Single Path Generation for Closed Contours via Graph Theory and Topological Hierarchy

Michael Borish*, Alex Roschli*, Charles Wade[1], Brian Post*, Liam White*, Cameron Adkins*

*Manufacturing Demonstration Facility, Oak Ridge National Laboratory, Knoxville, TN 37932
[1]University of Colorado Boulder, Boulder, CO 80309

Abstract

Slicing converts a 3D object into a set of 2D polygons that are filled with multiple path types. These paths involve travels where the extruder of the machine must stop building, lift, travel to the next path, lower, and resume construction. Travels are considered wasted time as construction of the object is not occurring. Further, the start/stop point, called a seam, causes both reduced aesthetic and weaker material properties. To address these issues, an algorithmic approach was developed to compute a continuous single path from closed contours. The algorithm utilizes graph theory and a topological hierarchy to produce a single path for an individual layer. This approach can be combined with spiralization techniques to compute a single path for entire objects. The resulting objects can be constructed quicker and have improved material properties as verified via tensile testing.

Keywords: slicing, additive manufacturing, spiralization, continuous fiber deposition, single path, graph theory, topological hierarchy

Introduction

The standard workflow for constructing an object with an additive manufacturing (AM) machine is to design a 3D object, place that object in some type of slicing software, have the slicing software create g-code, and then upload the g-code to the machine for printing.

Slicing is a necessary step in this workflow and is generally a purely geometric approach [1, 2]. Before slicing can begin, an object is typically converted from CAD (computer aided design) to an STL (stereolithography) file. An STL represents all surface geometry using triangles [3]. This STL is then intersected by a horizontal plane at incremental heights corresponding to the preferred layer height of construction on the AM machine. These intersections create multiple 2D polygons that must be filled with pathing. Typical pathing styles include perimeters, insets, skins, and infill. These styles can be generally classified as closed contour or open loop pathing. These paths are modified by numerous user settings, but commonly begin with perimeters and insets. Perimeters and insets are closed-loop paths generated by offsetting the boundary of the polygon by the bead width of the machine. Once these paths have been created, skins and infill are used to fill the remaining void using one of many pattern styles. Regardless of the type and style of paths, many of those paths are disconnected from one another. As a result, travels are added as part of the slicing process.

Travels connect various build paths. In terms of a standard MEX (Material Extrusion) process, a travel involves the machine stopping material flow, lifting the nozzle, moving to the next build path, lowering the nozzle, and resuming material flow. Because this motion does not include deposition, travels are often viewed as wasted motions that do not contribute to the construction of an object. The ideal path then would not involve any travels and would result in a lower construction time compared to the same object whose build paths included travels. Additionally, these travels create start and stop points of material flow. The start and stop point, called a

seam, creates a non-uniformity on the surface of the object. This seam, besides being aesthetically unpleasing, creates a material weakness in the resultant object [4]. An example of such a seam is shown in Fig. 1.

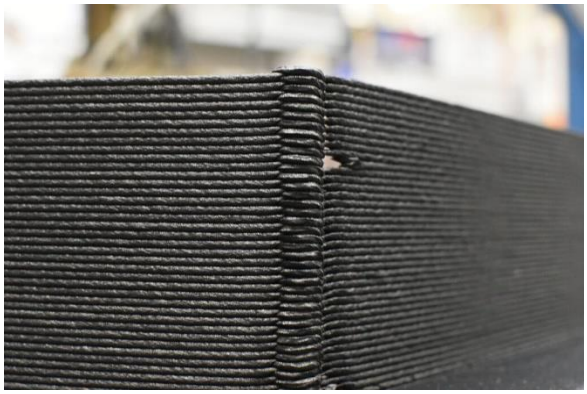

Figure 1: Example of a seam - a point of structural weakness and reduced aesthetic quality.

To tackle the presence of travels and resultant seams, researchers at Oak Ridge National Laboratory developed a single path algorithm. This algorithm focuses on converting the closed-loop contours of perimeters and insets into a single path without travels using graph theory combined with a topological hierarchy. These layer-based single paths were then augmented using spiralization techniques. Spiralization is the conversion of a 2D path to a 3D path via interpolation of the z-value. To validate this new path planning algorithm, the researchers constructed several samples and tested mechanical properties of the object both with and without the single path algorithm.

Related Work

As will be shown, two advantages of this algorithm are the ability to construct a part quicker and have the resultant part be stronger. This algorithm is not the first to attempt to optimize the AM process to such an effect. Indeed, much research has been devoted to hardware solutions. Work by McQueen et al. and Ali et al. are but two examples [5, 6]. In this work, multiple effectors or nozzles were investigated to speed up the construction process. Besides hardware approaches, research has also investigated software approaches.

Ding et al. proposed multiple types of adaptive pathing adjustments for wire arc metal AM [7, 8]. These adaptive adjustments were meant to improve process stability and the material properties of the resultant object. Similarly, Liu et al developed single paths based on raster style patterns [9]. This approach focused on correcting sharp angles in the resultant pathing. Other approaches such as the work done by Michel et al. illustrates path planning that is not purely geometric [10]. This work focused on splitting 2D polygons into regions over which specific process parameters could be applied. In this way, the pathing could be adapted to the appropriate local conditions for optimal construction.

Work has also been conducted on path planning for extrusion-based processes. In work by Jin, He, & Du, a new type of “wavy” path design was developed to address issues with the build process [11]. In work by Fok et al. and Ganganath et al., alterations to the path planning were based around heuristics [12, 13]. In this work, heuristics were proposed to obtain good results with the traveling postman algorithm while attempting to reduce the overall computation complexity. There are also specific examples of extrusion-based single path approaches. One example is work done by Xia et al. where the authors attempt to generate single path solutions using a raster pattern [14]. Similarly, work by Guang et al. used graph theory to develop single paths for scaffolding primarily dealing with various infill patterns [15]. Dreifus et al. attempted to use graph theory to solve for single pathing through hexagonal infill patterns [16].

This graph-based work has also been combined with topological information. A relevant example is Choi and Cheung who applied a simple topological hierarchy to their pathing approach for multi-material construction

[17]. This work builds upon these approaches by applying graph theory and topological information in a similar way to construct single paths. However, unlike these approaches, the representation in this work is unique and rather than produce entirely new pathing, modifies existing pathing. As a result, this single path algorithm can be thought of as an optimization strategy rather than a pure path planning approach.

Finally, in comparison to the other approaches and algorithms already discussed. Research has also investigated manual approaches based on object design. One example is work done by Roschli et al. [18]. In this work, the authors discuss designing objects with very small cuts in appropriate locations to "trick" standard slicing software into creating a single path. This previous work relates directly to the algorithm presented here as the algorithm is meant to provide an automated solution to this manual design process.

Algorithm Implementation

The first major step in a slicing process is to cross-section the object. The cross-section represents one of the multiple layers that will be constructed to reproduce the object. The boundary of this cross-section is called a perimeter and is a closed contour defining a 2D polygon. This perimeter is then offset towards the center of the object by some value, usually the bead width of the AM machine the object will be constructed on. Each successive offset fills in more of the polygon and is called an inset. Ultimately, this creates a set of closed contours as shown in Figure 2.

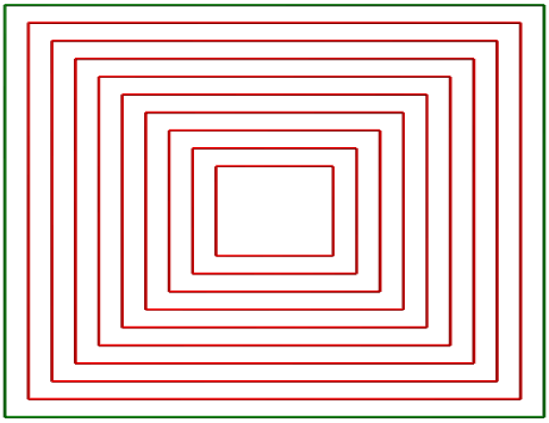

Figure 2: Typical set of closed contours in a cross-section. The green contour represents the perimeter while the red contours are insets.

Once this is complete, the typical slicing process would then insert travels between the contours. The construction behavior would thus be to travel to a contour, construct it, travel to the next contour, construct it, and so on. The single path algorithm proposed in this paper functions as an optimization to the typically generated path before an evaluation for travels takes place. The only requirement of the algorithm is that the pathing be composed of closed contours. While the focus will be on perimeters and insets due to the heavy use of such pathing, the approach is equally valid for any set of closed-loop contours such as concentric skins or infill.

Once the typical contours for the pathing have been generated, this algorithm inserts bridges between two contours. A bridge is a modification to the pathing to allow a move from one contour to another. Ultimately, this bridge allows a return to the original contour as well and allows two contours to be merged to form a single contour. An example of a simple object, its original contours, and the modified contours with bridges is shown in Figure 3.

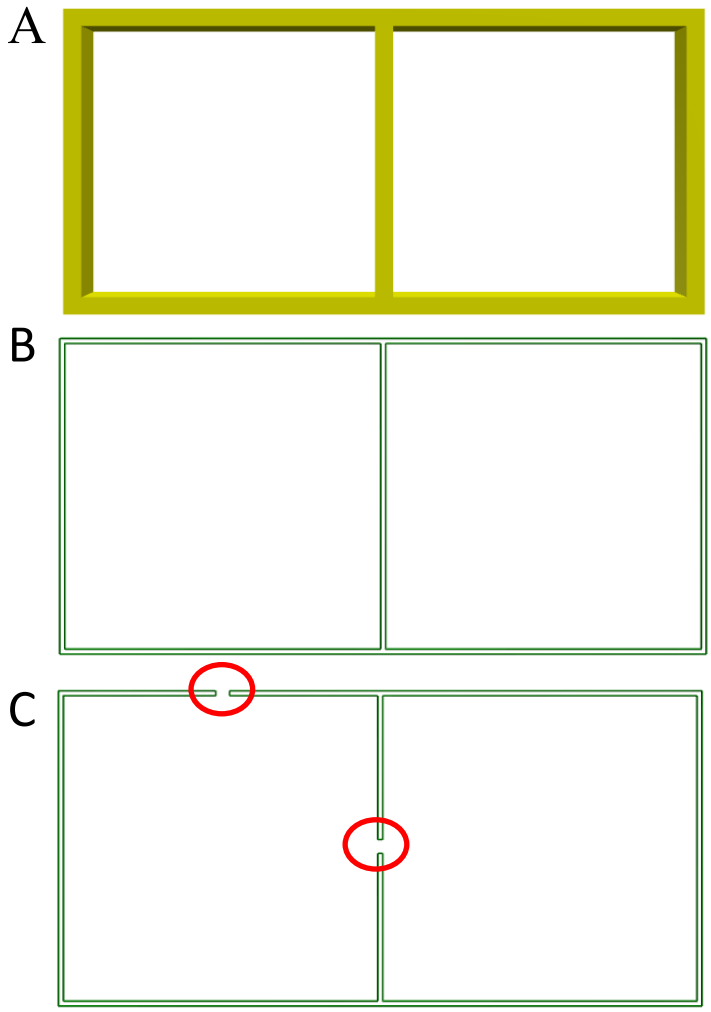

Figure 3: 3A) Original STL, 3B) Traditional pathing using a single outer perimeter with two smaller inner perimeters (green), 3C) Pathing with single path optimization applied (green) and bridges highlighted (red)

It is important to note in Figure 3B and 3C as well as others that the pathing is represented by the centerline. This means the pathing has physical width and despite the appearance of gaps, the pathing for this rectangular object is solid. Further, the gaps of the bridges that have been inserted into Figure 3C are based on the bead width and are solid as well. The use of these bridges to construct a single path does not affect an object in terms of fill.

To convert a set of contours into a single path, the algorithm utilizes graph theory. This allows the problem to be broken down into two major phases: graph generation and graph touring. The first phase, graph generation, concerns the conversion of the various path segments in the contours into node/edge representations in an undirected graph. The second phase, graph touring, concerns the choice of how to walk the graph to create the desired single path. In addition to the construction and evaluation of the graph, a simple topological hierarchy is necessary to guide the directionality of the contours. Each of these steps will be discussed in detail as well as practical limitations and extensions applied to and after this algorithm has been completed as part of the slicing process.

Graph Generation

The graph representation can be supplied by numerous software libraries and this implementation uses the Boost Graph library [19]. Once the slicing process has cross-sectioned an object, the slicing moves to the

generation of closed contours via perimeters and insets as described in the last section. With the contours created, the algorithm begins by evaluating all contour edges for possible bridge locations. Candidate edges are those edges between two contours that are spatially close. For the current implementation, the spatial limit was set to one bead's width. This guarantees that if a segment of a contour is selected to host a bridge, the resulting geometry will still be solid. Each candidate location for a bridge on the segment must also be at least two bead's width long. Two bead's width of length is the minimum size of a bridge as the algorithm must connect one contour to another and return to the original contour to create a single path.

These candidate locations are also checked to make sure that they are a minimum distance away from the vertices that comprise the individual line segments. This ensures that bridges will not be placed around the corners of a contour or straddling the beginning/ending of two segments. There is no limitation on the number of bridges a particular edge can host. For example, a particularly long segment may host multiple bridges to smaller, interior contours. This means that each segment will create a list of candidate locations for bridges. Figure 4 shows a simple evaluation of contours with the candidate bridge locations highlighted in yellow.

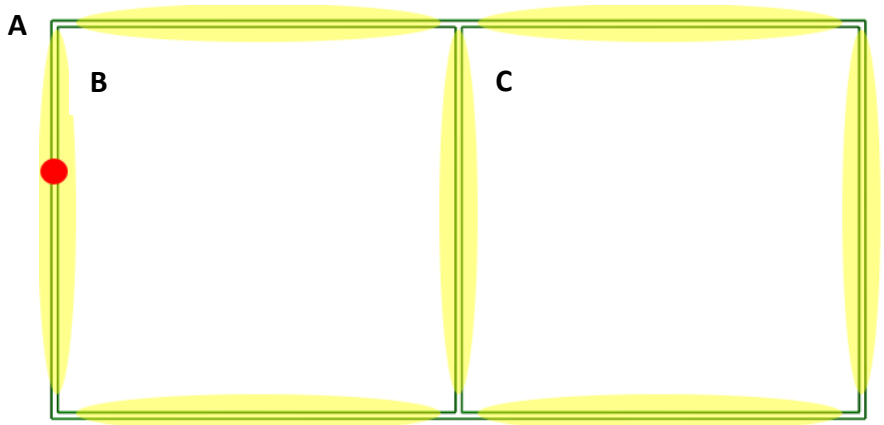

Figure 4: Contour evaluation for outer, rectangular contour A, left inner square contour B, and right inner square contour C. The yellow highlighted areas show potential bridge locations. The red dot represents the location of a single bridge that is two bead's width in size. Each highlighted section then will generate numerous potential bridges that will be inserted into the edges of the graph for evaluation.

With all candidate bridge locations identified along all possible segments between two contours, the information can now be transformed into a graph. Before this graph transformation occurs, however, the list of candidates can potentially be reduced due to two additional features of the algorithm. Both are related to the removal of bridge candidates based on previous layers. These practical adjustments will be described in a subsequent section as neither is strictly required to create a single path. These adjustments do however assist with practical effects of the algorithm's application and improve surface quality.

Once the candidate locations are computed, the information is placed into a weighted, undirected graph. Each vertex in the graph represents a contour while each graph edge represents the list of potential bridge locations between two contours. To be clear, each edge maintains a list of all candidate bridge locations between two contours along all geometric segments. For Figure 4 then, the edge in the graph between contours A and B would have candidate bridges from all three segments that are within the appropriate spatial distance. Similarly, for the edge between contours A and C. Contours B and C would have candidates from only one common segment from the middle of the object.

Each graph edge is also assigned a weight that could represent any type of optimization factor. This optimization factor allows a user to prioritize bridges between specific contours based on some criteria. A simple example is the edge from the graph vertex that represents the perimeter to any other graph vertex has a very high weight, while all other edges have a very low weight. This causes the algorithm to prioritize placing bridges on the interior of the object and guarantees a single bridge on the exterior if all contours can be connected. Such an optimization would improve the surface quality of an object by minimizing bridges on the

exterior perimeter and is the default behavior of the algorithm. However, this edge weight could represent anything such as stress or thermal characteristics and be substituted accordingly. A simple graph representation for Figure 4 is shown in Figure 5.

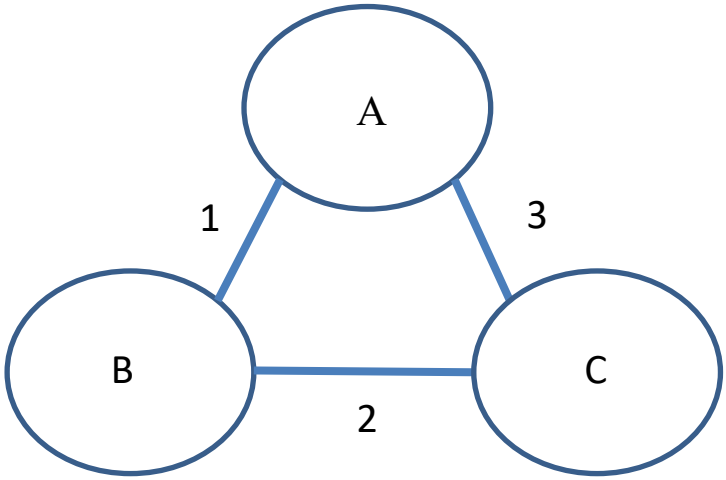

Figure 5: Graph representation of Figure 6. Vertex A represents the outer contour, while vertex B and C represent the inner left and right contours respectively. Edges 1, 2, and 3 represent the list of potential bridge locations between the respective contours. Edge 1 and 3 contains candidates from three segments, while edge 2 represents candidates from a single common segment.

Even though the graph is undirected, there is an imposed shape. As mentioned in the last section, by virtue of how the contours are generated, that is by taking the perimeter and stepping some distance in, all contours are guaranteed to never intersect. As such, all contours are either contained within one another or next to each other within some other contour. The lack of contour intersections creates a tree-like structure where a single root node represents the outermost perimeter, each level of the tree has some number of nodes representing the insets, and the tree height equals the deepest contour.

The lack of intersections also gives rise to the topological hierarchy necessary to control the winding order of the geometric vertices during the graph touring described in the next section. The winding order of vertices is the directionality of the vertices in a closed contour, either clockwise (CW) or counterclockwise (CCW). This hierarchy is composed of two relationships: parent-child and sibling. Both relationships are illustrated in the simple object in Figure 4. The parent-child relationship represents one contour contained inside another, while the sibling relationship represents two contours that are not contained inside one another. To be clear, the sibling relationship is established between two contours that ultimately reside at the same depth of the tree-like graph that will be generated. The sibling contours themselves do not need to be "spatially" close as the topological hierarchy has no bearing on whether two contours may be connected via bridges.

These two relationships give the tree-like graph its distinct shape. As parent-child contours are encountered, a new level is added to the graph, increasing its depth. Conversely, as siblings are encountered, nodes are added to the graph at the current level, increasing its breadth. As the graph is eventually toured, these relationships determine the directionality of the vertices of a contour. If a bridge is inserted that allows motion between two contours that are a parent-child relationship, the vertices of the second contour must be reversed. For sibling contours, the ordering can remain unchanged. This relationship is stored in the graph edges along with the potential candidate locations and weighting factor.

In this way, the algorithm can create a forest of trees where each tree represents a solid piece of geometry. Each solid piece of geometry is typically called an island in slicer software. If the geometry is fully connected, that is, has potential bridges between all contours, there will be a single root node and tree. If any geometry is disconnected, either being spatially distant or having insufficient locations for bridges, there will be multiple roots each with a single path. The single path algorithm then guarantees a single path only when the geometry is fully connected, otherwise, it will connect only what geometry is possible to connect. In the latter case, partially connecting the geometry still guarantees the minimal number of travels. A disconnected example is shown in Figure 6.

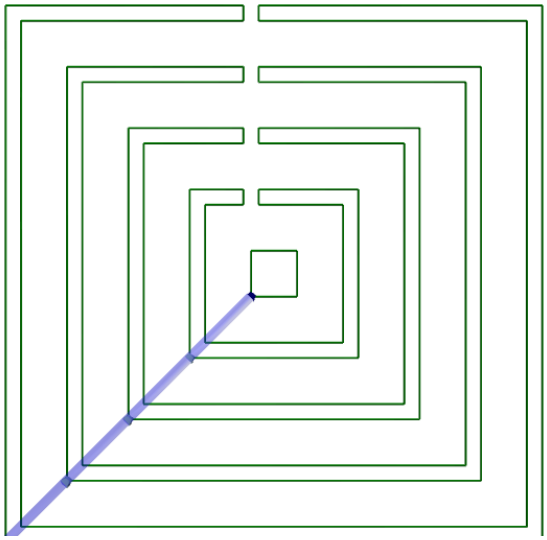

Figure 6: Example of minimal travels between multiple single paths. Except for the innermost square, all other squares consisted of two contours joined via a bridge.

Graph Touring

With this forest of trees created, the algorithm can proceed to the second phase of computation: graph touring. In this phase, an appropriate graph tour must be identified and walked thereby selecting specific bridges from among the candidates of each edge and connecting the contours. To determine an appropriate tour, Kruskal's algorithm is applied. Kruskal's algorithm is a well-known minimum (min) spanning tree algorithm [20]. A min spanning tree is a fully connected graph of minimum weight. At a high level, Kruskal's algorithm creates a minimum spanning tree by first creating a temporary set or forest composed of all the vertices of the already existing graph. Then, all the edges of the original graph are sorted and evaluated sequentially. From this sorted list, the minimum weight edge is removed and added to the temporary set. The two vertices connected by this edge are combined, and the process repeats. This process continues selecting minimally weight edges until all vertices in the temporary set are connected. This determination is made possible by comparing the connectivity of the temporary set to the original graph. At that point, the minimally weighted number of edges has been used to connect all vertices into a single tree. A simple min spanning tree for the geometry in Figure 4 is shown in Figure 7.

Recall that the edge weights in the previous phase prioritize specific edges. By default, interior contours will have low weights while the exterior perimeter will have a high weight. By weighting edges that represent internal contours lightly, these edges will be selected first by Kruskal's which places bridges on the internal contours first. This approach also lends itself to replacing those weights with other optimizations such as overlaid stress fields to minimize stress.

With the min spanning tree created, the graph can be walked using a depth first search (DFS) [21]. A DFS is possible due to the tree-like structure imposed on the graph by virtue of the way the contours are created. Just like Kruskal's algorithm, a DFS is a well-known algorithm. In this case, the DFS begins with the root node

representing a perimeter and follows graph edges until finding the deepest node representing the innermost contour. At each step in the search, the edge is evaluated, and a bridge is selected. Recall that each edge in the graph contains a list of candidate bridge locations between the two nodes that represent the contours. The choice of which bridge selected from that list can be arbitrary if connectivity is the only concern, however, as the experimental results will show, specific criteria should be applied to prioritize bridge selection. Specifically, bridges should be placed in the middle of the longest common geometric edges. Once the innermost node along the path in the graph is reached, the DFS returns to the highest node with a branch and begins again. In this way, the entire graph is evaluated.

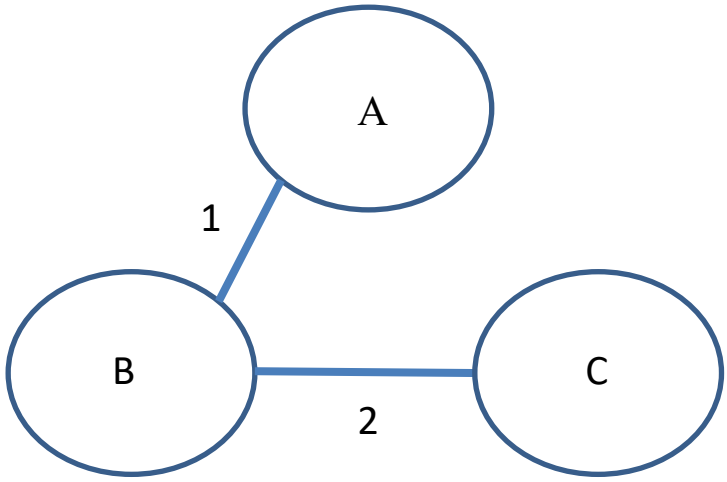

Figure 7: One possible min spanning tree. In this case, edge 3 was not needed. The minimization of the cost when computing this tree is driven by the weights in each edge.

When an edge is visited by the DFS and a bridge is selected, this bridge is removed from any other edges in the graph with connectivity to the two contours in question. This prevents a bridge location from being reused in the event it is shared among more than two contours. This process continues until all edges in the min spanning tree have been visited and bridges selected. From this point, a final recursive walk down the structure is performed. As each graph edge is visited, the appropriate geometric segment is added to the final contour. If there is no bridge on a particular geometric segment, the vertices of the segment can simply be added. However, if there is a bridge, the bridge is inserted into the final contour and the algorithm recurses into the next contour to continue.

Insertion of a bridge requires the addition of four points. At the bridge location, the first two points of the bridge are inserted into the final contour immediately. This requires breaking both the current and destination geometric segment. The other two points are held temporarily as additional recursion may require more bridges to be inserted. As mentioned previously, to keep vertices in the correct order, the topological hierarchy must be tracked. The relationship that was embedded into the graph edges is now referenced to control the CW or CCW directionality of the geometric vertices.

This recursion continues until all contours have been iterated over, and their vertices and bridges added to the final contour. As the recursion unwinds, the second of the four points are inserted for each contour in the same way that the initial two points were inserted. The single path for a particular island and its connected geometry is now complete. Naturally, if there is disconnected geometry somewhere, the algorithm then moves to the next root and the process begins again. Thus, the algorithm will provide the path of minimal travels if the geometry is disconnected or a single path if the geometry is connected. The resulting first few layers for the object presented in Figure 4 is shown in Figure 8.

Additional Implementation Details

As mentioned, when first evaluating geometric edges and identifying candidate bridge locations, the list of locations can be reduced before transforming the information into a graph. There are two ways in which the list can be reduced: previous layer exclusion and zippering. Both reductions arise from practical considerations of constructing an object using an AM process.

Previous layer exclusion simply refers to excluding potential bridge locations that were close to bridges in the most immediate layer. For example, consider the simple geometry of a cube composed of two contours. Since the geometry does not change from layer to layer, the algorithm would place bridges at the exact some location for each layer. This leads to all the bridges stacked upon on another over the course of object construction. Not only does this reduce the aesthetic quality of the object but could cause a reduction in material properties by concentrating the bridges. By excluding the most recent bridge locations from the current choices, the algorithm is forced to distribute the bridges across the object even for geometry that does not change from layer to layer.

Zippering occurs after exclusion and refers to prioritizing a set of bridges. If zippering is used, candidate bridges within a minimum spatial distance to the bridges from two layers ago are prioritized. This means that bridges from layer one and three would be placed in the same area, while layer two and four would be similar, and so on. This causes a bouncing between bridges much like the teeth of a zipper. The concept is illustrated in the following figures.

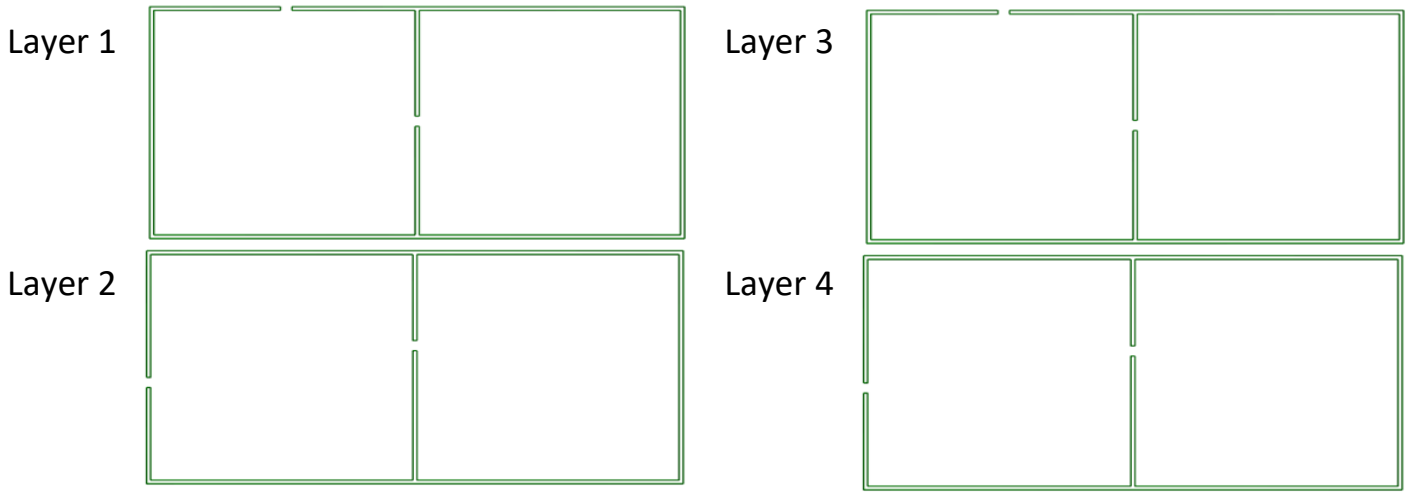

Figure 8: Example of zippering. Layer 1 and 3 and Layer 2 and 4 have bridges in the same location that allow alternating from layer to layer like the teeth of a zipper.

Finally, it should be noted that the single path algorithm described to this point is a layer-based approach meaning that though a single path might be created for a specific layer, layers still exist. However, the algorithm as presented can be combined with simple spiralization approach to create a single path for entire objects. This approach is z interpolation and assumes full connectivity as described in the last section. Recall that a fully connected set of paths creates one single path for each layer. This path can then have its z value interpolated based on a simple distance ratio: D_s / D_p. D_s is the distance of each segment, while D_p is the distance of the entire path. In this way, each segment contributes a percentage of z interpolation relative to its contribute of length with respect to the entire path. As a result, explicit layers are no longer necessary as the pathing will now interpolate across an entire layer height's worth of distance.

Experiment

To demonstrate the efficacy of the algorithm, an object was sliced and built both with and without this algorithm. The chosen geometry is a dog bone shaped object, the ASTM D638 Type V Tensile Bar, shown in Fig. 12. The object was constructed from PLA plastic from Push Plastic.

Figure 9: Test object used for tensile tests and path evaluation.

Using this geometry, two test sets were developed: one using the single path algorithm and traditional layers and one using the single path algorithm in combination with spiralization. For the layer-based test set, four conditions were evaluated: two using traditional pathing and two using the single path algorithm previously described. In addition, each pathing type also had the location of the start/stop or bridge altered to test the impact of placement on mechanical properties. The tests are as follows:

- Condition #1 – used traditional pathing consisting of perimeters and insets with starts/stops in the corner of each contour.
- Condition #2 – used traditional pathing consisting of perimeter and insets with start/stops closest to the center of the object.
- Condition #3 – used the single path algorithm with start/stops on the shortest edge of a contour. That is, bridge locations were placed on the shortest edge.
- Condition #4 – used the single path algorithm with start/stops on the longest edge of a contour. That is, bridge locations were placed on the longest edge.

For each condition, five copies of the object in Figure 12 were constructed for a total of 20 samples. To better visualize the pathing of each test, the g-code visualization is shown in Figure 13. The first image represents the pathing for Condition #1 and Condition #2. Both tests consist of a typical path consisting of a perimeter and set of insets. Each closed contour is disconnected from all others and requires a travel to move from one to the next. The only difference between these two tests are the general start/stop locations for each contour. The next two images represent the pathing for Condition #3 and Condition #4 respectively. Condition #3's bridges are located on the shortest edge of each contour while Condition #4's bridges are located on the longest edge.

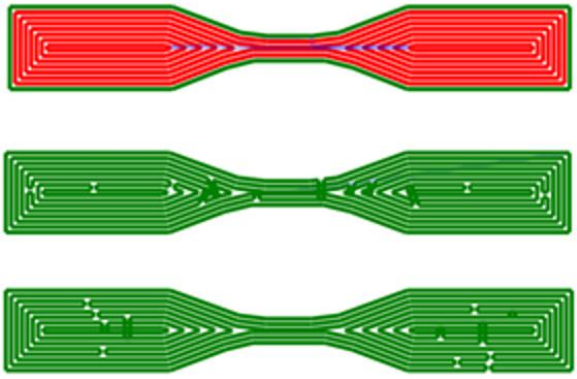

Figure 10: Pathing for tests. Top image represents pathing for conditions regarding traditional pathing. The middle image represents the pathing for single path shortest edge. The bottom image represents pathing for single path longest edge.

The setup for the second test set regarding single path and spiralization is similar. For this test set, three conditions were evaluated:

- Condition #1 – used traditional pathing consisting of perimeters with starts/stops in the corner of each contour.
- Condition #2 – used the single path algorithm with bridges on the shortest edge of a contour.
- Condition #3 – used the single path algorithm with bridges on the longest edge of a contour.

For each condition, five copies of the object in Figure 12 were constructed for a total of 15 samples. The style of pathing used is the same as the previous test set visualized in Figure 13. The primary difference is that Condition #2 and #3 have no explicit layers due to the spiral behavior of the pathing.

All objects were then tested for tensile strength with a speed of 1 mm/min to evaluate the impact of each pathing approach on its mechanical properties. To facilitate this testing, each object was kept relatively small at approximately 3 inches wide. The time taken to construct the object was also evaluated.

Results and Discussion

The mechanical impact of the pathing algorithm will be presented first followed by an evaluation of the construction time and finally some simulated and practical construction examples besides the test samples.

Mechanical Impact

The min, max, and average stress for each condition is shown in Table 1 and 2. Table 1 shows the results for the first test set using the single path algorithm but retaining explicit layers while Table 2 shows the results for the second test set in which spiralization was used in place of layers.

Table 1: Layer-based Test Results

Stress (MPa)	Condition 1	Condition 2	Condition 3	Condition 4
Min	57.37	52.11	56.19	58.13
Max	59.40	54.49	58.73	63.20
Average	58.28	53.48	57.20	61.53

Table 2: Spiralization Test Results

Stress (MPa)	Condition 1	Condition 2	Condition 3
Min	52.74	30.68	54.12
Max	54.68	43.37	56.33
Average	53.92	35.99	55.46

As seen from Table 1, the best performing pathing approach when still retaining layers is Condition #4 in all categories. On average, this approach produces an object that can withstand 5.3%, 13.1%, and 6.1% more stress compared to the other three approaches respectively. As mentioned previously, travel start/stop locations are points of mechanical weakness in an object. So, it is no surprise that a path without travels would perform better than a path with travels. However, intelligent placing of the bridges in the single path algorithm is also required to realize the benefits of this algorithmic approach. In general, placing the bridges on longer edges produces a stronger object. These bridges should also be distributed about the object as practical testing has shown that stacking the bridges in the same location across layers creates weak points in the object. Though positive results were seen in this example, this also presents the opportunity for future research. Bridge placement strategies and its impact on the resultant object are a clear path of improvement.

Similarly for the second set of data in Table 2, Condition #3 produced the best results in each category. Condition #3 in this test set corresponds to Condition #4 in Table 1. That is, bridges were placed along the longer edges of the object. This produces the strongest object even when removing explicit layers as was done for this test set. In addition to these mechanical benefits, time savings can also be realized, and will be discussed next.

Construction Time

To facilitate mechanical testing, the objects presented thus far have been relatively small. The object used in the test sets were identical in size and approximately 3 inches in width. At that size, regardless of pathing approach, time differences are negligible compared to overall print time. Travels are short and quick, so the overall time savings is minor.

However, as this object is scaled up, time lost to travel becomes a more significant issue. To evaluate time savings for a large-format system, the object was scaled to have a width of approximately two feet. A two-foot-wide object is typical for larger machines. At this size, the object consisted of eight layers of identical pathing when evaluating conditions #1-#4 for the first test set.

For the traditional pathing, construction of the object was estimated to take 8:48, while the single path version was 7:38. This represents a time savings of approximately 13.3%. Naturally, the time savings for the conditions in the layer-based test set will be lower than the spiralized test set. This is because time is still lost transitioning between layers. With the introduction of a spiralized approach, time savings are increased since explicit layers no longer exist. Importantly, as the object becomes larger, this savings increases. With larger objects, travels become a larger percentage of the time not spent constructing the object. So, as objects become larger, the pathing provided by the presented algorithm will have greater importance as greater time savings can be realized.

Additional Practical Examples

In addition to the test samples evaluated for mechanical properties, several other simulations were run to investigate the efficacy of the single path algorithm. Several examples of pathing solutions are shown in Figure 11.

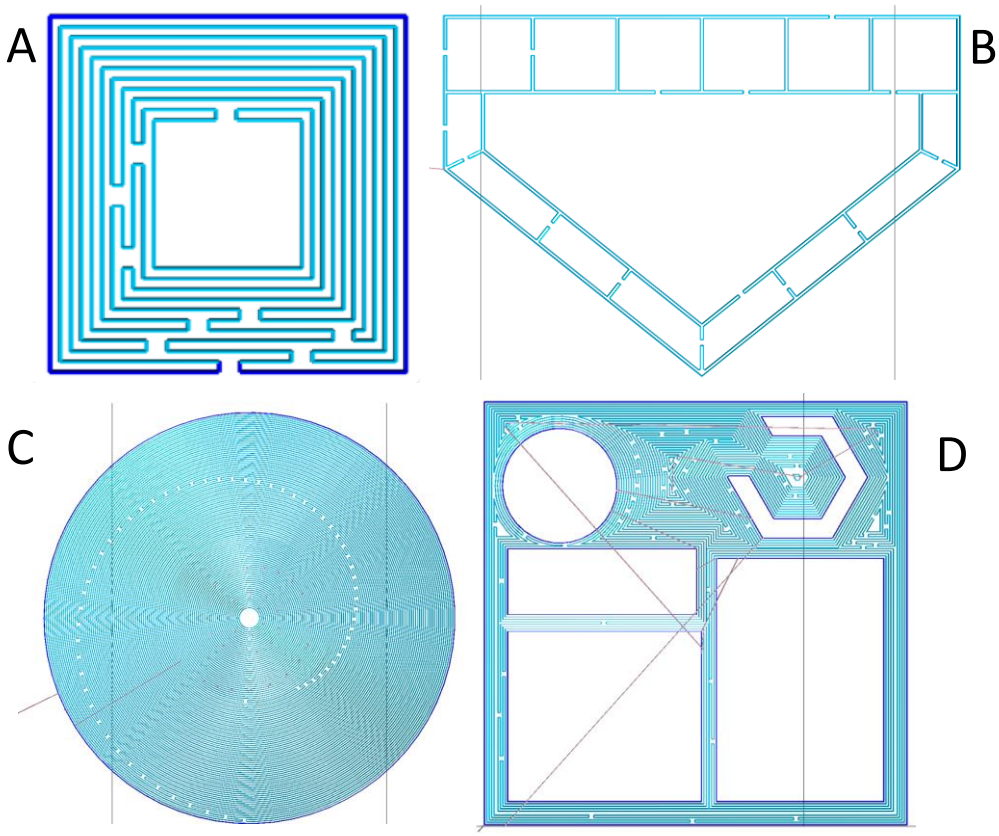

Figure 11: Single Path Examples. 11A/B - fully connected paths without travels. 11C/D - some connected geometry but travels remain.

Figure 11A and B demonstrate objects with full connectivity. For these objects, a single path can be constructed for each layer as bridges can connect every contour. Figure 11C and D demonstrate objects that do not have full connectivity. In these cases, the algorithm inserts as many bridges as possible, but not all geometry can be connected. As a result, travels are inserted where necessary. For Figure 11C, as the pathing progresses to the interior of the circle, segments become so small that they cannot host bridges. Recall that the algorithm is prevented from placing bridges across the boundaries of segments. If that condition were relaxed, this object

could be connected. Similarly, in Figure 11D, bead widths were chosen that were purposely incorrect. This leads to gaps between the contours that are larger than one bead width of distance. Again, the algorithm as discussed is limited in the distance that can be traversed when inserting bridges and once again, could connect this geometry if the restrictions were relaxed. However, even when maintaining these restrictions, there is a significant reduction in the number of travels in these objects compared to the traditional pathing approach.

Finally, there is an example construction shown in Figure 12. This object, referred to as the "bicycle wheel" demonstrates the necessity for single path approaches. This object was constructed previously using traditional pathing approaches. However, because of the number of insets and features within this design, the resultant object was too mechanically weak to be practical. On more than one occasion, the object simply broke when removing it from the build table. Using the single path approach though led to a fully connected solution. This allowed constructions without travels and removed an enormous number of seams. As a result, the construction of this object is now practical with enough mechanical rigidity to survive removal from the build table.

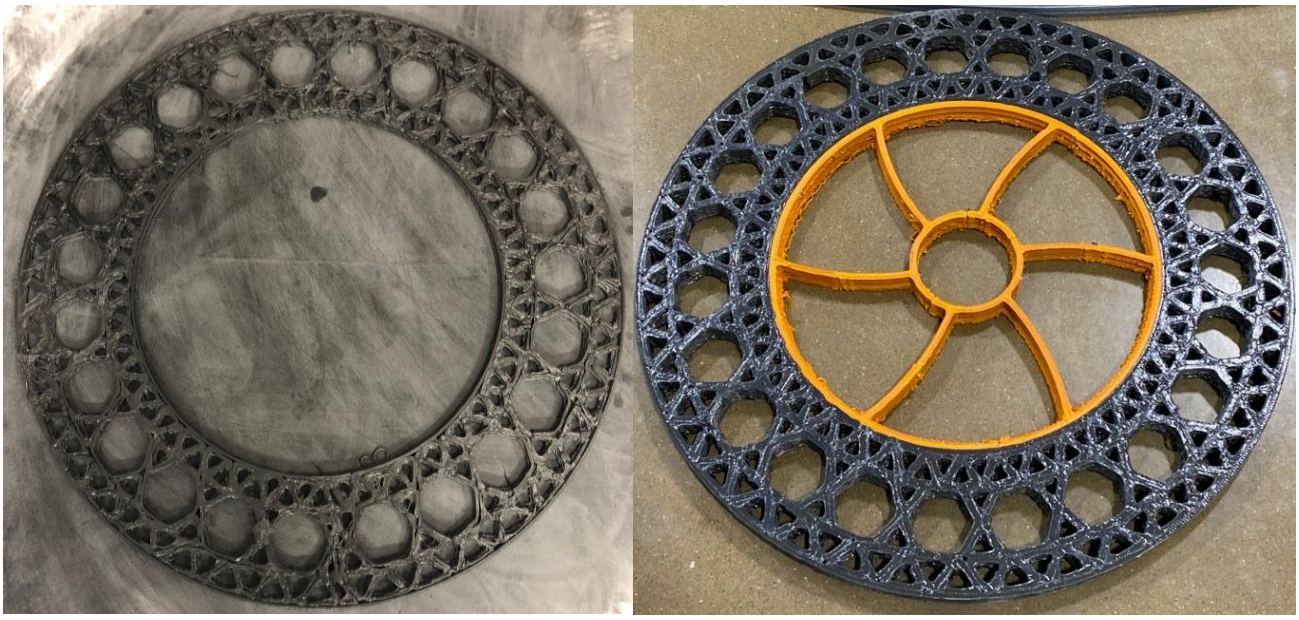

Figure 12: Example construction using single path. Traditional construction was not possible due to weakness introduced by all the seams in this object's design.

Conclusion

In this paper, a new approach for generating a single path for closed contour geometry was presented. This pathing approach converted the polygon into a graph representation to evaluate the placement of bridges. These bridges allowed the connection of one closed contour to another resulting in a single path. This pathing was then combined with spiralization techniques and tested against a traditional pathing approach. The single path approach was found to be both faster and produce parts that are more mechanically robust.

Despite the positive results produced thus far, there is still room for improvement and future work is planned. Currently, optimization of bridge location is a priority. The current methodology to weigh individual contours in the graph representation does not account for physical properties. This leads to a straightforward way to select bridge locations; however, this process could be improved dramatically by linking to analyses in the form of stress or thermal calculations.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Energy Efficiency and Renewable Energy, Office of Advanced Manufacturing, under contract number DE-AC05-00OR22725. This work is described in US patent application, Serial Number 16/842,274 (filed on April 7, 2020.

Bibliography

[1] Wang, W., et al. (2014). Saliency-Preserving Slicing Optimization for Effective 3D Printing. Computer Graphics Forum, 33(5), 1 – 12.

[2] Suh, Y., Wozny, M. (1994). Adaptive Slicing of Solid Freeform Fabrication Processes. Solid Freeform Fabrication, 5, 404-411

[3] Chen, Y. H., et al. (1999). Generation of an STL File from 3D Measurement Data with User-Controlled Data Reduction. The International Journal of Advanced Manufacturing Technology, 15(2), 127 – 131.

[4] Chesser, P., et al. (2019). "Extrusion control for high quality printing on big area additive manufacturing (baam) systems." Additive Manufacturing 28: 445-455.

[5] McQueen, K., Darensbourg, S., Moore, C., Dickens, T., & Allen, C. (2018). Efficient Path Planning of Secondary Additive Manufacturing Operations. In MATEC Web of Conferences (Vol. 249, p. 03011). EDP Sciences.

[6] Ali, M. H., Mir-Nasiri, N., & Ko, W. L. (2016). Multi-nozzle extrusion system for 3D printer and its control mechanism. The International Journal of Advanced Manufacturing Technology, 86(1-4), 999-1010.

[7] Ding, D., Pan, Z. S., Cuiuri, D., & Li, H. (2014). A tool-path generation strategy for wire and arc additive manufacturing. The international journal of advanced manufacturing technology, 73(1-4), 173-183.

[8] Ding, D., Pan, Z., Cuiuri, D., Li, H., van Duin, S., & Larkin, N. (2016). Bead modelling and implementation of adaptive MAT path in wire and arc additive manufacturing. Robotics and Computer-Integrated Manufacturing, 39, 32-42.

[9] Liu, H. H., Zhao, T., Li, L. Y., Liu, W. J., Wang, T. Q., & Yue, J. F. (2020). A path planning and sharp corner correction strategy for wire and arc additive manufacturing of solid components with polygonal cross-sections. The International Journal of Advanced Manufacturing Technology, 106, 4879-4889.

[10] Michel, F., Lockett, H., Ding, J., Martina, F., Marinelli, G., & Williams, S. (2019). A modular path planning solution for Wire+ Arc Additive Manufacturing. Robotics and Computer-Integrated Manufacturing, 60, 1-11.

[11] Jin, Y., He, Y., & Du, J. (2017). A novel path planning methodology for extrusion-based additive manufacturing of thin-walled parts. International Journal of Computer Integrated Manufacturing, 30(12), 1301-1315.

[12] Fok, K. Y., Ganganath, N., Cheng, C. T., Iu, H., & Chi, K. T. (2019). A Nozzle Path Planner for 3D Printing Applications. IEEE Transactions on Industrial Informatics.

[13] Ganganath, N., Cheng, C. T., Fok, K. Y., & Chi, K. T. (2016, April). Trajectory planning for 3D printing: A revisit to traveling salesman problem. In 2016 2nd International Conference on Control, Automation and Robotics (ICCAR) (pp. 287-290). IEEE.

[14] Xia, L., Ma, G., Wang, F., Bai, G., Xie, Y. M., Xu, W., & Xiao, J. (2022). Globally continuous hybrid path for extrusion-based additive manufacturing. Automation in Construction, 137, 104175.

[15] Zhang, G., Wang, Y., He, J., & Xiong, Y. (2023). A graph-based path planning method for additive manufacturing of continuous fiber-reinforced planar thin-walled cellular structures. Rapid Prototyping Journal, 29(2), 344-353.

[16] Dreifus, G., Goodrick, K., Giles, S., Patel, M., Foster, R. M., Williams, C., ... & Kunc, V. (2017). Path optimization along lattices in additive manufacturing using the chinese postman problem. 3D Printing and Additive Manufacturing, 4(2), 98-104.

[17] Choi, S. H., & Cheung, H. H. (2006). A topological hierarchy-based approach to toolpath planning for multi-material layered manufacturing. Computer-Aided Design, 38(2), 143-156.

[18] Roschli, A., et al. (2019). Creating Toolpaths without Start and Stops for Extrusion-based Systems. Solid Freeform Fabrication Symposium Proceedings.

[19] Schäling, B. (2011). The boost C++ libraries. Boris Schäling.
[20] Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical society, 7(1), 48-50.
[21] Even, S. (2011). Graph algorithms. Cambridge University Press.