depending on the mesh size and the number of layers. To speed up the slicing process, the mesh faces can be sorted to reduce the number of faces need to be intersected [7], and others developed information reuse in the context of customization [8] or applied adaptive slicing to reduce the number of layers [9]. After the contours are generated, the infill pattern can be generated within them [10, 11].

The contour slicing requires the mesh connectivity and quite a number of intersection operations, but there are also other technologies developed as alternatives. Instead of computing face-plane intersections, some use voxelization [12] – 3D pixels – to find out the volume with/out material, and some even do not work on mesh, but directly slice on the CAD model [13], point clouds [14], fitted moving least square surfaces [15] or other implicit surfaces [16, 17]. A similar voxelization strategy has been developed at the 2016 Formlabs Hackathon [18], which essentially is a ray-tracing algorithm that uses a stencil buffer to determine the in/out of the model. Wang et al. [19] has also developed a ray representation called Layered Depth Normal Image (LDNI) that can be used for slicing purpose [20, 21]. A LDNI is a 2D image, and each pixel shoots a ray to intersect the model and stores a sequence of intersected points. Therefore, the CAD model is sparsely represented by a set of sampling points, which is a compact representation. As this representation

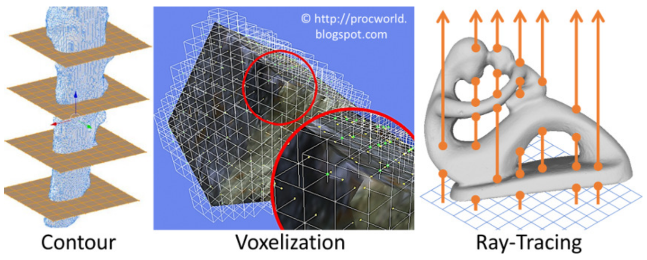

Fig. 1. Three slicing technologies for DLP are tested: Contour [28], Voxelization, and Ray-Tracing.

can solve various defects in mesh model [22], many usages have been developed with LDNI, including offsetting operation [23] and adaptive slicing [24]. With the volumetric or implicit representation of the object, the infill pattern can also be defined as a function [25]. One of the advantages is that the infill can be determined upon demand [26], without the need of reconstructing the internal structures in a CAD format first. Representing objects with multiple materials or functionally graded material is also possible [27].

2 Slicing Technologies for DLP

Although there are methods slicing a CAD model without tessellation, slicing a triangulated mesh model is still the commonly used method in 3D printing caused by its format (i.e., STL) is widely adopted in software and machine. Therefore, this paper mainly focuses on the slicing technologies that take meshes as input, and specifically those for DLP 3D printing. These technologies can be summarized into three categories (see Fig.1), and they are listed in the following with the online-available software used as the representative of the category for comparison.

1. Contour (RayWare Version 1.4.1 [29])
2. Voxelization (Voxelizer Version 2.0.0 [30])
3. Ray-tracing (LDNI-based Solid Modeling [31])

First, the contour method is the traditional slicing process that generates the cross-sectional information by intersecting the input model with a set of horizontal planes. As the input model is tessellated into faces (e.g., triangles defined in the STL), the slicing operation is actually a number of face-plane intersections, each of which is a segment. In a layer, the intersection between the model and a slicing plane is one or more polygons (contours), which are constituted by the segments. These contours define the 2D profile of the model, and material should be put inside them. For DLP 3D printing, the contours are converted into images by rasterization. The software RayWare [29] using the contour method is developed by SprintRay Inc. for their MoonRay DLP 3D printer, and thus it is very qualified to serve as a representative of the contour technology.

Second, the voxelization method creates a 3D array of voxels that can cover the whole volume of the input model, and then decides whether each voxel is inside or outside the model. The in/out determination is challenging, because the mesh is just a set of faces in the 3D space without the information of inside or outside. The software Voxelizer [30] developed by ZMorph Inc. is a representative of this method, which can be seen from its name. It starts a propagation from the outermost voxels which is stopped by the mesh surface, so that the regions that are outside the model can be identified. Voxelizer can theoretically work for different 3D printers, and especially for DLP printing as the voxels from the same height directly form a slice image. Unfortunately, the current version doesn't provide this function. For the experiments in this paper, the infill is set to nearly 100% to mimic a complete fill and its preview from the software is screen-captured.

Third, the ray-tracing method starts with a 2D image and determines in/out for every pixel in a slice, similar to the point-in-polygon testing. Here, the testing is done by casting a ray from each pixel to intersect with the model, and finding out if the ray reaches the interior or exterior of the model at a particular height. The representative selected for this technology is the Layered Depth Normal Image (LDNI) [31], which is an extension of the ray representation in solid modeling. Based on a discrete sampling approach, a structural set of LDNIs consists of x-, y-, and z-LDNI along their axes respectively. A LDNI in each axis is a 2D image, and the three images are located to cover the whole volume of the input model. Each ray intersects with the model and stores a sequence of intersection points with its normal. Therefore, it is a sparse representation of a volumetric model, and any position that is in between each pair of intersection points on a ray is inside the model. The original implementation of LDNI generates the slice images inside the GPU, however it limits the image resolution due to the GPU memory. As the source code is available, it is modified to generate images in CPU in this study.

All the software selected have applied parallel computing, either multi-core CPU or GPU. The extent of acceleration depends on the ability of the algorithm to be parallel. Therefore, it is fair to compare them as is.

3 Common Defects and Comparison

Created by 3D Systems, the STL file format (a.k.a. Standard Tessellation Language) has become the de facto standard data transmission format, and almost all CAD systems can generate an STL file. An STL file contains a set of triangular faces to define the shape of a CAD model, which means that the CAD model needs to be tessellated. The triangulated faces may look good on the screen, but it could have defects the user cannot see. The defects could disrupt the slicing and make it unreliable. For example, the mesh may have bad connectivity of triangles or small triangle that is flipped over, which may be not noticeable, but it will cause problems to manufacture from that data. This section studies and compares the performance of the three slicing technologies in handling different defects. Following the common STL file errors defined by Varotsis [32], this paper summarizes and defines four categories of defects, and they are discussed in each subsection.

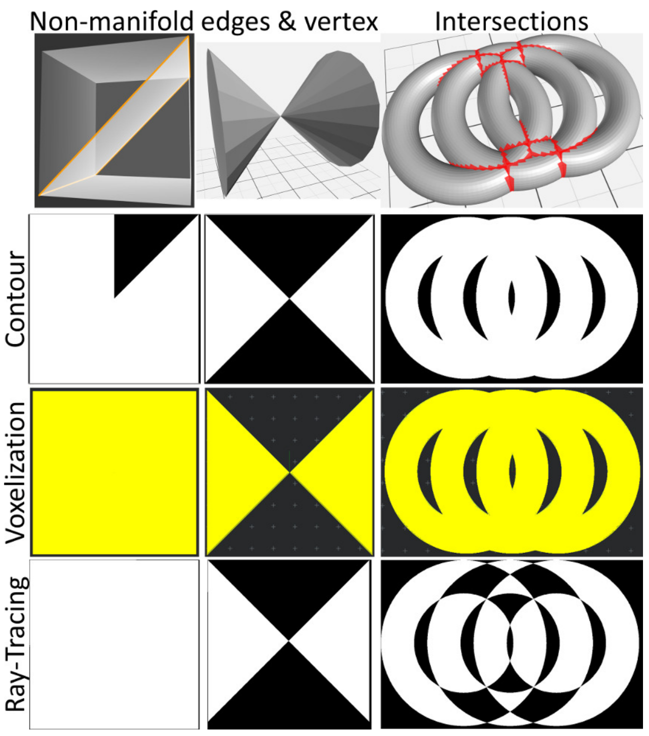

Fig. 2. Three models are designed to test on non-manifolds and intersections. Left-to-right: Box with an extra internal surface; Two cones sharing a same vertex; Three tori are put as an assembly. The slice images are taken at the layers of $800^{th}/1600$, $764^{th}/1527$, $145^{th}/290$, respectively.

3.1 Non-Manifold and Intersection

There are parts that look good at first, but create problems in slicing. These parts do not have missing data or wrong normal, and the slicing process can be completed normally. However, the results are usually not what the designer intends. In such cases, the problems could be due to the topological errors, i.e., non-manifolds and intersections. Non-manifold is a geometry which cannot exist in the real world, and that is why it is a mesh defect for 3D printing. There are two common causes of non-manifold geometry. First, the edges of an intact surface should connect to exactly two faces, so an edge is non-manifold when there are more than two faces connected to it. Edge with only one face will be discussed in the section of hole defect. "Non-manifold edge" would be found if an extra surface is defined in the interior of the model. One example is the box shown in Fig.2(left). Second, when there is some area with no thickness, it degenerates into a point, which is the case of "non-manifold vertex". It can also be seen as the vertex is shared by more than one body, as shown in Fig.2(middle). "Intersections" occur when two surfaces overlap or cross each other. This error is common when multiple bodies are occupying the same space, but they are not combined into a single solid, e.g., assemblies. Intersection messes up the definition of inside and outside of the solid model, and thus could lead to failures during slicing. As an example, three ring tori are put together without combining the mesh as shown in Fig.2(right) for the test.

"Non-manifold vertex" is treated like two separated bodies, so all methods do not have any problem in this defect. "Non-manifold edges" affect the mesh connectivity. As contouring method needs to reconstruct the connectivity of the segments, it could be confused by the non-manifold edges. In this test case, the contour runs in the internal face and jumps back to the exterior face, missing a portion of the square in the image. The voxelization and ray-tracing methods do not depend on the connectivity, so they are not affected, unless the extra surface form an enclosed volume inside the model, which will be similar to the shell case that will be studied in a later section. In the case of "Intersections" between different bodies, the contour method treats the bodies separately and combine them into an image, so the results are good. Noted that the overlapping is not a issue only for DLP, it could be a problem for FFF if the planned toolpaths cross over each other with extra material. The volxelization method considers all bodies at once. As the outside surfaces of any tori would stop the propagation of the method, the tori are naturally treated as a larger complete surface without affecting by the intersections, and it generates correct results. The ray-tracing method shoots rays to intersect with all bodies, and the overlapping of the bodies messes up the order of the intersection points on the rays, which confuse the in/out determination resulting in some portions of the slice images are flipped. In summary, the voxelization method seems to be the best in this category of defect. However, as non-manifolds do not exist in the real world, it is hard to say which extra surface (exterior or interior) in the case "Non-manifold edges" should be ignored is correct, so it is fair to say that the contour method is also well-performing. The implementation of ray-tracing method only considers the ray-intersected points, and it fails in the case "Intersections", which could be potentially solved by considering the normal of the points to clear the confusion.

3.2 Flipped Normal and Zero/Negative-Area Face

Each face in a mesh has a normal, which is a vector that is perpendicular to the plane of the face. The normal is often used to determine a surface's orientation, and all faces have to be oriented the same way with the right-hand rule, such that the vector is pointing outwards from the model. When the normal vector (i.e., the order of vertices) of a face is flipped, it can lead to difficulties in identifying the inside/outside of the model, and thus where to lay material. In addition, if a face has zero-area, mathematically its normal is undefined. Three models, as shown in Fig.3, are designed to test the slicing technologies in handling these normal problems. Note that, there is no connectivity defect in all three cases and the meshes are intact and complete, but only the order or the position of vertices is changed. First, a vase model with all its faces flipped is designed. In other words, the surface is completely "Inside Out", and the volume enclosed by the surface should be empty by definition. Second, a cup model is designed with only a portion of its faces being flipped, i.e., "Incoherent Normal". Third, a Moai model is designed with some "Bad Faces". The bad faces are the zero-area faces by moving some vertices onto other vertices

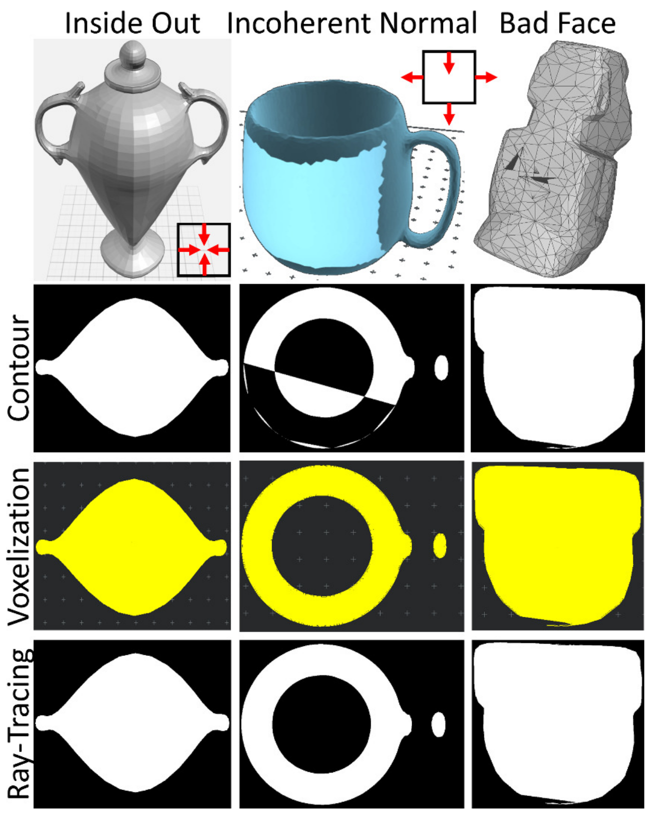

Fig. 3. Three models designed to test on flipped normal. Inside Out: all normal vectors are pointing inward. Incoherent Normal: the normal of the highlighted region is pointing oppositely from other region. Bad Face: existing some zero/negative-area faces. The slice images are taken at the layers of $1465^{th}/2559$, $204^{th}/1267$, $1156^{th}/2972$, respectively.

or edges of their associated faces. To make it even worse, some vertices are moved across other faces, such that some faces have negative area and their normal vectors point to an opposite direction from their neighbors'.

In the case of "Inside Out", all slicing technologies give the same result of having material in the enclosed area, just like the normal vectors do not matter. However, the contour method gets the wrong neighboring information in the region with "Incoherent Normal" and has the corresponding regions flipped in the images. It is not a problem for the voxelization and ray-tracing methods, as the implementations of the selected slicers do not take normal into account, but only the geometry. In the case "Bad Face", as the surface is complete, the slicers treat the shape of the surface as is and complete the slicing successfully. Even though the surface is poorly shaped, the images generated are valid. Therefore, this defect is not a problem for DLP, but it would complicate the toolpath-based processes. To summarize, if a technology does not consider normal during slicing, this category of defects is not a problem. Nevertheless, taking normal into account could have richer information that might create new possibilities (e.g., solving the "Intersections"), and it would be worth to explore some ways to consider normal but not suffer these defects.

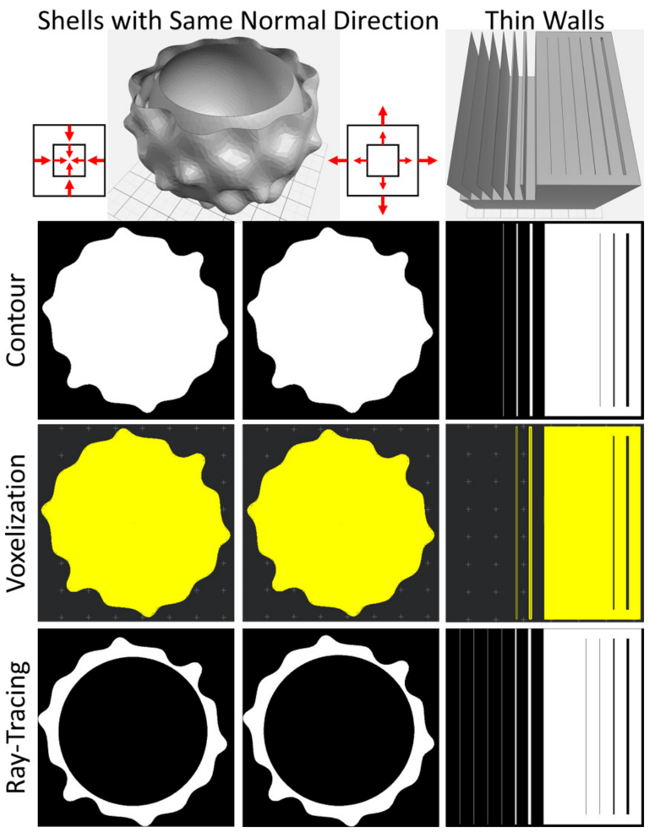

Fig. 4. Three cases designed to test on shells and thin walls. Two cases of shell are using the same model, one with all normal vectors pointing inward, another one all pointing outward. Thin walls different thickness are designed, and similarly for the cavities. All the slice images are taken at the layer of $800^{th}/1400$.

3.3 Shell and Thin Wall

A shell is a hollow model that is formed by disconnected (exterior and interior) surfaces to define the internal voids of the model. To generate correct slices, the volume of the interior surface needs to be subtracted from that of the exterior surface, which is mathematically defined as the Boolean operation – Difference. In principle, to define inside and outside of the model properly, the exterior surface should have its normal vectors pointing outward and the normal vectors of the interior surface should point inward. Otherwise, the actual volume of the model is not clear to the slicer, and the slicing result may not be the one intended. Another problem is when a shell gets too small, it becomes redundant and serves little purpose as the slicer might even not able to handle it. Therefore, three cases are designed to test for the shell defects as shown in Fig.4. The first two cases use the same "Shell with Same Normal Direction" (a bumpy sphere enclosing a sphere), but both pointing inward for the first case, and both pointing outward for the second. For the third case, different "Thin Walls" are designed with the thickness of $1mm$, $500\mu m$, $100\mu m$, $50\mu m$, $10\mu m$, and $5\mu m$. They are located on the left half of the model. The same setting is also designed to create thin cavities on the right half.

Similar to the case of "Inside Out", all methods do not have a problem dealing with the normal vectors of the exterior surface is flipped. However, because the exterior and interior surfaces are having the same direction, the contour method has done a union Boolean operation rather than a difference operation in both first two cases. Although the voxelization method is not affected by normal vectors, as it propagates to identify the region outside the model, it fails to reach the enclosed volume. The results are the same for both cases, missing the internal void. Even though both the contour and voxelization methods give the same result, it should be noted that they are due to different reasons, and the contour method can produce a hollow result if correct normal directions are given (tested but not shown in this paper), but the voxelization method still misses the internal void. In contrast, the ray-tracing method bases the in/out determination on the intersection points of the rays, and the changes in normal directions have no effect on the results. For the third case "Thin Walls", $50\mu m$ is used for the voxel and pixel sizes, and the one of slice image is $100\mu m$. The contour and ray-tracing method intersect the model with slice planes and rays, so they would not miss the tiny features theoretically. However, they have different slice images. The contour method stops to present the features under $100\mu m$, while the ray-tracing method presents those walls with one-pixel width. The voxelization method can only reconstruct the feature as small as $500\mu m$, due to the reason that small features cannot stop the propagation in identifying the regions outside the model. In summary, this category of defects is indeed an issue for the voxelization method, although the shell problem could be potentially solved by an implementation of multi-source propagation (could create problems in the case "Intersection"). The other two methods can successfully locate the shells and the thin walls. Depending on the intention, it is really up to the user to determine whether a union or a difference Boolean operation should be performed on shells with same normal direction and whether the walls that have a thickness smaller than one-pixel size should be printed.

3.4 Hole and Incomplete Surface

To properly define a solid object, the digital model needs to be watertight. Watertight means that the mesh of the surface is complete, the lines of the mesh create valid elements, and the elements are properly connected together at the edges so that the volume is fully enclosed. When there are holes or missing data on the surface, the model does not represent a closed volume. Imagine if this model is filled with water inside, the water would leak and thus it is not watertight. This occurs when some mesh edges are connected to only one face, or the adjacent faces fail to share two common vertices. The printer could not print the design correctly, since the model does not have a valid solid volume, so watertight is normally a requirement of slicing for 3D printing. Fortunately, the modern technologies can still do the slicing on non-watertight models under certain assumptions. For example, if there is a hole in a slice plane, then the slicing algorithm could connect the points of the void with a linear line. Nevertheless, it would alter the original design, and it is hard to predict how the missing data will be interpreted in

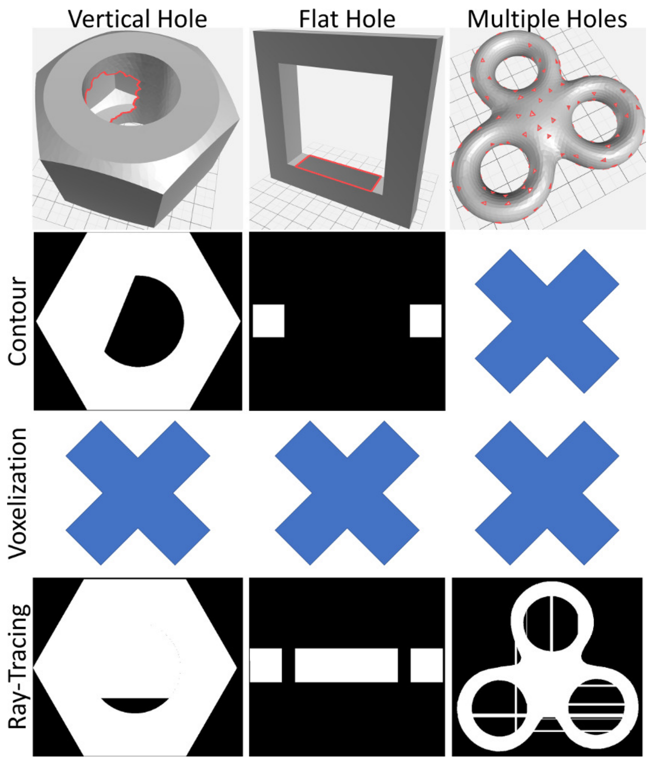

Fig. 5. Three models designed to test on holes. Left-to-right: Nut with a vertical hole on the inner surface (along the print direction); Square-torus with a flat hole perpendicular to the print direction; Triple-torus with multiple small holes. The slice images are taken at the layers of $630^{th}/1109$, $59^{th}/2007$, $60^{th}/301$, respectively.

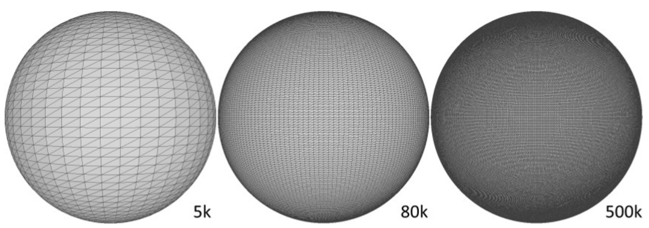

Fig. 6. Spheres of diameter $80mm$ with $5k$, $80k$ and $500k$ faces are used for computational test.

Table 1. Statatics of computation time. s: sec, m: min, h: hour.

Method	**Model**	Layer Thickness		
	(#Face)	$100\mu m$	$50\mu m$	$20\mu m$
Contour	5k	50s	1.5m	3.8m
	80k	1m	1.7m	4.1m
	500k	1.3m	2.5m	6m
Voxelization	5k	**17s**	1.2m	>7h
	80k	**19s**	1.2m	>7h
	500k	**23s**	1.2m	>7h
Ray-Tracing	5k	27s	**50s**	**2m**
	80k	24s	**47s**	**2m**
	500k	24s	**49s**	**2.1m**

different situations or even at different heights.

To understand how different slicing technologies handle the non-watertight models, three test cases in Fig.5 are used for the experiment: (1) "Vertical Hole" along the printing direction in a nut model, (2) "Flat hole" perpendicular to the printing direction in a square-torus, and (3) "Multiple Holes" on a triple-tori. In this experiment, there are slicing errors for the contour and the voxelization methods. Due to the fact that the voxelization method propagates to identify the in/out volume, a surface with any holes would result in empty volume, and thus none of the test cases get any images except black. The contour method computes the face-plane intersections and connects the segments based on closest-point searching. As expected, the hole in a slice plane is connected by a linear line, which can be seen in the case of "Vertical Hole". The "Flat Hole" is located and affects only one slice plane for the contour method, and all other images are nicely generated. However, when there are "Multiple Holes", the topology of segments becomes very complicated, resulting a fatal error causing the contour program exited without any images generated. The ray-tracing method can successfully generate images in all three test cases, but many wrong in/out determinations occur along the rays (e.g., x-axis for the vertical hole, and z-axis for the flat), due to the missing intersection points on the holes. In summary, the hole defects resulting non-watertight models are very challenging to handle for all slicing technologies. Among which, the ray-tracing method is the most robust one in handling these defects, but it still doesn't produce acceptable results.

4 Computational Resources

Besides the performance in handling different defects, another factor to consider in slicing technologies is the consumption of computational resources. Two resources are studied: one is the computation time – the time it takes to complete the slicing operation, another one is memory space – the amount of Random-Access Memory (RAM) needed while slicing. The tests in this section are done on a Workstation with 64-bit Win-10 operation system, Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, 8.00 GB RAM, and a graphics card – NVIDIA Quadro K620. To compare the computational side of the slicing technologies, a $80mm$-diameter sphere is designed with different meshes, which have $5k$, $80k$, and $500k$ faces as shown in Fig.6. Alongside with the three mesh sizes, three layer thicknesses are tested: $100\mu m$, $50\mu m$, and $20\mu m$. The time statistic is reported first, and it is followed by the memory.

4.1 Computation Time

The experiments are conducted using the online-available software by importing the designed STL files. With the limited access to the source code, the only way to measure the computation time is to count by a stopwatch from the time clicking the button to start slicing till the process ends. There might be other overhead time in between, and thus the results might not be exactly the time used for slicing alone. However, as their differences are not at a level of a few seconds, it is good enough to provide a mean of comparison, and the overhead or the measurement error is negligible in this test.

With three mesh sizes, three layer thicknesses, and three slicing technologies to be compared, there are in total 27 combinations, and the statistics of computation time are listed in Table 1. It is obvious that the layer thickness is a parameter for all the methods, and it is because the smaller the thickness, the more slice images need to be generated, regardless of how the in/out determination is done. Yet, the relationships between the layer thickness and different methods are different. As a geometric approach, increasing the resolution simply means to increase the number of slices for the contour method, and thus its computation time increases pretty linear with the decrease of layer thickness. Both the voxelization and the ray-tracing methods are the volumetric techniques, but the former is based on voxels – a 3D array, while the latter is based on rays shooting out from 2D images. The voxel resolution needs to be higher in order to have a smaller layer thickness, and thus the amount of data needs to be processed in the voxelization method increases exponentially with the resolution. With the layer thickness $20\mu m$, the data is too large to fit into a computer's main memory at one time and out-of-core process is needed resulting an extremely long computation time. In principle, a ray stacked with intersection points can generate an infinite number of layers without the need of any additional data. Therefore, the time complexity is also linear with the resolution for the ray-tracing method similar to the contour method.

On the other hand, the mesh size is not always a key parameter for different slicing technologies in terms of computation time. As the contour method computes the face-plane intersection on each face, the increasing trend of computation time can be seen with the mesh size. However, the voxelization method uses the mesh as a stopping condition of the propagation, and the ray-tracing method intersects the rays with the mesh through rendering. Although they are not totally independent of the mesh size, the primary factors are the voxels and the rays. Therefore, it is demonstrated by the results that there is not much variation among different mesh sizes for both the voxelization and the ray-tracing methods. The voxelization method tends to be faster than the ray-tracing method in low resolution, because the voxel is a direct access data structure, compared with the list data structure in a ray. That being said, there is a trade-off between time and memory, which is discussed in the following section.

4.2 Memory Space

For the same aforementioned reason, it is not possible to measure the actual memory consumed in each slicing technology. An alternative way is to use the Task Manager in Windows to monitor manually the memory usage during the slicing process. The memory used by the corresponding process is recorded before the slicing, and the highest value of the memory is also recorded during the slicing process. The difference between the two values is taken as the memory needed for each technology. Again, there might be memory used for purposes other than the slicing itself, but the results are distinct enough from each other.

Table 2. Statistics of memory usage. 1 GB = 1024 MB.

Method	**Model**	Layer Thickness		
	(#Face)	$100\mu m$	$50\mu m$	$20\mu m$
Contour	5k	30 MB	30 MB	40 MB
	80k	40 MB	50 MB	80 MB
	500k	120 MB	130 MB	190 MB
Voxelization	5k	900 MB	4 GB	6 GB
	80k	900 MB	4 GB	6 GB
	500k	900 MB	4 GB	6 GB
Ray-Tracing	5k	170 MB	170 MB	170 MB
	80k	190 MB	190 MB	190 MB
	500k	200 MB	200 MB	200 MB

Among all, the voxelization method takes up significantly much more memory than other methods, which is not a surprise. For a sphere with diameter $80mm$, the dimension of the 3D array (voxels) with size $100\mu m$ is $800 \times 800 \times 800$, which in turns equal to 512 MB memory for 1 byte per voxel (1 MB = 1×10^6 bytes). It grows exponentially to 4 GB and 64 GB (1 GB = 1×10^9 bytes) when the size is $50\mu m$ and $20\mu m$, respectively. Subtracting the memory reserved for other Windows processes from the computer's total memory 8 GB, the maximum available memory for the slicing process is about 6 GB. It is far not enough for the voxelization method to deal with the 64 GB data, and thus extensive data exchange between RAM and hard-drive is done to complete the process, which takes more than 7 hours as mentioned in the previous section. Incorporating an octree data structure or rectangular voxels in the implementation could reduce the memory usage, but the problem would not be completely solved, especially the 3D printing parts (e.g., lattice structure) are getting more and more complex externally and internally nowadays.

The contour and the ray-tracing methods compute intersections on-the-fly. Although they store the intersection results, those do not take up much memory and require only less than 200 MB in all cases. The contour method generates the segments by intersecting the mesh faces with each slice plane, and thus the total number of segments depends on the mesh size, the structural complexity, and the number of layers. It is supported by the results that with the increase in mesh size or resolution, the memory usage also increases. The ray-tracing method in general requires more memory than the contour method, because it needs to generate images with the rays covering the whole volume of the model, while the contour method needs only the surface. However, because of the same reason, the memory usage of the ray-tracing method is more controllable and it does not depend on the structural complexity. The memory usage slightly increases with the mesh size as more memory is used to render a larger mesh. Recall that a ray can theoretically generate infinite number of layers without any additional information, so there are no obvious changes in memory usage with the smaller layer thickness.

5 Conclusion

To benchmark the slicing technologies for Digital-Light-Processing (DLP) printing that uses images as fabrication tool, three implementations: contour, voxelization, and ray-tracing, are tested in this paper. A number of common defects found in STL files are designed for the testing, and they are also put online available to facilitate further development and testing of slicing technology. In a nutshell, the three technologies have their own characteristics. As a geometric approach, the contour method fails in handling non-manifolds and incoherent normal. The voxelization method makes use of a 3D array and determine in/out for each voxel, so that it can process models with topology and normal problems, but it suffers from the defects of shells and thin wall. The ray-tracing method shoots rays that penetrate the model and thus has no problem in locating shells or small features, but the order of intersection points in the rays are swapped when there are intersections between different bodies. As each of them has its own advantages and disadvantages, one possible solution is to take a step of error detection and decide which slicing technology to apply. Unfortunately, they all have problems in slicing models with holes. Therefore, it is highly suggested to fill in the holes before slicing.

That being said, Table 3 summarizes the defects with the level of difficulty to repair them. The suggested way of repairing is also described in the table. Some defects could be fixed easily or even automatically, such as those related to normal vectors, but some require extensive user intention like removing extra faces of the non-manifolds. Small holes are pretty simple to fill as the geometry is very much defined, but large holes are challenging, especially when there are complete surfaces missing from a part of questionable data. By linking the difficulty level in repairing and the capability of each slicing technology in handling the defects, it could draw some means of suitability and ways in applying different slicing technologies. The contour method was originally developed for the methods based on toolpath, which requires the least memory, but it is actually indirect to compute the contours first and then convert to images for DLP printing. The shell problem is an important issue for the voxelization method. While multi-source propagation is a possible solution, it could create other problems, which requires further testing to validate different implementations of the voxelization method. Moreover, fine resolution is normally needed for 3D printing, so the high memory usage of voxels should also be addressed. The ray-tracing method is the fastest in most cases, and it needs a moderate amount of memory (200 MB for a model with 500k faces). It looks to be a good balance between computation time and memory space. It would be optimal if the intersection problem can also be handled without creating other problems.

Table 3. Level of difficulty (L) to repair: 1 (easiest) to 3 (hardest).

Defects	**L**	**How to repair**
Non-manifold	3	Users carefully select and delete extra faces or reconnect mesh properly
Intersection	2	Apply union Boolean operation, but may require further user attention
Inside Out	1	Flip all normal vectors
Incoherent Normal	1	Automatically flip the normal vectors to be coherent with their neighbors
Bad Face	3	Users remove bad and add new faces
Shell (same normal)	2	Users select the shell with wrong normal direction and flip all its normal vectors
Thin Wall	1	Not a defect, but users shouldn't design features smaller than print resolution
Large Hole	2	Apply hole filling, smoothing, and fairing operations
Small Hole	1	Apply hole filling operation

Acknowledgements

This paper acknowledges the support of the Natural Sciences & Engineering Research Council of Canada (NSERC) grant #RGPIN-2017-06707.

References

[1] Matte, C.-D., Pearson, M., Trottier-Cournoyer, F., Dafoe, A., and Kwok, T.-H., 2018. "Multi-material digital light processing printer with material tower and spray cleaning". In ASME 13th International Manufacturing Science and Engineering Conference, Vol. 4, p. V004T03A063.

[2] Livesu, M., Ellero, S., Martnez, J., Lefebvre, S., and Attene, M., 2017. "From 3d models to 3d prints: an overview of the processing pipeline". *Comput. Graph. Forum,* ***36***(2), pp. 537–564.

[3] Gibson, I., Rosen, D. W., and Stucker, B., 2009. *Additive Manufacturing Technologies: Rapid Prototyping to Direct Digital Manufacturing*, 1st ed. Springer Publishing Company, Incorporated.

[4] Zhang, Z., and Joshi, S., 2015. "An improved slicing algorithm with efficient contour construction using stl files". *The International Journal of Advanced Manufacturing Technology,* ***80***(5), Sep, pp. 1347–1362.

[5] Wang, Q., Yang, P., Han, W., Wei, Q., and Wang, N., 2016. "A new slicing method for amf model with topology structure". In 13th International Conference on Embedded Software and Systems (ICESS), pp. 125–130.

[6] Minetto, R., Volpato, N., Stolfi, J., Gregori, R. M., and da Silva, M. V., 2017. "An optimal algorithm for 3d triangle mesh slicing". *Comput. Aided Des.,* ***92***, pp. 1 – 10.

[7] Chakraborty, D., and Choudhury, A. R., 2007. "A semi-analytic approach for direct slicing of free form surfaces for layered manufacturing". *Rapid Prototyping Journal,* ***13***(4), pp. 256–264.

[8] Kwok, T. H., Ye, H., Chen, Y., Zhou, C., and Xu, W., 2017. "Mass customization: Reuse of digital slicing for additive manufacturing". *ASME. J. Comput. Inf. Sci. Eng.,* ***17***(2), pp. 021009–8.

[9] Wang, W., Chao, H., Tong, J., Yang, Z., Tong, X., Li, H., Liu, X., and Liu, L., 2015. "Saliency-preserving slicing optimization for effective 3D printing". *Comput. Graph. Forum*.

[10] Zhao, H., Gu, F., Huang, Q.-X., Garcia, J., Chen, Y., Tu, C., Benes, B., Zhang, H., Cohen-Or, D., and Chen, B., 2016. "Connected fermat spirals for layered fabrication". *ACM Trans. Graph.,* ***35***(4), July, pp. 100:1–100:10.

[11] Jin, Y., He, Y., and Du, J., 2017. "A novel path planning methodology for extrusion-based additive manufacturing of thin-walled parts". *Int. J. Comput. Integr. Manuf.,* ***30***(12), pp. 1301–1315.

[12] Ahsan, A., Xie, R., and Khoda, B., 2018. "Heterogeneous topology design and voxel-based bio-printing". *Rapid Prototyping Journal,* ***24***(7), pp. 1142–1154.

[13] Sun, S. H., Chiang, H. W., and Lee, M. I., 2007. "Adaptive direct slicing of a commercial cad model for use in rapid prototyping". *The International Journal of Advanced Manufacturing Technology,* ***34***(7), Oct, pp. 689–701.

[14] Qiu, Y., Zhou, X., and Qian, X., 2011. "Direct slicing of cloud data with guaranteed topology for rapid prototyping". *The International Journal of Advanced Manufacturing Technology,* ***53***(1), Mar, pp. 255–265.

[15] Yang, P., and Li, K amd Qian, X., 2011. "Topologically enhanced slicing of mls surfaces". *ASME. J. Comput. Inf. Sci. Eng.,* ***11***(3), pp. 031003–9.

[16] Steuben, J. C., Iliopoulos, A. P., and Michopoulos, J. G., 2016. "Implicit slicing for functionally tailored additive manufacturing". *Comput. Aided Des.,* ***77***, pp. 107 – 119.

[17] Adams, D., and Turner, C. J., 2018. "An implicit slicing method for additive manufacturing processes". *Virtual and Physical Prototyping,* ***13***(1), pp. 2–7.

[18] Keeter, M., 2016. Formlabs DLP slicer. https://www.mattkeeter.com/projects/dlp.

[19] Wang, C. C. L., Leung, Y.-S., and Chen, Y., 2010. "Solid modeling of polyhedral objects by layered depth-normal images on the GPU". *Comput. Aided Des.,* ***42***(6), pp. 535 – 544.

[20] Zeng, L., Lai, L. M.-L., Qi, D., Lai, Y.-H., and Yuen, M. M.-F., 2011. "Efficient slicing procedure based on adaptive layer depth normal image". *Comput. Aided Des.,* ***43***(12), pp. 1577 – 1586.

[21] Huang, P., Wang, C. C. L., and Chen, Y., 2013. "Intersection-free and topologically faithful slicing of implicit solid". *ASME. J. Comput. Inf. Sci. Eng.,* ***13***(2), pp. 021009–13.

[22] Chen, Y., and Wang, C. C. L., 2013. "Regulating complex geometries using layered depthnormal images for rapid prototyping and manufacturing". *Rapid Prototyping Journal,* ***19***(4), pp. 253–268.

[23] Chen, Y., and Wang, C. C. L., 2011. "Uniform offsetting of polygonal model based on layered depth-normal images". *Comput. Aided Des.,* ***43***(1), pp. 31 – 46.

[24] Mao, H., Kwok, T.-H., Chen, Y., and Wang, C. C. L., 2018. "Adaptive slicing based on efficient profile analysis". *Comput. Aided Des.*

[25] Feng, J., Fu, J., Lin, Z., Shang, C., and Niu, X., 2019. "Layered infill area generation from triply periodic minimal surfaces for additive manufacturing". *Comput. Aided Des.,* ***107***, pp. 50 – 63.

[26] Yaman, U., Butt, N., Sacks, E., and Hoffmann, C., 2016. "Slice coherence in a query-based architecture for 3d heterogeneous printing". *Comput. Aided Des.,* ***75***(C), June, pp. 27–38.

[27] Zhang, Z., and Joshi, S., 2017. "Slice data representation and format for multi-material objects for additive manufacturing processes". *Rapid Prototyping Journal,* ***23***(1), pp. 149–161.

[28] Sass, L. R., Khani, M., Natividad, G. C., Tubbs, R. S., Baledent, O., and Martin, B. A., 2017. "A 3d subject-specific model of the spinal subarachnoid space with anatomically realistic ventral and dorsal spinal cord nerve rootlets". *Fluids and Barriers of the CNS,* ***14***(1), Dec, p. 36.

[29] SprintRay, 2018. Rayware version 1.4.1. https://sprintray.com/software.

[30] ZMorph, 2018. Voxelizer 2. https://voxelizer.com.

[31] Leung, Y.-S., 2014. LDNI-based solid modeling. http://ldnibasedsolidmodeling.sourceforge.net.

[32] Varotsis, A. B. Understand and fix common STL file errors. https://www.3dhubs.com/knowledge-base/fixing-most-common-stl-file-errors.