Solid Freeform Fabrication 2024: Proceedings of the 35th Annual International
Solid Freeform Fabrication Symposium – An Additive Manufacturing Conference
Reviewed Paper

Fundamental Path Optimization Strategies for Extrusion-based Additive Manufacturing

Alex Roschli[1,2], Liam White[2], Michael Borish[2], Cameron Adkins[2], Ashley Gannon[2], Adam Stevens[2], Thomas A. Feldhausen[1,2], Brian Post[2], Eric MacDonald[1,2]

[1] University of Texas at El Paso

[2] Oak Ridge National Laboratory, Manufacturing Science Division

Abstract

Extrusion-based additive manufacturing processes begin with a software program, called a slicer, that generates layer geometry and fits toolpaths to each layer to define where material is to be extruded or deposited. Before the toolpaths are output as g-code for the additive manufacturing system to execute, the toolpaths should be optimized. Many complex optimization approaches using graph theory, Chinese postman problem, and other complex mathematical models exist, but these approaches are rarely used in daily printing operations and are not available through common slicing programs such as Cura and PrusaSlicer. Instead, path planning and optimization typically revolves around simpler, fully automated approaches such as inside out and next closest. This paper will explore the fundamental optimization strategies for toolpath planning and document a new implementation, available via open-source slicing software, that allows for greater control of the path planning process.

Keywords: Slicing, toolpath generation, path planning, path optimization, g-code

Introduction

Slicing is the title given to the entire software process of going from a CAD (computer aided design) model to g-code instructions that a 3D printer can read and execute to build a part. This title is given even though slicing is just one step of the process, and often one of the fastest computational steps. The slicing step is where the CAD model is “sliced” into layers through a process called cross-sectioning [1]. After the slicing step, toolpaths are fit to the layer, then optimization steps are applied to order the printing operations. Finally, the toolpaths are converted to g-code instructions formatted specifically for the machine that will read and execute the g-code.

The optimization step can be broken down into many parts including ordering and travel insertion [2]. Simple strategies for ordering allow definition of the start and stop position on the outermost path, called the seam, and path order such as inside out or outside in. Prusa Slicer, a common open-source slicer [3], allows for seam optimization including random, aligned, nearest, rear, and user defined via a UI

painting tool. Cura, another common open-source slicer [4], also allows similar seam optimization settings and adds a custom point path optimization strategy applicable to first path of the layer.

Advanced optimization strategies for path planning have been popular topic of research for years in the truck driving industry [5-8] and for guiding tractors around farms [9-12]. However, additive manufacturing is a relatively young industry, and the slicing and toolpathing operations are not well researched or documented at this time including the standard path optimization strategies. However, there is some university research showcasing advanced mathematical approaches for path optimization. Dreifus showed the application of the Chinese Postman Problem (CPP) to create pathing along a lattice that minimized print time [13]. Kim applied a complex stress analysis to generate a graded infill structure and traversed the pathing using CPP [14]. Lechowicz proposed two hybrid path optimization strategies: Greedy Two Opt and Greedy Annealing to reduce path length and decrease print time for 3D printing [15]. Many authors used an ant colony algorithm to optimize pathing and reduce print time [16-21] whereas Liu used the traveling salesman approach [22]. Dong showcases the use of a Hopfield Neural Network for filament-based printing [23]

What all of these optimization and pathing approaches have in common is that none of them are in use outside of university and research environments. Industrial manufacturing environments and other end users don't make use of these advanced pathing strategies because of the complexity of implementation and the limited use cases. Millions of desktop 3D printers are in use [24], but none of the software programs including Cura, PrusaSlicer, Simplify3D, KiSSSlicer, MatterControl, and OrcaSlicer include any means of path optimization or planning based on graph theory, traveling salesman, Chinese Postman, ant colony, or any other complex mathematical model.

Some of these strategies may be computationally efficient and only add negligible time to the slicing process for an average geometry, but the easy ability to make use of them still hasn't been implemented commercially or transitioned to industry. Instead, these slicing programs and users rely on much simpler algorithms that are calculated efficiently and effectively for a wide variety of geometries. This paper will explore these fundamental, automated optimization and path planning strategies, then show a restructuring and new implementation, via open-source software, that allows users more control of the path planning process. This new implementation is in use today in industrial environments and seeing commercial adoption.

Creating Toolpaths for a Layer

When a mesh, typically an STL for 3D printing applications, is sliced into layers, a polygon is generated to represent the bounds of the layer. Complex geometries can create multiple distinct printable areas within a layer, each represented by a separate polygon called an island. Multiple islands also occur when multiple parts are printed at one time. Once the islands, or island, for a layer have been found, toolpaths can be fit to each island within the layer.

The toolpath creation process for an island is done by toolpath region. The most common regions are perimeter, inset, skin, infill, skeleton, and support. Not all regions are used on each print, and not each region will have toolpaths on each layer. A region can and often will have multiple paths for the same layer. Perimeter and inset always generate closed-loop paths, which is a path that has the same start and end point. Skeleton paths are always open loop, meaning a path that has different start and end points.

Infill, skin, and support can generate both open loop and closed-loop paths depending on the slice settings.

Optimizing the Layer

Once the paths are planned for every region in every island, optimization strategies can be implemented to define how to execute the paths. These optimization strategies can be divided into three main categories including ordering of the islands, ordering of the paths within the regions, and ordering of the points within the paths. Ordering of the regions within the islands is also something that can be modified, but no slicer currently offers anything beyond outside in and inside out. This is partly because some slicers don't separate paths by regions, and because some slicers treat path and region optimization as the same optimization by applying the same strategy to each. Because of this, ordering of the regions will not be a sub-section and will instead be discussed in more detail in the next section, where a new solution based on user preference will be introduced. The following sub-sections will define the existing optimization strategies for the three main categories and introduce a few other strategies that can easily be applied.

Island Optimization Strategies

The first optimization strategy is to determine the order of the islands. Optimization strategies for islands include next closest, next farthest, random, custom location, and least recently visited. With the exception of least recently visited, these strategies are shared with path optimization strategies and will be defined in the next section.

Least recently visited is useful for thermal processes to help evenly distribute heat among all islands by always going to the least recently visited island, which would typically imply going to the coldest island for processes that involve deposition of hot material. When a new island appears, typically when an existing island splits into multiple islands, this new island is at the beginning of the list and gets printed first. More robust implementations can detect an island splitting into multiple islands and keep the new islands at the same order as the original island. Because island order optimization isn't necessary for a lot of builds, and because the optimization strategies are shared with the path optimization strategies, the remainder of this paper will focus on single island optimization strategies for both path order and point order.

Path Optimization Strategies

There exist several common path optimization strategies which serve to define the order of the paths within a region. Typically, slicers order paths in one of two ways: outside in and inside out. Other methods of optimization include random, next closest, next farthest, and custom point. The following path optimization strategies will share a common region order: perimeter, inset, infill, and skeleton.

Outside In and Inside Out

Outside in and inside out are very simple strategies that work as expected. For outside in, paths on the outer edge of the region are printed first, then the next path towards the center of the layer is printed and so on until all paths within the region are printed. For instances when multiple paths exist towards the center, the path closest to the current path is printed next. Inside out is simply the opposite, the path within the region that is closest to the center of the layer is printed first and then the next path

outwards is printed next and so on. Figure 1 shows the toolpathing for a square with perimeter (dark blue), inset (light blue), and infill (green) paths printed outside in and inside out.

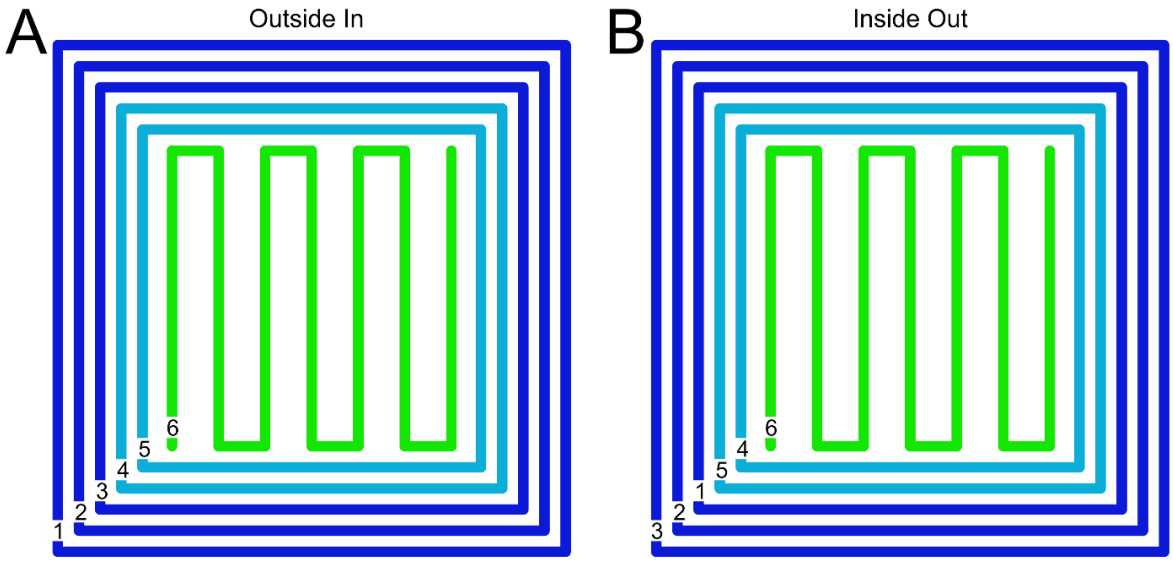


Figure 1: A square shape showing outside in (A) and inside out (B) pathing. The numbers indicate the print order for the paths.

Notice that Figure 1b, inside out, still prints the three perimeter contours first rather than infill first. The layer itself isn't being printed inside out, just the paths within each region. For some slicing packages, selecting inside out forces not only paths but also regions to be printed inside out; however, this section is focusing on path optimization on a per-region basis. An upcoming section will talk about the potential to re-order the regions to allow for all regions and paths to be printed inside out.

Outside in and inside out get more complicated for objects that have multiple "outsides". Take for example the hollow square shape shown in Figure 2a. For a configuration with just perimeter and infill paths, the typical implementation of outside in starts by printing the outermost square perimeter and working inwards toward the infill. Next, the pathing does the outermost perimeter for the inner square and begins working inward. Finally, the infill is printed. This pathing is illustrated in Figure 2b.

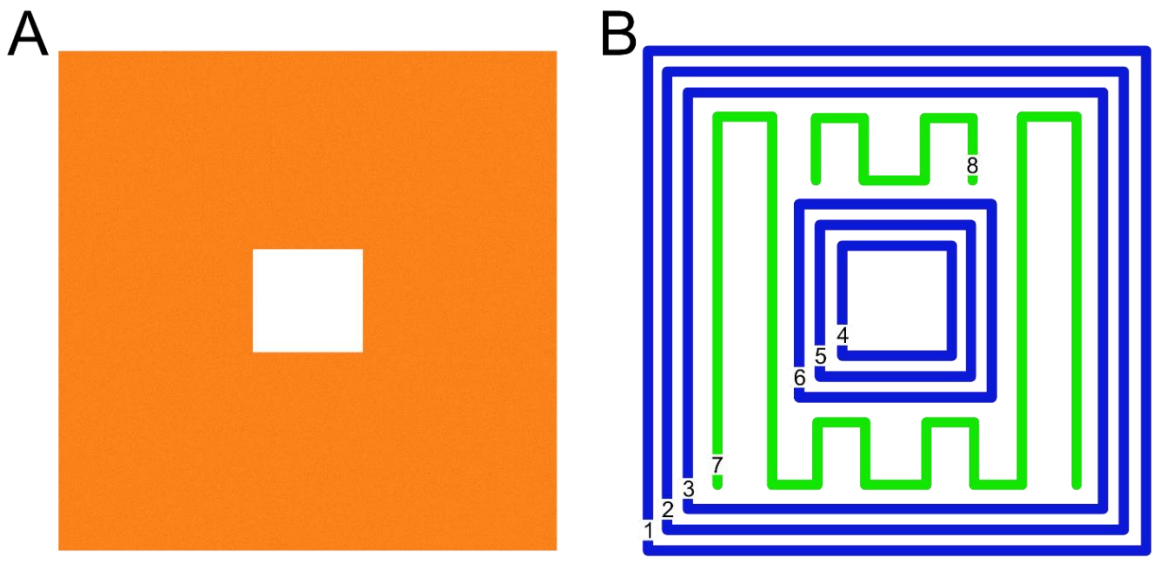


Figure 2: A hollow square (A) and the resultant pathing from outside in (B).

The path ordering from Figure 2 changes if the regions for the closed contours changes. In Figure 2, all closed contours are perimeters (dark blue). If the pathing is implemented with perimeters and insets (light blue), the order changes so that all of the perimeter paths are printed first and then the inset paths are printed (Figure 3). This is because the paths for one region, perimeters, are printed and then the paths for the next region are printed. The motivation for making the inner closed contours insets rather than perimeters is to print these insets with different parameters, typically a faster speed, than the perimeter contours which will be exposed.

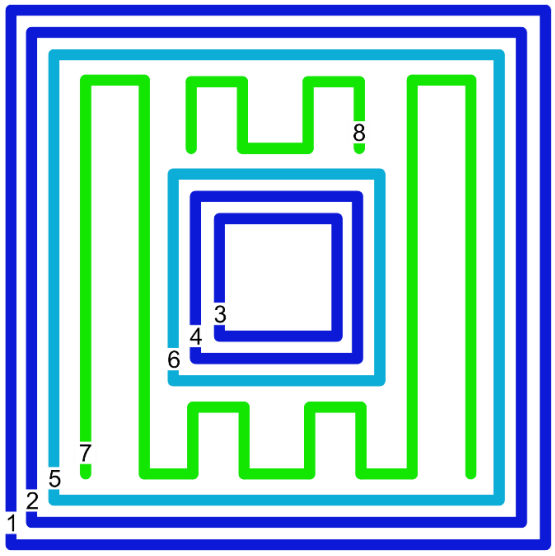


Figure 3: Changing the third perimeter in Figure 2B to an inset alters the printing order.

While outside in and inside out seem quite simple, many complex edge cases can be encountered that make it complicated to determine which path to print next. For example, the figure-eight shape shown in Figure 4a has two closed contours (A&B) inside of the large outermost closed contour. After

printing the outermost contour, both remaining interior contours are equally close to the center of the layer. To decide which contour to print next, a next closest calculation is implemented to select the closer path to print followed by the remaining path. The distance to point A is less than the distance to point B, so path A gets printed first (Figure 4b).

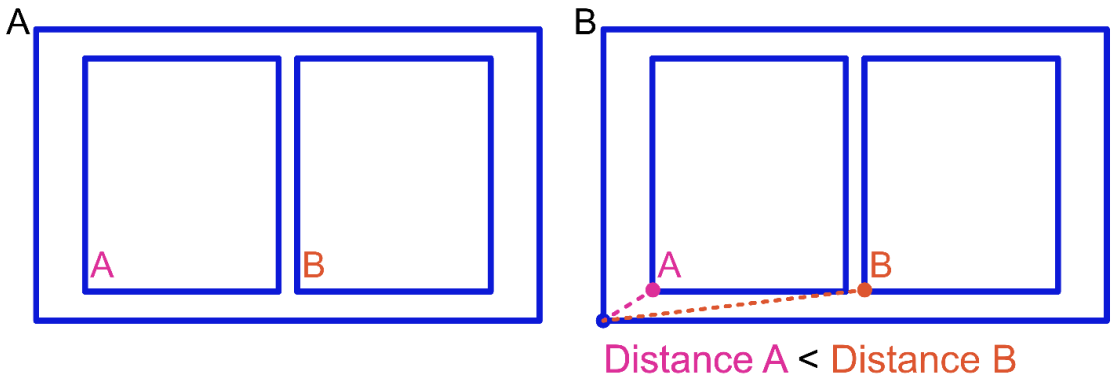


Figure 4: Figure eight shaping pathing where both interior perimeters are equal distance from the center (A), and the next closest calculation to determine whether path A or path B is printed next (B).

Outside in and inside out pathing are used depending on the needed geometric accuracy. For example, outside in is used when the geometric accuracy of the outer bounds is most important. The downside of this approach is that while printing moves toward the centroid, any excess extrusion has nowhere to go and ultimately causes overfilling that exceeds the top of the layer as seen in Figure 5.

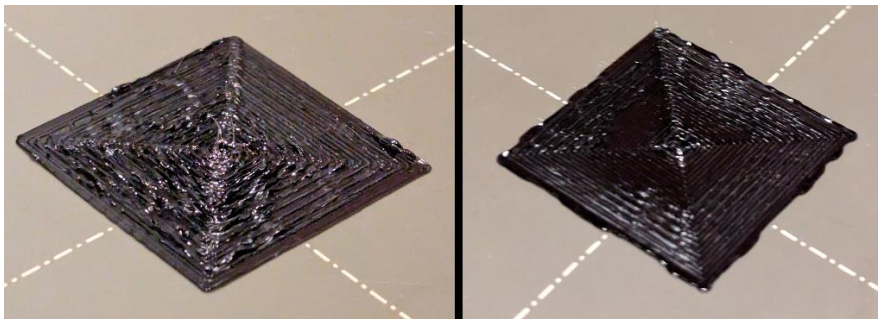


Figure 5: A 25mm square printed concentrically from outside in causing center overfill (A), and the same 25mm square printed inside out (B). Notice that the outside in version has clean outer lines but appears rough in the center while the inside out version is rough on the outer lines and smooth in the center.

Inside out isn't as likely to have the bulging and overfill issue seen in Figure 5 because excess material can be pushed outwards. However, because excess material is pushed out, the result can cause the outermost perimeter bead to miss the defined outer boundary of the layer. Figure 6 shows a measurement of the outer dimensions for both outside in and inside out pathing for the same shape.

Figure 6: A 25mm square printed outside in (left) and inside out (right). Notice that the outside in square (left) perfectly matches the dimensions while the inside out version (right) is oversized because of the material being pushed outward during printing.

Random Path Optimization

Random path optimization is exactly like it sounds paths placed in an unspecific, unrepeatable order. Once the toolpaths are created for a layer, an algorithm is used to randomly order the paths. The benefit of this strategy is that it helps prevent buildup issues from one layer to the next by causing paths to print in a different order each time. Because of the bulging of beads mentioned during Outside In and Inside Out, the space where a path lies during random optimization can slightly shift and therefore have a higher chance of success. The downside is that material properties can be inconsistent from the random distribution of paths that causes a random distribution of heat. Figure 7 shows a random path order for the same toolpaths seen in Figure 1.

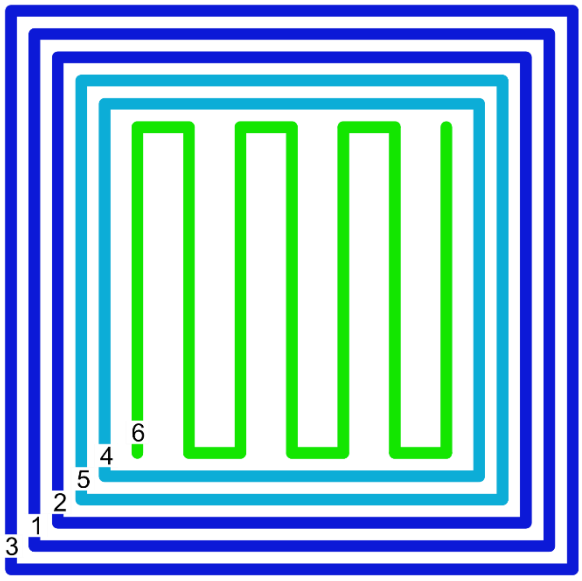


Figure 7: A square shape where the paths within each region are randomly ordered.

Next Closest

As the name implies, the next closest path optimization works by finding the closest path to the current location. This is often done with a Greedy Algorithm [25] to speed up the computation time. At the beginning of a build, the current location is assumed to be the origin of the workspace, though a custom point can be defined which will be explained in a further section. Once a path has been printed, the end point of the path is used as the current location for the purposes of calculating the next closest path.

Two different methods of calculating the next closest can be used. A point-to-point calculation can be used to compare the current location to every point on every remaining path, ultimately defining the path with the closest point as the next path. That closest point isn't necessarily used as the starting point for the next path, which will be explained in the next section. The second method involves creating a convex hull from each remaining path such that the center of the hull can be used to calculate a distance from the current point. The path with the shortest distance from center point to current location is chosen as the next path. The issue with the latter implementation is that concentric paths may generate a convex hull with the same center point. When this happens, a different calculation must be used to determine which path to print. This often means using the first calculation method of comparing every point on each path to the current location. Figure 8 shows the next closest path implementation for the same paths in Figure 1, using a next closest implementation based on comparing all remaining points on all remaining paths.

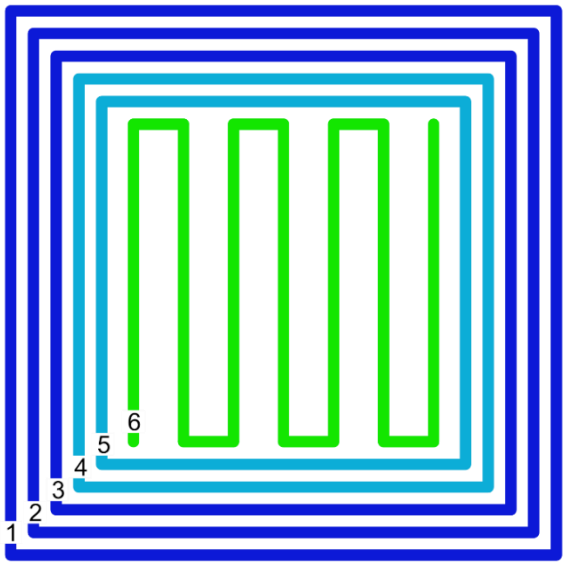


Figure 8: Next closest path optimization for a square.

The path order shown in Figure 8 is identical to that of Figure 1a. This is because the outside in and inside out pathing optimizations use a next closest calculation to find the next path when moving towards or away from the center of the part. By using outside in and inside out, the user can define which direction to order the pathing. Certain implementations of next closest pathing can also be used to ignore region order and instead find the next closest path of any type within the layer.

Next Farthest

In contrast to the next closest strategy is the next farthest strategy. Next farthest is implemented using the same Greedy Algorithm calculations as next closest but instead selects the path or point along a path that generates the largest distance [26]. Next farthest isn't often used, but it is helpful for systems where the material being deposited needs time to cool and solidify before another bead is printed alongside. The result of implementing the next farthest optimization can be seen in Figure 9.

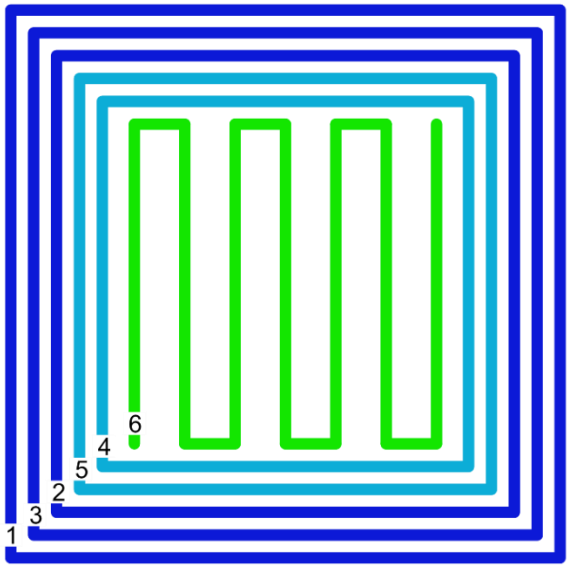


Figure 9: Next farthest path ordering for a square.

Custom Point

The last path optimization strategy is the custom point method. This approach requires the user to define a custom point, typically just an X and Y location, as the location to feed into the next closest algorithm rather than the current location. After each path is completed, the next closest path to the optimization point is calculated and printed. Figure 10 shows the result of two different custom point optimizations for the same toolpaths used in Figure 1. Notice that the second implementation, Figure 10b, causes the perimeter to print inside out and the insets to print outside in.

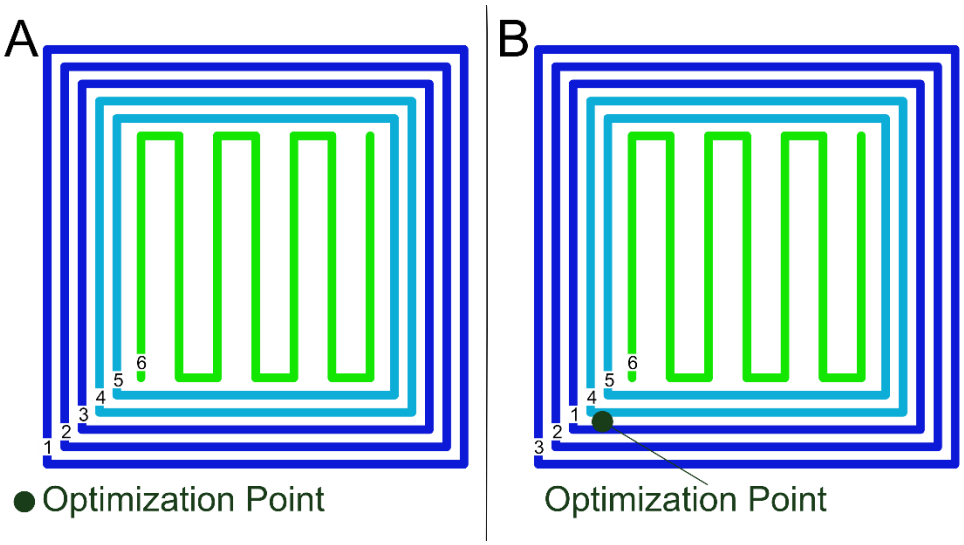


Figure 10: Two scenarios for custom point path order optimization.

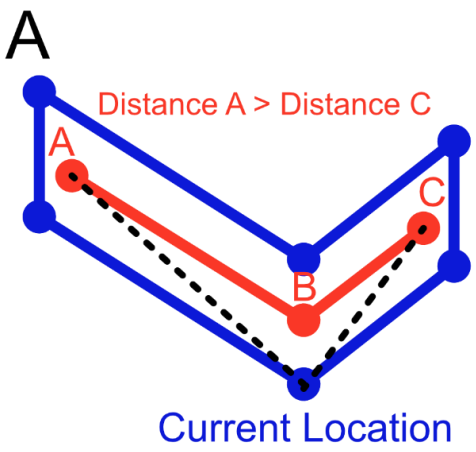
Point Optimization Strategies

There exist several common point optimization strategies which are implemented to define the start point of the path. These optimization strategies are sometimes referred to as seam selection or seam optimization because they are used to define where the seam is on the exterior of the part. The seam is typically the name given for the start/stop point on the outermost path which can visibly be seen as a minor defect.

For closed-loop paths, the point optimization strategies can also be used to define the order of the points within the path, or the direction of traversal. Because open loop paths have two distinct ends, the optimization strategies are used to determine which end point is the start point, but traversal direction cannot be modified. The following sections will define point optimization strategies that can be used to determine start point and point order.

Next Closest and Next Farthest

These two strategies were previously defined for path optimization, but a similar implementation can also be used for determining the start point. For an open-loop path, the implementation works by comparing the current location to the two end points of the open loop path that is next in the path order. Whichever end point is closer, or farther depending on whether next closest or next farthest is being implemented, is selected as the start point. This may result in the list of points for the path being printed in their existing order, or it may require the list of points to be reversed. Figure 11a shows the implementation for an open-loop path inside of a closed-loop path. In this scenario, point A is used for next closest, and C is used for next farthest. Figure 11b shows how the order of points changes for the open-loop path based on using next closest or next farthest.

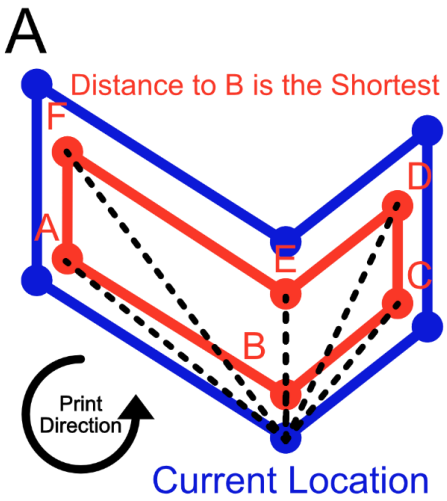


B

Original Point Order	Updated Point Order
A	C
B	B
C	A

Figure 11: Finding the point order for a red skeleton path inside a blue closed loop path (A) and showing how the order of points changes based on the distance calculation (B).

For a closed-loop path, the distance is calculated between the current point and all points on the next closed-loop path. Whichever point is determined to be the shortest distance, or farthest distance if using the next farthest implementation, becomes the start point. Programmatically, this can be implemented by rotating the list or array of points that define the path so that the desired start point is the first item in the array. The order of points in the path is not changed during this rotation. Additional modifications can be used such as clockwise versus counterclockwise to change the order of points and will be discussed in an upcoming section. Figure 12a shows the implementation for a closed-loop path nested within another closed-loop path. The outer path is calculated and printed first, and then the start point for the nested closed-loop path can be defined. Figure 12b shows how the order of points changes for each implementation.

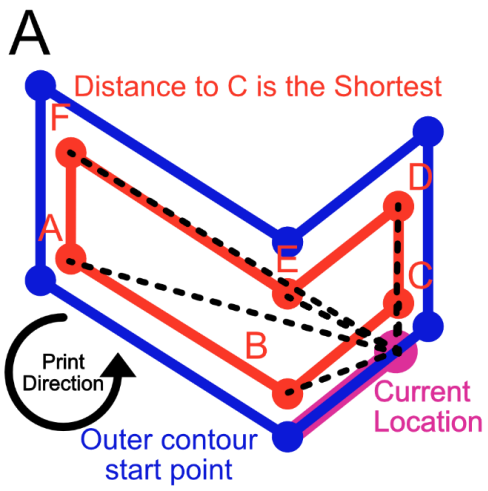


B

Original Point Order	Updated Point Order
A	B
B	C
C	D
D	E
E	F
F	A

Figure 12: A red closed loop contour is added inside the outer blue contour (A). To find the start point using shortest distance, the distance from each point on the red contour to the current location is calculated. The distance B is shortest, so that point becomes first in the print order (B).

The previous examples assume that the start and end point of the path are the same point, shown as the current location. However, thermoplastic pellet extrusion systems often implement path modifiers, such as tip wipes, that add additional moves to the end of the path such that the end point is not shared with the start point [2]. This could cause the point optimization to move to a different vertex on each path because the end point of the tip wipe, rather than the end point of the print path, is used for the next closest calculation. Figure 13 adds a pink tip wipe to the blue outer contour causing the current location to be altered which generates a new result for next closest.



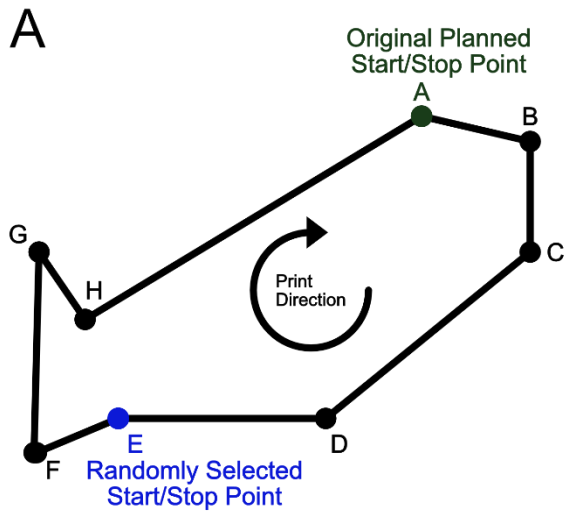
B

Original Point Order	Updated Point Order
A	C
B	D
C	E
D	F
E	A
F	B

Figure 13: Adding a pink tip wipe to the paths from Figure 12 causes the shortest distance calculation to change.

Random

As previously defined in the path order optimization, random optimizations work by arbitrarily selecting an index among the list of available points. For point order optimization, this means randomly selecting a point along the path to be the first point of the path. Programmatically, this is implemented by randomly selecting an index in the array of points on the path, then rotating the array until the randomly selected point is the first index of the array. The random point optimization doesn't change the direction of the printing, or the order of the points, it just modifies where the path starts and ends printing. Figure 14 shows the random path optimization applied to an object with many vertices.



B

Original Point Order	Updated Point Order
A	E
B	F
C	G
D	H
E	A
F	B
G	C
H	D

Figure 14: An eight-sided polygon with labeled start points (A). Point A is initially designated the start point, but point E gets randomly selected as the first point. This table shows the updated point order (B).

The advantage of random ordering is that the seam is constantly moving to a new location which helps distribute the weak interface created by the seam. The downside is that the seam isn't quite as hidden since it's distributed everywhere and visible from every angle. This effect is more drastic on round objects where a corner isn't available to help hide the seam. Figure 15 shows a cylinder shape with random seam optimization.

Figure 15: FDM printed cylinder with random seams showing like "zits" around the exterior surface.

Consecutive

The consecutive point order is helpful for objects that have the same or similar layer geometries from one layer to the next because it allows the start point to move or rotate among vertexes layer to layer. For example, a hexagon could be configured so that each layer starts from a different vertex. With a hexagon, as shown in Figure 16, this could mean starting layer 1 at vertex A, layer 2 at vertex B, and so on until layer 7 is back to vertex A.

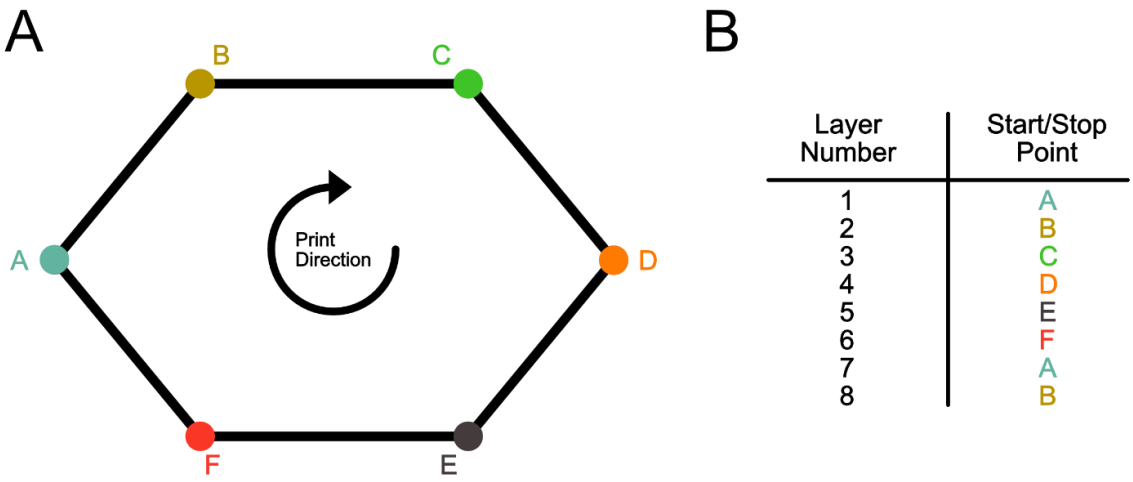


Figure 16: A hexagon using consecutive start point (A) to rotate among the start points from layer to layer (B).

The amount of distance to move/rotate along the path from layer to layer can be a user configurable setting so that each layer doesn't move to the next vertex. The rotation could instead skip 1 or more vertices. For more complex objects, this rotation distance could be configured as a minimum distance so that it's not necessarily jumping by the same number of vertices each time, and instead it is moving a set distance away from the previous object. Figure 17 shows an example of a complex polygon where the minimum consecutive rotation distance means moving by a different number of vertices each time.

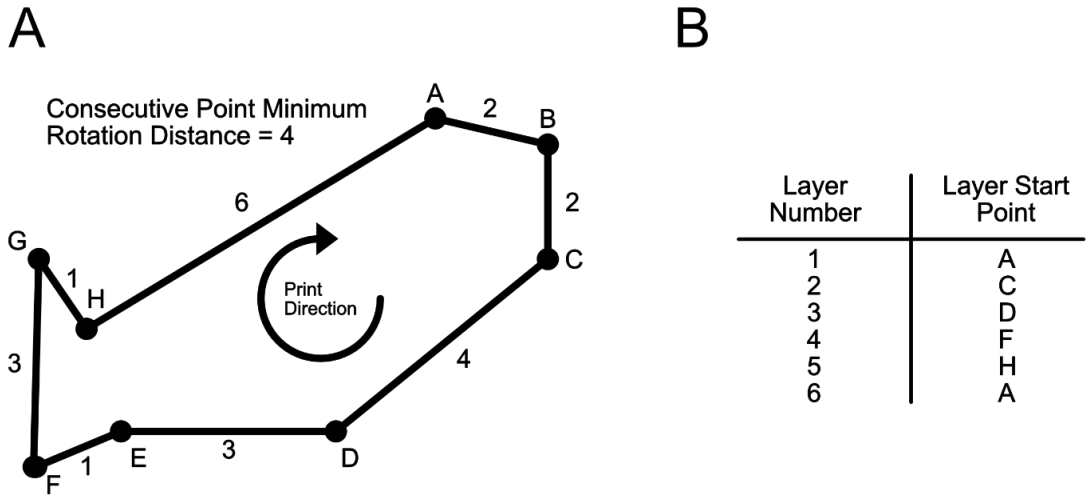
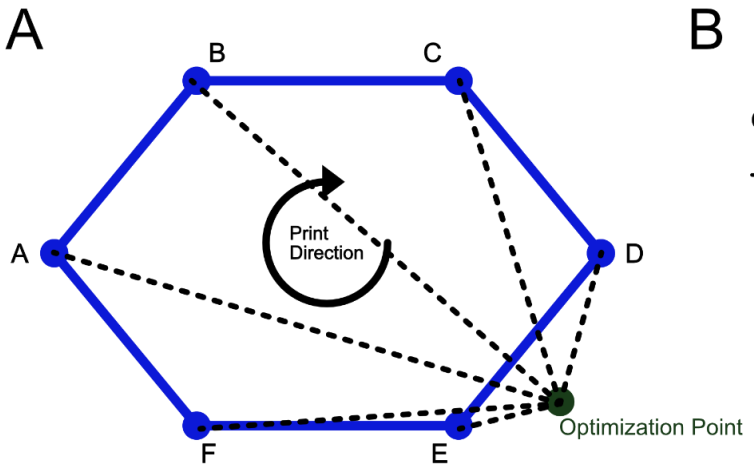


Figure 17: An eight-sided polygon with distances shown for the length of each side (A). The resulting start point for each layer based on a consecutive point minimum rotation distance of 4 (B).

Custom Point

The custom point implementation uses the next closest implementation, but the current location used for the distance calculation is replaced by a user defined point such that after every path, the point on the next path closest to the optimization point is selected as the start point. The custom point location can be defined as an X/Y value via textbox. The custom point approach is often used to define the seam as being in one certain area of the part for the entire build, ideally somewhere that is less noticeable or

hidden. Figure 18 shows custom point being implemented on a hexagon shape to define the start point of the first path of the layer.

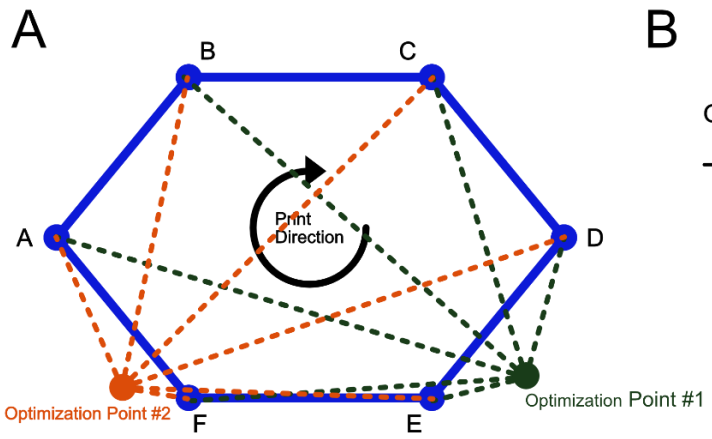


B

Original Point Order	Updated Point Order
A	E
B	F
C	A
D	B
E	C
F	D

Figure 18: The blue outer contour of a hexagon with a green optimization point showing the basis for the distance calculation to be used for the next closest algorithm (A). The black dashed lines represent the distance to each point from the optimization point. The table shows the original point order and the updated order based on the optimization point (B).

The custom point approach can be implemented with two user defined custom points to allow the start/stop point to alternate between two locations from layer to layer. This is useful for situations where the continued buildup of material at the seam could cause a print failure on a long build. It is also helpful to distribute the weak joint caused by the seam to two separate locations. Figure 19 shows the hexagon from Figure 18 with a second optimization point added. In this scenario, the first optimization point is used for all odd numbered layers, and the second point is used for all even numbered layers.



B

Original Point Order	Updated Point Order for Odd Layers	Updated Point Order for Even Layers
A	E	F
B	F	A
C	A	B
D	B	C
E	C	D
F	D	E

Figure 19: Adding a second optimization point, orange, to the hexagon from Figure 18 (A) causes a different point order for even and odd layers (B).

It should be noted that custom point, and all other point optimization strategies discussed in this section, cannot create or modify points within the path. The defined custom point may be very close to the path, but farther from an existing point on the path, but it cannot create a point on the path to serve as the new start point. Figure 20 shows a square where the custom point is close to the side of the square, but the algorithm must pick one of the four blue vertexes to be the start/stop point of the path. The

orange created point shows the shortest distance from the optimization point to the path, but a point does not exist at that location and cannot be created to use as a start point.

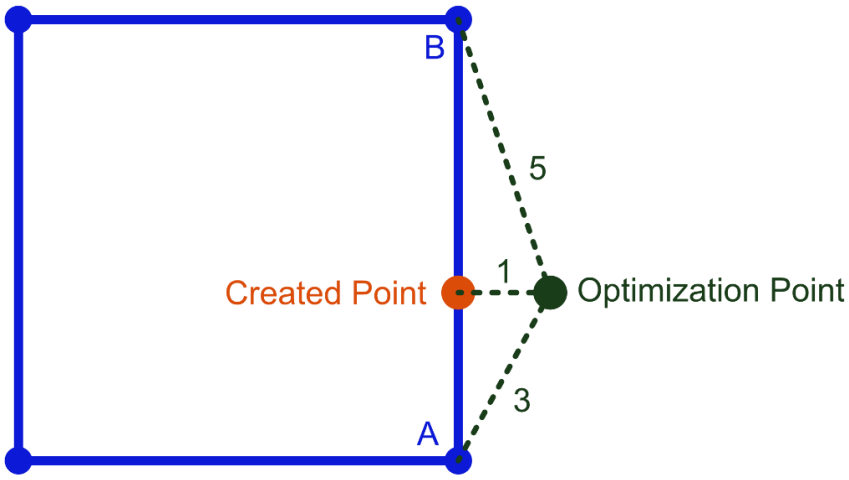


Figure 20: In this instance, a point along the line (shown as the orange created point), is the closest to the optimization point. However, this point isn't one of the polygon vertices and therefore can't be used as the start/stop point. In this scenario, point A will be the start/stop point.

For many objects, after the first path has been ordered, the next closest point using the current location yields the same result as the custom location point. This can be seen in Figure 8 and Figure 10a. The pathing result for the interior path when using next closest and custom point is the same. However, certain geometries can yield a different result based on the location of the custom point. Figure 21 shows one such instance where the start point of the interior path would not be the same with custom point and next closest. Similarly, the custom point strategy, unlike next closest, can be used to ensure that the start/stop for an object always stays in the same location regardless of path modifiers being added.

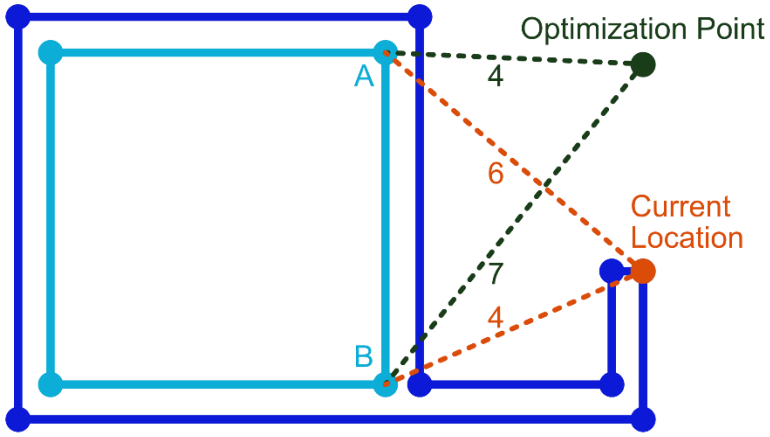
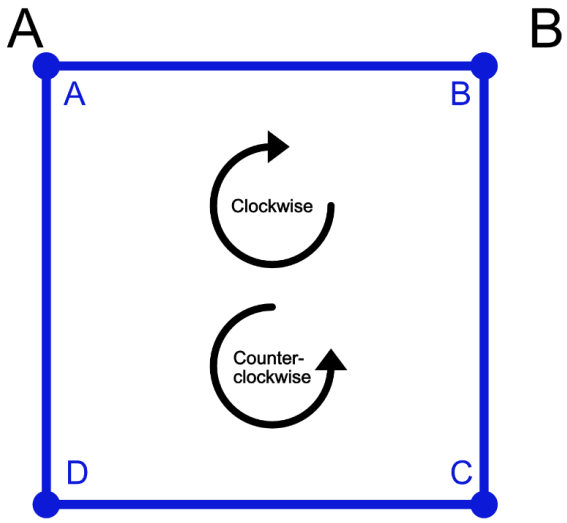


Figure 21: In this instance, the current location is closer to B meaning that point B would be used for the next closest optimization strategy. However, point A is closer to the optimization point and is therefore used for the custom point optimization strategy.

Clockwise and Counterclockwise

For a closed-loop path, the order of the points along the path, called the winding order, is either clockwise or counterclockwise. It can be useful to change the direction of the printing by adjusting the order of the points so that the material flows in a different direction through corners and starts/stops. This is a simple modification that takes the list of points and reverses the order as necessary, but does not change which point is the start point. Figure 22 shows a square with clockwise and counterclockwise point order.



Clockwise Point Order	Counterclockwise Point Order
A	A
B	D
C	C
D	B

Figure 22: A square illustrating clockwise and counterclockwise print directions (A). The resultant point order for the two print directions (B). Note that point A remains the first point for both directions.

New Optimization Strategies Implementation

The previous sections outlined a variety of optimization strategies for islands, paths, and points and talked about how they are implemented or could be implemented. Some of these are available as settings in various slicing packages, some are hard coded as the default operation, and others aren't available at all. To give users full control and flexibility with the optimization of their path planning, a new implementation has been developed that encompasses all of the previous strategies and more. This includes configurable island optimization strategy, region order, path optimization strategy, point optimization strategy, and point direction (CW vs CCW). All of the following implementations are immediately available via ORNL Slicer 2, an open-source slicing software developed by Oak Ridge National Laboratory (ORNL) [27].

Region Order

The typical slicing approach generates pathing by region but doesn't allow the user to define the printing order for the regions. For example, Cura allows inside out and outside in, but that can create confusion if multiple regions, such as skin and infill, exist on the same layer and are equidistant from the center. A more robust and customizable solution allows a user to define the order of the regions as well as individual optimization strategies for island order, path order, and point order. Figure 23 shows a list box that allows the user to rearrange the path printing order. This setting has no impact on the order of path generation, which still begins with perimeter and ends with infill, but instead impacts the order of pathing when output to the g-code which ultimately defines the order of printing.

Figure 23: A rearrangeable list of regions to configure the path printing order.

Island, Path, and Point Optimization

The crux of the issue with most slicing implementations is a lack of customization and separation for the optimization settings. For example, an inside out region order can't be implemented with an outside in path order because selecting inside out for these programs means printing paths and regions from inside out. By separating region and path order so that paths are outside in and regions are inside out, the amount of overfill in the center of the part can be minimized, but the outermost bead can still be printed in the proper position to maintain geometric accuracy. Other combinations such as a custom start stop point with a random path order or next closest path with next farthest point cannot be achieved.

These issues can all be overcome by adding more settings and user configuration such that different strategies can simultaneously be applied to island, path, and point optimization. Figure 24 shows the new implementation with the addition of drop-down settings menus for each of island order optimization, path order optimization, and point order optimization. Island order optimization options include next closest, next farthest, least recently visited, random, and custom point. Path order optimization options include next closest, next farthest, random, outside in, inside out, and custom point. Point order optimization strategies include next closest, next farthest, consecutive, random, and custom point. For each of the optimization categories, a custom point X and Y value can be defined. With the point optimization, a checkbox is available to enable a second custom point location. When two path points are enabled, the first point is used for all odd numbered layers and the second point is used for all even numbered layers. A setting is also available for the consecutive distance threshold for point optimization.

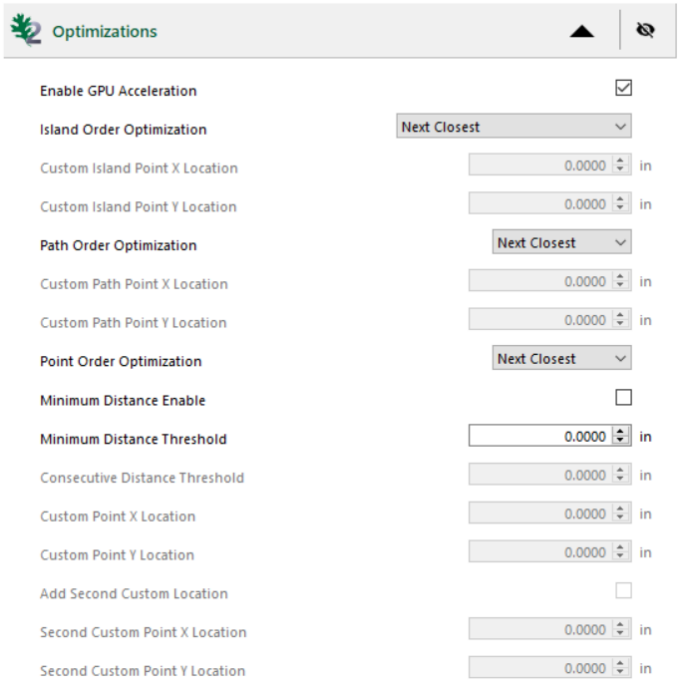


Figure 24: List of settings for the new implementation of optimization strategies.

The various custom point location settings allow the user to define an X and Y location within the build area to use as the optimization point. While this is effective, it can be hard for some users to understand where that point is in space with respect to the object being printed. To overcome this, a visualization element has been added such that a small sphere appears within the build area that can be positioned near the object (Figure 25). One sphere is available for each of the four optimization points: island point (dark blue), path point (dark green), first path point (light blue), and second path point (light green).

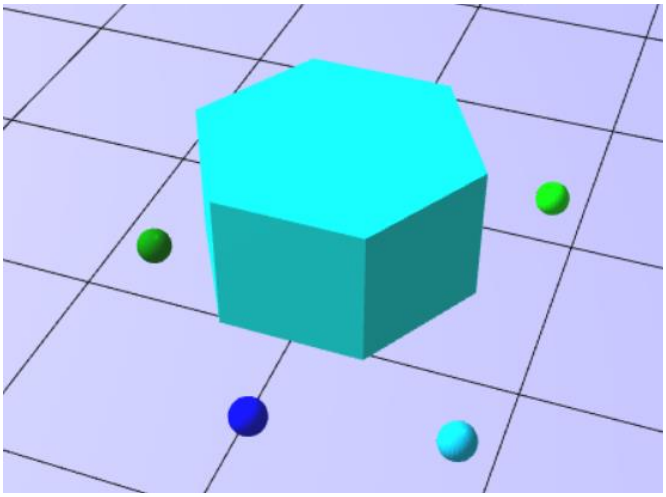


Figure 25: Hexagon object to be printed with four spherical optimization point markers positioned around the object.

One new setting not previously mentioned also exists for the next closest optimization: minimum distance threshold. The minimum distance is only available when the next closest path optimization

strategy is being used, and it allows the user to define a minimum distance from the current distance to start the next path. This is implemented by drawing a circle, with radius equal to the minimum distance threshold, around the current position (typically the end point of the current path) and finding the next closest point on the next path where the point is outside the circle. Figure 26 shows an example of the minimum distance threshold where the next closest point falls within the threshold and is therefore not used as the start point for the interior path.

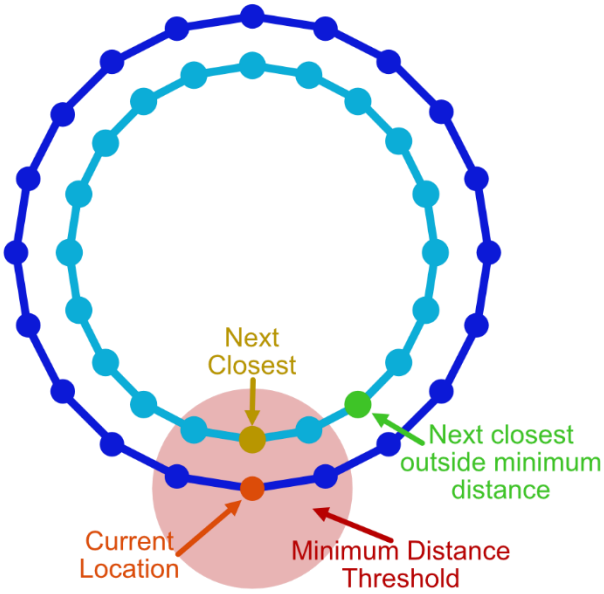


Figure 26: A circular toolpath showing the implementation of minimum distance threshold.

Point Direction (CW vs CCW)

The last missing optimization strategy is defining the print direction for closed loop paths. This is particularly important for perimeters and insets, and less so for closed loop infill such as the concentric pattern. By default, the order of points for a polygon boundary, the winding order, will be counterclockwise. For an interior hole, the order will be clockwise. It can be desirable to change this order to allow the material to extrude over the seam in a different direction. This is implemented by reversing the list of points in the closed loop path so that they print in the opposite direction. To allow the user to select this, two drop-down menus have been added: one for perimeters and one for insets (Figure 27). The default is to print with the standard winding order, but an optimization to reverse the order is available, and a third option exists to reverse the order on every other layer.

Figure 27: Drop-down menus for reversing the print direction on perimeters and insets.

Conclusion

This paper reviews many of the common path optimization and ordering strategies available in existing slicing packages such as inside out, custom point, CW vs CCW, and more. It also outlines some fundamental additions to these strategies to add more versatility. While not all edge cases can be covered

in path optimization due to the unlimited number of geometries that can be created for 3D printing, this paper aims to cover all standard examples with a few of the more common edge cases.

Further, an implementation was created, via the open-source slicing package ORNL Slicer 2, that adds significant flexibility and capability improvements for the user to have complete control over path ordering and optimization. This includes breaking out optimization strategies into three major categories: island order optimization, path order optimization, and point order optimization. Each of these optimization strategies comes with many options and some additional settings such as custom point locations and a minimum distance threshold. The new implementation also adds a region order optimization where the user can click and drag to rearrange the print order of the regions.

All strategies implemented in this paper are designed to run as part of the slicing process. The calculation for the optimizations is all automated, based purely on the geometry already within the slicer, and requires no human input beyond defining slicing settings. The strategies are designed to work with all geometries and not be limited to one-off use cases. The results of this work are immediately available via GitHub.

Future Work

This paper focused on defining a new approach to optimization settings within the slicing process. Future work will focus on different implementations of these optimization approaches, including how the different approaches can be combined to construct objects. This future work will also investigate how the optimization strategies impact print time by modifying total path distance, with an emphasis on non-extrusion path distance, called travel paths. Additional work will explore the impact of different optimization approaches on heat distribution and cooling rate. Work will also look at the computational expense of the strategies for various geometries.

References

1. Borish, Michael, et al. "Cross-Sectioning." *Motion and Path Planning for Additive Manufacturing* (2023): 71.
2. Borish, Michael, et al. "Travels, optimizations, and ordering." *Motion and Path Planning for Additive Manufacturing* (2023): 149.
3. “PRUSA3D/Prusaslicer: G-Code Generator for 3D Printers (RepRap, Makerbot, Ultimaker Etc..).” *GitHub*, Prusa Research, github.com/prusa3d/PrusaSlicer. Accessed 10 Feb. 2024.
4. “Ultimaker/CuraEngine: Powerful, Fast and Robust Engine for Converting 3D Models into G-Code Instructions for 3D Printers. It Is Part of the Larger Open Source Project Cura.” *GitHub*, Ultimaker, github.com/Ultimaker/CuraEngine. Accessed 10 Feb. 2024.
5. Li, Bai, Kexin Wang, and Zhijiang Shao. "Time-optimal trajectory planning for tractor-trailer vehicles via simultaneous dynamic optimization." *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015.
6. Li, Bai, and Zhijiang Shao. "An incremental strategy for tractor-trailer vehicle global trajectory optimization in the presence of obstacles." *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2015.
7. Li, Bai, et al. "Optimization-based maneuver planning for a tractor-trailer vehicle in a curvy tunnel: A weak reliance on sampling and search." *IEEE Robotics and Automation Letters* 7.2 (2021): 706-713.

8. Oliveira, Rui, et al. "Optimization-based on-road path planning for articulated vehicles." *IFAC-PapersOnLine* 53.2 (2020): 15572-15579.
9. Liang, Chuandong, et al. "Multi-node path planning of electric tractor based on improved whale optimization algorithm and ant colony algorithm." *Agriculture* 13.3 (2023): 586.
10. Wang, Liang, et al. "Path tracking control of an autonomous tractor using improved Stanley controller optimized with multiple-population genetic algorithm." *Actuators*. Vol. 11. No. 1. MDPI, 2022.
11. Zhao, Xin, et al. "An obstacle avoidance path planner for an autonomous tractor using the minimum snap algorithm." *Computers and Electronics in Agriculture* 207 (2023): 107738.
12. Han, Xiao, Yanliang Lai, and Huarui Wu. "A path optimization algorithm for multiple unmanned tractors in peach orchard management." *Agronomy* 12.4 (2022): 856.
13. Dreifus, Gregory, et al. "Path optimization along lattices in additive manufacturing using the chinese postman problem." *3D Printing and Additive Manufacturing* 4.2 (2017): 98-104.
14. Kim, Seokpum, et al. *Graded infill structure of wind turbine blade accounting for internal stress in big area additive manufacturing*. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2018.
15. Lechowicz, Piotr, et al. "Path optimization in 3D printer: algorithms and experimentation system." *2016 4th International Symposium on Computational and Business Intelligence (ISCBI)*. IEEE, 2016.
16. Ma, Zongfang, et al. "an approach of path optimization algorithm for 3D concrete printing based on graph theory." *Applied Sciences* 12.22 (2022): 11315.
17. Fok, Kai-Yin, et al. "An ACO-based tool-path optimizer for 3-D printing applications." *IEEE Transactions on Industrial Informatics* 15.4 (2018): 2277-2287.
18. Fok, Kai-Yin, et al. "Accelerating 3D printing process using an extended ant colony optimization algorithm." *2018 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 2018.
19. Cheong, Kah Jun. *Path Optimization For Cooperative Multi-Head 3d Printing*. Diss. UTAR, 2020.
20. Hashim, Saipul Azmi Mohd, Mohd Zaki Abdul Manap, and Mohd Kamarul Ariffin Salaudin. "OPTIMIZATION IN CONTROLLING EXTRUDATE SWELL 3D PRINTING OF TISSUE ENGINEERING SCAFFOLD USING ANT COLONY OPTIMIZATION." *Young* 25: 3.
21. Sridhar, Sundarraj, et al. "A bioinspired optimization strategy: to minimize the travel segment of the nozzle to accelerate the fused deposition modeling process." *Bulletin of the Polish Academy of Sciences. Technical Sciences* 71.4 (2023).
22. Liu, Hao, et al. "Minimizing the number of transitions of 3d printing nozzles using a traveling-salesman-problem optimization model." *International Journal of Precision Engineering and Manufacturing* 22 (2021): 1617-1637.
23. Dong, Yuwei, and Bo Hu. "Optimized Control Method for Fused Deposition 3D Printing Slice Contour Path Based on Improved Hopfield Neural Network." *Applied Artificial Intelligence* 37.1 (2023): 2219946.
24. “3D Printing Market: 10 Million 3D Printers to Sold by 2030 Thanks to Declining Cost and Advancing Technology.” *GlobeNewswire News Room*, SkyQuest Technology Consulting Pvt. Ltd., 8 Aug. 2022, www.globenewswire.com/en/news-release/2022/08/08/2494063/0/en/3D-printing-Market-10-million-3D-Printers-to-Sold-by-2030-Thanks-to-Declining-Cost-and-Advancing-Technology.html.
25. Furqan, Mhd, Rifki Mahsyaf Adha, and A. Armansyah. "Determination of The Closest Path Using The Greedy Algorithm." *IJISTECH (International Journal of Information System and Technology)* 7.5 (2024): 333-340.
26. Tao, Yufei, et al. "An efficient cost model for optimization of nearest neighbor search in low and medium dimensional spaces." *IEEE Transactions on Knowledge and Data Engineering* 16.10 (2004): 1169-1184.
27. Roschli, Alex, Borish, Michael, Barnes, Abigail, Wade, Charles, Crockett, Breanne, White, Liam, and Adkins, Cameron. *ORNL Slicer 2 - Open Source Copyright.* Computer Software. https://github.com/ORNLSlicer/Slicer-2. USDOE Office of Energy Efficiency and Renewable

Energy (EERE), Energy Efficiency Office. Advanced Materials & Manufacturing Technologies Office (AMMTO). 06 May. 2024. Web. doi:10.11578/dc.20240520.1.