TRAvel Slicer: Continuous Extrusion Toolpaths for 3D Printing

Jaime Gould
University of New Mexico
Albuquerque, NM, USA
egould@unm.edu

Camila Friedman-Gerlicz
University of New Mexico
Albuquerque, NM, USA
friedmangerlicz99@unm.edu

Leah Buechley
University of New Mexico
Albuquerque, NM, USA
buechley@unm.edu

(a) TRAvel Slicer ***(b)*** Cura ***(c)*** TRAvel Slicer ***(d)*** Cura

Figure 1: Four artifacts printed on a Direct Write (DW) 3D printer. TRAvel Slicer creates a continuous extrusion toolpath that results in successful prints of complex models, (a) and (c). Toolpaths generated with a traditional slicer result in failed prints for the same models, (b) and (d). Here, travel movements are erroneously printed because extrusion cannot be quickly stopped and restarted on DW printers.

ABSTRACT

In this paper we present Travel Reduction Algorithm (TRAvel) Slicer, which minimizes travel movements in 3D printing. Conventional slicing software generates toolpaths with many travel movements–movements without material extrusion. Some 3D printers are incapable of starting and stopping extrusion and it is difficult to impossible to control the extrusion of many materials. This makes toolpaths with travel movements unsuitable for a wide range of printers and materials.

We developed the open-source TRAvel Slicer to enable the printing of complex 3D models on a wider range of printers and in a wider range of materials than is currently possible. TRAvel Slicer minimizes two different kinds of travel movements–what we term Inner- and Outer-Model travel. We minimize Inner-Model travel (travel within the 3D model) by generating space-filling Fermat spirals for each contiguous planar region of the model. We minimize Outer-Model travel (travels outside of the 3D model) by ordering the printing of different branches of the model, thus limiting transitions between branches. We present our algorithm and software and then demonstrate how: 1) TRAvel Slicer makes it possible to generate high-quality prints from a metal-clay material, CeraMetal, that is functionally unprintable using an off-the-shelf slicer. 2) TRAvel Slicer dramatically increases the printing efficiency of traditional plastic 3D printing compared to an off-the-shelf slicer.

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0628-8/24/10…$15.00
https://doi.org/10.1145/3654777.3676349

CCS CONCEPTS

• **Applied computing** → **Computer-aided design**; • **Computing methodologies** → *Computer graphics.*

KEYWORDS

Slicing Software, g-code generation, Traveling Salesman Problem, Direct Write 3D Printing, Clay 3D Printing, Fused Filament Fabrication

ACM Reference Format:
Jaime Gould, Camila Friedman-Gerlicz, and Leah Buechley. 2024. TRAvel Slicer: Continuous Extrusion Toolpaths for 3D Printing. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3654777.3676349

1 INTRODUCTION

Direct Write (DW) 3D printing—also known as Robotcasting—is an approach to 3D printing where parts are built up from an extruded paste [2]. Popular direct write printers include clay 3D printers, which build ceramic objects from soft clay [7] and food printers, which create edible objects from a variety of edible pastes including chocolate [28] [34] [26] and dough [46] [31][6]. DW printers are taking an increasingly prominent role in digital fabrication research as they become more accessible—commercial models like the Eazao Zero [11] and Tronxy Moore [43] now sell for under $1000 USD—and the range of materials that can be printed with them expands.

DW printers are powerful platforms for material design and exploration because they are capable of extruding almost any clay or paste. They have been one of the primary platforms used to

explore the printing of biomaterials including cellulose, starch, gelatin, calcium alginate [39], coffee grounds [38], and mussel shells [35]. Direct write printers are also used to print cement [10] [15] adobe [9] [16] and other experimental materials including metal [5] and glass-based inks [36].

Despite the many advantages of DW printing, the process presents significant challenges. In particular, traditional slicing algorithms depend on travel movements—movements taken while extrusion is halted—to generate viable toolpaths. Most off-the-shelf slicing software, including Cura [4] and Simplify3D [40] were developed for Fused Filament Fabrication (FFF) 3D printers and thermoplastic materials. It is easy to start and stop the extrusion of thermoplastic materials on FFF printers. However, many direct write 3D printers, including most PotterBot [1] models, do not support quick starts and stops in extrusion; paste flows continuously out of the printer nozzle for the duration of a print. Materials with non-linear rheology may continuously flow out of a printer nozzle for the duration of a print even when they are employed in a DW printer that supports extrusion control. For these printers and materials, toolpaths that include travel moves lead to failed prints. Material continues to extrude during travel moves and builds up on or around the desired part, as can be seen in Figure 1 (b) and (d).

TRAvel Slicer was designed to overcome these challenges by reducing or eliminating travel movements in toolpaths, thus enabling the DW 3D printing of complex geometries in a wide variety of materials.

In this work we demonstrate TRAvel Slicer's ability to generate toolpaths that produce viable prints for intricate geometries in continuous extrusion materials that otherwise fail to print successfully when sliced with a traditional slicer, Cura. We slice five models with TRAvel Slicer and Cura, and print them in a bronze-based paste with non-linear rheology [5] on an Eazao Zero [11], and in Polylactic Acid (PLA) on a Creality Ender 3D [8]. The main contributions of this work are:

- TRAvel Slicer produces toolpaths that result in viable 3D printed artifacts in continuous-extrusion materials that otherwise fail to print.
- TRAvel Slicer dramatically improves printing efficiency by decreasing print time and material waste as a result of reducing the number and length of the travel movements.
- We combine two search spaces and methods of travel path minimization that have not previously been integrated together.
- We expand upon prior methods continuous path infill, improving the visual appearance of the outer surface of 3D printed artifacts by separating the wall(s) from the infill.
- We leverage an understanding of the 3D printer nozzle dimensions to reduce the search space of model layer ordering.

2 RELATED WORK

2.1 G-code Generation

G-code is the language that provides instructions to most digital fabrication machines, including 3D printers. A g-code file for 3D printing consists of a list of spatial coordinates–that determine the path followed by the print head–coupled with machine-specific commands that control parameters like speed and material extrusion. Slicers generate g-code files based on input geometries and printer parameters. The g-code file, when interpreted by a 3D printer reproduces the input geometry. Commercial slicers like Cura and Simplify3D can be configured with DW printer parameters like nozzle size and build volume. However, their toolpaths are limited to non-continuous infill patterns–all standard infill patterns require travel movements–and sequential layer ordering with respect to z–all areas of a model at a given height are printed before the machine moves up in z. The only continuous toolpaths that can be generated with these slicers are "vase mode" prints, in which a continuous spiral path follows the outer contour of a model. This is a limited feature that can only produce single-walled prints for non-branching solid models.

Bypassing traditional slicers and generating g-code files directly enables fine-grained control over printer behavior. There is a robust body of previous literature that explores custom g-code generation. Custom g-code generators have been developed to print textiles [13], hair [27], and foam-like [29] "meta-materials" on FDM printers. They have also been employed to generate prints with novel surface textures [37, 42]. Silkworm is a Rhino and Grasshopper-based tool that allows for the direct generation of toolpaths via Grasshopper programming–lines that are generated in Grasshopper are turned into g-code commands [33]. CoilCAM [3] is a Grasshopper program developed to support the design of 3D printable ceramic vessels based on a series of mathematical operations performed on cylinders. All of these tools generate g-code files, but they do not function as slicers. They do not slice arbitrary input geometry.

Xylinus [19] is a slicer implemented in Grasshopper that reproduces the slicing behavior of commercial tools. Its utility stems from its integration of slicing with Grasshopper programming. A user can generate geometry and slice that geometry in a single application and program.

Vespidae [14] is what might be termed a meta-slicer. This tool functions as a standard slicer that also provides opportunities to directly edit g-code files. For example, after slicing a model, a user can replace an infill pattern on a given layer with a custom hand-drawn pattern. Vespidae also allows users to manually re-order the printing of different sections of a model. Though it is possible to manually control the printer toolpath with some of these previous tools, none of them provide travel minimization options.

2.2 Travel Minimization

Some traditional slicers such as Simplify 3D [41] and Prusa [24] have a functionality called sequential printing. Sequential printing allows the user to print different models or copies of models in a non-planar positive-z order, but it does not account for collisions and requires the user to manually consider path planning between separate objects on the print bed. Additionally, sequential printing only allows the user to print layers linearly within a single model, it does not account for branching structures within the object itself.

Jiang and Ma compiled and reviewed work related to path planning strategies for 3D printing [23]. They discuss the need for a path-planning platform that allows the user to prioritize different strategies of travel path optimization depending on specific

goals. They make the point that different optimization approaches–including travel minimization–may be employed to achieve different goals. For instance achieving a goal of high print quality may require a different approach than improving print time efficiency.

Hergel et al. [18] approach travel minimization in DW clay 3D printing by generating a large supportive shell that encloses the model. All travel movements occur in this shell. The shell functions as a support structure for the model and is removed after the print is complete. Their strategy allows the printer to continuously extrude while maintaining a sequential layer ordering with respect to height. They demonstrate how their slicer enables the DW printing of a range of ceramic artifacts. The downside of this approach is that their toolpaths require the printing of a significant amount of extra material (for the shell) and a post-processing step to remove the shell from the model.

Zhong et al. [49]'s algorithm decomposes the surface of a model in order to print its outer shell with as few travel movements as possible, and similarly takes height-dependency, overlap, and nozzle height constraint into consideration. However it only prints hollow shells of models with no infill.

2.2.1 Reducing Travel Between Regions. Work done in minimizing travel between branches of a part–minimizing what we term Outer-Model travel–includes Liu et al.'s approach [30], in which they create collections of "sub-parts" representing stacks of neighboring planar regions that can be printed together in a non-planar order. They implement metaheuristics to find reasonable paths between these sub-parts. (Metaheuristics are employed to reduce the time spent searching.) They test their algorithm by printing two different models using PLA on a standard plastic 3D printer and metal on a selective laser sintering machine. Kaplan et al. lengthened a traditional FFF 3D printer nozzle in order to increase the space between the tip of the nozzle and the large heat block of the print head. This similarly allows for the printing of neighboring planar regions in a non-planar order. The lengthened nozzle supports non-planar ordering because it reduces the likelihood of collisions between the print head and already printed layers [25]. They tested their algorithm by printing a number of branching models in PLA and TPU on standard plastic 3D printers.

2.2.2 Reducing Travel Within a Planar Region. Work done in minimizing travel within planar regions–what we term Inner-Model travel–includes approaches that minimize travel between walls and infill via toolpath *reordering*. Gupta et al. [17], Flemming et al.[12], and Hu et al. [20] achieve a modest reduction in travel by reordering g-code commands for a individual planar regions within a model. More useful techniques *eliminating* travel movements within planar region altogether. Zhao et al. employ a continuous low-curvature path composed of connected Fermat spirals that begin and end at the same location [48]. Zhai et al.'s continuous path is generated by decomposing porous structures with Voronoi diagrams and then filling these subdomains with Fermat spirals [47]. Xia et al. generate "hybrid" continuous paths by connecting walls to regions of zigzag infill [45].

Our algorithm integrates and improves on these two categories of travel reduction techniques. We develop an approach that was informed by Kaplan et al.'s [25] and Liu et al.'s [30] prior work to minimize travel between branches of a model and employ and extend Zhao et al.'s connected Fermat spirals [48] to minimize travel movements within a closed region of the model at a given layer.

2.3 Connected Fermat Spirals

A connected Fermat spiral [48] is a continuous space-filling path of a complex shape that starts and stops at the same point and is created by spiraling and connecting its equidistantly spaced inner isocontours. This path completely fills the shape without the need for travel movements between different areas of infill (see Figure 2 for an example). More information about how this path is generated can be found in Appendix D.

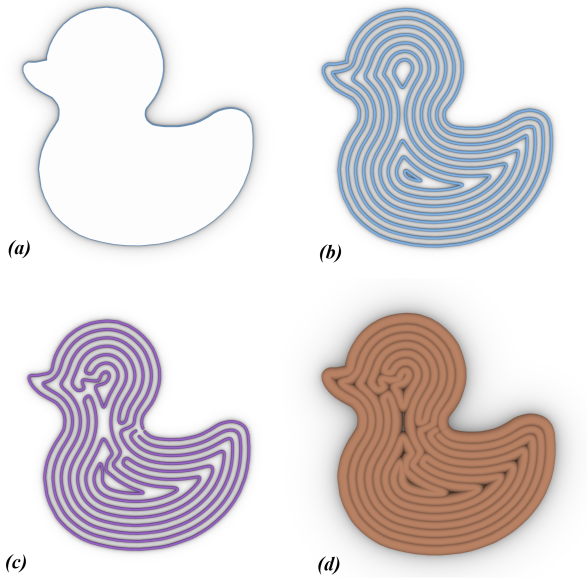


Figure 2: An example of a space-filling continuous connected Fermat spiral toolpath of a complex shape. (a) A duck shape. (b) The equidistantly spaced isocontours of the shape, with two local "maxima", or separated inner contour regions. (c) The connected Fermat spiral path. (d) A visualization of the path in the material with the specified extrusion width.

3 OVERVIEW

Travel Reduction Algorithm (TRAvel) Slicer is a general purpose slicing tool that takes a 3D model as input and generates a toolpath in the form of a g-code file with few to no travel movements (see Figure 3). TRAvel Slicer is an open source Python library written to be used in Grasshopper and Rhino3D and is available in a public Github repository [22]. (See Appendix A for more details.)

Conventional 3D printer software slices the model along the z-axis at intervals of $distance = l_{height}$ parallel to the x-y plane, where l_{height} is the layer height and is determined by nozzle size, desired print resolution, and material properties. The slices are printed linearly with respect to the z-axis; all isolated regions at layer l_i are printed before the printer nozzle advances to the next $z = l_{i+1} \cdot l_{height}$. We define isolated regions as the following:

Definition 3.1 (Isolated Region). A single closed planar area bounded by an outer contour that is generated when the model intersects an x-y–parallel plane. The region may also be bounded by inner

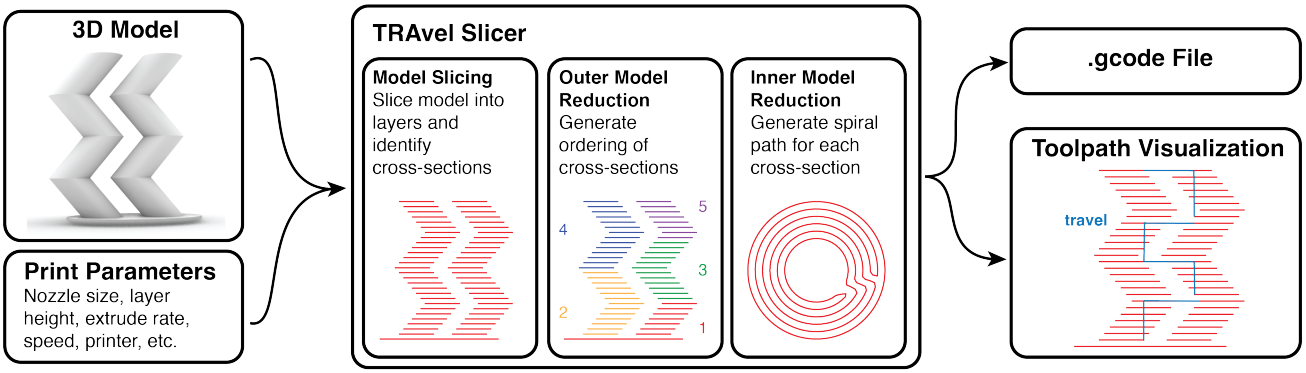


Figure 3: An overview of the TRAvel Slicer algorithm and software.

contours if there are holes in the model (see example in Figure 12). There may be multiple isolated regions at any given layer height due to branching in the 3D model.

To build the model, 3D printers deposit material within these isolated planar regions (infill), and along the contours defining their boundaries (walls), building the model up layer by layer.

Travel – or transfer – movements can occur within a given isolated region (see Figure 4 (a)), and between isolated regions when the 3D model contains branching structures (see Figure 4 (b)).

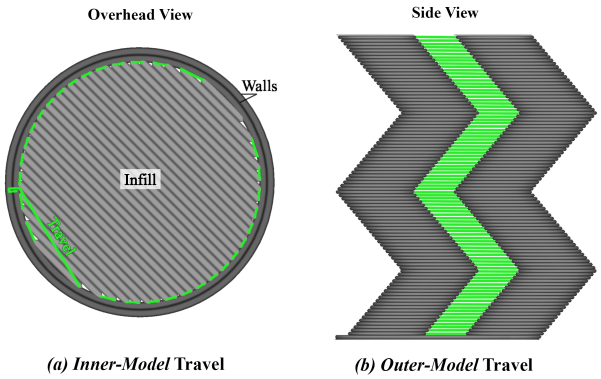


Figure 4: Types of travel movements in 3D printing, shown in green. (a) Inner-Model travel within an isolated region. (b) Outer-Model travel between isolated regions.

The travel path reduction search can be separated into these two loosely connected problem spaces, which we refer to as Inner-Model and Outer-Model Travel Reduction.

Definition 3.2 (Inner-Model Travel). Given a single closed planar contour belonging to a slice of the model at a given layer l_i, we define Inner-Model travel as any travel movements within this contour (Figure 4 (a)).

Definition 3.3 (Outer-Model Travel). We define Outer-Model travel as any travel movements between isolated closed regions. Outer-Model travel is either due to a branching 3D model or to multiple models present on the print bed (Figure 4 (b)).

Our algorithm first slices the model into layers (see Model Slicing in Figure 3). Layers with more than one isolated region (layers with branching structures) are split. We then generate a printable ordering of these nodes that reduces travel between them, thus reducing[1] Outer-Model travel (see Outer-Model Travel Reduction in Figure 3). Once this ordering has been generated, our algorithm generates a continuous travel path for each isolated region, reducing Inner-Model travel (see Inner-Model Travel Reduction in Figure 3). These two steps of our algorithm are described in detail in the next two sections. The end result is a visualization of the printer toolpath and a g-code file.

4 PARAMETERS AND MODEL SLICING

Our algorithm begins by slicing a 3D model into layers. Slicing is based on parameters associated with the printer and/or print material that is being used.

4.1 Parameters

TRAvel Slicer employs the same parameters that traditional slicers do, including layer height, referred to as l_{height}, and extrusion width, referred to as w_e. The extrusion width of the nozzle (its inner diameter) determines the spacing of the walls and infill toolpath.

TRAvel Slicer also takes two additional parameters for Outer-Model travel reduction, nozzle width (noz_{width}) and nozzle height (noz_{height}). Nozzle width represents the maximum outer diameter of the nozzle. The nozzle height is the measurement representing the vertical clearance from the tip of the extruder to the rest of the printer (see Figure 5).

4.2 Slicing

Our algorithm slices a 3D model into layers based on these parameters. The number of layers is determined by the layer height (l_{height}) and the height of the model.

Each slice consists of at least one closed curve. These curves are generated by the intersection of the model with a plane at height $z = l_i \cdot l_{height}$. Once we have generated these curves, we group them into isolated regions. An isolated region may consist of a single

[1]Note that our algorithm does not necessarily generate an *optimal* print ordering in all cases. However, it reliably generates a very good approximation.

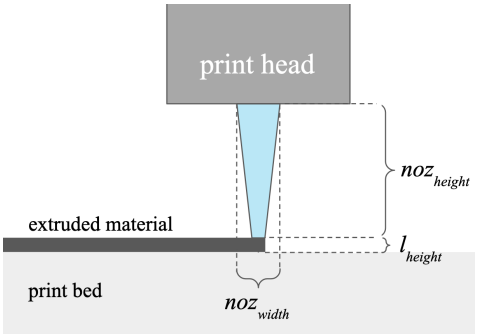


Figure 5: A side view diagram of a 3D printer nozzle and corresponding parameters layer height (l_{height}), nozzle width (noz_{width}), and nozzle height (noz_{height}).

closed curve if the region is solid. Alternatively it may consist of multiple outer curves if the region has holes (see Figure 12).

We then pass these isolated regions to Outer-Model travel reduction to generate a non-planar printable order.

5 OUTER-MODEL TRAVEL REDUCTION

Outer-Model travel reduction generates a print ordering for the isolated regions that minimizes travel while ensuring that the print head does not collide with the model during printing.

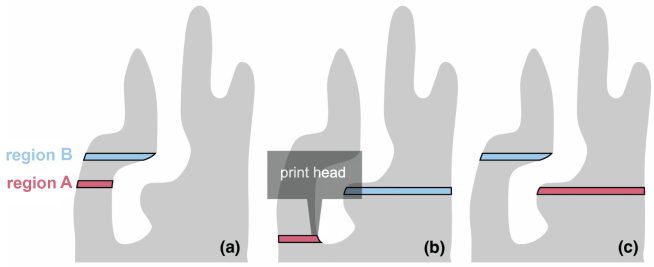


Figure 6: Constraints that dictate layer ordering; (a) height dependency, (b) nozzle height restriction, and (c) collision avoidance. Region A must be printed before region B in all three of these instances.

The ordering must fulfill the following constraints. First, region B cannot be printed before region A if region A is *underneath* region B, Figure 6 (a) and (b). Note that in Figure 6 (b), the printer cannot return to print region A after printing region B because of the nozzle height restriction; the armature will collide with the previously printed part of the model. Lastly, note that in Figure 6 (c), region A must be printed before region B to avoid a collision due to overlap.

To first satisfy the height-dependency constraint, we begin constructing our search space with a height-dependence tree.

5.1 Height-Dependence Tree

TRAvel Slicer's Outer-Model travel optimization begins by constructing height-dependence tree H (see Figure 7). In a branching 3D model, the intersection of the model and x-y parallel plane at some layer l_i can result in multiple isolated regions (each corresponding to at least one closed curve) $C_i = \{c_i^1, c_i^2, ..., c_i^n\}$. In order to simplify H and later reduce the number of necessary comparisons when constructing a path, we compare the isolated regions within each layer height to determine potential overlap, using the approach described in Appendix B. If any two isolated regions $c_i^j, c_i^k \in C_i \mid j \neq k$ are a distance of less than or equal to $noz_{width}/2$ from each other they cannot be printed in a non-planar order, therefore we group them in the same node H.

Given consecutive layers i and $i+1$ and corresponding sets of isolated regions C_i and C_{i+1}, we define the height dependence of a given isolated region $c_{i+1} \in C_{i+1}$ on region $c_i \in C_i$ to be true if the their projections onto the same plane results in overlapping areas.

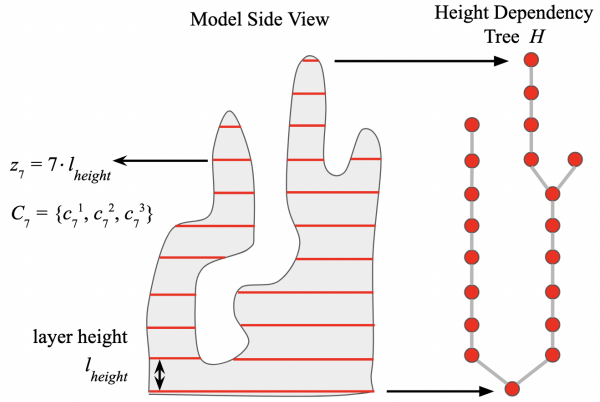


Figure 7: Construction of Height-Dependency tree H. Every isolated region (red lines) c_i becomes a node in H if $distance(c_i^j, c_i^k) \geq noz_{width}/2$ for all other $c_i \in C_i, j \neq k$.

Note that this could technically result in multiple nodes satisfying the definition of a parent for a subsequent node, producing a Directed Acyclic Graph (DAG) rather than a tree. In order to reduce the number of comparisons in our algorithm later on, we arbitrarily assign the first node that satisfies this constraint as the parent, resulting in a tree data structure.

5.2 Nozzle-Height and Overlap Tree

Once we have created a tree of isolated regions that captures height dependencies, we begin to identify and eliminate potential collisions that may occur during printing.

5.2.1 Nozzle-Height Tree. We begin by leveraging Kaplan et al.'s insight that the nozzle's width is very small compared to the print head dimensions [25], also utilized in Liu et al.'s [30] and Zhong et al.'s work [49]. Our algorithm avoids a large category of collisions by requiring that all layers within a nozzle height are printed before layers above that nozzle height—everything within a given height range $z_o \leq z \leq z_o + noz_{height}$ must be printed before the toolpath can advance any higher. To capture this constraint, we generate a new tree H' whose nodes v' contain contiguous sets of nodes $\{v_i, v_{i+1}, ..., v_{i+k}\}$ from H in order to reduce the size of the shortest toolpath search space in section 5.3. The nodes in H' must satisfy the following conditions as can be seen in Figure 8:

- Any two nodes $(v_k, v_m) \in v'$ where $k \neq m$ are lineal descendants; v_k is either a descendant or ancestor of v_m, but they

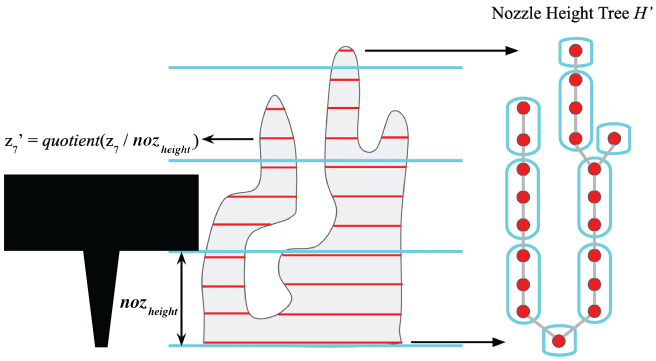


Figure 8: The construction of H' from H. The number of nodes $|H'| \leq |H|$.

cannot be "siblings" or "cousins" within H'. This is to satisfy the condition that they can be printed consecutively without travel movements.

- For any $v_i \in H$, its regions have the same height z_i. We compute the segmented height z_i' of v_i with respect to the nozzle height of the printer: $z_i' = quotient(\frac{z_i}{noz_{height}})$. All $v \in v'$ must have the same segmented height z'.
- If a given node v has multiple children in H, it must be the last node in its node v' of H'. Relationships between a parent node with more than one child and its children H are preserved in H' (for example, in Figure 8 the very first layer contains a single node, but layer 1 contains two nodes in H. Because this node in H has two children, it remains a single node in H').

5.2.2 Sub-Divide by Overlap. There can still be collisions when traveling between indirectly related nodes in H' within a given z' height if any subset of the set of regions within a node $v' \in H'$ overlaps above and below any set of regions in another node $u' \in H'$. We perform an additional overlap comparison of each node v' with its "siblings" and "cousins" at the same z' height. If any region within v' overlaps u' both above and below its region, we split v' at the transition into two new nodes (see Figure 9).

The nodes in this tree indicate which groups of layers can be printed together without generating collisions. However, we still need to determine a valid ordering for these layer groups.

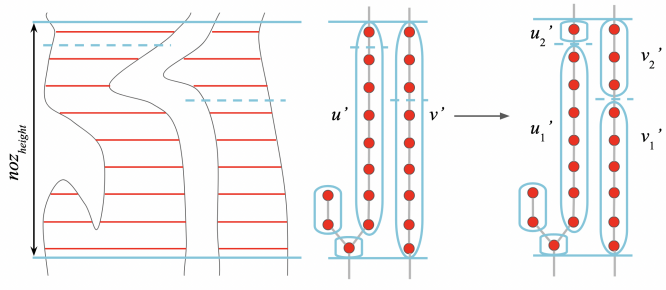


Figure 9: The comparison of indirectly related nodes u' and v' in H', resulting in a subdivision of u' into u_1', u_2' and v' into v_1', v_2' due to overlap.

5.3 Connected Graph and Hamiltonian Path

Once we have our sub-divided H' we can search for a valid travel path between the nodes at each given nozzle height section $0 \leq z' \leq quotient(\text{model height}/noz_{height})$. We iterate through z', constructing a subset $S \subset H'$ where $S = \{v_i' \mid z_i' = z', v_i' \in H'\}$.

We create a directed weighted graph $G = (V, E)$, where there exists a node in V for every node in S, and an edge (v_k, v_m) if v_k is the parent node of v_m in H' or v_k is an indirectly related node and v_m does not occlude it. The weight of these edges is equal to the distance between the last contour in v_k and the first contour in v_m. We search for a Hamiltonian path within the graph, beginning with any node containing contours at $z = z' \cdot noz_{height}$. These nodes become the start nodes for our Hamiltonian Path Search, and their initial weights are equal to the distance from the last node in the path found in the previous nozzle height segment $z = z' - 1$. Travel minimization at each height segment is dependent on the path found in the previous segment.

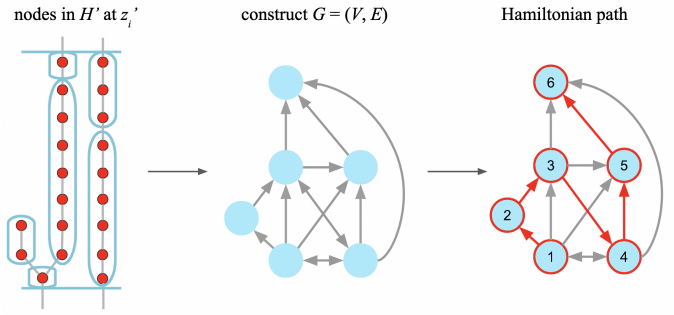


Figure 10: The construction of graph G from subset $S \subset H'$ and resulting Hamiltonian path.

TRAvel Slicer performs a depth-first search on G in order of minimum weighted edges to find a Hamiltonian path. When adding a node to the path, we account for height dependency by verifying that the parent of that node in H' has been printed. We then check that no node in the current path occludes the node we are attempting to add, otherwise the path is not viable due to overlap.

We iterate the search until either: we have traversed the entire tree, we have found a specified number of Hamiltonian paths, or we hit a specified search time limit. While iterating, a path search in progress is halted and discarded if the summation of its edge weights exceeds the minimum total weight of any path already found.

5.4 Outer-Model Travel Reduction Output

Once a complete path of the model has been found and TRAvel Slicer has generated an ordered sequence of isolated regions with minimized travel movements, there is an extra step in which we verify that a path parallel to the x-y plane at height equal to the higher of the two nodes we are traversing between will not result in a collision with the bounding boxes of nodes that have previously been printed. If it will, we traverse between them at the highest z that has been printed so far, satisfying the requirement that the path does not collide with the model (see Figure 11).

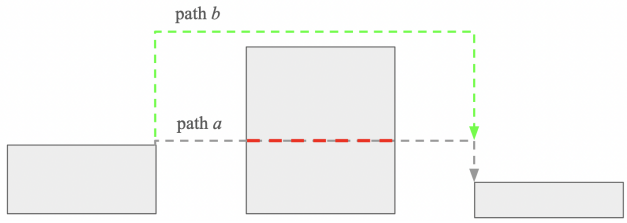


Figure 11: Side view of traversal between nodes in H' during printing. Path a collides with a previously printed part of the model, so TRAvel Slicer uses path b.

The result of this stage of our algorithm is an ordering of isolated regions that minimizes Outer-Model travel and ensures that no collisions will occur. Note that we have not yet generated a toolpath, only an ordering of printable isolated regions. The actual toolpath is generated in the next step of our algorithm.

5.5 Improvements on Previous Work

TRAvel Slicer's Outer-Model travel minimization significantly reduces the search space and improves performance by: 1) combining information about Inner- and Outer-Model travel to optimize travel paths between regions, 2) generating a searchable graph based on height-dependent nodes that are sectioned by nozzle height (Section 5.2) and, 3) identifying overlaps through curve comparison as opposed to bounding box or area centroid comparison (Section 5.1 and Appendix B)).

TRAvel Slicer waits to generate the infill path of a region until after the Outer-Model travel reduction step, allowing us to start and stop extrusion paths based on the ordering of the sections from this step. We are able to improve performance by integrating the two approaches. In contrast, NozMod's [25] algorithm works by reordering the horizontal printing of isolated regions–the horizontal toolpaths are generated by traditional slicing tools. This limits their ability to optimize the travel paths between regions. Furthermore, while Kaplan et al. construct a tree (grouped into nozzle-height chunks), they do not construct a graph of possible paths or search for a shortest path through a graph. Instead, they opt for a depth-first search search on their tree taking overlap into account. The toolpath they generate is viable and does reduce travel movements, but does not search for the shortest path throughout the entire model.

Liu et al. [30] does search for a shortest path. However, they provide limited information about how they construct their searchable graph. What is clear from the paper is that their search space ends up being so large it requires metaheuristics to solve. Our algorithm produces a much smaller graph and corresponding search space. Additionally, they group regions according to their area centroids, which leads to inaccuracies when dealing with complex models that include nested geometry or interlocking curves (see Appendix B). The algorithm they describe generates erroneous travel movements between these regions. Our algorithm identifies and minimizes travel movements in these especially difficult cases.

6 INNER-MODEL TRAVEL REDUCTION

Once Outer-Model travel has been reduced and returned a print order for the isolated regions of the model, we turn to generating a continuous toolpath for each region. We compute a connected Fermat Spiraling [48] for each of these isolated regions in order to minimize travel paths generated due to printing and infill. A Fermat spiral is a continuous space filling curve that first spirals in to the center of an arbitrary shape and then spirals back out.

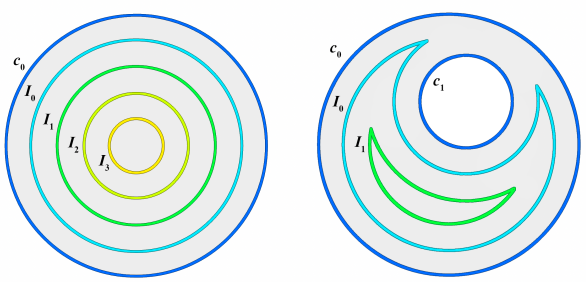


Figure 12: The region on the left has a clear parent-child relationship between the outer curve c_0 and the subsequent isocontours: $c_0 \rightarrow I_0 \rightarrow I_1 \rightarrow I_2 \rightarrow I_3$. The region on the right has ambiguity with respect to parent-child relationships. c_1's parent can be either $c_0 \rightarrow c_1$ or $I_0 \rightarrow c_1$.

We use connected Fermat spirals as our infill toolpath in TRAvel Slicer due to their flexibility in setting the start and end point of the path. Controlling the location of this point allows us to set the "seam" of an individual branch, and start our path at the closest location on an isolated region to our previous location, reducing the length of the Outer-Model travel between branches and avoiding depositing excess material over a previously printed layer due to continuous extrusion.

Before generating a connected Fermat spiral path, we connect outer walls to their inner walls in isolated regions with holes and generate isocontours for each region. Isocontours are equidistantly spaced contour curves within an isolated region that form the basis of our Fermat spiraling.

6.1 Regions with Holes: Connect Inner and Outer Contours

Generally, isocontour generation in regions with holes–regions that contain more than one curve–is done in two directions. Both the outer and inner curves generate contours that march toward each other and meet in the middle of the region, forming a single closed curve where they intersect [44]. This can result in ambiguity in the order of parent-node and child-node relationships when generating the tree data structure necessary for connecting contours within the region. In regions without holes the order is very simple; the outermost contour is the parent node, and the next isocontour or isocontours within it are its children. In regions with holes there may need to be an additional search to determine the closest neighbor of an isocontour and thus its parent curve (see Figure 12).

In order to simplify these relationships, we connect the outer and inner contours of closed regions before generating isocontours. By connecting contours, we create a single closed curve for each isolated region. This approach simplifies subsequent steps in our algorithm. It also allows us to print the walls of the isolated region separately from the infill.

We connect inner and outer contours by creating a fully connected weighted graph $G = (V, E)$, where there exists a vertex

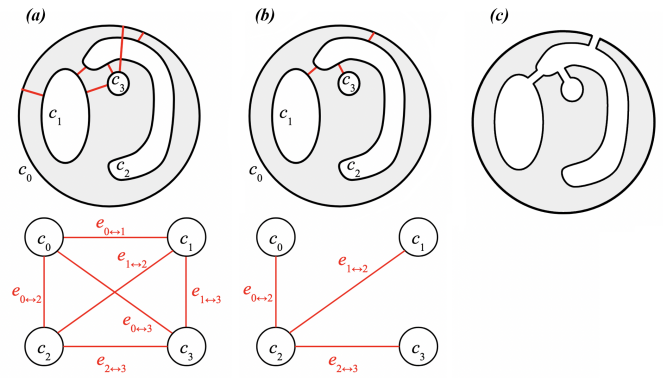


Figure 13: The Minimum Spanning Tree between an outer contour and its inner hole contours.

$v \in V$ for every curve in the region, and an edge $e \in E$ between every node with weight equal to the shortest distance between those two curves (13 (a)).

We search for the minimum-weighted spanning tree of this graph, which gives us the shortest connection points between our inner and outer curves (13 (b)). We break the curves at these locations and reconnect them to one another in order to create our single outer contour (13 (c)). This is somewhat similar to Zhai et al.'s method [47], though they employ Voronoi diagrams which they then recombine before filling with Fermat spirals.

6.2 Infill

6.2.1 Generate Isocontours. Isocontours, or equidistantly spaced contours within a closed curve, are used in the generation of walls and concentric infill in traditional slicers. When the distance between these contours is equal to the extrusion width of the printer nozzle, the toolpath following them is space-filling. For our algorithm, isocontours serve as the foundation for our Fermat Spirals.

The built-in contour curve generation functionality in Rhino fails to produce correct isocontours for complex non-convex geometry. As a result, slicers written in Rhino and Grasshopper often use packages available on food4rhino such as Clipper [44] to generate isocontours for walls. However, these packages have limitations with respect to what types of geometry they support as input. We implemented our own method for finding isocontours that we call "point marching", which is based off of the pixel marching (or marching squares) method [32]. A detailed description of our method can be found in Appendix C.

6.2.2 Fermat Spiral. Once we have the isocontours for an isolated region, we use them to generate a connected Fermat spiral toolpath. We implemented Zhao et al.'s method to generate these paths [48]. First we generate a tree data structure of the isocontours. Then, we identify consecutive contours that can be Fermat-spiraled, and finally we connect all contours. The result is a single continuous path that starts and ends an extrusion width w_e apart, while completely filling the isolated region (see Figure 14). Detailed information about this step in our algorithm can be found in Appendix D.

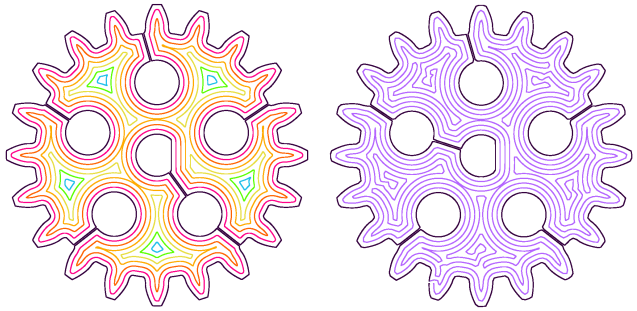


Figure 14: The isocontours (left) and toolpath (right) of an isolated region with holes. The wall is in black, the infill in purple.

6.3 Walls

Our algorithm allows a user to choose between two slicing modes, solid infill mode or wall mode. In solid infill mode, our algorithm generates a space filling spiral path that fills each isolated region completely. In wall mode, a user specifies a number of walls and our algorithm generates a spiraling path that generates that number of walls only.

Traditional slicing methods often designate walls and infill as separate toolpaths with Inner-Model travel movements between them. In Zhao et al.'s approach, no distinction is made between walls and infill; each region is filled with a single spiraling path [48]. In contrast, TRAvel Slicer separates the outermost contour from the inner isocontours, which results in a cleaner outer surface on the final print. Furthermore, TRAvel Slicer starts and stops the infill toolpath at the closest point to the start point of the wall, resulting in zero Inner-Model travel movements between walls and infill (see Figure 14, right).

In rare circumstances, it is possible for infill to be separated into two or more regions with Inner-Model travel paths between them. In this case, TRAvel Slicer begins the connected Fermat-spiral infill of each region at the closest point to the previous region, resulting in short Inner-Model travel movements.

6.4 Inner-Model Travel Reduction Output

Once a continuous path has been generated for every isolated region, TRAvel Slicer has finished reducing both Outer-Model and Inner-Model travel movements. It outputs a g-code file for the specified 3D printer along with a visualization of the extrusion and travel paths.

6.5 Improvements on Previous Work

Connecting the outer contour to its hole inner contours made defining relationships within the isocontour tree straightforward (Section 6.1 and Appendix D), and allows us to specify the print order of walls and infill without excessive Inner-Model travel movements. Instead, it treats the wall as one continuous path with a single start and end point in the same location.

Early tests of our implementation of connected Fermat spirals [48] showed a noticeable seam on the outside of models. Separating the outermost contour (the wall) from the infill significantly decreased the visibility of these seams.

7 REAL-WORLD APPLICATION

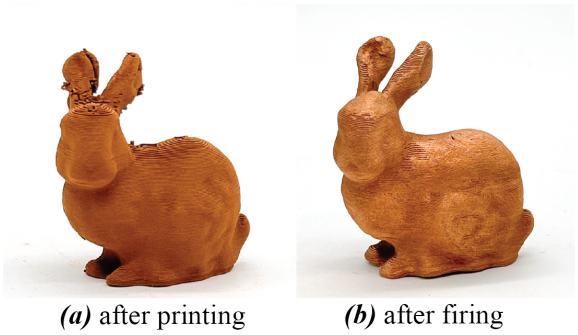
7.1 Printing Process

To test our slicer and demonstrate its utility, we employed it to print a range of complex models on two different printers and materials: 1) a DW clay 3D printer using a material called CeraMetal, and 2) a traditional FFF printer using standard Polylactic Acid (PLA) filament. On both printers, we contrast prints generated by TRAvel Slicer with those generated by a traditional slicer (Cura). We used the same settings for layer height, wall thickness, speed, and extrude rate (flow) in both slicers to facilitate comparison.

7.1.1 DW Printing in CeraMetal. For DW printing, we employed an Eazao Zero [11], a small and affordable desktop DW printer that was developed for clay 3D printing. Models are printed from CeraMetal, a novel "metal clay" material [5]. This material has a non-linear rheology. Even when printed with an auger-based printer like the Eazao Zero, it is impossible to start and stop extrusions; material continues to ooze from the nozzle long after a stop extrusion command is given. CeraMetal provides a clear illustration of the benefits of TRAvel slicer for 3D printing in these kinds of materials.

For all models printed in CeraMetal, we used a nozzle with an inner diameter of .6 mm, a print and travel speed of 1000 mm/minute, and an extrude rate of .25 mm of filament per mm traveled. Models were sliced with a layer height of .3 mm.

The Outer-Model travel minimization portion of TRAvel Slicer takes additional parameters nozzle height (noz_{height}) and maximum nozzle width (noz_{width}). We sliced these models with nozzle dimensions noz_{height} = 14 mm, noz_{width} = 6 mm.



(a) after printing ***(b)*** after firing

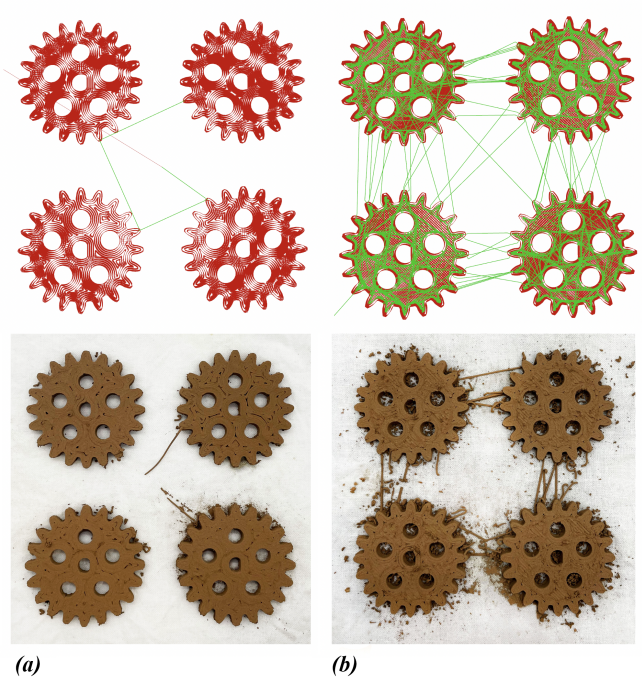
Figure 15: A Stanford Bunny test model sliced with TRAvel Slicer and printed in CeraMetal before (a) and after (b) firing.

Once a part is printed in CeraMetal, it is dried and then fired in a kiln. During firing, the clay sinters into a solid bronze metal part. Figure 15 shows images of a Stanford bunny model that was sliced with TRAvel Slicer before (a) and after (b) firing.

7.1.2 Traditional Thermoplastic Printing. We used a Creality Ender 3D Pro [8], a commercial desktop FFF printer to conduct traditional PLA printing. All models in PLA were printed at a print speed of 1000 mm/minute, a travel speed of 9000 mm/minute, and an extrude rate of 0.033 mm of filament per mm traveled. Models were sliced at a layer height of .2 mm.

The Creality Ender's default nozzle height is approximately 2.5 mm from the heat box of the printer. We purchased and installed a custom brass nozzle and insulator from IAMG/NonPlanar [21] with an inner diameter of .4 mm, and dimensions noz_{height} = 30 mm, noz_{width} = 13 mm.

7.1.3 Models. We sliced and printed models that were designed to test and demonstrate both the Inner- and Outer-Model travel reduction capabilities provided by TRAvel slicer. We group these tests into five categories: duplicates and horizontal holes, vertical nesting, horizontal nesting, branching, and vertical holes.



(a) ***(b)***

Figure 16: Test: Duplicates With Holes. Four copies of a gear model with six holes sliced with TRAvel Slicer (a) and Cura (b). The top images show toolpath visualizations. Red: extrude paths, green: travel paths.

7.2 Duplicates and Holes

In traditional slicing, printing multiple instances of a model requires traveling to each copy of the model for every layer printed. We tested TRAvel Slicer on four copies of a gear model. Each gear also has six holes. This model tests path planning for Outer-Model travel reduction between the duplicates. It also tests the outer contour to hole contour connection for Inner-Model travel reduction and demonstrates that TRAvel Slicer can generate Fermat Spiraling paths for complex non-convex 2D geometries.

The top two images in Figure 16 are an overhead view of the generated toolpaths, with travel movements shown in green and print movements shown in red. The top left image is TRAvel Slicer's toolpath, which has reduced travel movements by printing each of the gears individually with a single continuous toolpath for each gear. This path has four travel movements in total. In contrast, Cura's toolpath, shown on the top right, has 51 Outer-Model travel movements and 2704 Inner-Model Travel movements (see Table 1). The bottom two images are the printed results.

The gears were printed with a solid infill. We can see excess material due to Outer-Model travel movements between the gears in the print generated by Cura's toolpath, and close to the gears due to Inner-Model Travel movement. The surface of the Cura gear has some lumps of extra material due to traversal between regions of infill. The print generated by TRAvel Slicer is cleaner, less excess material is generated due to travel movements, and the top layer of the print is smoother.

7.3 Vertical Nesting

Models in which vertical nesting occurs — where portions of the model overlap below and above other portions with respect to the z-axis — test TRAvel Slicer's overhang and collision detection. We test these features with a model that consists of two zig-zagging pillars, seen in Figure 17.

This model was printed with three walls and no infill. The top two images in the figure show toolpaths for the zig-zag model. The corresponding prints are shown in the bottom two images. The toolpath generated by TRAvel Slicer contains four travel movements between the two pillars. The toolpath generated by Cura contains 275 travel movements between the pillars; two travel movements for each layer that includes pillars, one to move from the starting pillar to the second pillar and another to return. As can be seen in Figure 17 (b), when printed with the continuously extruding CeraMetal, these travel movements obscure the form of the print, completely filling the space between the two pillars.

Figure 17: Test: Vertical Nesting. A model with two zig-zagging pillars sliced with TRAvel Slicer (a) and Cura (b). The top images show toolpath visualizations. Red: extrude paths, green: travel paths.

7.4 Horizontal Nesting

Models with horizontally nested geometry, or geometry that contains inner and outer structures, test TRAvel Slicer's curve comparison methods and Outer-Model path planning between the inner and outer structure.

We test TRAvel Slicer's ability to print horizontally nested geometry with a model consisting of a small solid heart nested within a larger heart-shaped shell (Figure 18). This model was printed in solid mode. TRAvel Slicer prints the base and the inner heart shape before printing the outer structure, resulting in a total of five travel movements. By contrast, Cura's toolpath includes 101 Outer-Model travel movements between the inner geometry and outer shell as a result of traversing back and forth at each layer height. Cura also generates 1799 Inner-Model travel movements as it travels between walls and areas of infill.

As can be seen in Figure (18 (b)), Cura's toolpath results in extra material being extruded between the inner and outer structures during Outer-Model travel movement. The outer wall of the shell collapses as a result of Inner-Model travel movement in this wall–extra material gradually builds up and eventually results in complete wall degradation.

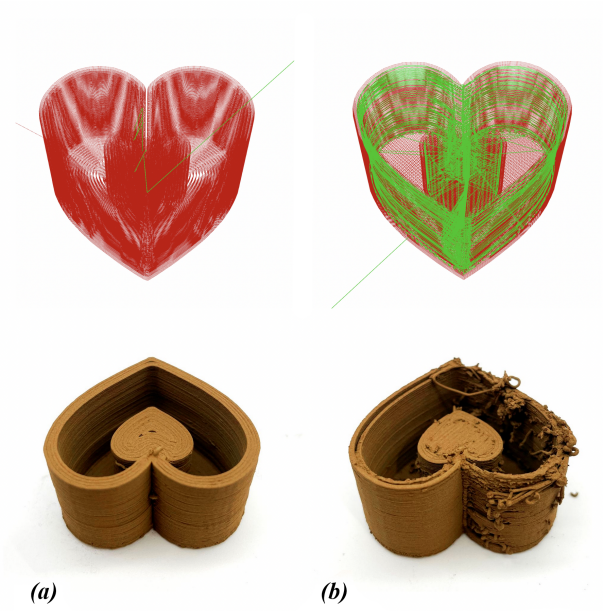


Figure 18: Test: Horizontal Nesting. A nested heart model sliced with TRAvel Slicer (left) and Cura (right). The top two images are toolpath visualizations. Red: extrude paths, green: travel paths.

7.5 Branching

For models with branches, traditional slicers produce an especially high number of travel movements, between all of the branches on each layer. We sliced a tree model with many branches to investigate performance on this particularly challenging class of shape. This model also tests TRAvel Slicer's overhang and collision detection and generates a very large search space for Outer-Model path planning.

Figure 19 shows a comparison of toolpaths for this model. Cura's toolpath results in excess material that is extruded between the branches of the tree, producing a print that deviates significantly from the original model. TRAvel Slicer produces a print with only a handful of significant Outer-Model travels. This toolpath also demonstrates how our algorithm generates a path that avoids collisions with the branches during travel movements (as described earlier and shown in Figure 11).

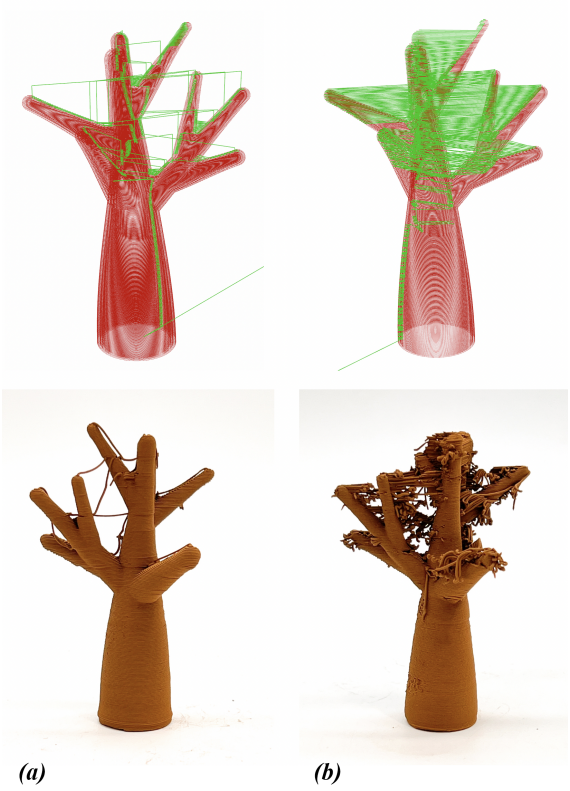


Figure 19: Test: Branching. A tree model sliced with TRAvel Slicer (left) and Cura (right). The top two images are toolpath visualizations. Red: extrude paths, green: travel paths.

7.6 Vertical Holes

Models with vertically-aligned holes, or branching structures that split apart at a lower layer and meet again at a higher region, result in travel movements back and forth within the empty space when sliced traditionally. Author 2 is a mathematician and ceramic artist who designed the sculpture seen in Figure 20. The model has branching structures and a hole in the center that closes at the top. We sliced this model with TRAvel Slicer and Cura. The toolpaths can be seen in Figure 20.

Cura's toolpath resulting in multiple travel movements between the branches of the structure, which can be seen as excess material at the center of the printed example and at the top of the model as it traversed between the two peaks (Figure 20, (b) lower). TRAvel Slicer's toolpath produced very few travel movements, resulting in less material extruded at the center of the model and a cleaner top.

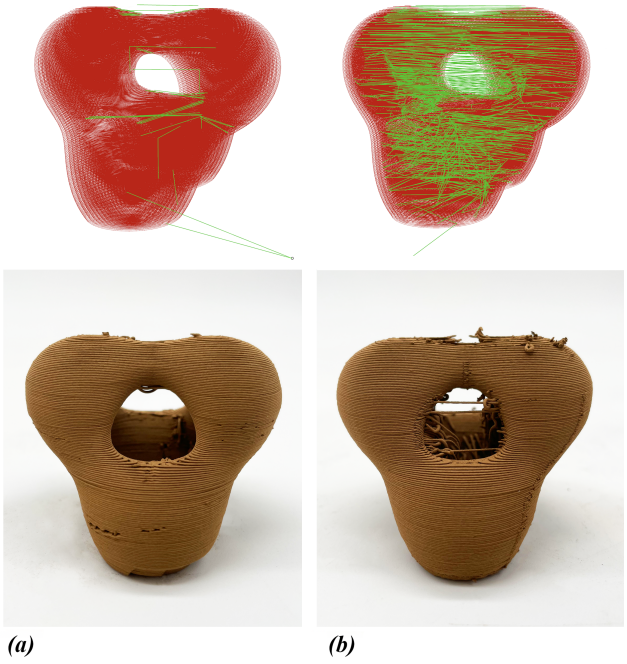


Figure 20: Test: Vertical Holes. A mathematical sculpture model designed by Author 2, sliced with TRAvel Slicer (left) and Cura (right). The top two images are toolpath visualizations. Red: extrude paths, green: travel paths.

7.7 PLA Prints

We printed the same models on the Creality Ender 3D as well (Figure 21). These prints demonstrate the versatility of TRAvel Slicer. TRAvel Slicer produces viable toolpaths for traditional PLA 3D printing. It also produces neater seams than Cura's toolpaths in several cases (Figure 21, bottom). By eliminating travel between branches of a model, TRAvel slicer produces a cleaner outer surface on branches.

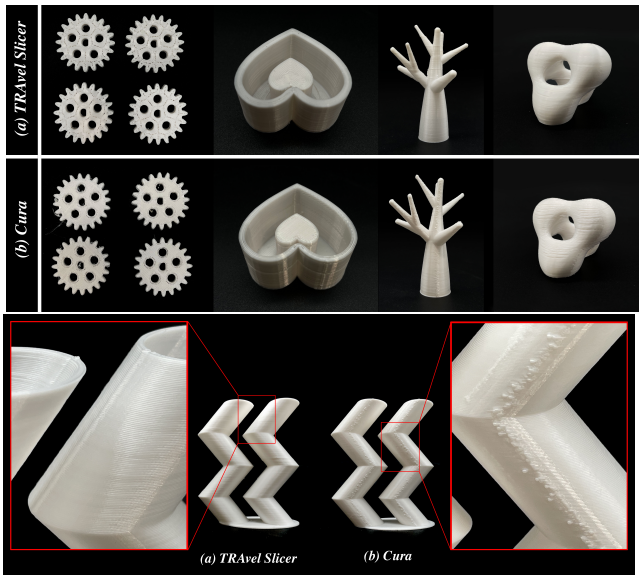


Figure 21: Top: All models printed in PLA with (a) TRAvel Slicer, (b) Cura. Bottom: Zoomed in views of the seams generated by each toolpath.

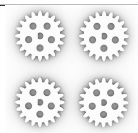
Model	Outer-Model Travels		Inner-Model Travels		Travel Length (mm)		Print Time (min)	
	TRAvel Slicer	Cura	TRAvel Slicer	Cura	TRAvel Slicer	Cura	TRAvel Slicer	Cura
Multiple Gears 	4	51	0	2704	112	14706	48	118
Zig Zags 	4	275	0	23	152	5934	20	26
Nested Hearts 	5	101	0	1799	50	7246	34	70
Tree 	172	467	1	288	1107	12335	30	50
Sculpture 	29	118	8	843	451	13359	107	134

Table 1: Results for CeraMetal. Nozzle Width: 6mm, Nozzle Height: 15mm.

Model	Outer-Model Travels		Inner-Model Travels		Travel Length (mm)		Print Time (min)	
	TRAvel Slicer	Cura	TRAvel Slicer	Cura	TRAvel Slicer	Cura	TRAvel Slicer	Cura
Multiple gears	4	71	0	6499	103	22337	175	223
Zig Zags	10	762	0	3432	207	22706	167	209
Nested Hearts	3	150	0	4764	36	15649	118	149
Tree	343	828	1	2870	1588	12358	71	93
Sculpture	57	155	6	2029	556	14184	295	355

Table 2: Results for PLA. Nozzle Width: 13mm, Nozzle Height: 30mm.

7.8 Print Results

Tables 1 and 2 provide a summary of printing information. The table includes information on travel movements, travel path length, and print time. The table shows that TRAvel slicer provides significant advantages, particularly in printing efficiency, over Cura in most cases. For all models, TRAvel Slicer prints had fewer travel movements, shorter travel lengths, and significantly shortened print times compared to Cura prints.

TRAvel Slicer's toolpath reduced the number of Outer-Model travel movements by a factor of 10 or more in the multiple gears, zig-zag, and nested hearts models in both materials. The number of Outer-Model travel movements in the tree model was reduced by more than half of the number of travel movements produced by Cura's toolpath. For the tree, the number of travel movements is somewhat misleading. For TRAvel Slicer, this number includes many small travel movements between branches that are very close together–too close to separate into different regions. These small travel movements are visible in the "vees" between branches in Figure 19. The number of travel movements between these branches could be reduced by using a nozzle with a smaller width.

Travel Slicer eliminated Inner-Model travels completely for the gears, zig zags, and nested hearts. The tree model has a single very short Inner-Model travel movement between two branches just before they diverge, resulting in two separate regions of infill from the outer wall. Similarly, the sculpture model also has multiple regions of infill separate from the outer wall on a few of its layers, resulting in 8 Inner-Model travel movements in the CeraMetal, and 6 in PLA.

The length of the travel movements generated by TRAvel Slicer is smaller than Cura's by an order of magnitude in all cases. In

the case of the multiple gears printed in CeraMetal TRAvel Slicer's travel length is smaller than Cura's travel length by a factor of 100.

It is also worth noting that print time is significantly shorter for all TRAvel Slicer prints. This is especially apparent in the multiple gears and the nested hearts. Print time was cut in half or lower in CeraMetal, and significantly reduced by comparison to Cura's print time in PLA.

7.9 Limitations and Improvements

In general, depending on the model geometry and the input parameters, TRAvel Slicer's slicing time typically takes between 20 seconds to 3 minutes. Though the slicing time is negligible compared to print time, there is room for potential improvement in the efficiency of the algorithm. Overall we consider the slicing time to be worth the improved print performance.

Currently travel movements are minimized only within a given height segment that is determined by the nozzle height. Travel paths could be minimized further, by considering all layers. However, this would significantly increase the size of the graph that we search. When traversing this graph, the search would need to perform an additional check that all nodes at a given nozzle height section z'_i have been printed before moving to a node within the next height section z'_{i+1}.

The geometry of the nozzle could additionally be considered. Currently TRAvel Slicer treats the nozzle as though it is a cylinder of height equal to the nozzle height noz_{height} and diameter equal to the maximum width of the nozzle noz_{width}. The nozzles employed on both the Eazao and the Creality Ender have some tapering that could be taken into consideration when computing collisions. This could decrease the number of short travels like the ones produced between nearby branches for the tree model.

Our code currently lacks support generation. However, if supports were generated, TRAvel Slicer's Outer-Model travel reduction section would operate the same as it does with the rest of the model and would print support structures before the (overhanging) layers they are intended to support. The infill of the support structures could be treated differently than the infill of the model. Infill patterns are currently limited to Fermat spirals. However, we should be able to employ other approaches to inner travel minimization (i.e. the zig zagging approach described by [45]) while maintaining our current approach to Outer-Model travel minimization since these two phases of our algorithm are distinct.

When comparing TRAvel Slicer's algorithm to other methods such as Liu et al. [30] and Kaplan et al. [25], we are limited in our quantitative assessment as a result of the lack of accessibility to their source code. We leave a more in-depth analysis of how TRAvel Slicer compares to other algorithms to future work.

8 CONCLUSION

In this work, we introduced a new Travel Reduction Algorithm (TRAvel) Slicer and demonstrated its utility and versatility by printing a range of complex geometry on two different printers and materials.

TRAvel Slicer makes it possible to print complex models in continuously extruding materials like CeraMetal. As our results demonstrate, these models *could not be printed* using traditional slicers. TRAvel Slicer also improves performance and efficiency in any 3D printing context. For models in which a traditional slicer generates travel movements, TRAvel Slicer decreases print time and, in some cases, also increases print quality.

In addition to our empirical results, our algorithm is unique in combining Inner- and Outer-Model travel reduction methods. We also expand on previous methods of continuous path infill to produce smoother outer surfaces with our methods of wall and hole curve connection. For Outer-Model travel, we generate a significantly reduced path search space that is easily and quickly traversed. Our approach also works for a broader class of geometries, including models with horizontally nested or interlocking features.

ACKNOWLEDGMENTS

This work is supported by National Science Foundation (NSF) Grant No. 2026218. We would like to thank Hand and Machine lab members Fiona Bell, Alyshia Bustos, Erin McClure, Alyssa Johnson, and Lasair Servilla.

REFERENCES

[1] 3Dpotter. [n. d.]. 3Dpotter. https://3dpotter.com/
[2] Shahriar Bakrani Balani, Seyed Hamidreza Ghaffar, Mehdi Chougan, Eujin Pei, and Erdem Şahin. 2021. Processes and materials used for direct writing technologies: A review. *Results in Engineering* 11 (Sept. 2021), 100257. https://doi.org/10.1016/j.rineng.2021.100257
[3] Samuelle Bourgault, Pilar Wiley, Avi Farber, and Jennifer Jacobs. 2023. CoilCAM: Enabling Parametric Design for Clay 3D Printing Through an Action-Oriented Toolpath Programming System. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3544548.3580745
[4] David Braam et al. 2016. UltiMaker Cura. https://ultimaker.com/software/ultimaker-cura/
[5] Leah Buechley, Jaime Gould, and Fiona Bell. 2024. CeraMetal: A New Approach to Low-Cost Metal 3D Printing with Bronze Clay. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3613904.3642155
[6] Leah Buechley and Ruby Ta. 2023. 3D Printable Play-Dough: New Biodegradable Materials and Creative Possibilities for Digital Fabrication. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3544548.3580813
[7] Zhangwei Chen, Ziyong Li, Junjie Li, Chengbo Liu, Changshi Lao, Yuelong Fu, Changyong Liu, Yang Li, Pei Wang, and Yi He. 2019. 3D printing of ceramics: A review. *Journal of the European Ceramic Society* 39, 4 (April 2019), 661–687. https://doi.org/10.1016/j.jeurceramsoc.2018.11.013
[8] Creality. 2023. Creality Ender 3D. https://www.creality.com/products/ender-3-3d-printer
[9] Alexander Curth, Barrak Darweesh, Logman Arja, and Ronald Rael. 2020. Advances in 3D printed earth architecture: Onsite prototyping with local materials.
[10] R. Duballet, O. Baverel, and J. Dirrenberger. 2017. Classification of building systems for concrete 3D printing. *Automation in Construction* 83 (Nov. 2017), 247–258. https://doi.org/10.1016/j.autcon.2017.08.018
[11] Eazao. 2022. Eazao Zero - Eazao. https://www.eazao.com/product/eazao-zero/
[12] Chloë Fleming, Stephanie Walker, Callie Branyan, Austin Nicolai, and Geoffrey Hollinger. 2020. Toolpath Planning for Continuous Extrusion Additive Manufacturing. (May 2020).
[13] Jack Forman, Mustafa Doga Dogan, Hamilton Forsythe, and Hiroshi Ishii. 2020. DefeXtiles: 3D Printing Quasi-Woven Fabric via Under-Extrusion. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA, 1222–1233. https://doi.org/10.1145/3379337.3415876
[14] Frikk H Fossdal, Vinh Nguyen, Rogardt Heldal, Corie L. Cobb, and Nadya Peek. 2023. Vespidae: A Programming Framework for Developing Digital Fabrication Workflows. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference (DIS '23)*. Association for Computing Machinery, New York, NY, USA, 2034–2049. https://doi.org/10.1145/3563657.3596106

[15] N. Gaudillière, R. Duballet, C. Bouyssou, A. Mallet, Ph. Roux, M. Zakeri, and J. Dirrenberger. 2019. Chapter 3 - Building Applications Using Lost Formworks Obtained Through Large-Scale Additive Manufacturing of Ultra-High-Performance Concrete. In *3D Concrete Printing Technology*, Jay G. Sanjayan, Ali Nazari, and Behzad Nematollahi (Eds.). Butterworth-Heinemann, 37–58. https://doi.org/10.1016/B978-0-12-815481-6.00003-8

[16] Mohamed Gomaa, Wassim Jabi, Alejandro Veliz Reyes, and Veronica Soebarto. 2021. 3D printing system for earth-based construction: Case study of cob. *Automation in Construction* 124 (April 2021), 103577. https://doi.org/10.1016/j.autcon.2021.103577

[17] Prashant Gupta, Bala Krishnamoorthy, and Gregory Dreifus. 2020. Continuous Toolpath Planning in Additive Manufacturing. *Computer-Aided Design* 127 (Oct. 2020), 102880. https://doi.org/10.1016/j.cad.2020.102880 arXiv:1908.07452 [cs].

[18] Jean Hergel, Kevin Hinz, Sylvain Lefebvre, and Bernhard Thomaszewski. 2019. Extrusion-based ceramics printing with strictly-continuous deposition. *ACM Transactions on Graphics* 38, 6 (Dec. 2019), 1–11. https://doi.org/10.1145/3355089.3356509

[19] Ryan Hoover. 2018. Ryan Hoover | Xylinus. http://www.ryanhoover.org/rd/xylinus.php

[20] Kaiyu Hu, Hailong Li, and Kou Xi. 2023. A Toolpath Optimization Algorithm for Layered 3D Printings based on Solving the TSP. *Journal of Physics: Conference Series* 2456, 1 (March 2023), 012039. https://doi.org/10.1088/1742-6596/2456/1/012039

[21] IAMG/NonPlanar. 2020. nonplanar.xyz. https://www.nonplanar.xyz/

[22] Jaime Gould. 2024. TRAvel_Slicer Github Repository. https://github.com/Hand-and-Machine/TRAvel_Slicer

[23] Jingchao Jiang and Yongsheng Ma. 2020. Path Planning Strategies to Optimize Accuracy, Quality, Build Time and Material Use in Additive Manufacturing: A Review. *Micromachines* 11, 7 (July 2020), 633. https://doi.org/10.3390/mi11070633 Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[24] Josef Prusa. [n. d.]. Sequential printing | Prusa Knowledge Base. https://help.prusa3d.com/article/sequential-printing_124589

[25] Daphna Kaplan, Shir Rorberg, Mirela Ben Chen, and Yoav Sterman. 2022. NozMod: Nozzle Modification for Efficient FDM 3D Printing. In *Proceedings of the 7th Annual ACM Symposium on Computational Fabrication (SCF '22)*. Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/3559400.3561999

[26] Matthew Lanaro, David P. Forrestal, Stefan Scheurer, Damien J. Slinger, Sam Liao, Sean K. Powell, and Maria A. Woodruff. 2017. 3D printing complex chocolate objects: Platform design, optimization and evaluation. *Journal of Food Engineering* 215 (Dec. 2017), 13–22. https://doi.org/10.1016/j.jfoodeng.2017.06.029

[27] Gierad Laput, Xiang 'Anthony' Chen, and Chris Harrison. 2015. 3D Printed Hair: Fused Deposition Modeling of Soft Strands, Fibers, and Bristles. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, Charlotte NC USA, 593–597. https://doi.org/10.1145/2807442.2807484

[28] Phoebe Li, Stephen Mellor, James Griffin, Charlotte Waelde, Liang Hao, and Richard Everson. 2014. Intellectual property and 3D printing: a case study on 3D chocolate printing. *Journal of Intellectual Property Law & Practice* 9, 4 (April 2014), 322–332. https://doi.org/10.1093/jiplp/jpt217

[29] Jeffrey I. Lipton and Hod Lipson. 2016. 3D Printing Variable Stiffness Foams Using Viscous Thread Instability. *Scientific Reports* 6, 1 (Aug. 2016), 29996. https://doi.org/10.1038/srep29996 Number: 1 Publisher: Nature Publishing Group.

[30] Wenyang Liu, Ling Chen, Guangzhen Mai, and Lijun Song. 2020. Toolpath Planning for Additive Manufacturing Using Sliced Model Decomposition and Metaheuristic Algorithms. *Advances in Engineering Software* 149 (Nov. 2020), 102906. https://doi.org/10.1016/j.advengsoft.2020.102906

[31] Yaowen Liu, Xue Liang, Ahmed Saeed, Weijie Lan, and Wen Qin. 2019. Properties of 3D printed dough and optimization of printing parameters. *Innovative Food Science & Emerging Technologies* 54 (June 2019), 9–18. https://doi.org/10.1016/j.ifset.2019.03.008

[32] William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM SIGGRAPH Computer Graphics* 21, 4 (Aug. 1987), 163–169. https://doi.org/10.1145/37402.37422

[33] Arthur Mamou-Mani and Adam Holloway. 2023. ProjectSilkworm/Silkworm. https://github.com/ProjectSilkworm/Silkworm original-date: 2013-02-17T11:04:21Z.

[34] Sylvester Mantihal, Sangeeta Prakash, Fernanda Condi Godoi, and Bhesh Bhandari. 2017. Optimization of chocolate 3D printing by correlating thermal and flow properties with 3D structure modeling. *Innovative Food Science & Emerging Technologies* 44 (Dec. 2017), 21–29. https://doi.org/10.1016/j.ifset.2017.09.012

[35] Marita. [n. d.]. Reprintable Mussel shell. https://commons.materiom.org/data-commons/recipe/649c36218e0f06dcab0b7d2c

[36] Du T. Nguyen, Cameron Meyers, Timothy D. Yee, Nikola A. Dudukovic, Joel F. Destino, Cheng Zhu, Eric B. Duoss, Theodore F. Baumann, Tayyab Suratwala, James E. Smay, and Rebecca Dylla-Spears. 2017. 3D-Printed Transparent Glass. *Advanced Materials* 29, 26 (July 2017), 1701181. https://doi.org/10.1002/adma.201701181 Publisher: John Wiley & Sons, Ltd.

[37] Franklin Pezutti-Dyer and Leah Buechley. 2022. Extruder-Turtle: A Library for 3D Printing Delicate, Textured, and Flexible Objects. In *Sixteenth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '22)*. Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/3490149.3501312

[38] Michael L. Rivera, S. Sandra Bae, and Scott E. Hudson. 2023. Designing a Sustainable Material for 3D Printing with Spent Coffee Grounds. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference (DIS '23)*. Association for Computing Machinery, New York, NY, USA, 294–311. https://doi.org/10.1145/3563657.3595983

[39] Eva Sanchez-Rexach, Trevor G. Johnston, Coralie Jehanno, Haritz Sardon, and Alshakim Nelson. 2020. Sustainable Materials and Chemical Processes for Additive Manufacturing. *Chemistry of Materials* 32, 17 (Sept. 2020), 7105–7119. https://doi.org/10.1021/acs.chemmater.0c02008 Publisher: American Chemical Society.

[40] Simplify3D. 2013. Simplify3D. https://www.simplify3d.com/

[41] Simplify3D Software. [n. d.]. Multi-Part Printing | Simplify3D Software. https://www.simplify3d.com/resources/articles/multi-part-printing/

[42] Haruki Takahashi and Homei Miyashita. 2017. Expressive Fused Deposition Modeling by Controlling Extruder Height and Extrusion Amount. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, Denver Colorado USA, 5065–5074. https://doi.org/10.1145/3025453.3025933

[43] Tronxy. 2024. Tronxy 3D Printers Official Store. https://www.tronxy3d.com/products/tronxy-moore-2-pro-ceramic-clay-3d-printer

[44] Arend van Waart. 2023. arendvw/clipper. https://github.com/arendvw/clipper original-date: 2014-07-05T09:15:32Z.

[45] Lingwei Xia, Guowei Ma, Fang Wang, Gang Bai, Yi Min Xie, Weiguo Xu, and Jianzhuang Xiao. 2022. Globally continuous hybrid path for extrusion-based additive manufacturing. *Automation in Construction* 137 (May 2022), 104175. https://doi.org/10.1016/j.autcon.2022.104175

[46] Fan Yang, Min Zhang, Sangeeta Prakash, and Yaping Liu. 2018. Physical properties of 3D printed baking dough as affected by different compositions. *Innovative Food Science & Emerging Technologies* 49 (Oct. 2018), 202–210. https://doi.org/10.1016/j.ifset.2018.01.001

[47] Xiaoya Zhai and Falai Chen. 2019. Path Planning of a Type of Porous Structures for Additive Manufacturing. *Computer-Aided Design* 115 (Oct. 2019), 218–230. https://doi.org/10.1016/j.cad.2019.06.002

[48] Haisen Zhao, Fanglin Gu, Qi-Xing Huang, Jorge Garcia, Yong Chen, Changhe Tu, Bedrich Benes, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. 2016. Connected fermat spirals for layered fabrication. *ACM Transactions on Graphics* 35, 4 (July 2016), 1–10. https://doi.org/10.1145/2897824.2925958

[49] Fanchao Zhong, Yonglai Xu, Haisen Zhao, and Lin Lu. 2023. As-Continuous-As-Possible Extrusion-Based Fabrication of Surface Models. *ACM Trans. Graph.* 42, 3 (March 2023), 26:1–26:16. https://doi.org/10.1145/3575859

A SOFTWARE

TRAvel Slicer is written in python and is implemented in Rhino 7 and Grasshopper3D. The code is open-source and publicly available in a Github repository [22]. The code includes a Grasshopper file that can be used to slice closed polysurfaces in Rhino that have either been created in Rhino's CAD interface or imported. It also allows the user to set the printer parameters (see Figure 22). TRAvel Slicer uses Pezutti et al.'s Extruder Turtle Library [37] to generate the g-code file, another open source python library used in conjunction with Rhino and Grasshopper. Extruder Turtle supports g-code generation for the Eazao Ceramic 3D printer [11] and the Creality Ender 3D printer [8].

B OVERLAP

In order to determine overlap between curves, we compare the distance between their projections onto the same x-y parallel plane. TRAvel Slicer can determine this by either bounding box comparison or Rhinoscript's built-in functions for planar curve comparison.

The bounding box method compares the overlap between the bounding boxes of two curves. If overlap in x and y exceeds $w_e/2$, we say the curves overlap. This method is much quicker than other overlap calculations, but it can fail when given more complex geometry in which the curves maintain a distance greater than $w_e/2$

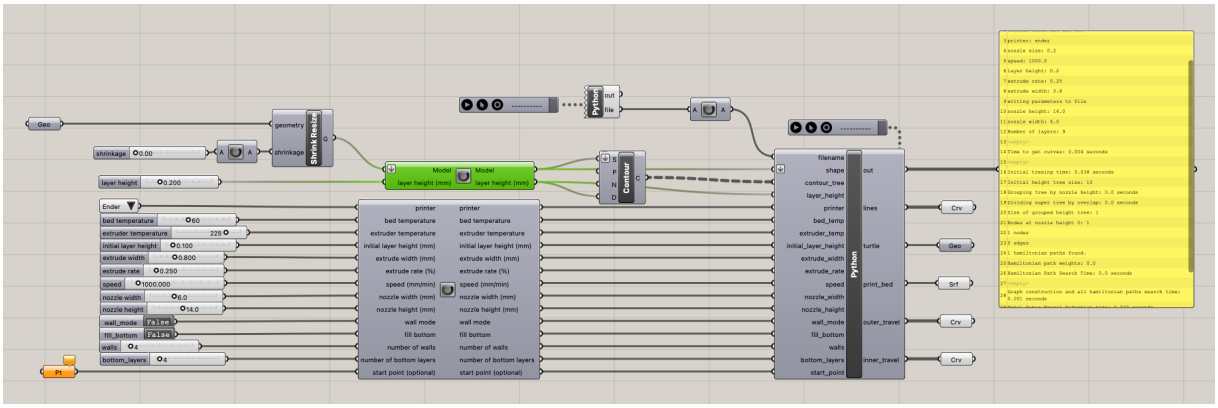


Figure 22: A screenshot of the Grasshopper file included in TRAvel Slicer's Github repository.

from one another but still have overlapping boundaries, see Figure 23.

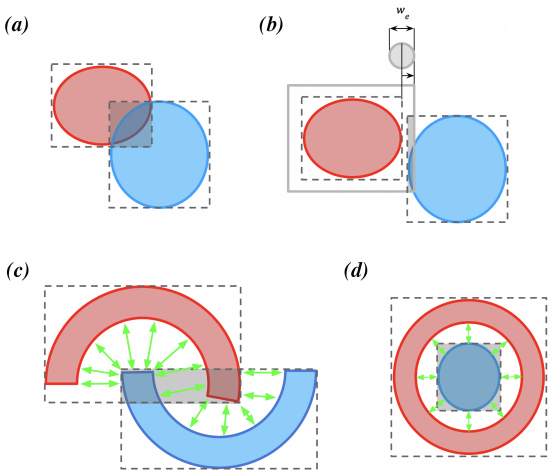


Figure 23: Bounding box comparison for overlap between contours. Green arrows between contours indicate space for nozzle travel. In (a) and (b) this method succeeds. In (c) it fails due to concave curves and (d) it fails due to nested contours.

Rhinoscript's built-in *PlanarClosedCurveContainment* function takes two curves as its input and an optional tolerance value. When the tolerance is set to $w_e/2$ we can determine overlap in the case of more complex geometry like in (c) and (d) of Figure 23. Though this method is slower than the bounding box comparison it is much more accurate, so we have found this trade-off with respect to time to be preferable.

C POINT MARCHING

Our method of "point marching" is inspired by Marching Squares [32].

We divide a given curve into $n = curve\ length/(w_e \cdot k)$ equidistant points, where w_e is the printer nozzle's extrude width and k is a coefficient that is determined by the precision of the printer. We have found a value of $k = 1/2$ to be sufficient. We then "march" all of these points inward by the extrusion width in order to find the next isocontour of the shape (see Figure 24).

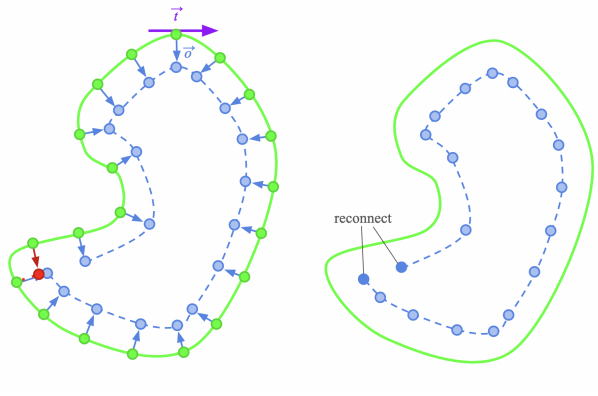


Figure 24: "Point Marching" for generating isocontours.

For all $p \in points$, we compute new points p':

$$\vec{t_i} = p_{i+1} - p_{i-1}$$

$$\hat{t_i} = \frac{\vec{t_i}}{|\vec{t_i}|}$$

$$\vec{o_i} = R_z \times (w_e \cdot \hat{t_i})$$

$$p'_i = p_i + \vec{o_i}$$

Where $\hat{t_i}$ is the normalized tangent vector at a given point p_i, $\vec{o_i}$ is the offset vector orthogonal to the tangent with magnitude equal to w_e, and R_z is a rotation matrix around the z axis. R_z rotates a given vector by 90° if the winding order of the points' indices is counter-clockwise, and −90° if the order is clockwise.

Marching these points inward can result in displacing them into the space between the curve and the new inner isocontour or even outside of the curve, particularly when the curve has sharp corners. As a result, we do an additional check to determine that all $p' \in new\ points$ are distance $\leq w_e$ from all $p \in points$.

In order to avoid comparing every p' to p, which results in $O(n^2)$ time complexity for a single contour, we utilize a two-dimensional

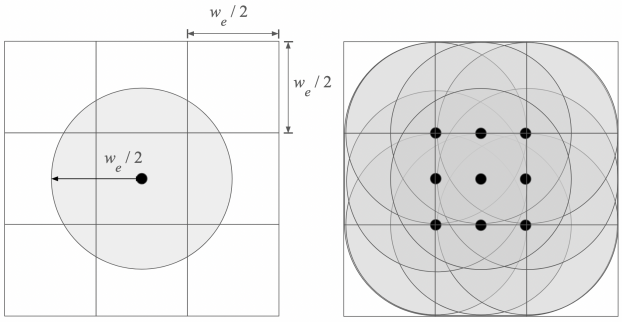


Figure 25: Grid $G[u][v]$ and its eight neighbors. The left image shows a point at the center $G[u][v]$. The gray shaded areas indicate distances of $w_e/2$ from that point. The right image shows the boundary of the search space within these grid squares.

grid data structure $G[u][v]$. We retrieve the minimum and maximum x and y coordinate values of all $p \, \epsilon \, points$ then construct $G[u][v]$ with dimensions $0 \leq u \leq quotient(\frac{x_{max}-x_{min}}{w_e/2})$ and $0 \leq v \leq quotient(\frac{y_{max}-y_{min}}{w_e/2})$.

We then place all $p \, \epsilon \, points$ into G by computing their indices:

$$u_{index} = quotient(\frac{p_i.x - x_{min}}{w_e/2})$$
$$v_{index} = quotient(\frac{p_i.y - y_{min}}{w_e/2})$$
$$G[u_{index}][v_{index}].append(p_i)$$

Given p_i', we compute its u_i and v_i in order to retrieve a subset of points S_i, which include the points at $G[u_i][v_i]$ and the points in its eight neighbors (see Figure 25):

$$S_i = \{G[u][v] \mid u_i - 1 \leq u \leq u_i + 1, \; v_i - 1 \leq v \leq v_i + 1\}$$

Then we compute the distance between p_i' and $p \, \epsilon \, S_i$. If any $|p_i' - p| < w_e$, we discard that p_i.

If no p' is discarded, we can then pass the computed new points as a single consecutive sequence directly to Rhino's curve interpolation function to generate our next isocontour.

If one or more p' has been discarded we have an additional step to reconnect broken line segments in the curve. We identify all point sequences with contiguous indices and take their first and last indices as start and end points. For every end point we determine the closest start point, which can belong to the same contiguous sequence in the case of an isocontour "pinching" into two separate contours. Once we have determined all connections between sequences, we pass the connected sequences to Rhino's curve interpolation function and return one or more isocontours.

For a solid infill, we continuously pass our isocontours to our point marching algorithm in order to generate the next set of one or more isocontours until the curve area is too small. If all p' are discarded, the given curve is the innermost contour of its region.

D CONNECTED FERMAT SPIRALS

D.1 Tree Isocontours

We define a tree data structure T that we use to construct a new tree, T' for identifying spiral-able regions and connecting all contours and regions for a single continuous path.

Tree T's nodes represent the individual contours of a single closed region. The root node of our tree contains the outermost curve. When we pass a curve to our code for generating isocontours, the returned contours become children nodes of that curve's node in T. Once the isocontours have been generated, T is complete.

We then construct T' by traversing T with Depth-First search, beginning from $T.root$. The nodes in T' each represent a set of contours rather than a single curve. We adopt Zhao et al.'s [48] terminology and classification and assign a "type" property to these nodes; type I nodes contain consecutive contours from T with one or fewer children. These are spiral-able collections of contours with a single local "maxima," or innermost contour. Type II nodes contain a single contour with two or more children (see Figure 26). Once we have constructed T' we spiral and Fermat-spiral all collections of isocontours in our type I nodes.

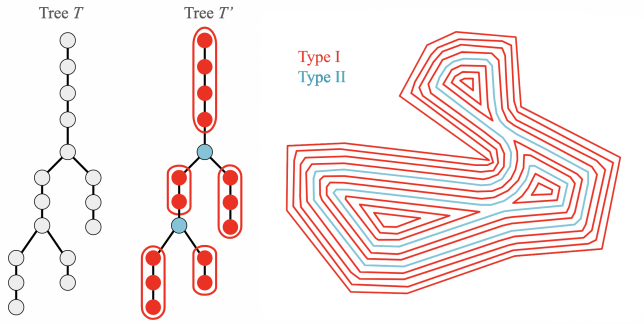


Figure 26: From left to right: Tree T, in which each node represents a single contour; Tree T', in which type I nodes contain a set of contours with a local maxima and type II nodes contain a single contour with two or more children in T; and the region containing all isocontours.

D.2 Spiral and Fermat-Spiral

As in [48], we spiral regions with a single local maxima by continuously breaking and connecting consecutive contours I_i, I_{i+1} in either clockwise or counter-clockwise order with an offset of extrusion width w_e, resulting in a single path beginning on the outermost curve and ending at the innermost curve.

We then create a Fermat spiral from the spiral. Our initial break point on the outermost contour for connecting to the subsequent isocontour becomes our last point in the path out in order to keep the start and end points at the same location (separated by w_e) to minimize travel. We compute a new point that acts as a connection inward, a distance of w_e counter-clockwise from the point acting as the "out" connection. The segment of the spiral between these two points is removed, and the new "in" point is connected to the next isocontour as shown in Figure 27. This is repeated for every other contour until the innermost contour is reached.

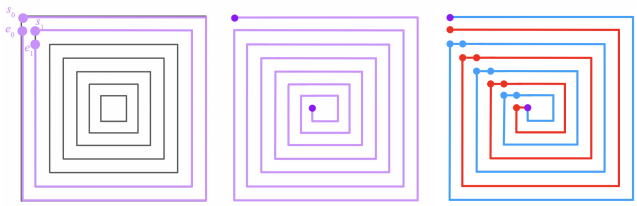


Figure 27: A collection of Type I isocontours that can be spiraled. Left: Finding connections between contours. Middle: The resulting spiral with start point at outermost contour and the end point at the center. Right: The resulting Fermat spiral with start and end points extrusion width w_e apart.

D.3 Connect All Nodes in Region for a Single Path

Beginning with the leaf nodes within T', we connect Fermat-spiraled type I nodes to their type II parent at the closest points along their curve to the "in" and "out" points of the Fermat spiral. We connect type II nodes to their parent contours, provided the connection does not conflict with any other connections in the region. Once all nodes are connected, the region contains a single space-filling toolpath that starts and ends at the same location (see Figure 28).

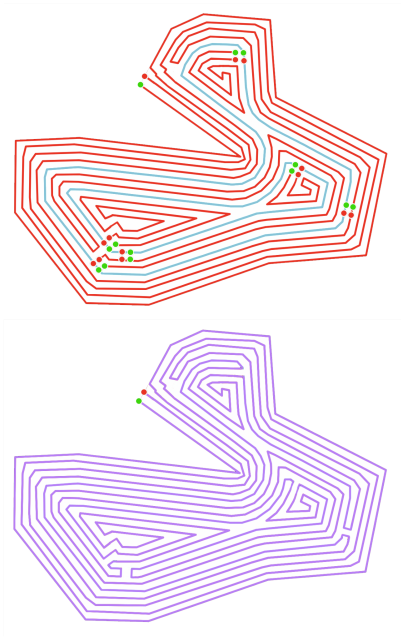


Figure 28: Connecting all type I and type II nodes to their parent node for a single continuous space-filling curve.