The Pennsylvania State University

The Graduate School

GRAPH THEORY DRIVEN TOOLPATH DESIGN FOR MATERIAL EXTRUSION ADDITIVE MANUFACTURING

A Thesis in

Mechanical Engineering

by

Logan J. Hutton

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

May 2024

The thesis of Logan J. Hutton was reviewed and approved by the following:

Joseph Bartolai

Applied Research Laboratory

Assistant Research Professor

Thesis Co-Advisor

Darren C. Pagan

Assistant Professor of Materials Science and Mechanical Engineering

Thesis Co-Advisor

Nicholas Meisel

Associate Professor of Engineering Design and Mechanical Engineering

Simon W Miller

Applied Research Laboratory

Associate Professor of Architectural Engineering

Robert Kunz

Professor and Associate Head for Graduate Mechanical Engineering

Abstract

Polymer material extrusion (MEX) additive manufacturing (AM) toolpath design has been driven by toolpath algorithms that are robust and computationally inexpensive. These contemporary algorithms fail to minimize travel moves – toolpaths where no material is deposited. It has been shown that *continuous toolpaths*, toolpaths that have no travel moves, increase the bond strength between adjacent roads deposited. This increase in bond strength leads to increased part strength when continuous toolpaths are used throughout the part. A reduction in travel moves is especially impactful in pellet-fed MEX AM systems, where precise control of the flow rate is not possible, leading to larger and more frequent defects at start-stop locations. This thesis presents a graph theory based toolpath generation algorithm, entitled *GRATER* – the ***GRA****ph* ***T****heory based slic****ER*** – that when compared to contemporary toolpath generation softwares, or "slicers", reduces travel moves by up to 95%, reduces travel distance by a factor of 3, and reduces build time by approximately 25%. Further advantages of continuous toolpaths are shown for parts which have regions within a layer built using different processing parameters, such as infill density. Contemporary slicers consider these regions independently, using discontinuous toolpaths to deposit material within the layer. By generating a continuous toolpath for the layer containing varying process parameters, the effective ultimate stress of parts is shown to increase 33% under various strategies.

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Code Snippets

Nomenclature

Symbols

β	number of brims
ϵ_{in}	Extrusion width of infill [mm]
ϵ_p	Extrusion width of perimeter [mm]
η	number of layers
θ	perimeter starting angle
λ_{in}	weight of the infill edges
λ_p	weight of the perimeter edges
μ	infill percentage
ν	angle from vector
ρ	number of perimeters
σ	layer height [mm]
τ_{bed}	printer's bed temperature
τ_{hot}	printer's nozzle temperature
ϕ	infill overlap percentage
χ	number of contours
ω	infill angle

Acronyms

2D	Two Dimensional
3D	Three Dimensional
AM	Additive Manufacturing / Additively Manufactured
MEX	Material Extrusion

Typeface

`Teletype`	Algorithms not developed in this research
Bold	Algorithms or toolpaths developed in this research
Underline	User-defined options in GRATER
Small Caps	MATLAB or open source functions used in GRATER

Acknowledgements

I would like to thank my friends at The Pennsylvania State University Applied Research Laboratory. In particular I would like to thank Simon Miller, Justin Valenti, Callie Zawaski, Thomas Jones, Zachary Renda, and Joseph Fisher for their guidance, willingness to listen, and extensive technical support. I would also like to express thanks for those whom I shared the Bull Pen with: Andrew Loughran, Victoria Lenze, and Christie Hasbrouck. Their friendship and support will not be forgotten. Many thanks to my readers for taking the time to review this thesis and providing helpful feedback. Last but not least, a special thanks to Joseph Bartolai for your guidance and support.

I would like to thank my family for their understanding, support, and love. A special thanks to my fiancée for braving this journey with me. I would also like to thank my dog, Dobby, for knowing when I needed a break and demanding a walk.

This thesis is based on work funded by Penn State's Applied Research Laboratory (ARL) through the Walker Scholar Program. The opinions, findings, conclusions, and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Penn State ARL or collaborators.

Remember - the enemy's gate is down.

Ender, Ender's Game

Chapter 1

Introduction & Motivation

The polymer MEX AM process is often relegated to prototyping due to its perceived poor and/or unpredictable mechanical properties of parts. Modern toolpath, generated instructions that define movement of the system such as the path, location, speed, and direction, algorithms were developed to be computationally inexpensive and robust, but are ignorant to the *performance* of the final part or the physics of the AM process. For example, modern toolpath algorithms fail to consider the bonding mechanisms of polymers in MEX AM. Holistically considering the physics of the AM process during the toolpath generation step is non-trivial as MEX AM slicers can change printing parameters for any number of defined regions within a part [1, 2]. The most common use for defining one of these regions is to define regions of sparse and dense infill to strategically strengthen or lighten the as-built part. When parameters that directly affect the toolpath are changed in one of these regions, the slicer will independently develop the toolpaths for each region. This leads to the part being fabricated *as if it were multiple parts* on the build plate that *just happen to be touching* one another. The boundaries of these adjacent regions are not printed sequentially, leading to a large temperature difference between the road being extruded and the adjacent, previously extruded road. The temperature difference between adjacent roads (also known as extrudate) can lead to poor bonding between the roads and is often the weak spot in the part, thereby limiting the utility of defining regions of varying parameters in a single part [3–6]. This is more simply stated as defining a region of dense infill at a high-stress region in a part does not increase the strength of the region as much as is possible due to the defects at the boundary of the dense infill region [3]. Furthermore,

the regions are deposited independently with no transition or load paths between adjacent regions, which is another toolpathing problem that diminishes the possible strength of the produced part.

Travel moves in the MEX AM process are toolpaths that stop extruding (often with a retraction command), travel to another area of the part, then resume extruding. These travel moves can cause defects called *travel defects*, which are uncommon on filament-fed MEX AM systems due to the precise extrusion control that the short melt zone provides and an accurate model of pressure affects (see Section 1.2.2 and Figure 1.3 for a discussion on extrusion control in filament-fed MEX AM systems) [7]. Precise extrusion control allows filament systems to stop extruding during travel moves without stringing molten filament between travel points. Screw-fed extrusion systems found on large-format MEX AM systems do not have precise extrusion control as the systems cannot quickly or reliably stop then start extruding for travel moves due to the highly non-linear extruder dynamics [8]. This lack of extrusion control greatly increases the chance a travel move can cause a travel defect as well as cause *stringing* — a phenomenon that occurs when the molten polymer is still under pressure causing it to ooze out of the print nozzle during travel moves. Stringing defects can also occur when the filament continues to flow from the nozzle while the extruder moves to another region of the print thereby inducing a travel defect as well.

1.1 Introduction of MEX AM

The ASTM / ISO 52900 standard defines the MEX AM process as an "additive manufacturing process in which material is selectively dispensed through a nozzle or orifice" [9]. MEX AM can be more intuitively explained by drawing an analogy to building a castle using a hot glue gun where you build the castle one layer at a time with the molten glue. These machines are the most ubiquitous by far as the process technology is mature, the range of materials processable is wide, the process has no hazardous powder or resin like other AM processes use, and the upfront and operating cost is low [10]. Entry-level machines cost around $150 at the hobbyist level and industrial machines capable of producing engineering-grade materials being in the six-figure range [11].

1.1.1 Desktop Machines from Hobbyist to Industrial

In 2005, Dr. Adrian Bowyer started the Replicating Rapid Prototyper (RepRap) project to bring inexpensive 3D printers to the masses [12]. RepRap is an open-source project to create self-replicating 3D printers and is still alive today with open-source printers being designed such that they can print most of their own parts. The patent for *fused deposition modeling* (FDM) expired in the year 2009 and, essentially overnight, the price for a MEX machine dropped dramatically from $10,000 to $1,000; hobbyist MEX AM machines can be bought today for only $150 [11, 12]. Numerous companies have entered the hobbyist MEX market, while passionate groups of hobbyists, such as VORON Design, have created open-source designs that anyone can contribute to [13]. Most of the companies were created by hobbyists who believed in keeping everything open-source – the hardware and software. This openness led to the rapid development of software tools for MEX AM such as the popular hobbyist tool from Ultimaker called `Cura` that is commonly forked by companies for their industrial machine [14]. It is also common in the hardware space for commercial companies to use open-source motion systems and micro-controllers. That said, not all of the innovation in the MEX AM space is happening at the hobbyist / open-source level. The capital required to research and build a large-format MEX machine has limited these machines to commercial enterprises.

1.1.2 Large Format MEX Machines

Big Area Additive Manufacturing (BAAM) has a variety of other names such as Large Area Additive Manufacturing (LAAM) or Wide Area Additive Manufacturing (WAAM) — WAAM has fallen out of fashion due to confusion with Wire Arc Additive Manufacturing that shares the same acronym. BAAM machines have "big" build volumes and use pellets for the material feedstock as a cost-effective alternative to filament, see Figure 1.1 [15]. Filament MEX systems have a positive displacement extrusion system where the volume of filament pushed into the hotend is assumed to be directly correlated to the volume of molten plastic extruded. By assuming the cross section of the extrudate is a stadium, the slicer can accurately model the dimensions of the extruded road. [2,16]. Pellet MEX systems are, however, not positive displacement with respect to the volume of plastic extruded. This small difference has created a litany of issues including the development of controller strategies that attempt to

Figure 1.1: World's largest 3D printer at UMaine's Advanced Structures and Composites Center with a build volume of 100×22×10 feet, reproduced from [17].

correlate changes in screw RPM to changes in the volume of plastic extruding [8]. In practice, this limits pellet systems to empirically determining parameters such as bead width, screw RPM, and layer height for a steady state flow rate regime [16]. The print quality degrades when not at a steady state, this can include decelerating for sharp corners or extrusion starts and stops [8]. The inability for precise extrusion control means that pellet systems cannot retract to stop extruding molten plastic and travel as filament systems are able. This small difference leads to challenges in material deposition consistency, mechanical properties, geometric accuracy, and in almost every aspect of process planning. Compounding this effect is that conventional toolpath algorithms do not attempt to minimize the travel moves, which are known to lead to defect-riddled parts when used in conjunction with pellet systems. These defects are known as travel defects. The tools developed in Chapter 2, seek to reduce travel moves and therefore travel defects.

1.2 MEX AM Software

There are two software components to most MEX AM machines: the slicer and the firmware. The slicer is the software usually hosted on a separate computer to the MEX AM system that translates the desired model (e.g., STL, CLI, AMF, and 3MF) into Gcode (a programming language used to control machines and 3D printers) that the firmware can read, interpret, and execute. The firmware takes in the Gcode that the slicer generates and translates it into a buffered set of machine commands. Slicing and path planning are integral for the MEX AM process as they are the means by which a desired design becomes a set of instructions that can be fabricated. Methods for slicing are reliable and robust, but toolpath planning requires a number of decisions (algorithmic, computational, and physical) to be taken into account to get the best results possible from the MEX AM process. As such, this thesis focuses on the toolpath algorithms to meet several of these needs. A brief aside into the details of slicers and printer firmware is presented in the following subsections to provide additional context into the individual components that will be built upon to develop the novel suite of tools in Chapter 2.

1.2.1 Slicers

The process planning for most polymer MEX AM systems is carried out in the slicer, which takes in the desired object model and performs a number of tasks before the model is ready to be printed. This sequence of tasks can be divided into four routines: (1) orienting and positioning (often user-driven), (2) dividing the model into 2D layers (slicing), (3) assembling those cut layers into polygons, and (4) planning the printer's toolpath [10]. The first step is elementary if user-driven and is used to position the part into the MEX AM systems' build volume and address many design for AM principles, e.g., overhang angles and material anisotropy. Slicing, the second routine, splits the model into 2D Layers using parallel-to-the-build plate planes that are intersected with the model. The spacing of these planes is commonly a constant, user-specified distance called the "layer height"; however, algorithms do exist to dynamically vary the distance between the parallel planes [10]. Most slicing algorithms assume the model is an unordered and unstructured mesh representation such as the ubiquitous STL file format that represents the 3D model as a triangularly tessellated mesh [18]. The optimal mesh slicing algorithm is run in $\mathcal{O}(n + k + m)$ by only operating on

the set of triangles sliced by each plane, where n is the number of triangles, k is the number of slicing planes, and m is the number of triangle-plane intersection [18]. Throughout the late 1990's and early 2000's, slicing algorithms got closer to optimal runtime until Minetto et al. (2017) published an optimal algorithm that could handle adaptive layer heights and creates the contours as part of the slicing process [18]. The slicer must then organize the 2D intersections of the planes with the model into closed polygons during the third step such that the inside and outside of the polygon are noted. This is usually done by following the convention that perimeters of closed polygons are clockwise while interior holes are counterclockwise. The fourth and final step of generating toolpaths will be discussed at length in Section 1.4. In brief; it consists of space-filling and toolpath planning routines specific to the MEX AM process. The output of a slicer is a file containing machine readable instructions of the toolpaths and various commands needed to build the part, usually in the form of a Gcode file.

Gcode was invented and first used in the MIT Servomechanisms laboratory in 1958 during the research of computer numerical control (CNC) machining systems [19]. Gcode was initially known as the Automatically Programmed Tool (APT) and an attempt was made to standardize Gcode with the RS-274 Electronic Industries Alliance standard in 1962 and the ISO 6983 standard first published in 1982 [20]. The ISO 6983 standard failed to adequately define Gcode due to a variety of reasons such as: the path of the cutter is with respect to the machine axis *not* part axis, semantics of program statements left unclear, uni-directional flow of information making changes on the shop floor impractical, and an incapability of handling splines, making Gcode unusable for five plus axis machines [20, 21]. This led to each vendor enhancing the language in non-standard ways (called flavors of Gcode), combined with the rise of five plus axis machines has seen Gcode replaced in new CNC machines. The novel suite of tools in Chapter 2 are built to work with the widely used `Marlin` flavor of Gcode [22].

The most common slicers are open-source slicers that benefit from both hobbyist creating new features in their free time and companies building on top of these open-source slicers for their commercial machines. The licensing for these open-source slicers often requires those who modify them to publish the changes forcing companies to give back to the community when they use an open-source slicer [23]. One of the most popular slicers is `Ultimaker Cura`, which as of 2019 had 500,000 unique users per month and processes 1.4 million parts per week [24]. Another popular

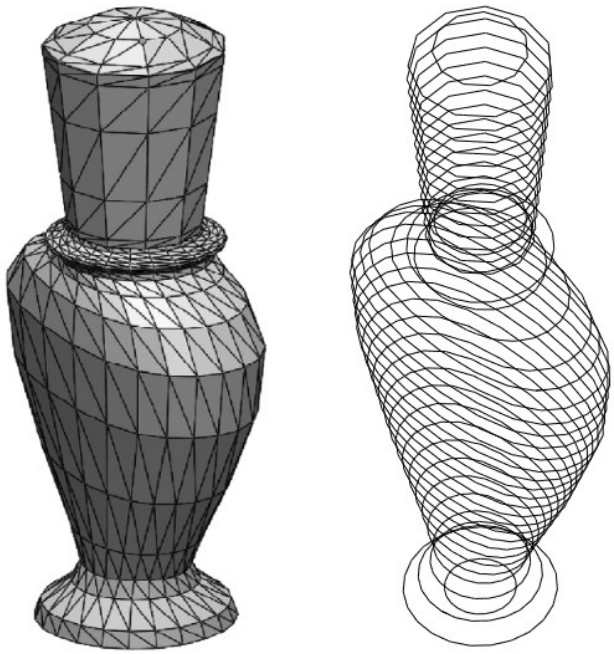


Figure 1.2: A triangle mesh of the surface of the 3D object (an STL) and the 3D object after being sliced. Reproduced from [18].

slicer is `PrusaSlicer`, a fork of one of the original open-source slicers, `Slic3r` [25]. `PrusaSlicer` quickly outgrew `Slic3r` with a dedicated team of developers overhauling the code base from Perl to C++ and revamping the user interface, but `PrusaSlcier` and `Slic3r` still merge features between each other [25]. Custom slicers from companies that do not build off of an open-source slicer do exist, but the they act as a black-box with no information about the unique toolpath algorithms used and little to no means of modifying the Gcode before use.

1.2.2 Firmware

Firmware is a low-level software that controls a device's hardware. For MEX AM machines, the firmware translates the Gcode from the slicer into command signals for the stepper motors, fans, heater, and any other miscellaneous hardware integrated into the MEX AM system. As discussed earlier, Gcode is not standardized among vendors, so each firmware is designed to translate a specific flavor of Gcode. The most popular firmware flavor is called `Marlin` and its widespread adoption has driven its dictionary to be the defacto standard [22]. `Marlin` grew out of the popular open-source CNC firmware `grbl` and an early MEX AM firmware called `Sprinter` [22, 26, 27]. Modern

firmwares will look ahead to determine cornering speeds between moves instead of fully decelerating at the end of every movement command as well as smooth out acceleration [7]. This helps mitigate disruptions to the steady state conditions that pellet MEX AM printing parameters are empirically determined.

Firmware performs other computation on the Gcode to attempt to balance commands versus actions in the open-loop control architecture of most MEX AM systems. One such computation is the determination of material flow. In a filament extrusion system, the deposition is considered positive displacement. This assumption is largely true when in a steady state of constant extrusion. A change in the volume of extrusion being pushed through the hot end pressure causes a lag in the volume of filament extruded. Modern firmwares use a model called pressure advance to accurately model the pressure effects in the extrusion system through a linear coefficient, see Figure 1.3 [7]. This linear coefficient is empirically determined and controls how much additional filament is used during accelerations and how much extra filament retracts during accelerations. Future work could pair the tools built in Chapter 2 with a modified pressure advance model for pellet MEX AM systems.

1.3 Bonding Mechanisms in MEX AM

The bonding mechanism in polymer MEX AM is driven by the thermal energy between adjacent roads [28]. If the temperature at the interface of the two roads drives the energy at the interface to or above the activation energy, then reptation across the interface will occur [4, 29, 30]. Reptation, the snake-like movement of polymer chains above the activation energy, entangles polymer chains across the interface boundary [4, 30]. As the entanglement across the boundary approaches the bulk entanglement of the bulk polymer, the interface strength approaches the strength of the bulk polymer and is considered fully healed [4]. This interface healing is depicted in Figure 1.4 which is reproduced from Grewell et al. (2007) [4]. The speed of reptation is exponentially dependent on the interface energy and thus the interface temperature [4, 30]. To maximize the strength of bonds between adjacent roads in MEX AM, the roads should be deposited sequentially to minimize the time the first deposited road has to cool. The infill combination strategies in Chapter 4 minimize the deposition times between adjacent roads with aspirations of increasing bond strength in the polymer.

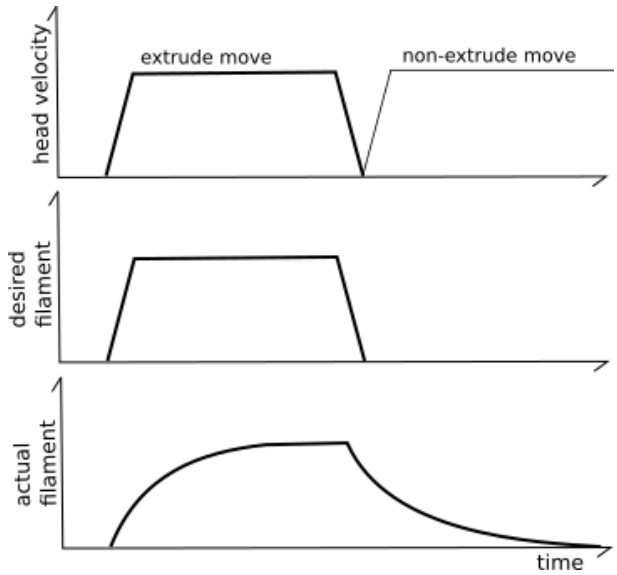


Figure 1.3: The pressure advance system models the extruder based on pressure, not on the assumption that the system is a purely positive displacement system. The pressure in the extruder increases when filament is pushed into the extruder and the flow rate dominates the pressure needed to extrude the molten plastic. The top figure depicts an extrusion move followed by a travel move, the other two figures depict the commanded amount of filament extruded and the actual amount of filament extruded without pressure advance. Reproduced from [7].

1.4 MEX AM Toolpath Design

There have been various studies on how infill patterns, the structure and shape of the material inside a part such as lines, grids, and gyroids, and relative density affect the final part strength. Yadav et al. and, separately, Abbas et al. specifically looked at the compressive strength of various infill patterns and found that it increased with infill percentage regardless of tested infill pattern type [31, 32]. A similar study of infill parameters' affect on tensile strength was conducted by Panzdzic et al., who found that ultimate and yield strength increases with infill density for every infill pattern tested [33]. The orientation of the part as printed and therefore the orientation of the infill as-printed has a dramatic effect on part strength [34]. Varying the infill pattern and percentage on a layer-by-layer basis to optimize the strength and weight of a part was studied by Dave et al. [35]. It was found that at lower infill densities, strength of the part can be optimized for specific loading scenarios by stacking specific sequences of different infill patterns [35]. Mechanically interlocking infill has been

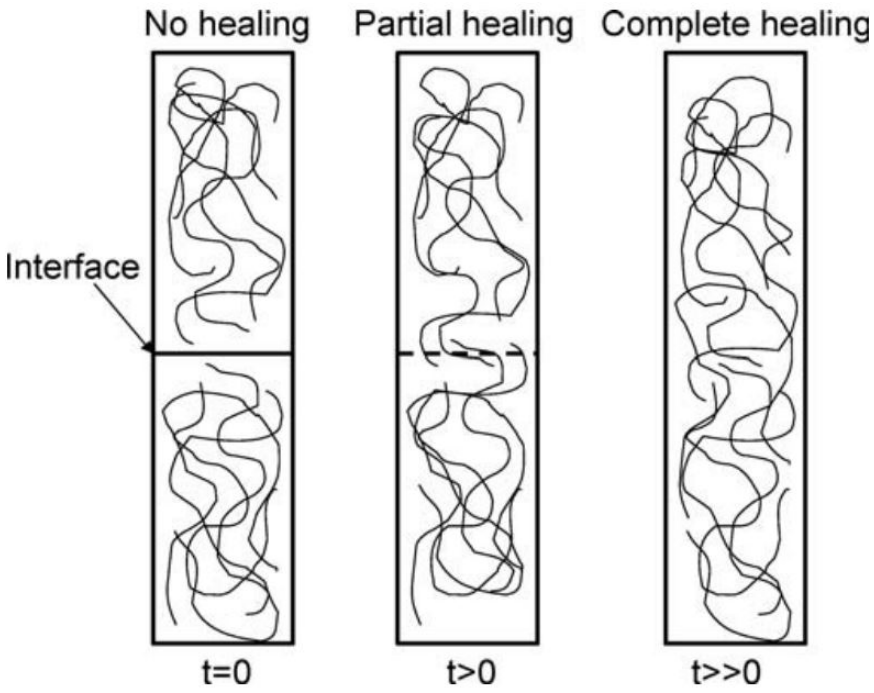


Figure 1.4: Polymer interface hearing when the interface is above the activation energy, reproduced from [4].

used to improve the interface strength of multi-material parts up to the bulk strength of the material [36].

There have been various attempts to create an algorithm that produces the optimal toolpath, with optimal being defined as the toolpath that gives the highest strength for a specific load case or the toolpath that completely fills space. Xia et al. (2020) designed a stress-based toolpath algorithm that constructs toolpaths parallel with the maximum principal stress directions with a depth-first search and then used Dijkstra's algorithm to minimize travel distance [37]. The limitation of Xia et al.'s (2020) work is the algorithm only considers 2D complex shapes with a single load case [37]. Shaikh et al. (2016) developed a toolpath algorithm that follows the Hilbert Curve, which is a continuous space-filling fractal [38]. This approach is implemented in most slicers and can be selected as the Hilbert Curve infill pattern. Yoo et al. (2020) used a Monte Carlo Tree Search algorithm to efficiently fill space and reduce travel moves, but the method was significantly slower than conventional space-filling toolpath algorithms [39]. Most recently Borish et al. (2023) developed a toolpath algorithm that generates a single path for closed contours by using graph theory and topological hierarchy [40].

This thesis takes a holistic approach to toolpath design, aspiring to create toolpath algorithms that reduce defects in parts and increase mechanical part properties.

ORNL `Slicer 2` is an evolution of ORNL `Slicer`, which was the first purpose-built slicing software for large-scale printing [41]. `Slicer 2`'s current approach for travel moves is to try and minimize their affect on print quality by modifying the beginning and ending of travel moves. For instance a *Tip Wipe*, which is a motion used to wipe the tip of the nozzle and break away the extrusion bead, is implemented at the beginning of travel moves [42]. This addressing of the symptoms caused by travel moves is effective, enough to consistently get adequate parts, up to a point. The tools presented in Chapter 2 seek to address the root of the problem, but future work that combines both approaches could greatly improve the chronic inconsistency of pellet MEX AM parts.

1.5 Graph Theory

Graph theory is a relatively new mathematical field that models the pairwise relation between objects. A graph is defined by an ordered pair (V, E) where V is a finite, nonempty set called *vertices*, and E is a set of unordered pairs of vertices called *edges* [43]. A vertex u is adjacent to a vertex v if the unordered pair $\{u, v\}$ is in E, i.e., the vertices are *connected* to one another via the edge $e \in E$ [44]. Graphs are commonly presented visually by representing each node as a point in a plane and each edge as a line connecting the vertices [45]. An example of this visual representation can be seen in Figure 1.5, where each node is labeled with a number, and the edges connecting the nodes are denoted as solid lines. An example of the set notation from Figure 1.5 would be the edge between nodes 2 and 7 being denoted in the edge set E as $\{2, 7\}$ or as $\{7, 2\}$ since edges are unordered pairs. The *degree* of a vertex is defined as the number of incident edges (edges that connect to the node) [46]. For example, every node in the Peterson graph, depicted in Figure 1.5, has a degree of 3.

The geometric position of the points, the path the edges take between nodes, and the length of the edges (typically) hold no meaning. The graph is usually drawn such that no edges intersect other edges, to aid in human readability [49]. A graph that can be drawn without such intersection is defined as a planar embedding of that graph and said graph is considered a *planar graph* [46]. A geometric graph is a graph whose nodes are defined by geometric means [50]. For our purposes, we will be discussing

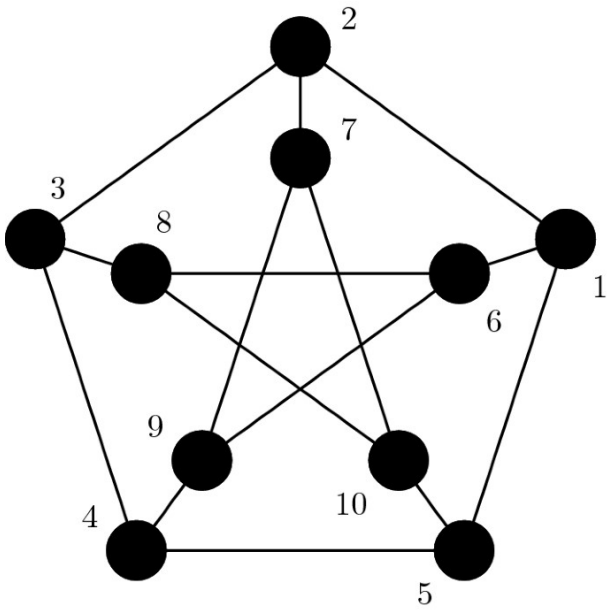
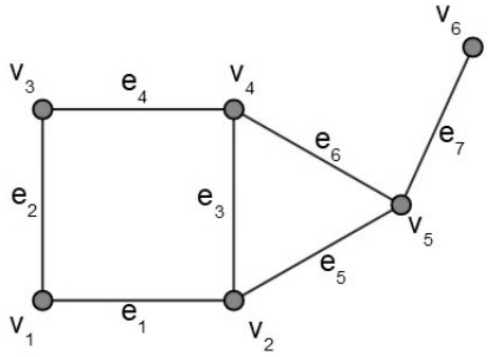


Figure 1.5: Peterson Graph with nodes labeled 1-10, reproduced from [47].

	v_1	v_2
e_1	1	2
e_2	1	3
e_3	2	4
e_4	3	4
e_5	2	5
e_6	4	5
e_7	5	7

(a) Adjacency List



(b) Graphical Representation

Figure 1.6: An adjacency list and its corresponding graphical representation. Each node is defined in a separate list as a vertex in 2D space. Reproduced from [48].

geometric graphs in the Euclidean plane, but other topological geometric graphs have a deep depth of research and practical applications [50]. Nodes in this work will be defined by their coordinates in a 2D coordinate space and the edges will be the unordered pair of the indices of the nodes [48]. The *edge list* will be referred to as the *adjacency list*; an example of an adjacency list and its graphical representation is shown in Figure 1.6.

For practical applications it is often useful to assign a weight to the edges to represent some cost or price, commonly referred to as distance in geometric graphs, to traverse that edge. Weights are used in a variety of applications such as connecting landline nodes or planning roads as they directly map to physical distances in the system [43]. A common practical problem to solve with a weighted graph is to find the shortest distance between two vertices within a graph. As such, a variety of robust algorithms have been developed to solve the so-called shortest path problem [43]. A popular greedy algorithm, a greedy algorithm is one that makes the locally optimal choices at each decision step to solve the problem, to find the shortest distance is Dijkstra's algorithm. [44]. This heuristic strategy rarely results in the optimal solution but it does, in many cases, provide a near-optimal solution significantly faster than the optimal solution could be found [44]. Dijkstra's algorithm is popular in solving the shortest path problem due to its runtime of $\mathcal{O}(|V^2|)$, or in other words, the runtime of the algorithm scales by the number of vertices in a graph squared [44]. There are faster shortest-path algorithms for certain types of graphs, but Dijkstra's is considered the fastest shortest-path algorithm that can handle any graph well. We will use these robust shortest-path algorithms as a framework for the toolpath algorithm that will allow the algorithm to robustly handle complex layer geometry.

Chapter 2

GRATER: GRAph Theory based slicER

This work has developed a suite of tools collectively known as the **GRA**ph **T**heory based slice**R** (GRATER) that seeks to improve the MEX AM process by reducing travel moves. As discussed in the previous chapter, travel defects are inherent to travel moves; therefore, reducing travel moves will reduce travel defects, reduce the amount of post-processing necessary, and reduce print time. The reduction of travel moves is especially relevant to the pellet extrusion systems found on large format MEX AM systems due to their inherent difficulties in extrusion control as discussed in Section 1.1.2. This chapter will discuss the structure and algorithms used by GRATER in detail with the next chapter will discuss the results of using GRATER to manufacture parts.

2.1 Introduction

GRATER was written in `MATLAB` and currently ingests ASCII STLs, binary STLs, or NaN-separated contours (similar to some laser additive manufacturing processes). GRATER currently outputs `Marlin` flavored Gcode for filament MEX AM systems or for the Juggerbot 3D's P3-44. The P3-44 uses a pellet-fed screw extrusion system that can output up to fifteen pounds of filament an hour [51]. The P3-44 runs a limited version of `Marlin`, with the main difference being that a screw RPM is commanded

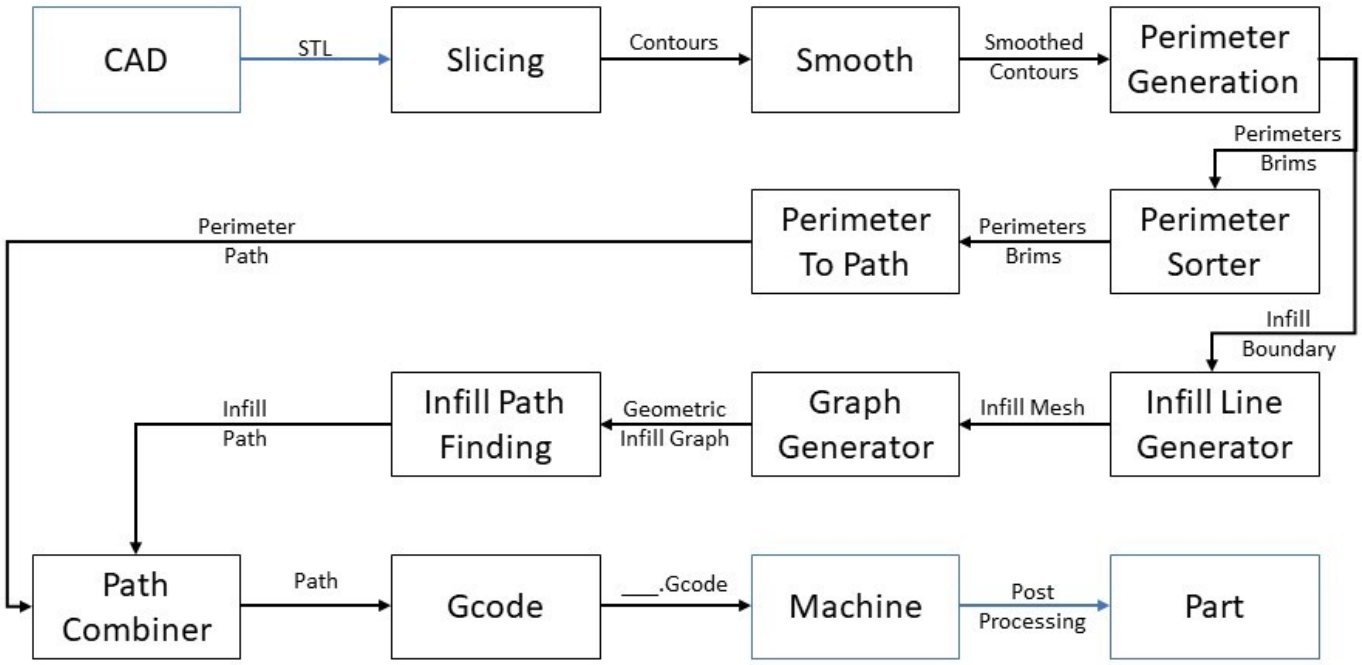


Figure 2.1: This diagram represents the structure of GRATER. Every black box is a separate `MATLAB` function and the inputs and outputs of those boxes is the major data being passed between functions. The blue boxes represent the surrounding steps in the MEX AM process with the blue inputs and outputs being inputs and outputs to GRATER.

instead of an extrusion value. The structure of GRATER is described in Figure 2.1. Each black box in Figure 2.1 represents a function in `MATLAB` and the inputs and outputs represent the variables being passed into and out of the functions. The blue boxes are the rest of the MEX AM process that are either an input into GRATER or the output of GRATER. Each function will be discussed at length in this chapter, with each function being a chapter subsection.

2.2 Main

The **Main** `MATLAB` file calls all the other functions for GRATER as well as housing the user-defined options that control the slicing and toolpath generation process. The user-defined options are listed in Table 2.1 along with brief descriptions. The user-defined options will be discussed in greater detail as the various functions of GRATER use them in the subsequent sections.

The decision to house all of the functions in **Main** and limit the number of times a function is called outside of **Main** allows for quick debugging modularity of the code. This is useful for factory methods when you have other MEX AM systems in the future. For instance, the **Graph Generator** function could have been called from the **Infill Line Generator** function. However, having the data from the **Infill Line Generator** function be passed into main and then passed into the **Graph Generator** function allows for debugging of the **Infill Line Generator** function and the **Graph Generator** function separately. The **Main** `MATLAB` file is sectioned into groups of functions with functions that take a particularly long time being grouped by themselves. This lets the user change user inputs such as hot end temperature without having to go through the entire slicing process. Re-slicing an entire part when a small change in slicing parameters is a common friction point with contemporary slicers. The **Main** `MATLAB` file housing all of the functions makes it trivial to add a GUI as well as bundle everything into an executable for the public release of GRATER.

2.3 Slicing

GRATER can accept STL files in either the binary or plain text format (ASCII). The recommendation is to use binary STLs as the plain text STLs become cumbersome to transfer between computers or folders, due to size, at the model scale GRATER is

Table 2.1: User Inputs to GRATER

Slicing Parameter	**Function**
Slice Height	Height in millimeters which the STL will be sliced.
Layer Height	Height of the printed layers in millimeters.
Number of Perimeters	The number of perimeters of the printed part.
Perimeter Extrusion Width	The width in millimeters of the printed perimeters.
Infill Extrusion Width	The width in millimeters of the printed infill.
Infill Percentage	The percentage of interior space filled by the infill, defined in decimal form.
Infill Overlap	The percentage that the infill overlaps the perimeter.
Bed Temperature	The temperature the bed of the printer will be commanded to maintain throughout printing.
Hot End Temperature	The temperature the nozzle of the printer will be commanded to maintain throughout printing.
Weight Perimeter	Weighting of the perimeter paths when generating the graphs used to find continuous toolpaths.
Weight Infill	Weighting of the infill paths when generating the graphs used to find continuous toolpaths.
Smooth Step	Resampling step for contours generated from slicing the STL in millimeters.
Number of Brims	Number of brims generated for the first layer.
Infill Angle	The angle in degrees from the X-axis the infill will be rotated to.
Perimeter Starting Angle	The counter-clockwise angle used to determine the start of perimeter paths.
Angle From	The point from which the perimeter starting angle is measured.
Gcode File Name	Name of outputted Gcode file.
STL Name	Name of STL to slice.

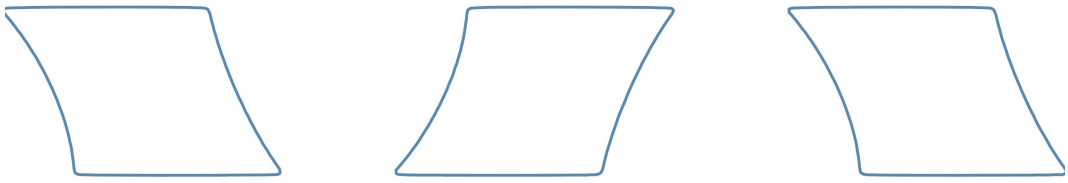


Figure 2.2: This is an example of an output from the **Slicing** function. Each of these closed contours is [NaN,NaN] separated.

designed for [52]. To extract closed contours from the STL, an open-source collection of `MATLAB` functions hosted on the `MATLAB` File Exchange[1] under the BSD[2] license was used. This collection of scripts developed by Sunil Bhandari is exceptionally fast and scales well with larger STL files to identify contours by intersecting the triangles of the STL with a plane to generate closed contours [53]. The user inputs the Slice Height that defines the interval between the planes (also known as the layer height) used to intersect the triangles. The closed contours are stored in a 1 by n cell where n is the number of layers generated. Within the cell, the contours are stored in an m by 2 array of x, y points, with NaNs separating separate closed contours. Figure 2.2 showcases the contours generated by Bhandari's open-source slicer.

2.4 Perimeter Generation

The inputs into the **Perimeter Generator** function in GRATER are the contours from the **Slicing function**, Perimeter Extrusion Width (ϵ_p), Infill Extrusion Width (ϵ_{in}), Infill Overlap Percentage (ϕ), and the number of Brims (β).The user inputs are further explained in Table 2.1. The perimeter and infill extrusion width were kept separate to allow the user more options as well as enable future features to be implemented without having to rewrite the function.

The structure of the perimeter generator function is detailed in Algorithm 2.1. The contours are read into the function then the function generates brims and perimeters using the infill and perimeter extrusion width to determine spacing for the POLYBUFFER function. The `MATLAB` function POLYBUFFER[3], which takes in a polygon and returns a polygon with boundaries that are buffered from the input polygon by the specified

[1] https://www.mathworks.com/matlabcentral/fileexchange/
[2] https://opensource.org/license/bsd-3-clause/
[3] https://mathworks.com/help/matlab/ref/polyshape.polybuffer.html

Algorithm 2.1: The **Perimeter Generation** algorithm takes in the contours generated by the **Slicing** function and leverages the `MATLAB` function POLYBUFFER to buffer perimeters, brims, and the infill boundary from the contour.

Input: Contours, ρ, ϵ_p, ϵ_{in}, ϕ, β
Output: Perimeters, Infill Boundary, Brims

```
function perimeterGenerator:
    Define Perimeters, Infill Boundary, Brims
    for n = 1 to η do
        if n == 1 then
            use Polybuffer to generate Brims
        end
        for j = 1 to ρ + 1 do
            use Polybuffer to generate Perimeters and Infill Boundary
        end
    end
```

distance. Brims and perimeters are separated by one Perimeter Extrusion Width and the infill boundary is separated from the last perimeter by one Infill Extrusion Width. The infill boundary is used in the infill generation and path-finding steps. If a road is coincident with the infill boundary, it will be connected to the perimeter. The perimeters and infill boundary buffered from the contour are shown in Figure 2.3. The structure of the output of the function is a $\{1, \eta, \rho\}$ cell of $(n, 2)$ arrays, where n is an arbitrary number of rows needed to define the geometry of the polygons. The arrays contain closed contours defined by $[x, y]$ points and separated by [NaN,NaN].

The brims generated are contained in a similar structure but the cell is a $\{1, 1, \beta\}$, as brims are only generated for the first layer. The brims help with adhesion to the buildplate and help mitigate warping. It is common to offset the brims and perimeter by a slight amount to aid in the removal of the brims. However, GRATER does not contain this feature (at this time) as warping is a significant issue when printing on large format MEX AM systems for which GRATER is designed. The level of warping at the large format MEX AM scale can lead to offset brims breaking away from the outer perimeter completely, negating all benefits of brims.

Figure 2.3: The first perimeter / brim is buffered half an extrusion width from the contour, see Figure 2.2, while the rest of the perimeters / infill boundary / brims are buffered a whole extrusion width from the previous.

2.5 Perimeter Sorter

The **Perimeter Sorter** algorithm in Algorithm 2.2 takes in the perimeters and brims from Algorithm 2.1 and outputs the perimeters in the same data structure, but the $(n, 2)$ lists are sorted such that one perimeter starts near the previous perimeter ended. The starting point for the first perimeter or brim on the first layer is defined by the point closest to the Perimeter Starting Angle measured from the Angle From user-defined option. The other perimeters or brims are sorted by the closest point from the end of the previous perimeter first. The perimeters generated from Algorithm 2.1 are ostensibly in the same direction and order, but frequently the direction of the perimeter (clockwise or counterclockwise) is not maintained when buffering from a previous perimeter, due to quirks of the POLYBUFFER function. If the perimeters were already in order then the arrays could simply be concatenated together from the outside-in or the inside-out, whichever is preferred. In practice, edge cases such as inner perimeters being squeezed into more contours than the perimeters before them break this assumption frequently. This is part of what lets GRATER start the next layer where the previous layer ended.

2.6 Perimeters To Paths

The **Perimeter To Paths** algorithm in Algorithm 2.3 takes in the sorted perimeters and brims from Algorithm 2.2, which are in the form $\{1, \eta, \rho\}$ and $\{1, \eta, \beta\}$, and returns the variable Perimeters Path in the form of $\{1, \eta\}$. If the perimeters contain

Algorithm 2.2: The **Perimeter Sorter** algorithm sorts the perimeters and brims such that their beginning vertex is the vertex closest to where the previous perimeter ended.

Input: Perimeters, θ, $\vec{\nu}$
Output: Perimeters Sorted

```
1  function perimeterSorter(graphs):
2      for n = 1 to η do
           // Loops over every Layer
3          for i = 1 to ρ do
               // Loops over every Perimeter
4              for j = 1 to χ do
                   // Loops over every Contour
5                  if n == 1 & i == 1 & j == 1 then
6                      Sort Contour by closeness to θ
                   // Sorts Contours within a single Perimeter
7                  else
8                      Sort Contour by closest point to previous contour
9                      Perimeters Sorted ← Sorted Contours
10                 end
11             end
12         end
13     end
```

different numbers of contours then the algorithm inserts [NaN,NaN] between the contours to signify a travel move is necessary, otherwise, the perimeters are simply concatenated. Once all of the perimeters are sorted by contours, the algorithm orders them such that the travel distance is minimized by choosing the first contour by its closeness to the end of the previous layer, and then the next contours are chosen by the closeness at the end of the first contour. The contours are reordered such that they start at the closest point found in the previous step. This does not guarantee the total distance traveled is minimal as the algorithm does not use any look ahead strategy, but it is fast and prevents any exceedingly long or illogical ordering within a layer. Future work could explore other algorithms to minimize total travel distance.

2.7 Infill Line Generator

The **Infill Line Generator** of Algorithm 2.4 creates the rectilinear infill pattern represented as a graph, such as the infill seen in Figure 2.4, by creating horizontal scan lines. The **Infill Line Generator** function then rotates the scan lines by the

Algorithm 2.3: The **Perimeter To Paths** algorithm takes in the sorted perimeters and brims from the **Perimeter Sorter** algorithm and combines them into a continuous path.

```
   Input: Perimeters Sorted, Brims
   Output: Perimeter Paths
 1 function perimeterToPaths(graphs):
 2     for n = 1 to η do
           // Loops over every Layer
 3         for i = 1 to ρ do
               // Loops over every Perimeter
 4             for j = 1 to χ do
                   // Loops over every Contour
 5                 Concatenate perimeters from separate cells to one cell
 6             end
 7         end
 8     end
 9     for k = 1 to η do
10         if k == 1 then
11             PerimeterPaths ← vertcat(brims, Contour{k, 1})
12         else
13             Sort Contour by closest point to previous contour or Last Point
14             Perimeter Paths ← Sorted Contours
15             LastPoint ← PerimeterPath{1, k − 1}(end, :)
16         end
17     end
```

user-defined input Infill Angle and intersects the rotated lines with the infill boundary generated in Algorithm 2.1. The `MATLAB` command POLYXPOLY[4] is used for the intersection as the intersection indices it generates are helpful when creating the graph in Algorithm 2.5 [54]. The vertical distance between the horizontal scan lines is defined by Equation 4.1. Figure 2.4 shows these scan lines in red intersecting the infill boundary in blue. To aid in the next algorithm, the intersection points found by intersecting the scan lines with the infill boundary are included in the array defining the infill boundary. The closest point in the infill boundary is replaced by the intersection point. Replacing the point instead of inserting a new point lets GRATER skip updating the intersection indices generated by POLYXPOLY. While this does slightly change the exact shape of the infill boundary, the computation gains and readability of the codebase are well worth the slight change. With a point on the infill boundary every 0.1mm, the change is never enough to affect the printed part. This

[4]https://www.mathworks.com/help/map/ref/polyxpoly.html

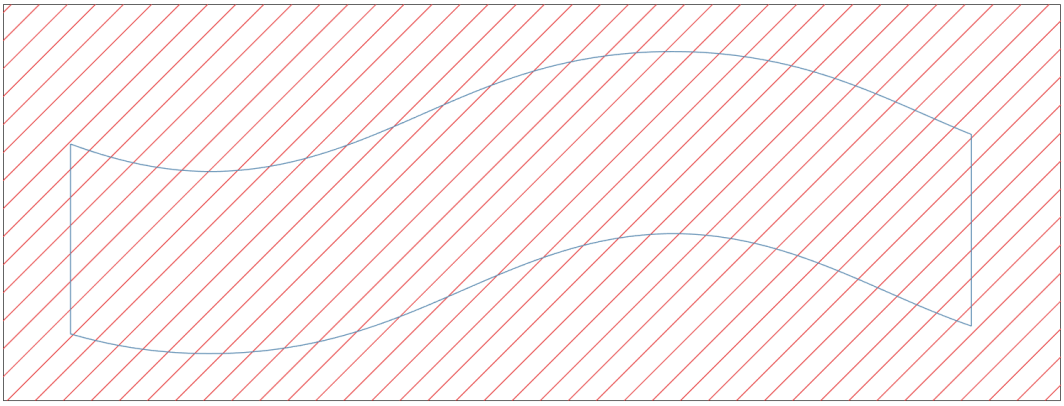


Figure 2.4: Infill lines (red) intersecting the infill boundary (blue). The intersections are recorded and used to make the infill for the infill path-finding function to traverse.

updated infill boundary is outputted as well as the variable *scanInt*, which has the form $\{1, \eta, k\}$ where $k = 1$ is the (x, y) intersection points and $k = 2$ is the line segment indices generated by POLYXPOLY. Edge cases, such as an infill line intersecting only one vertex on the infill boundary, or an infill line intersecting the infill boundary more than twice must be considered, or the infill will end up outside the contour. Such as in a C shape, there will be intersections of the infill lines connecting the convex part of the C. If an infill line has more than two intersections the resulting infill line is checked to see if it is inside the contour. This ensures infill lines connect to the infill boundary and are interior.

2.8 Graph Generator

The **Graph Generator** in Algorithm 2.5 operates on the modified infill boundary and the scanInt variables from Algorithm 2.4 and the Weight Perimeter and Weight Infill parameters. The **Graph Generator** algorithm generates a weighted, geometric graph, and stores it in a variable called graphs. The variables are a $\{1, \eta, k\}$ cell with $k = 1$ being the node list, $k = 2$ the edge list, and finally $k = 3$ the weight list for every layer η. The node list is an $(n, 2)$ array of (x, y) points. The edge list is also an $(n, 2)$ array where each row contains the indices of the two nodes in the node list that the edge connects. The weight list is a column vector of the same length as the edge list where the weight in row n corresponds to the weight of the edge in row n of the edge list. The algorithm creates two separate graphs and then combines them: (1) the interior

Algorithm 2.4: The **Infill Line Generator** algorithm creates rectilinear infill by intersecting scan lines with the infill boundary.

```
   Input: ϵ_in, μ, ω,Infill Boundary
   Output: scanInt, updated Infill Boundary
1  function infillLineGenerator(graphs):
       // This Algorithm is specifically for Rectilinear Infill
2      Define: χ
3      for n = 1 to η do
           // Alternates infill Angle
4          if mod(n, 2) == 0 then
5              χ ← ω − 90
6          else
7              χ ← ω
8          end
9          Generate horizontal lines
10         Rotate horizontal lines by χ in 2D
11         Intersect rotated lines with Infill Boundary
12         scanInt ← intersection points & indices
13         for n = 1 to size(scanInt) do
14             updated Infill Boundary ← Replace closest point in Infill Boundary with
                scanInt(n,:)
15         end
16     end
```

infill making up the edges spanning perimeters in the rectilinear infill, and (2) the graph of the infill boundary. Concatenating these graphs forms the rectilinear infill, this concatenation happens at the end of the algorithm to form one geometric graph. Creating the graphs separately and then combining them vastly simplifies creating the graphs.

> **NOTE:** The **Graph Generator** algorithm can have *any pattern* passed into it including non-traditional infill patterns and 2D meshes so long as the input follows the data structure described above — the final result will always be a geometric graph. The pattern must intersect the infill boundary for the algorithm to find a path that inlcudes the patern. The weights of the edges are currently user-defined variables that require some manual iteration to yield the desired output. For example, if the weight ratios are off then the infill will be zigzag instead of rectilinear. In future work, the weight of the edges will be mathematically determined.

Algorithm 2.5: The **Graph Generator** creates geometric graphs that contain the infill boundary and the infill.

Input: updated Infill Boundary, scanInt, λ_{in}, λ_p
Output: graphs

```
function graphGenerator(graphs):
    Define: tempNodesB, tempnodesI, edgeS, edgeB weightS, weightB,
     tempNodesCombined, tempEdgesCombined, tempWeightCombined, graphs
    for n = 1 to η do
        // repeat for each contour if necessary
        tempNodesB ← updated Infill Boundary{n,1}
        tempNodesI ← scanInt{1,n,1}
        Use the indices in scanInt to add edges between the nodes in scanInt
         to the corresponding nodes in updated Infill Boundary
        edgeS ← indices found above
        // The Infill Boundary nodes are simply connected sequentially
        for k = 1 to length(tempNodesB) − 1 do
            edgeB(k,:)=[k, k+1]
        end
        weightS ← zeros(length(edgeS(:, 1)), 1) + λ_in
        weightB ← zeros(length(edgeS(:, 1)), 1) + λ_p
        tempNodesCombined ← vertcat(tempNodesB, tempNodesS)
        tempEdgesCombined ← vertcat(edgeB, edgeS)
        tempWeightCombined ← vertcat(weightB, weightS)
        graphs{1,n,1} ← tempNodesCombined
        graphs{1,n,2} ← tempEdgesCombined
        graphs{1,n,3} ← tempWeightCombined
    end
```

2.9 Infill Path Finding

The **Infill Path Finding** in Algorithm 2.6 leans heavily on the `MatGeom` library for its robust geometric graph theory functions [55]. For each contour in each layer, the GRAPHPERIPHERALVERTICES (from `MatGeom`) function is used to find the peripheral vertices of the graph [55]. The eccentricity of a vertex is the maximum distance from that vertex to any other vertex in the graph [45]. The peripheral vertices are the vertices whose eccentricity is the maximum, and the maximum eccentricity is the diameter of a graph. This finds the two furthest, by edge weight, vertices from each other so we can find the longest continuous path. This function takes up most of the runtime, but simply using the linear distance between vertices in the x, y plane does not guarantee the furthest path between two nodes. This is apparent when using a

Algorithm 2.6: The **Infill Path Finding** algorithm uses Dijksra's algorithm to find the longest path in the graph generated by **Graph Generator**.

Input: graphs
Output: infill Path

function `infillPathFinding(`*graphs*`):`
  Define: infillPath, nodesSplit, edgesSplit, weightsSplit, nodes, edges, edgeWeights, ssIndices, shortPathIndices, shortPath
  for $n = 1$ **to** η **do**
    `// splitting polygons`
    nodesSplit $\leftarrow$ $splitPolygons(graphs\{1, n, 1\})$
    edgesSplit $\leftarrow$ $splitPolygons(graphs\{1, n, 2\})$
    weightsSplit $\leftarrow$ $splitPolygons(graphs\{1, n, 3\})$
    for $k = 1$ **to** $length(nodesSplit)$ **do**
      nodes $\leftarrow$ nodesSplit{k}
      edges $\leftarrow$ edgesSplit{k}
      edgeWeights $\leftarrow$ weightsSplit{k}
      `// finds the nodes with max eccentricity`
      ssIndices $\leftarrow$ $graphPeripheralVertices(nodes, edges)$
      `// finds shortest path between the two nodes found above`
      shortPathIndices $\leftarrow$ $grShortestPath(nodes, edges, ssIndices(1)...$
      $..., ssIndices(end), edgeWeights)$
      shortPath $\leftarrow$ $nodes(shortPathIndices)$
      if $k == 1$ **then**
        infillPath{1,n} $\leftarrow$ shortPath
      else
        infillPath{1,n} $\leftarrow$ $vertcat(infillPath\{1, n\}, [nan, nan], shortPath$
      end
    end
  end

convex geometry such as a C shape. The function uses Dijkstra's algorithm to create a shortest path tree and determine the eccentricity of every vertex [43].

These peripheral vertices and the geometric graph are passed into the grShortestPath function from `MatGeom` to find the shortest path between the vertices [55]. The grShortestPath function can take in negative weights which can be used to find the longest path. This is how GRATER identifies the longest path between the two peripheral vertices while using a shortest-path algorithm. This function also uses Dijkstra's algorithm; however, it uses the classical version of Dijkstra's algorithm to only find one path instead of creating a shortest path tree. An example of a path found by this algorithm can be seen in Figure 2.5 This ensures the function is fast, but since Dijkstra's algorithm is a greedy algorithm, GRATER does not provide any

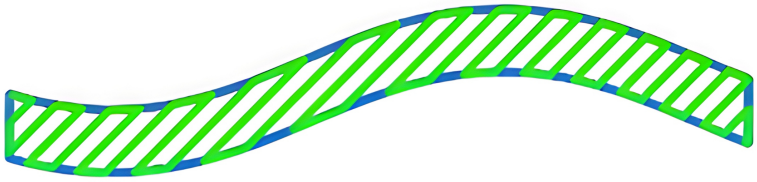
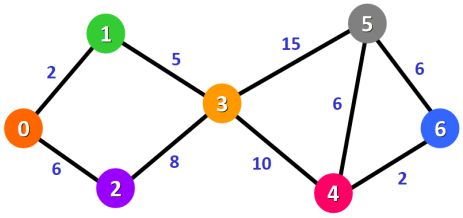
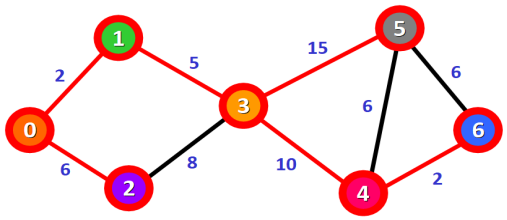


Figure 2.5: The blue lines represent the edges of the infill boundary and infill. The green line is the continuous path that Algorithm 2.6 found that is deposited during the MEX AM process.



(a) Starting Graph



(b) Red lines denote the shortest path from node 0 to every other node.

Figure 2.6: This figure is an example of how Dijkstra's algorithm works for a simple graph. The algorithm finds the shortest path from node 0 to every other node. Reproduced from [56].

guarantee that the path chosen is the "shortest path" [43]. This means the path found may not capture all of the infill, future work could explore using non-greedy shortest path algorithms to guarantee capturing all of the infill.

An example of Dijkstra's algorithm is given in Figure 2.6 [56]. Dijkstra's algorithm either starts at a node you choose and finds the shortest path between the starting node and every other node in the graph. For the purpose of GRATER we will use negative edge weights to get the longest path. By definition this path will be the longest, continuous, visit no edge twice, path within the graph from the starting node to the largest node. The red edges in Figure 2.6b represent the paths Dijkstra's algorithm found. In order to capture the greatest possible amount of infill we do this process starting at every node to determine each node's eccentricity. The path Dijkstra found between the two nodes with the highest eccentricity is the longest path possible in the graph, an example of this is in Figure 2.7. This longest path possible is chosen to be the infill path.

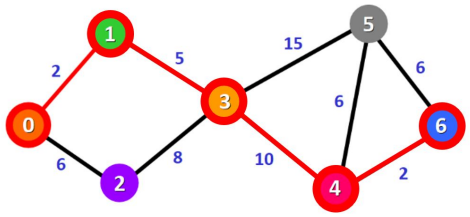


Figure 2.7: The longest minimal path between two nodes that Dijkstra's algorithm finds when starting at node 0. Modified figure from [56].

2.10 Path Combiner

The Path Combiner in Algorithm 2.7 takes in the infill path from Algorithm 2.6, and the perimeter path from Algorithm 2.3, and combines them into one continuous path. The closest perimeter point to the beginning and the end of the infill is found. Then depending on a user-defined input, either a short travel is inserted to reach the closest point on the next layer from where the previous layer ended or a small portion of the next layer is not traveled. If the index of the perimeter point closer to the beginning is higher than the index of the perimeter point closest to the end of the infill, then the perimeter is truncated to that index and concatenated with the infill path to form one path. If the opposite is true then the perimeter is truncated to the index closest to the end of the infill and the perimeter is truncated with the flipped infill to form one path. The truncation means, the inner perimeter may not be fully printed and should be considered a sacrificial perimeter. The ability to flip the infill is unique to GRATER as each closed contour is guaranteed to have a single continuous infill path. This process is repeated for every contour on each layer.

2.11 Gcode

The Gcode algorithm takes in the Path variable from Algorithm 2.8 and creates a Gcode file that a 3D printer can read. Specifically, the Gcode algorithm writes Gcode for machines that run Marlin firmware or firmware loosely based on Marlin, like the Juggerbot Tradesman Series P3-44. The Gcode algorithm calls a start function at the beginning and an end function after going through the entire Path variable. These functions write printer-specific Gcode to the Gcode file which is usually some combination of homing the printer, heating the bed and hotend, wiping the nozzle, and turning motors and heaters off. Most Gcode flavors start with an alphanumeric

command at the start of the line, then alphanumeric values relevant to that command. Most commands start with G followed by a number hence the name Gcode. The command for a move has the following form: *G1 X10 Y10 Z10 E50* such that (X, Y, Z) is where the toolhead travels to from its current position and E is the value the extruder axis travels. The E value is calculated by determining the volume of the extruded plastic based on the linear distance traveled. The linear distance is calculated by the current point and the points in the $G1$ command. The volume is determined by assuming the cross-section of the extruded plastic is a stadium, whose cross-sectional area is given by Equation 2.1 where ϵ is extrusion width and σ is layer height, then multiplying the cross-sectional area by the linear distance being traveled [2].

$$A_{\text{stadium}} = (\epsilon - \sigma) * \sigma + \pi(\sigma/2)^2 \tag{2.1}$$

The volume of the extruded path is then divided by the cross-sectional area of the filament to get a linear distance of filament needed to fill the volume of the path to determine the E value. If the path point consists of [NaN,NaN] then the algorithm inserts a travel more, i.e., a move without an E value. The convention is to denote a travel move by using the $G0$ command and a regular move by a $G1$ command. GRATER follows this convention and uses $G0$ for all travel moves. GRATER does not support the $G2$ and $G3$ commands for arc and circle moves as these moves get discretized to short straight-line moves by the firmware [57].

The Juggerbot P3-44 and similar pellet-fed screw extrusion printers do not accept an E value. Instead, extrusion is controlled by controlling the screw RPM. The volume of plastic extruded cannot be mathematically correlated to screw RPM due to the buildup of pressure in the screw system. The volume of plastic to the screw RPM has to be empirically determined for every feedrate. GRATER has a user-defined input for screw RPM when using GRATER for a pellet-fed MEX AM system.

Algorithm 2.7: The **Path Combiner** combines the longest path found in **Infill Path Finding** and the perimeter path found in **Perimeter To Paths** into a continuous path.

Input: infill Path, perimeter Path
Output:

function `pathCombiner(`*graphs*`):`
  Define: perimSplit, infSplit, tempInf, tempPerim, infillBeg, infillEnd, indxPerimBeg, indxPerimEnd, tempPath, pathOut
  for $n = 1$ **to** η **do**
    perimSplit $\leftarrow$ $splitPolygons(perimPath\{1, n\})$
    infSplit $\leftarrow$ $splitPolygons(infPath\{1, n\})$
    if $length(perimSplit) == length(infSplit)$ **then**
      for $i = 1$ **to** $length(infSplit)$ **do**
        tempInf $\leftarrow$ infSplit{i}
        tempPerim $\leftarrow$ perimSplit{i}
        infillBeg $\leftarrow$ tempInf(1,:)
        infillEnd $\leftarrow$ tempInf(end,:)
        indxPerimBeg $\leftarrow$ $findClosestPoint(infillBeg, tempPerim)$
        indxPerimEnd $\leftarrow$ $findClosestPoint(infillend, tempPerim)$
        if $indxPerimBeg > indxPerimEnd$ & $length(infSplit) == 1$ **then**
          tempPath $\leftarrow$ $vertcat(tempPerim(1 : indxPerimBeg, :), tempInf)$
        else
          tempPath $\leftarrow$ $vertcat(tempPerim(1 : indxPerimEnd, :), flip(tempInf))$
        end
        if $i == 1$ **then**
          pathOut{1,n} $\leftarrow$ $vertcat([nan, nan], tempPath)$
        else
          pathOut{1,n} $\leftarrow$ $vertcat(pathOut\{1, n\}, [nan, nan], tempPath)$
        end
      end
    `// Perimeters and infill have an unequal number of contours`
    else
      pathOut{1,n} $\leftarrow$ $vertcat([nan, nan], perimPath\{1, n\}, [nan, nan], tempPath)$
    end
  end

Algorithm 2.8: The **Gcode** algorithm outputs the continuous path from **Path Combiner** and encodes it a format the printer can read.

Input: Path, ϵ_{in}, ϵ_p, σ, τ_{bed}, τ_{hot}, file name
Output: file name.Gcode

function `gcodeGenerator()`:
  Define: speed, filamentCSA, filW, pathCSA, A, movelength, pathVolume, E
  `// assume cross section of extruded road is a stadium`
  pathCSA $\leftarrow (\epsilon_{in} - \sigma) * \sigma \ + \ (\frac{\sigma}{2})^2 * \pi$
  filamentCSA $\leftarrow \frac{\text{filW}}{2}^2 * \pi$
  create a file called file name.Gcode
  A $\leftarrow$ file name.Gcode
  print start Gcode for specific printer to A
  for $n = 1$ **to** $size(Path, 2)$ **do**
    `// printing the speed once at the beginning is specific to the juggerbot`
    if *n==1* **then**
      print speed Gcode command to A
    end
    for $i = 1$ **to** $length(path\{1, n\}$ **do**
      if $isnan(Path(i))$ **then**
        if *i==1* **then**
          print to A move to start point
        else
          print to A hop up and travel
        end
      else if *i==1* **then**
        calculate moveLength using last point from previous layer
        pathVolume = $movelength * pathCSA$
        $E = E \ + \ pathVolume/filamentCSA$
        print extrusion move to A
      else
        calculate moveLength using
        pathVolume = $movelength * pathCSA$
        $E = E \ + \ pathVolume/filamentCSA$
        print extrusion move to A
      end
    end
  end
  print end Gcode for specific printer to A

Chapter 3

GRATER Results

To determine if GRATER achieves its goal of reducing travel defects, we will first test GRATER on a small scale filament extrusion MEX AM machine. Then move on to large scale testing where we will compare GRATER's travel moves, travel distance, and build time to other slicers.

3.1 Desktop Benchmark Testing

GRATER was first tested on a desktop filament MEX AM system before moving on to large-scale pellet MEX AM system tests. A benchmark part was designed as it represents an end-use part that is extremely difficult to print on large-scale MEX AM systems with current slicers. Retractions, when filament MEX AM systems retract

(a) `Slic3r` (b) GRATER

Figure 3.1: A 2.7 inch gyroid section sliced using `Slic3r` with retraction off versus one processed with GRATER on a desktop filament MEX AM system. Note the excessive amount of stringing between the two regions as a result of excessive travel moves.

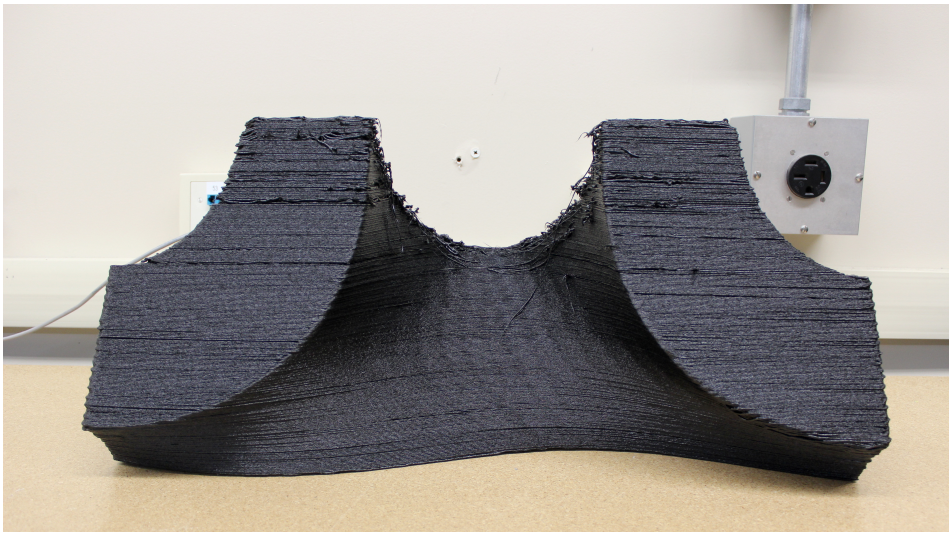
filament from the hotend to prevent stringing and defects during travel moves, were disabled for both prints to simulate the inability of a large-scale pellet MEX AM system to stop extruding plastic during travel moves. As seen in Figure 3.1a, the `Slic3r` sliced part has significant stringing between the two peaks at the top of the print. The `Slic3r` sliced part also has stringing on the bottom layers similar to Figure 3.2a, but this stringing is difficult to capture in an image. The GRATER sliced part in Figure 3.1b has slight stringing between the two peaks at the top of the print since it has to travel across the empty space once per layer; this stringing was insignificant compared to the `Slic3r` part. The bottom of the parts were similar. The GRATER sliced part had no stringing on the bottom layers, unlike the `Slic3r` part.

3.2 Large-Scale Benchmark Performance

After the success of the small-scale prints, seen in Figure 3.1, the same shape was made at full scale on a Juggerbot P3-44 large-scale pellet-fed MEX AM system. `Simplify3D`, a slicer similar to `Ultimaker Cura` and `Slic3r,` was used instead of `Slic3r` because `Simplify3D` is the manufacturer recommended slicer with a `Simplify3D` profile being provided by Juggerbot. The `Simplify3D` part failed due to the print head crashing into the print while traveling, but the point of comparison for travel defects is evident with the unfinished benchmark part. This failure is consistent with previous attempts to print this and similar geometries on the P3-44 with the `Simplify3D` slicer. The excessive travel moves used by `Simplify3D` and other contemporary slicers lead to build failures due to excessive stringing and travel defects. The part was printed with no infill, as no top surface needed infill to support it. GRATER would perform even better if infill was included as we will see later. The GRATER part was completed with minimal stringing between the two peaks and no stringing along the bottom of the part that consists of *a single* contour! Figure 3.2 showcases this dramatic improvement.

GRATER guarantees one continuous path per layer and starts the next layer near where the previous layer ended, so that the single contour layers only had one small travel move. This is in stark contrast to the multiple travel moves per layer that the `Simplify3D` sliced part had, which led to the abundance of stringing seen in Figure 3.2a. After these prints, the Juggerbot P3-44 underwent a series of firmware and hardware upgrades that came along with a profile for Oak Ridge National Lab's

(a) `Simplify3D`



(b) GRATER

Figure 3.2: A 27 inch gyroid section sliced with `Simplify3D` and GRATER on a pellet MEX AM system. The `Simplify3D` build failed due to travel defects from excessive travel moves.

`Slicer` [41, 42]. Therefore the following tests were done with `ORNL Slicer` instead of `Simplify3D`.

3.3 GRATER Performance Metrics

GRATER was built with the aspiration of reducing travel defects by reducing both the number of travel moves and the length of travel moves. Every travel move invalidates the steady-state extrusion flow condition that the process parameters were empirically determined under. This leads to defects such as over and under-extrusion before and

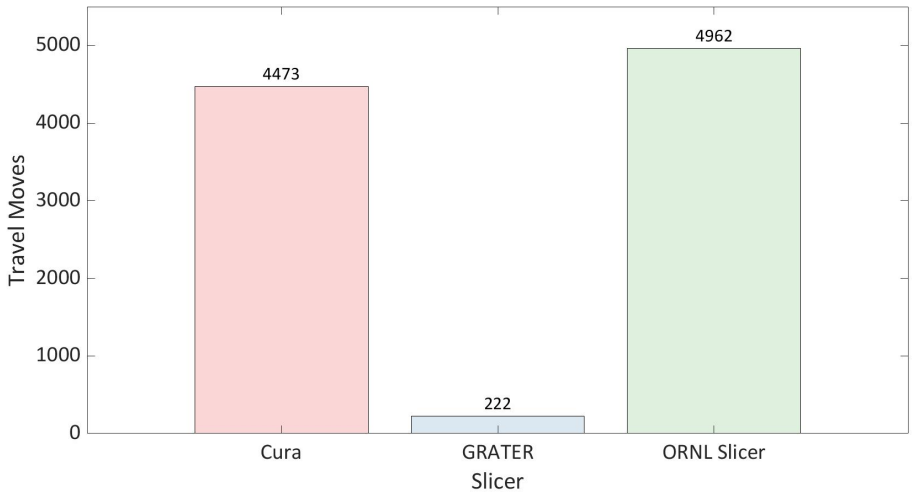


Figure 3.3: Total travel moves of the lattice section on the Juggerbot P3-44. GRATER has an order of magnitude fewer travel moves than both `Cura` and `Slicer`.

after every travel move. Molten plastic continually oozes out of the nozzle during travel moves in pellet-fed MEX AM systems. This ooze can even cause prints to fail such as in Figure 3.2a. To test if GRATER lives up to aspirations, it will be compared against the manufacture-provided slicer (`ORNL Slicer`) and the most popular open-source slicer (`Ultimaker Cura`).

A new end-use part was used for the following tests and can be seen in Figure 3.4. The same geometry was sliced in three slicers (`Cura`, GRATER, and `ORNL Slicer`) for the Juggerbot P3-44. Slice parameters were held the same across all of the slicers to isolate the effect of toolpath planning. The part was 221 layers tall and had a single contour on every layer. GRATER had one travel move per layer plus one travel move to move from home to the start of the first layer. Figure 3.3 shows the number of travel moves each slicer had for the same part. GRATER has *an order of magnitude fewer* travel moves than both `Ultimaker Cura` and `ORNL Slicer` for the geometry tested.

This reduction in travel moves leads to a reduction in build time, specifically on the Juggerbot P3-44. The P3-44 pauses every single time the extruder is stopped or started, which happens during every travel move. As is evident from Figure 3.5, these pauses result in *thousands* of stop-start events that aggregate into more than a 20% increase in total build time for both `Cura` and `Slicer` as compared to GRATER's continuous toolpath planning algorithm for the geometry tested. The `Cura` build time

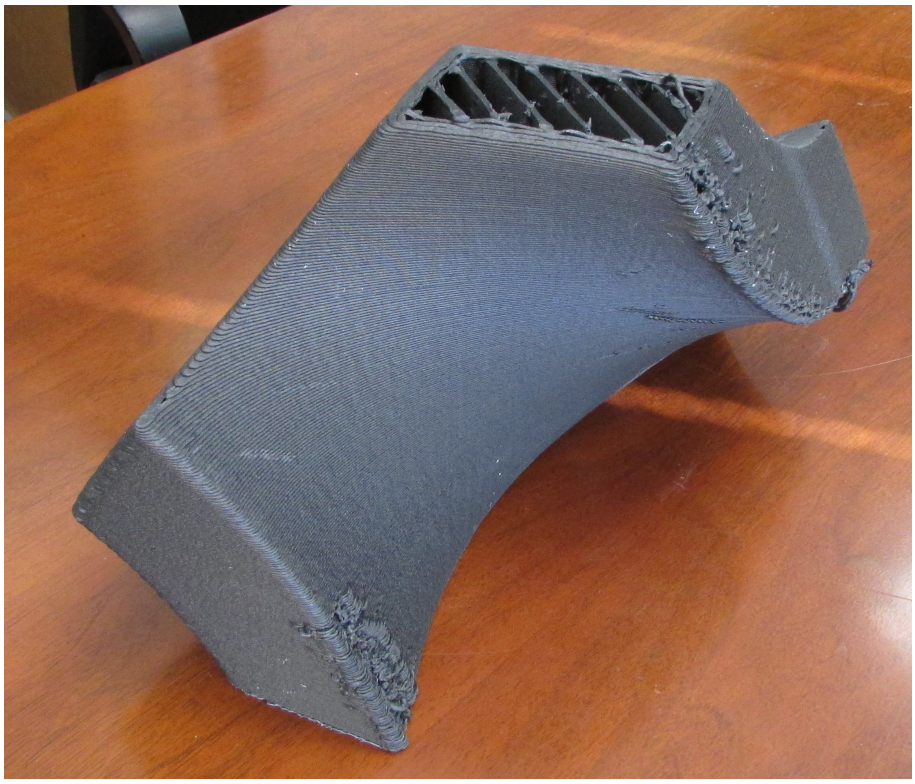


Figure 3.4: This test part was chosen as it was an end-use part that had a variety of cross-sections through the build direction.

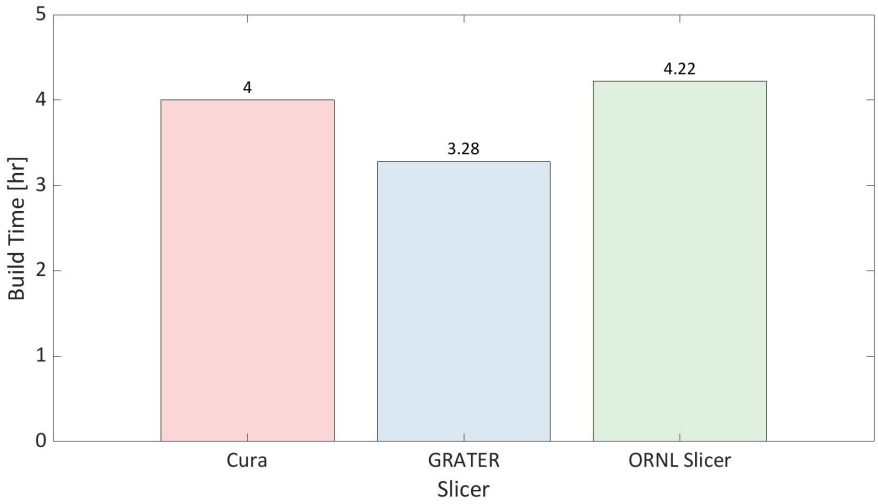


Figure 3.5: Build times of the lattice section on the Juggerbot P3-44. The Cura build time is an estimation as no Cura sliced part was printed successfully. The reduction in travel moves has the benefit of reducing print time on the Juggerbot P3-44.

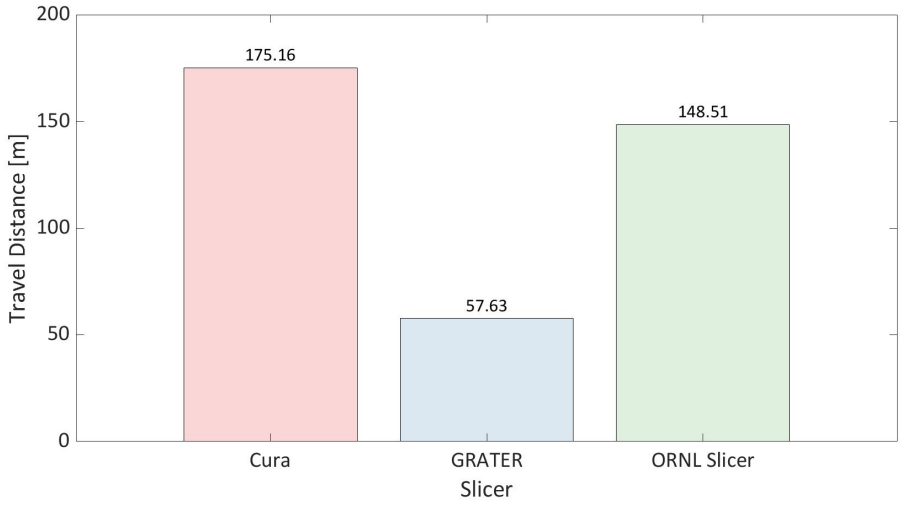


Figure 3.6: Travel distance of the lattice section on the Juggerbot P3-44. GRATER reduces the total distance traveled, reducing the amount of stringing due to traveling.

is, however, an estimate based on the expected build time as no benchmark parts were able to be successfully printed. The estimate is based on the time it took to print the failed part, combined with the estimated build time. This experiment clearly shows the benefit of reducing start-stop events on these pellet MEX AM systems to increase part production rates.

Pellet MEX AM systems ooze material as they travel leading to defects, GRATER sought to try and reduce the distance traveled to mitigate this problem. To calculate travel distance a `MATLAB` script was developed for each of the three slicers, one of which can be seen in Code 3.1 and the rest are in Appendix 5.3. Each of the three slicers denotes travel moves differently, making a script for each one necessary. The distance was the linear distance in the (x, y) from the starting location to the location being traveled. All three slicers implement a z-hop, an upwards travel in z to avoid crashing into the part when traveling, but this distance was not calculated due to it not contributing to the ooze defect. GRATER reduced the total distance traveled by a factor of three for the geometry tested!

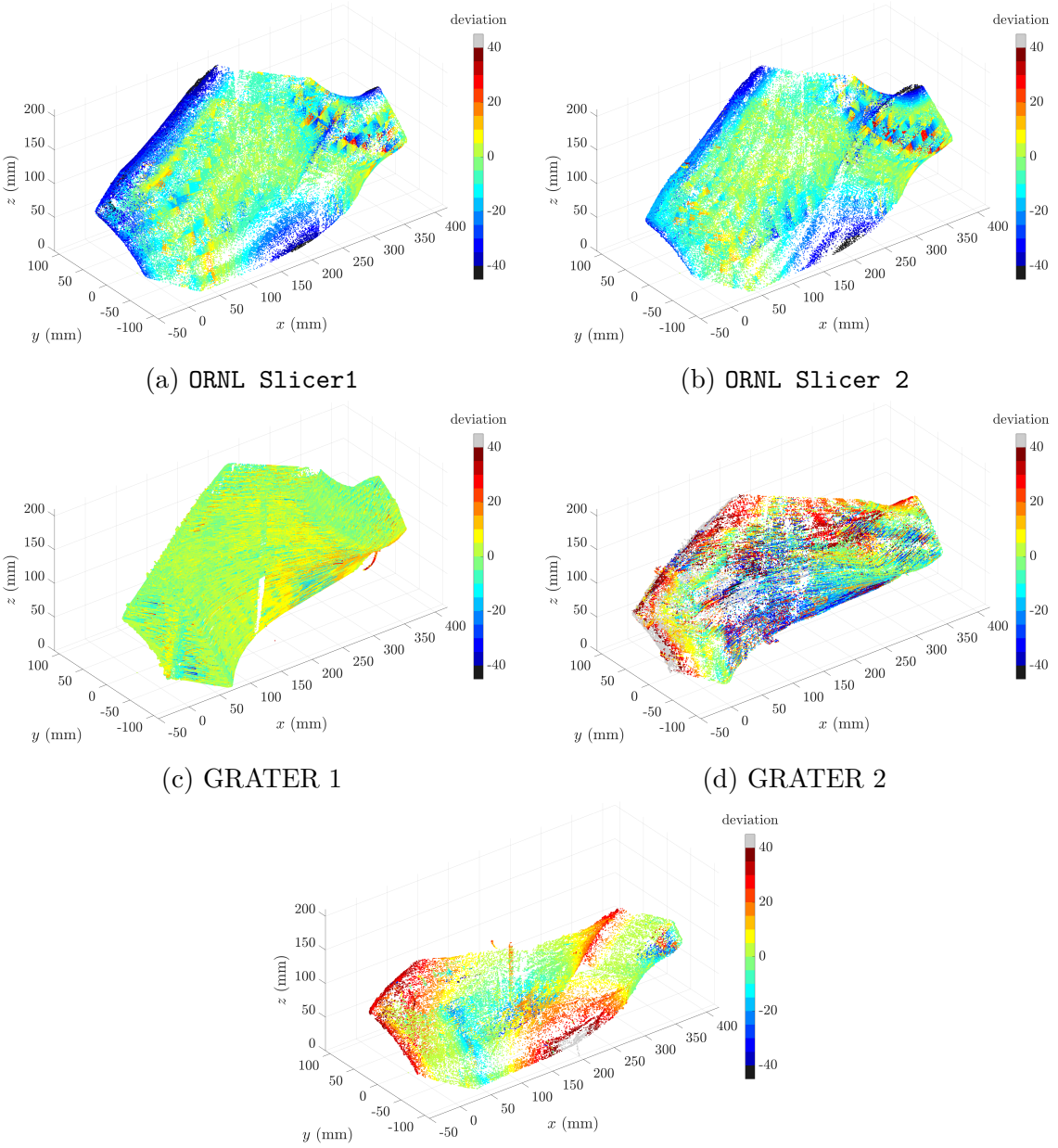
Code 3.1: The MATLAB script used to calculate the distance GRATER traveled.

```
close all; clear all; clc;

file = 'GRATER2.gcode';

D = readlines(file);

R = D(20:13781, 1);
total_distance = 0;

for n = 1:1:length(R)

    if isempty(R{n}) == 1
        continue
    end

    hop_check = strfind(R{n}, '; hop up');
    move_check = strfind(R{n}, ';move');

    if isempty(hop_check) == 0
        X_location = strfind(R{n - 2}, 'X');
        Y_location = strfind(R{n - 2}, 'Y');
        spaces = strfind(R{n - 2}, ' ');
        X_end = find(spaces > X_location);
        X_value = str2double(R{n - 2}(X_location + 1:spaces(X_end(1))));
        Y_end = find(spaces > Y_location);
        Y_value = str2double(R{n - 2}(Y_location + 1:spaces(Y_end(1))));
        loc_B = [X_value, Y_value];
    end

    if isempty(move_check) == 0
        X_location = strfind(R{n}, 'X');
        Y_location = strfind(R{n}, 'Y');
        spaces = strfind(R{n}, ' ');
        X_end = find(spaces > X_location);
        X_value = str2double(R{n}(X_location + 1:spaces(X_end(1))));
        Y_end = find(spaces > Y_location);
        Y_value = str2double(R{n}(Y_location + 1:Y_location + 6));
        loc_A = [X_value, Y_value];
        total_distance = total_distance + sqrt((loc_B(1) - loc_A(1))^2 + (loc_B(2) - loc_A(2))^2)
    end

end
```

3.4 Toolpath Quality

This section compares the effect of the different slicer's toolpath planning algorithms on the final part geometry. The goal is to capture travel defects such as stringing and the over-extrusion defects seen in Figure 3.4. To quantify the geometric accuracy of the part as a function of the toolpath, a FARO laser scanner was used to capture the as-built geometry [58]. For incomplete builds the Gcode was truncated to match the point at which the build failed. This point cloud was then compared to the Gcode used to build the part using a method pioneered by Valenti et al. (2022) [59]. The authors graciously provided the MATLAB script they used. The algorithm it uses is shown in Algorithm 3.1.

The printed parts were scanned with a Faro arm laser scanner to capture the stringing and defects travel moves created in large format pellet extrusion MEX AM systems. That scan was then compared to the Gcode points to capture the maximum deflection, the scanned point farthest from its respective Gcode point, and the RMSD value. All of the metrics captured are provided in Table 3.1. One of the Cura and



(a) `ORNL Slicer1`

(b) `ORNL Slicer 2`

(c) GRATER 1

(d) GRATER 2

(e) Cura 1

Figure 3.7: Heatmap showing deflection of the scanned point cloud to the Gcode. GRATER 2 and Cura 1 are both failed prints.

Algorithm 3.1: Algorithm to compare a point cloud to a Gcode file, reproduced from [59].

Input: Gcode, Point Cloud
Output: RMSD, Deformation
for *each Gcode Layer* **do**
  Find all Point cloud points within half a layer height;
  for *Each Point Cloud Point, p_i, at the current layer* **do**
    Find the closest Gcode point at the layer, g_j to p_i
    Determine whether g_{j+1} or g_{j-1} is the next closest to p_i
    Calculate the local deformation, δ_i, defined as the perpendicular distance between p_i and the line defined by the two closest Gcode points;
  end
end
Calculate RMSD of all δ values to find the RMSD deformation of the as-built part;

Table 3.1: Point cloud to Gcode comparison

Slicer & Run	RMSD (mm)	Max Deviation (mm)	Quartiles (mm) (25%, 50%, 75%)
GRATER-1	6.6	33.8	$(-1.7, 1.5, 5.9)$
GRATER-2*	27.9	62.6	$(-10.7, -6.1, 32.4)$
Slicer-1	18.9	35.8	$(-22.5, -9.0, -1.4)$
Slicer-2	15.5	34.8	$(-17.5, -6.7, -0.7)$
Cura-1*	15.6	65.1	$(-1.68, 3.4, 15.8)$

**results from partial builds*

one of the GRATER prints, GRATER-2 and Cura-1, failed due to a printer error. These runs are included for posterity, but the RMS and max deflection values are not representative of those slicers. The `Slicer` prints seen in Figures 3.7a and 3.7b have surface defects on the corner closest and the flat face farthest away from the figure viewpoint. These surface defects lead to higher RMSD values compared to the complete GRATER print. The complete GRATER print had one string that drooped down, seen in red, on the overhang, but overall has a better surface finish compared to the `ORNL Slicer` prints. While the method pioneered by Valenti et al. (2022) gave some interesting results, it is better suited for its original purpose, of measuring local deformation from thermal warping than measuring small defects such as over and under extrusion [59]. The data gathered fails to adequately capture travel defects such as stringing and over or under-extrusion.

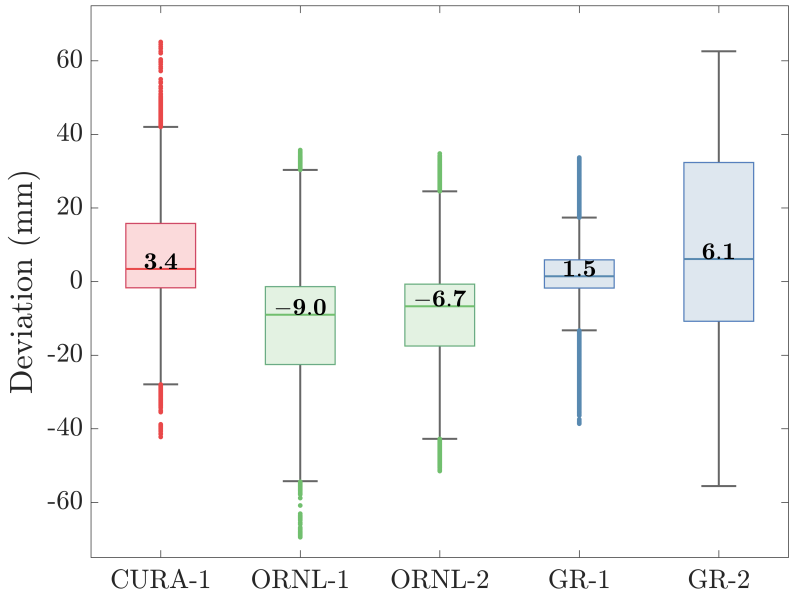


Figure 3.8: Box & Whisker plots of the geometric deviation measured for the scanned point clouds using Algorithm 3.1.

3.5 Summary

GRATER was built to reduce travel defects, specifically to make the pellet MEX AM process more reliable by reducing the travel defects inherent to the process. GRATER accomplished this by utilizing graph theory to find continuous toolpaths. GRATER was successful in this aspect as measured against both a contemporary filament MEX AM slicer and the manufacturer's recommended slicer in the following three metrics for the geometry tested.

1. GRATER reduced travel moves by an *order of magnitude* or 95%.
2. GRATER reduced total distance traveled by a *factor of three*.
3. GRATER reduced the print time by approximately *25%*.

The next chapter will expand on the continuous toolpath and explore how sequential, continuous toolpaths used to combine different infill regions affect part properties.

Chapter 4

Effects of a Continuous Toolpath on Integrating Dense and Sparse Infill

Slicers and preprocessing software for MEX AM builds can be used to assign any arbitrary region of a part to have different *processing properties*. This method is most commonly used to adjust the infill percentage in these regions to locally strengthen the part. Contemporary methods of generating toolpaths for said regions are performed independently without regard to the integrity of the final produced part, resulting in poor bonding between the regions. This chapter explores novel ways to connect the boundaries of the different infill regions *sequentially* and *continuously* to strengthen bonding and subsequently the overall part strength.

4.1 Methods

ASTM D638-22 provides a standard test geometry (aka "dogbone" geometry) to determine mechanical properties by loading the specimen in tension until failure [60]. The geometry printed to evaluate the effect of toolpath continuity at the interface between solid and sparse infill was selected as the Type 1, 4mm thick test specimens. The gage section of the specimen is used as the experimental domain to compare infill strategies with the grip regions defaulted to fully dense infill. All process parameters for a desktop MEX AM printer (a Lulzbot Mini2) were kept constant for all tensile test specimens with the relevant slicing parameters given in Table 4.1. Constant process

Figure 4.1: ASTM D638 Type 1 specimen geometry with an overlay of the dense and sparse infill regions within the tensile specimen. Sparse regions are shown in red and are 45mm in length; dense regions are shown in blue.

parameters allow for the effect of toolpath continuity on mechanical performance to be isolated simply. Further, the proposed toolpath planning is compared to contemporary slicers of `Slic3r` and `Ultimaker Cura` so that a baseline could be determined to compare against the proposed infill combination strategies detailed in Section 4.2.

Table 4.1: Printing Parameters

Slicing Parameters	Parameter Value
Extrusion Width	0.5 mm
Layer Height	0.2 mm
First Layer Printing Speed	25 mm/s
Printing Speed	60 mm/s
Infill Overlap	15%
First Layer Part Cooling Fan	0%
Part Cooling Fan	40%
Bed Temperature	60°C
Nozzle Temperature	240°C

`Slic3r` and `Ultimaker Cura` both have built-in *modifiers* that were used to modify the sparse region to consist of 20% infill, with the dense region consisting of 100% infill. The sparse region is in the gage region and is 45mm long [60] illustrated in Figure 4.1. The sparse region is shown in red and the dense regions are shown in blue. All the specimens were printed with no solid top and bottom surfaces to avoid different top and bottom surface patterns confounding the testing. The difference in weight between infill combination strategies was less than one standard deviation of the difference in weight between replications. The sparse and dense infill angle is ± 45 degrees to the specimen's load axis.

4.2 Toolpath Combination Strategies

Novel infill combination strategies are generated using `MATLAB` code. The `MATLAB` program first reads in an STL geometry file and assumes that the x-y cross section is constant for every z height, so only intersects the STL at one z height. The `MATLAB` function POLYBUFFER is used to generate perimeters and to define the infill regions, which allows for control of the infill overlap percentage. For the novel infill algorithm to find a continuous infill path between multiple infill regions there are two conditions that must be met:

(1) Higher infill density must be collinear with the lower infill density

(2) Infill travel directions must be the same along collinear paths

To guarantee that these conditions are satisfied, the following equations are used to define the sparse and dense infill spacing. The dense infill spacing is defined by Equation 4.1 and can be seen in Figure 4.2a.

$$\text{Dense Spacing} = \frac{\text{Extrusion Width}}{\text{Dense Infill Percentage}} \tag{4.1}$$

The spacing of the sparse infill lines alternates between the spacing found above in Equation 4.1 and the spacing found below in Equation 4.2. This generates pairs of lines where each pair is separated by the sparse spacing equation of Equation 4.2. The unique sparse infill spacing can be seen in Figure 4.2b.

$$\text{Sparse Spacing} = \frac{\text{Extrusion Width}}{\text{Sparse Infill Percentage}} \tag{4.2}$$

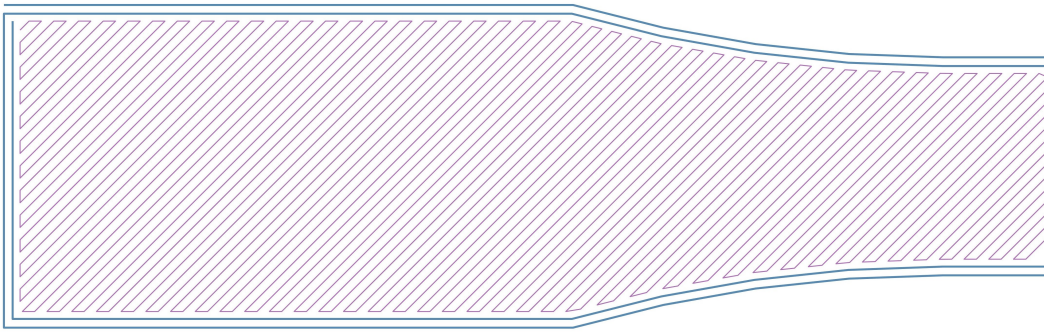
This guarantees that every sparse line is *collinear* with a dense line thereby fulfilling the first condition listed above. This collinearity is apparent in Figure 4.2c. Lines with either dense or sparse spacing are generated and then rotated to the desired infill angle. The lines are then intersected with their respective region boundaries to generate the intersection points on the region perimeter. These intersection points are ordered in such a way that rectilinear infill is generated. The downside of this infill generation method is the *two infill densities must be odd integer multiples* for condition (2), listed above, to be true. If the infill percentages are not odd multiples then the infill lines will still be collinear, but the sparse infill will be offset. This offset

means that the tool head would be traveling in the opposite direction necessary to travel into the sparse infill from the dense infill or vice versa.

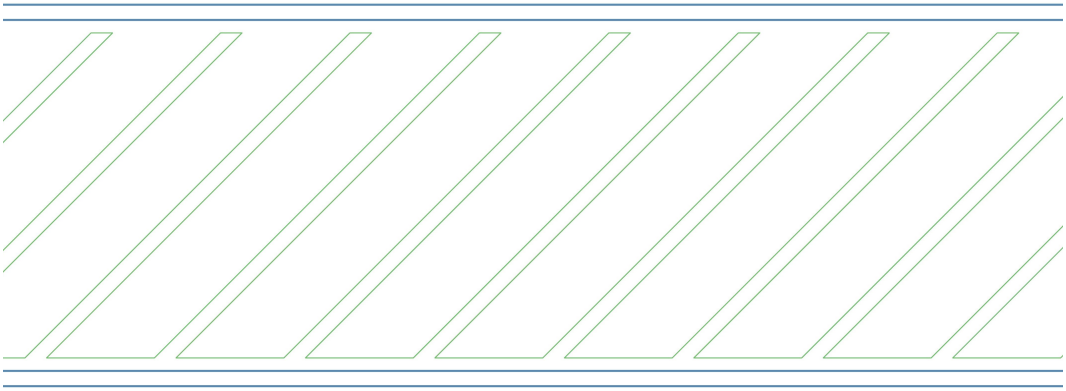
Five different ways to connect the sparse and dense infill regions are explored in this thesis. Illustrations of each of the build strategies used are shown in Figures 4.3, 4.4, and 4.5. Each color in the figures represents a single continuous toolpath. Each strategy produces a different toolpath plan that is mechanically tested to assess its performance.

The **Continuous** infill combination strategy, shown in Figure 4.3, builds all three infill regions and the perimeters in a single continuous toolpath. The **Continuous** infill combination starts with the left dense region and transverses along the intersection points previously established. While traveling along the intersection points, the algorithm checks if there is a sparse line collinear with the line it is traveling. If a sparse line is found to be collinear with the current dense line, the algorithm travels along the collinear sparse line instead of stopping when the dense line stops. This collinear spacing is demonstrated in Figure 4.2c; when the purple lines are collinear with the green lines the algorithm will traverse along the green lines. This naturally leads the path into the sparse region where it uses the same logic to travel back into the dense region. Traveling back is guaranteed because the sparse lines that intersect the dense-sparse boundary are collinear with a dense line and the sparse infill region is ordered such that the sparse infill travels along the bottom perimeter. The sparse infill traveling along the bottom perimeter is ideal for lining up the sparse region with the dense region, but it is not ideal for transitioning from the sparse infill region into the right dense region. The algorithm would travel into the dense region for only the lines that are collinear with the sparse region until the sparse region is completely traversed. This can be seen in Figure 4.2c, if the toolpath would travel from the green lines along the orange lines it would not naturally be led back into the sparse infill region. An extra road is inserted in the right side of the sparse infill to avoid the algorithm skipping a large portion of the right dense infill region. The **Two Continuous** strategy, seen in Figure 4.3, avoids this extra road by traversing the sparse region completely, then traveling to traverse the right dense infill region.

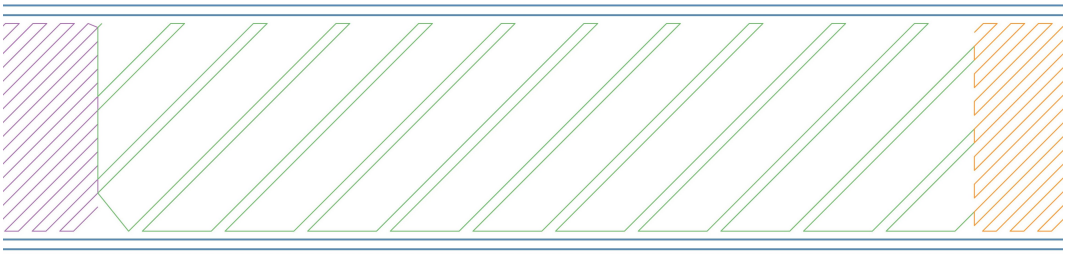
The **Weak Sparse** infill combination strategy (seen in Figure 4.4) makes one continuous toolpath by weakening, removing a road from one of the pairs in the middle of the sparse infill, the sparse infill geometry. The sparse region travels along the



(a) Dense Spacing

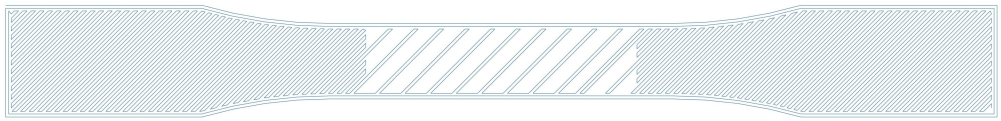


(b) Sparse Spacing

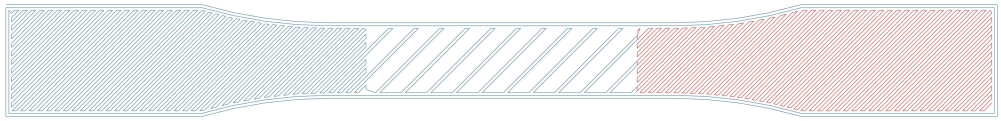


(c) Collinearity of Dense and Sparse infill due to the unique sparse infill spacing

Figure 4.2: Infill spacing showcasing the unique sparse infill and the forced collinearity. This collinearity is necessary to find a continuous toolpath through all infill regions.

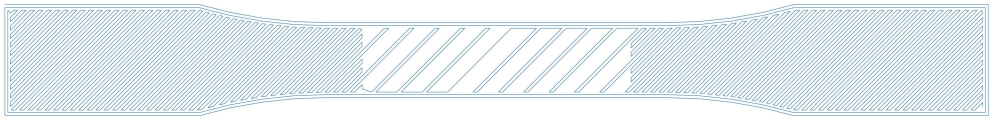


(a) Continuous

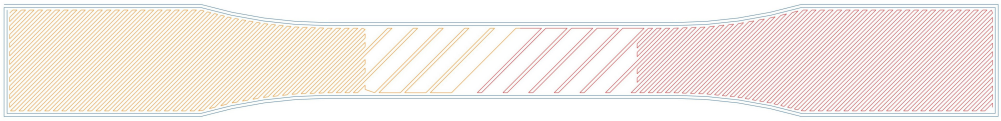


(b) Two Continuous

Figure 4.3: Continuous and Two Continuous infill combination strategies. Each color represents a continuous toolpath. The Continuous infill combination strategy consists of one continuous toolpath while the Two Continuous infill combination strategy consists of two continuous toolpaths with a travel move separating them.

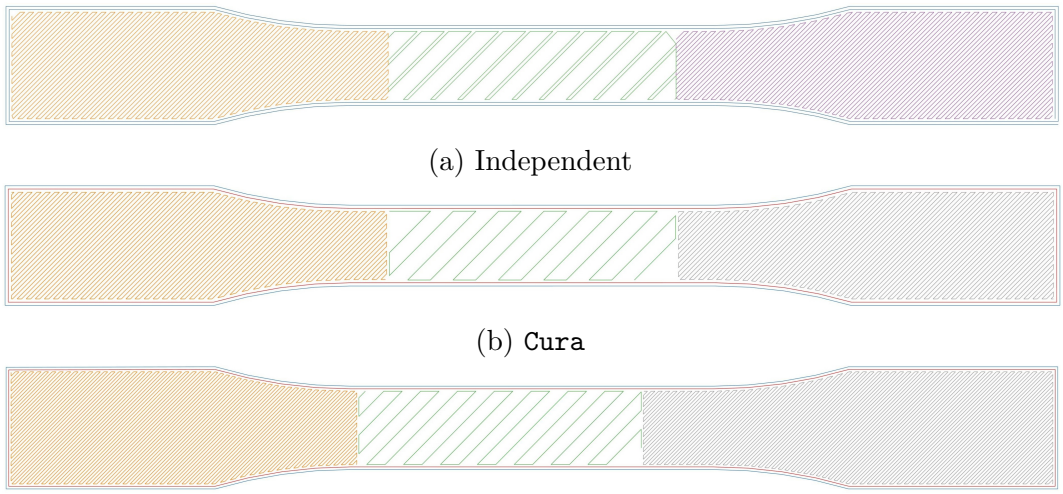


(a) Weak Sparse



(b) Out-In

Figure 4.4: Weak Sparse and Out-In infill combination strategies. Each color represents a continuous toolpath. The Weak Sparse infill combination strategy has a singular road in the middle of the sparse section to force a single continuous toolpath. The Out-In infill combination strategy is the same geometry as weak sparse, but the infill consists of two toolpaths each starting from the left and right respectively, and stopping in the middle of the sparse section.



(a) Independent

(b) `Cura`

(c) `Slic3r`

Figure 4.5: Independent, `Cura`, & `Slic3r` infill combination strategies. The Independent infill combination strategy has the same amount of discontinuities as `Cura` and `Slic3r`. The `Cura` and `Slic3r` infill combination strategy of completing each infill region independently is typical of contemporary slicers. Each color represents a continuous toolpath.

bottom perimeter in the left half which allows for the smooth transition between the left dense region and the sparse region. The sparse region travels along the top perimeter in the right half which allows for the algorithm to smoothly transition into the right dense region from the sparse region without an extra road. The transition in the sparse region from traveling along the bottom to the top requires an odd number of roads. The **Out–In** infill combination strategy seen in Figure 4.4, uses the same geometry as the **Weak Sparse** infill combination strategy, but after extruding the left dense region and half the sparse region, it then travels to the right dense region and traverses the rest of the geometry from right to left.

The **Independent** infill combination strategy starts from the left dense infill region and traverses the geometry from left to right, with travel moves between each region. This infill strategy is similar to `Cura` and `Slic3r`; however, it traverses the geometry from left to right sequentially. The `Cura` infill combination traverses the perimeters separately, then it traverses the left dense infill region starting from the bottom right corner. After the left infill dense region is traversed it travels to the right dense infill region and traverses it starting from the top left corner. Finally, it traverses the sparse infill region starting from the bottom right corner. The `Slic3r` similarly traverses the perimeters separately then it traverses the left dense infill region starting

from the bottom left corner, then traverses the right dense infill region starting from the top right corner. Finally, the `Slic3r` infill combination traverses the sparse region starting from the top right corner.

All tensile specimens were printed using Essentium PLA XTR feedstock on a Lulzbot Mini 2.0 MEX AM machine. The build order of the parts was randomized to minimize outside influences affecting the builds. Tensile testing was performed using an Instron 3345 test frame equipped with a 1.5kN load cell with BlueHill 3 control and DAQ software. Five replications for each infill combination were tested at a strain rate of 0.1 inches per minute except for one specimen. The **Weak Sparse** infill combination which was tested at 0.2 inches per minute. The strain rate was chosen to guarantee failure occurred between half a minute and five minutes as prescribed in ASTM D638-22 [60]. All samples failed in the gage region and within the half a minute and five-minute window. Effective ultimate stress was calculated according to the ASTM D638-22 standard using the dense gage region's cross-sectional geometry. The effective ultimate stress calculated does not represent the stress in the sparse region, but it allows for direct comparison across all infill combination strategies [61]. Hence the *effective* qualifier when discussing ultimate stress for the rest of this chapter. Nominal strain was measured by dividing the change in grip separation by the original grip separation and then multiplying by 100 as specified in ASTM D638-22. A one-way Analysis of Variance (ANOVA) table was calculated to compare means between infill combination categories with a significance level of 0.05. Tukey method comparisons with a significance level of 0.05 were used to compare differences in means. A one-tailed two-variance hypothesis test with a significance level of 0.05 was used to compare the variances between infill combinations.

4.3 Results & Discussion

The average effective ultimate stress and average nominal strain at break results from testing five replications of each infill combination strategy are shown in Table 4.2. Yield stress was not recorded due to all specimens demonstrating brittle failure. The average effective ultimate stress for `Slic3r` and `Ultimaker Cura` is lower than every other infill combination strategy. Similarly, the average nominal strain at break for `Cura` and `Slic3r` is lower than every other infill combination strategy.

Table 4.2: Tensile Test Results

Infill Combination Strategy	Average Effective Ultimate Stress (MPa)	Average Nominal Strain at Break (%)
`Slic3r`	8.01 (0.56)	1.51 (0.063)
`Ultimaker Cura`	7.98 (1.5)	1.59 (0.11)
Two Continuous	8.61 (1.6)	1.93 (0.20)
Out–In	8.69 (0.52)	1.69 (0.12)
Independent	8.94 (1.4)	1.87 (0.17)
Continuous	9.16 (0.79)	2.01 (0.80)
Weak Sparse	9.60 (0.94)	1.62 (0.13)

standard deviations are listed in parenthesis

A one-way ANOVA test to compare the mean effective ultimate stress of the infill combination strategies was performed. The results of the test showed that, at a 5% significance level, there is not enough evidence to claim a difference in mean effective ultimate strength for at least one of the infill combination strategies. This claim is apparent when looking at Figure 4.6, as every boxplot overlaps every other boxplot. One proposed benefit of a continuous toolpath is a reduction in defects, leading to more consistent parts. The standard deviation of the **Continuous** infill combination strategy is smaller than the standard deviation of the `Cura` infill combination strategy, but the `Slic3r` infill combination strategy has a lower standard deviation than both **Continuous** and `Cura`. The variation in the MEX AM process leads to a wide range of as-built mechanical properties, such that only drastic changes in mechanical properties can be proven statistically significant.

A one-way ANOVA test to compare the mean nominal strain at break for the various infill strategies was performed. It is found that, at a 5% significance level, there is enough evidence to claim that at least one of the infill combination strategies has a different mean nominal strain from at least one other infill combination strategy. The boxplots in Figure 4.7 showcase this difference in mean nominal strain between the infill combination strategies. Tukey method comparisons at a 5% significance level were carried out to evaluate differences in the nominal strain at break true means because the one-way ANOVA found a statistically significant difference in mean nominal strain. Table 4.3 showcases the results of the Tukey method simulations. From the Tukey method comparisons, it is possible to claim that **Continuous**, **Two Continuous**, and the **Independent** infill combination strategies have a higher true mean nominal strain at break than `Cura` and `Slic3r`.

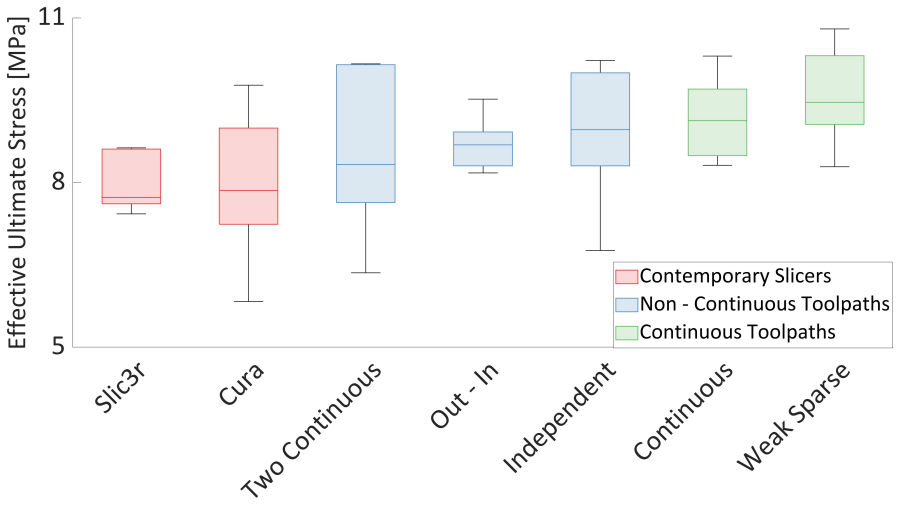


Figure 4.6: Effective ultimate stress boxplots showing that the contemporary slicers performed poorly, the non-continuous infill combination strategies performed better on average than the contemporary slicers, and the Continuous infill combination strategies performed the best.

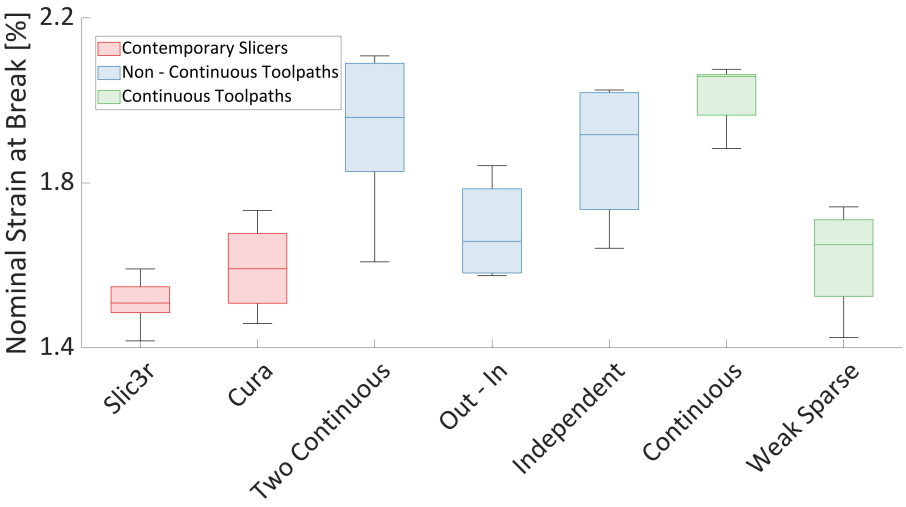


Figure 4.7: Nominal strain at break boxplot showing that the contemporary slicers performed the worst while the novel infill combination strategies performance was correlated with if they were sequential as explained in Section 4.4.

Table 4.3: Nominal strain at break Tukey method comparison results

Infill combination strategy	Mean	Grouping			
`Slic3r`	1.51				D
`Cura`	1.59				D
Two Continuous	1.93	A	B		
Out-In	1.69		B	C	D
Independent	1.87	A	B	C	
Continuous	2.01	A			
Weak Sparse	1.62			C	D

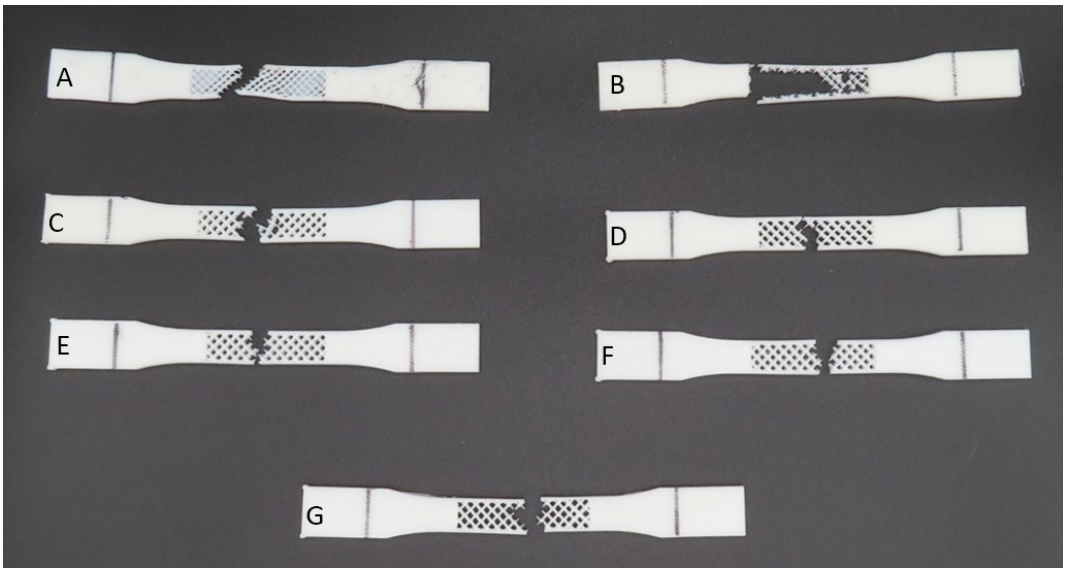


Figure 4.8: Representative Failed Tensile Specimens:
(A) Slic3r, (B) Cura, (C) Out-In, (D) Weak Sparse, (E) 2 Continuous,
(F) Independent and (G) Continuous

Infill combination strategies that displayed a statistically significant higher nominal strain at break (those that belong to group A), break in approximately the center of the gage region as seen in Figure 4.8. **Out–In** and **Weak Sparse** consistently break in known weak points in the geometry, the single road for **Weak Sparse** and the discontinuity point in **Out–In**. The `Cura` and `Slic3r` test specimens broke at a variety of locations, indicating that these infill combination strategies suffered from defects. Specimens produced using `Cura` showcased brittle fracture of the sparse infill region from the perimeters throughout loading as can be seen in Figure 4.8. `Slic3r` specimens frequently broke along one perimeter first leading to the warped sparse infill region seen in Figure 4.8, then the other perimeter failed in a multi-stage failure.

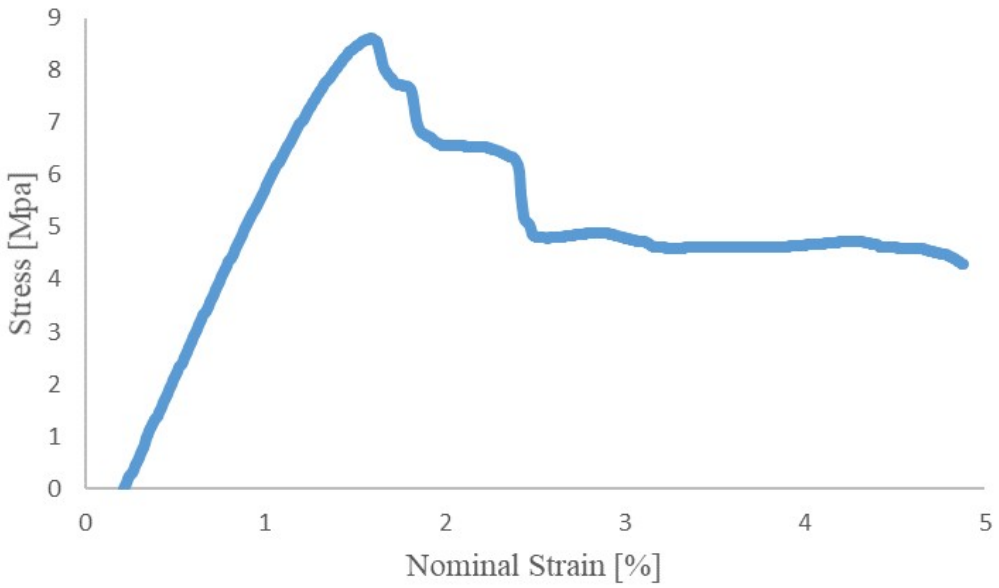
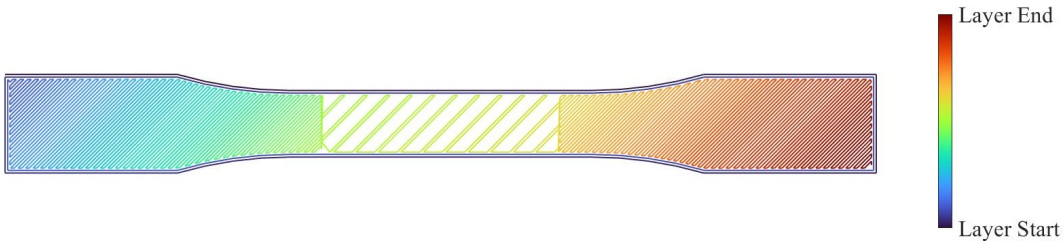


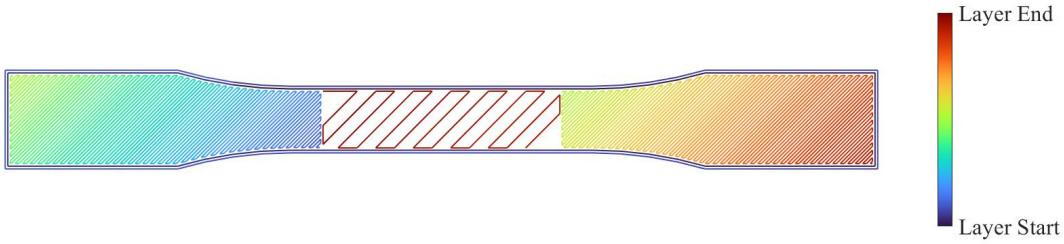
Figure 4.9: The `Slic3r` infill combination strategy consistently showcased multi-stage failure where one perimeter would break significantly earlier than the other perimeter resulting in the warped broken specimens as seen in Figure 4.8.

`Slic3r` specimen's multi-stage failure is apparent when looking at figure 4.9. The `Slic3r` specimens appear to be fracturing at defects, poor or lack of bonding between adjacent roads, with a significantly lower nominal strain than expected. The other portions of the specimen continue to resist load to a larger nominal strain. The statistically significant lower nominal strain along with the observations of the failure behavior both point to `Slic3r` and `Cura` infill combination strategies having significantly more defects than the more continuous toolpaths.

The **Independent** infill combination strategy had a statistically significant higher nominal strain and a higher average effective ultimate stress than **Out–In** and **Weak Sparse** strategies even though it has more discontinuities. The **Independent** infill combination strategy has a similar number of toolpaths as the `Cura` strategy but the roads are printed sequentially from left to right unlike in the `Cura` strategy. The difference in the order of the toolpaths can be seen in Figure 4.10. This suggests that the poor bonding in discontinuous adjacent roads is diminished if the adjacent roads are sequentially extruded. The sequentially extruded roads have the least amount of time to cool before an adjacent road is placed leading to the interface between the roads maintaining the critical temperature for reptation for longer. The good mechanical



(a) **Independent**



(b) `Cura`

Figure 4.10: The **Independent** and `Cura` infill combination strategy print order. This figure is colored by the order in which the toolhead traverses the layer. This showcases how the **Independent** strategy is sequentially compared to the random order of `Cura`.

properties of the **Independent** infill combination strategy suggest that discontinuities are preferable to the single roads found in **Weak Sparse** and **Out-In** strategies so as long as the discontinuous paths extruded adjacent roads sequentially. The **Two Continuous** strategy has a statistically significant greater average nominal strain at break than **Weak Sparse**, providing more proof. The large standard deviation for the **Independent** infill combination strategy is possibly due to the increased chance of defects at toolpath discontinuities, further complicating finding an optimal toolpath.

4.4 Summary

This research has compared the average effective ultimate stress and average nominal strain at break across the seven infill strategies separately. Figure 4.11 aggregates the average effective ultimate stress and average nominal strain at break in an Ashby-style chart. The error bars in all charts of this nature in this chapter represent one standard deviation. For comparison, preference is placed on producing parts with higher effective ultimate stress, and a higher nominal strain at break is considered to be better. Higher effective ultimate stress correlates to better bonding adjacent roads and a low nominal strain is an indicator of fewer defects.

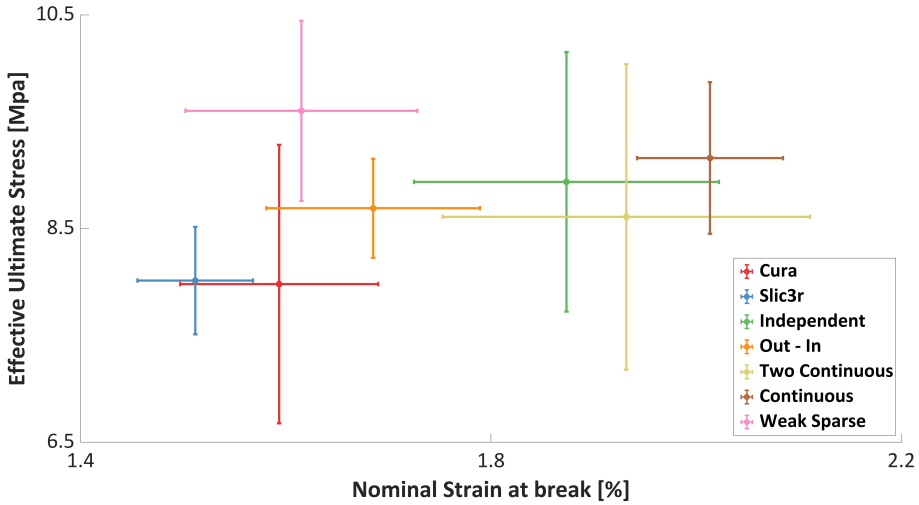


Figure 4.11: Ashby-style chart comparing the different infill combination strategies. Higher effective ultimate stress and higher nominal strain at break are desirable for high-performance parts.

As expected, the contemporary slicers perform poorly in *both* metrics. The **Continuous** infill combination strategy performs the best with the **Two Continuous** and **Independent** infill combination strategies within one standard deviation of the **Continuous** infill combination strategy. The **Two Continuous** and **Independent** infill combination strategies performed similarly, which is to be expected as they are the same, except that the **Two Continuous** infill combination strategy has one fewer travel moves. The **Weak Sparse** infill combination strategy was a continuous infill strategy that weakened the sparse infill to force continuity. The poor performance of the **Weak Sparse** infill strategy shows that weakening the infill to force continuity will lead to a weaker part than having a discontinuity in the infill. The **Out-In** infill combination strategy performs poorly in both metrics. The long delay between laying down the two adjacent roads in the middle of the infill led to poor bonding. This poor bonding did not result in only the perimeter resisting the load like the `Cura` infill combination strategy, but the part did fail in the middle of the infill at the two adjacent roads with poor bonding. Overall, the sequential infill combination strategies performed well in both metrics with non-sequential infill combination strategies performing poorly in both metrics.

The **Independent** infill strategy was created to be similar to contemporary slicers to act as a control. The **Independent** infill strategy was named so because it deposits

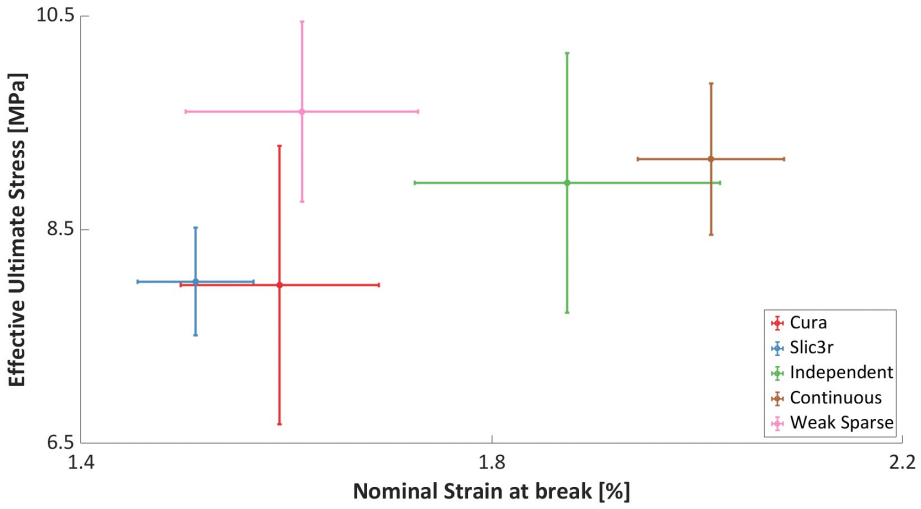


Figure 4.12: Ashby-style chart comparing the independent infill combination strategy to the **Continuous** infill combination strategies and the contemporary slicers. The **Independent** infill combination strategy performing well indicates that sequential toolpath strategies increase part properties.

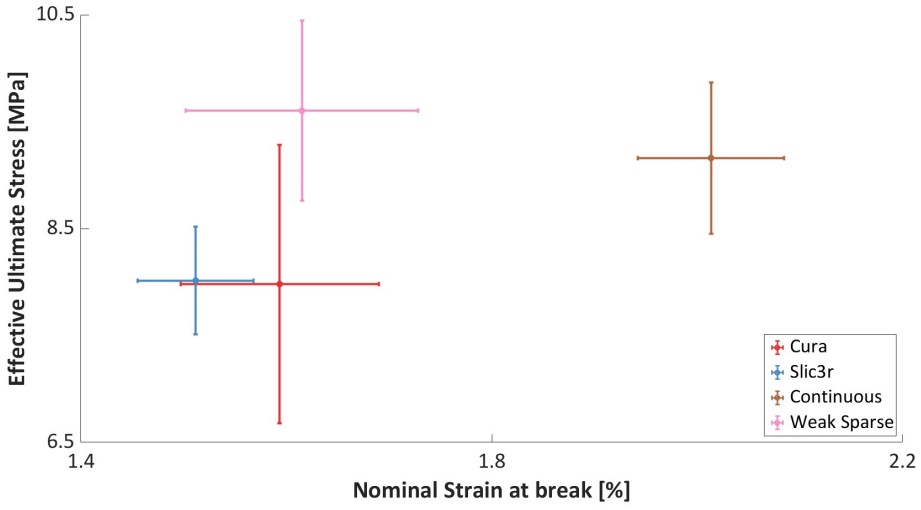


Figure 4.13: Ashby-style chart comparing the continuous infill combination strategies to contemporary slicers. This showcases that weakening the infill, the **Weak Sparse** infill combination strategy, to force continuity does not result in an infill combination strategy that performs well in both metrics.

the infill in each region independently. This is how contemporary slicers treat each infill region, but the **Independent** infill strategy sequentially traverses the infill region from left to right. This is showcased in Figure 4.10. The contemporary slicers do not traverse the infill regions sequentially, both `Cura` and `Slic3r` fill both dense infill sections first before filling in the sparse infill section. This leads to the roads at the sparse-dense interface cooling significantly before their adjacent road is extruded. This leads to a decrease in bond strength which can be seen in the decrease in average effective ultimate stress in Figure 4.13.

The novel infill combination strategies presented in this chapter sought to increase the mechanical properties of end-use parts by combining the infill regions using sequential, continuous toolpaths. The good performance of the **Independent** infill combination strategy compared to the contemporary infill combination strategies demonstrates the effect of sequential toolpaths on bonding between adjacent roads. The good performance of the **Continuous** infill combination strategy compared to the **Independent** infill combination strategy showcases the effect of continuous toolpaths on bonding between adjacent roads. Overall, this research saw a 20% increase in average effective ultimate strength, and an increase of 33% average nominal strain at break when comparing the novel infill combinations presented in this chapter to contemporary slicers.

Chapter 5

Contributions & Future Work

Based on the research results presented in this thesis, MEX AM parts can be fabricated with fewer travel moves and improved bonding between adjacent roads. This is especially relevant for large format MEX AM systems that use pellet extrusion systems. The hope is that the research presented in this thesis will make large-format MEX AM systems a more viable manufacturing process by improving the reliability of the process and the predictability of part properties.

5.1 Contributions

A novel method of combining dense and sparse infill was discussed in Chapter 4. This method of combining dense and sparse infill using continuous and sequential toolpaths showed an increase in average effective ultimate stress and a statistically significant increase in average nominal strain. These results were part of the motivation to create a new robust toolpath algorithm to create a single continuous toolpath for any contour. This thesis purely looked at rectilinear style infill, but the toolpath algorithm presented in this thesis can accept any infill pattern or mesh. The other part of the motivation is the lack of precise extrusion control in pellet extrusion MEX AM systems leads to travel move defects. It was shown in Chapter 3 that the novel toolpath algorithm presented in this thesis reduced travel moves by an order of magnitude when compared to other contemporary toolpath algorithms. Reducing travel moves was shown to reduce the overall build time of parts on a pellet MEX AM system. The novel toolpath algorithm also reduced the total distance traveled by a factor of three.

To support this novel toolpath algorithm a slicer was developed using a mix of open source libraries and new functions. This new slicer can reliably handle large STLs, a necessity when slicing parts for large format MEX AM. Slices (contours) can be used instead of STLs, similar to other additive manufacturing processes, eliminating the obtuse and time-consuming step of creating an STL.

The main contributions of this work can be summarized as follows:

1. A new toolpath algorithm that leverages graph theory to generate a single continuous toolpath for any closed contour.
2. A slicer (GRATER) oriented for medium to large format polymer material extrusion additive manufacturing systems that uses the aforementioned toolpath algorithm. For the geometry tested in this thesis, it was found that GRATER had the following results:
    - GRATER reduced travel moves by an *order of magnitude* or 95%.
    - GRATER reduced total distance traveled by a *factor of three*.
    - GRATER reduced the print time by approximately *25%*.
    - GRATER can reliably handle large STLS.
    - Contours can be used instead of STLS, eliminating the obtuse and often time-consuming step of creating an STL.
3. A novel method of combining dense and sparse infill in a continuous, sequential manner that improves mechanical properties of end-use parts.
    - These methods showed a *20% increase* in average effective ultimate stress.
    - These methods showed a *33% increase* in average nominal strain at break.

The number of geometries tested in this thesis is limited. Further refinement and characterization to quantify GRATER's performance on a range of MEX AM machines, geometries, etc, must be explored before definitively claiming that GRATER outperforms other slicers in every scenario.

5.2 Limitations

The code in this thesis is written for research, not for production. This means that GRATER and accompanying code has not been optimized for computational efficiency. GRATER was made to be able to handle large STL files and the overall process planning time is comparable to contemporary slicers, but no time studies were run due to computational efficiency not being the purpose of GRATER and accompanying code. Leveraging graph theory in the path finding algorithm of GRATER allows the algorithm to be inherently robust, but all path finding algorithms in graph theory inherently scale poorly. This poor scaling is due to the brute force approach required for the shortest path problem.

GRATER has not been tested for geometries with internal holes. GRATER does follow the standard for contour direction that denotes these internal feature and the graph theory section was made such that it can accept internal holes. As such, GRATER should be able to handle these geometries, but this is purely speculation and edge cases may exist that would require overhauling the GRATER codebase to handle internal features. If an infill mesh that does not intersect the infill boundary is passed into the **Graph Generation** algorithm, the **Infill Path Finding** algorithm can not find an infill path that includes the islands of infill. This will result in the **Infill Path Finding** algorithm finding a path that traverses the infill boundary, essentially resulting in one extra perimeter instead of the desired infill.

GRATER seeks to holistically improve part properties by minimizing travel moves; this maybe at the detriment of mechanical properties for certain geometries or slicer settings. This thesis only tested GRATER with rectilinear infill for ease of comparison to other slicers. Not every process parameter has been tested to extremes with GRATER, as such process parameters such as too sparse infill percentage could effect GRATER's ability to reduce travel moves.

5.3 Future Work

The toolpath algorithm developed in this thesis may be useful for other additive manufacturing processes, but testing the extensibility of the algorithm fell outside the scope of this thesis. Future work could explore using the continuous toolpath algorithm developed in this thesis for other additive manufacturing technologies that

benefit from continuous pathing. The slicer, discussed in Chapter 2, could be extended to become a robust fully featured slicer. Another option could be to push the toolpath algorithm as a unique feature or infill strategy to existing fully featured slicers. More exotic shortest-path algorithms could be implemented in GRATER that might improve run time as well as capture more of the infill. A mathematical way to determine the correct weight ratio for GRATER would improve the usability of the slicer.

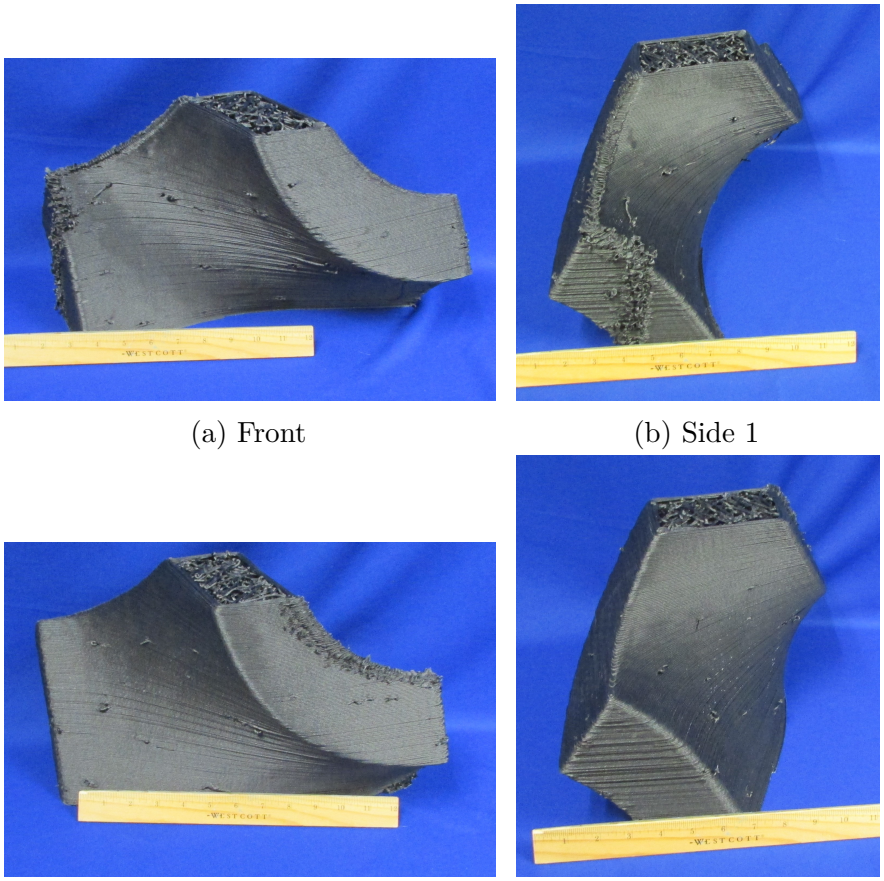
Appendix

Code A.1: The MATLAB script used to calculate the distance Cura traveled.

```
close all; clear all; clc;

file = 'CURA_Thesis.gcode';

D = readlines(file);

R = D(27:87116, 1);
total_distance = 0;

for n = 2:1:length(R)

    if isempty(R{n}) == 1
        continue
    end

    E_check = strfind(R{n}, 'E');

    if isempty(E_check) == 1
        %find previous points
        X_location = strfind(R{n - 1}, 'X');

        if isempty(X_location) == 1
            continue
        end

        Y_location = strfind(R{n - 1}, 'Y');
        spaces = strfind(R{n - 1}, ' ');
        X_end = find(spaces > X_location);
        X_value = str2double(R{n - 1}(X_location + 1:spaces(X_end(1))));
        Y_end = find(spaces > Y_location);

        if isempty(Y_end) == 1
            Y_value = str2double(R{n - 1}(Y_location + 1:strlength(R{n - 1})));
        else
            Y_value = str2double(R{n - 1}(Y_location + 1:spaces(Y_end(1))));
        end

        loc_B = [X_value, Y_value];
        % find current point
        X_location = strfind(R{n}, 'X');
        Y_location = strfind(R{n}, 'Y');
        spaces = strfind(R{n}, ' ');
        X_end = find(spaces > X_location);
        X_value = str2double(R{n}(X_location + 1:spaces(X_end(1))));
        Y_end = find(spaces > Y_location);

        if isempty(Y_end) == 1
            Y_value = str2double(R{n}(Y_location + 1:strlength(R{n})));
        else
            Y_value = str2double(R{n}(Y_location + 1:spaces(Y_end(1))));
        end

        loc_A = [X_value, Y_value];
        total_distance = total_distance + sqrt((loc_B(1) - loc_A(1))^2 + (loc_B(2) - loc_A(2))^2)
    end

end
```

Code A.2: The MATLAB script used to calculate the distance ORNL Slicer traveled.

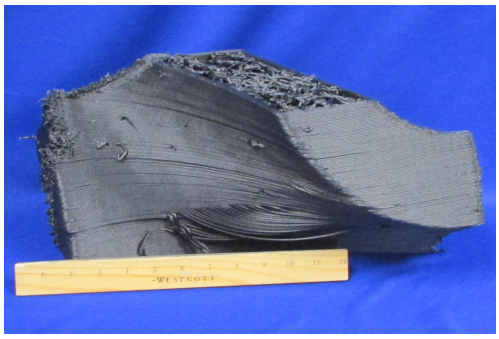
```
close all; clear all; clc;

file = 'Logan_ORNL_Test.gcode';

D = readlines(file);

R = D(446:68537, 1);
total_distance = 0;

for n = 1:1:length(R)
    G_check = strfind(R{n}, 'G');
    travel_check = strfind(R{n}, '; TRAVEL');

    if isempty(travel_check) == 0
        zhop_check = strfind(R{n}, 'Set');

        if isempty(zhop_check) == 0
            continue
        end

        X_location = strfind(R{n}, 'X');
        Y_location = strfind(R{n}, 'Y');
        spaces = strfind(R{n}, ' ');
        X_end = find(spaces > X_location);
        X_value = str2double(R{n}(X_location + 1:spaces(X_end(1))));
        Y_end = find(spaces > Y_location);
        Y_value = str2double(R{n}(Y_location + 1:spaces(Y_end(1))));
        loc_A = [X_value, Y_value];

        X_location = strfind(R{lastG}, 'X');
        Y_location = strfind(R{lastG}, 'Y');
        spaces = strfind(R{lastG}, ' ');
        X_end = find(spaces > X_location);
        X_value = str2double(R{lastG}(X_location + 1:spaces(X_end(1))));
        Y_end = find(spaces > Y_location);
        Y_value = str2double(R{lastG}(Y_location + 1:spaces(Y_end(1))));
        loc_B = [X_value, Y_value];

        total_distance = total_distance + sqrt((loc_B(1) - loc_A(1))^2 + (loc_B(2) - loc_A(2))^2)
    end

    if isempty(G_check) == 0
        X_check = strfind(R{n}, 'X');

        if isempty(X_check) == 0
            lastG = n;
        end

    end

end
```



(a) Front (b) Side 1

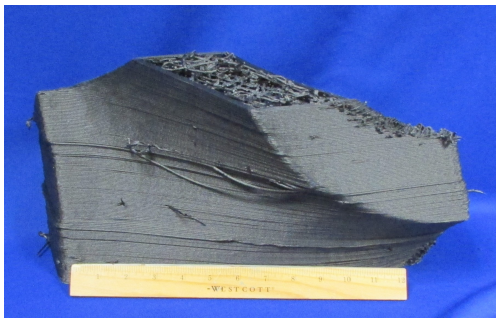
(c) Back (d) Side 2

Figure A.1: GRATER - 1 photos

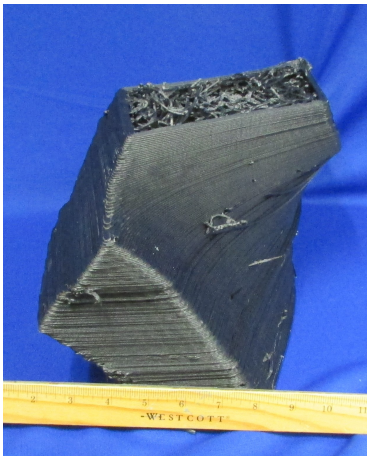


(a) Front

(b) Side 1

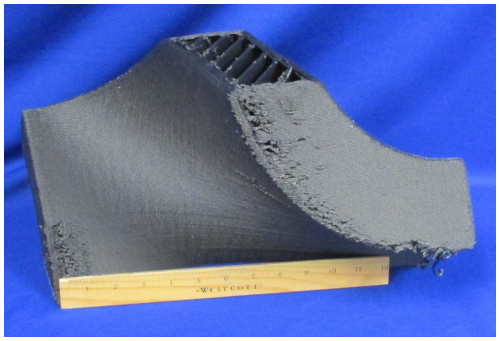


(c) Back

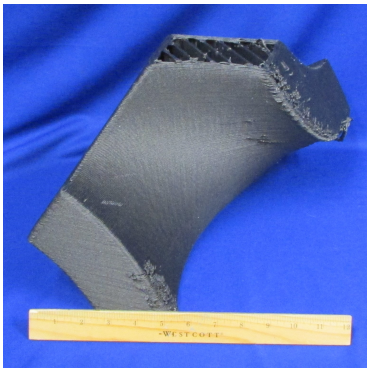


(d) Side 2

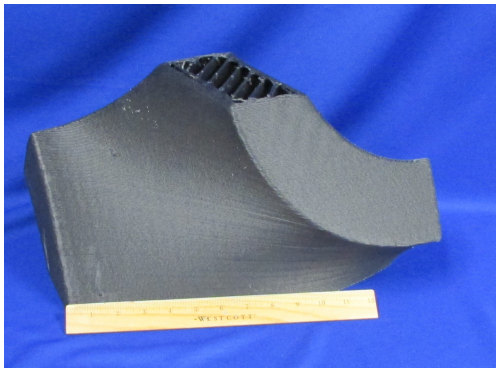
Figure A.2: GRATER - 2 photos



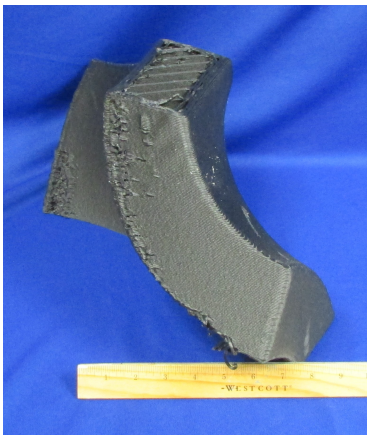
(a) Front



(b) Side 1

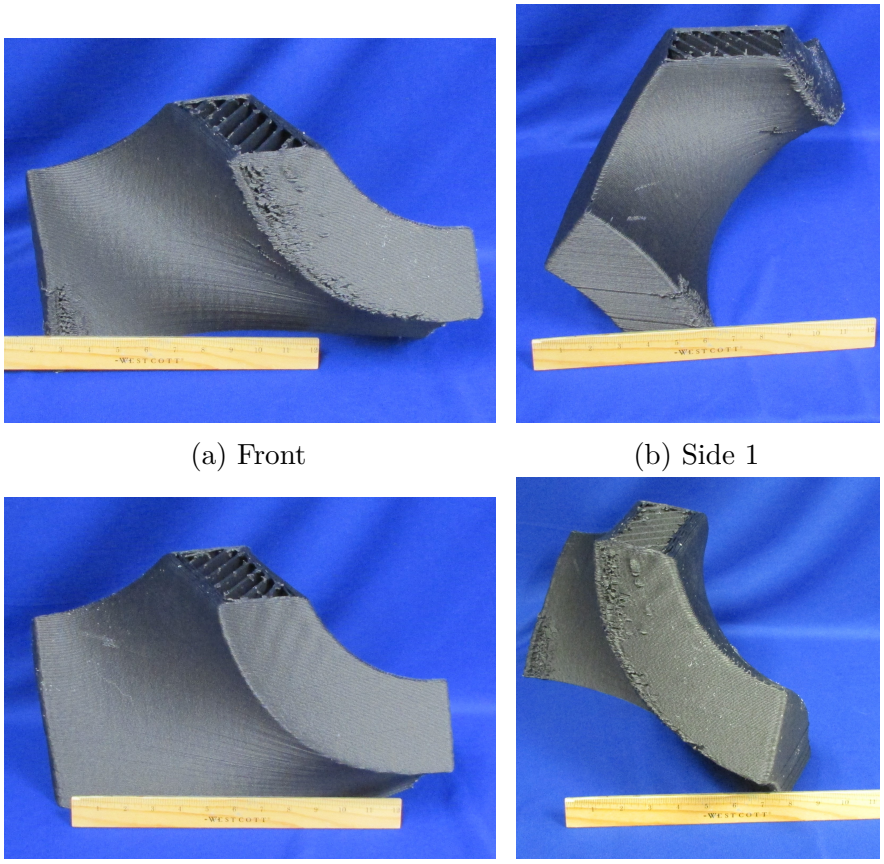


(c) Back



(d) Side 2

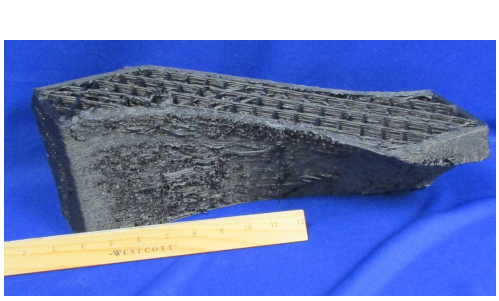
Figure A.3: ORNL Slicer - 1 photos



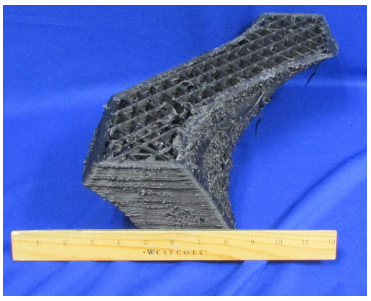
(a) Front (b) Side 1

(c) Back (d) Side 2

Figure A.4: ORNL Slicer - 2 photos

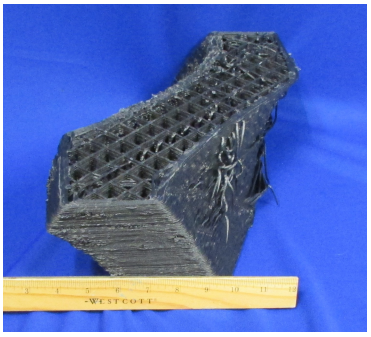


(a) Front



(b) Side 1

(c) Back



(d) Side 2

Figure A.5: Cura - 1 photos

References

[1] Makerbot. How to adjust print settings of a part of my model in Ultimaker Cura. Accessed: 2023-05-18. [Online]. Available: https://support.makerbot.com/s/article/1667417981430

[2] slic3r.org. Slic3r Manual — Welcome to the Slic3r Manual. Accessed: 2023-05-18. [Online]. Available: https://manual.slic3r.org/

[3] J. Bartolai, "Predicting and Improving Mechanical Strength of Thermoplastic Polymer Parts Produced by Material Extrusion Additive Manufacturing," Ph.D. dissertation, The Pennslyvania State University, 2018.

[4] D. Grewell and A. Benatar, "Welding of Plastics: Fundamentals and New Developments," *International Polymer Processing*, 2007, doi:10.3139/217.0051.

[5] M. Roy, R. Yavari, C. Zhou, O. Wodo, and P. Rao, "Prediction and Experimental Validation of Part Thermal History in the Fused Filament Fabrication Additive Manufacturing Process," *Journal of Manufacturing Science and Engineering*, 2019, doi:10.1115/1.4045056.

[6] R. Thomas, "Modeling the Fracture Strength Between Fused-Deposition Extruded Roads," in *International Solid Freeform Fabrication Symposium*, 2000.

[7] Kinematics — Klipper documentation. Accessed: 2024-02-06. [Online]. Available: https://www.klipper3d.org/Kinematics.html

[8] P. Chesser, B. Post, A. Roschli, C. Carnal, R. Lind, M. Borish, and L. Love, "Extrusion control for high quality printing on Big Area Additive Manufacturing (BAAM) systems," *Additive Manufacturing*, 2019, doi:10.1016/j.addma.2019.05.020.

[9] BSI, "Additive manufacturing. General principles. Fundamentals and vocabulary," British Standards Institution, Standard, 2022, doi:10.3403/30448424.

[10] I. Gibson, D. Rosen, B. Stucker, and M. Khorasani, *Additive Manufacturing Technologies*, 3rd ed. Springer International Publishing, 2021. doi:10.1007/978-3-030-56127-7.

[11] Official Creality Ender 3 3D Printer Fully Open Source with Resume Printing Function DIY 3D Printers Printing Size 8.66x8.66x9.84 inch: Amazon.com: Industrial & Scientific. Accessed: 2023-11-08. [Online]. Available: https://a.co/d/89hW8nQ

[12] C. Edge. It's Time For 3D Printing To Evolve From The Hobbyist Market. Accessed: 2023-12-14. [Online]. Available: https://www.bootstrappers.mn/post/it-s-time-for-3d-printing-to-evolve-from-the-hobbyist-market

[13] VORON Design. Accessed: 2023-12-14. [Online]. Available: https://vorondesign.com/

[14] A. Chapman. (2022) The complete history of 3D printing. Accessed: 2023-12-14. [Online]. Available: https://ultimaker.com/learn/the-complete-history-of-3d-printing/

[15] T. G. Crisp and J. M. Weaver, "Review of Current Problems and Developments in Large Area Additive Manufacturing (LAAM)," in *International Solid Freeform Fabrication Symposium*, 2021. [Online]. Available: https://repositories.lib.utexas.edu/bitstream/handle/2152/90746/2021-127-Crisp.pdf

[16] A. Roschli, M. Borish, L. White, C. Adkins, C. Atkins, A. Barnes, B. Post, Z. DiVencenzo, C. Dwyer, G. Rudiak, and B. Zellers, "Transmitting G-Code with Geometry Commands for Extrusion Additive Manufacturing," in *International Solid Freeform Fabrication Symposium*, 2023.

[17] The University of Maine. (2022) UMaine 3D Prints Two New Large Boats for U.S. Marines, Breaking Previous World Record. Accessed: 2023-11-30. [Online]. Available: https://composites.umaine.edu/2022/04/11/umaine-3d-prints-two-new-large-boats-for-u-s-marines-breaking-previous-world-record/

[18] R. Minetto, N. Volpato, J. Stolfi, R. M. Gregori, and M. V. da Silva, "An optimal algorithm for 3D triangle mesh slicing," *Computer-Aided Design*, 2017, doi:10.1016/j.cad.2017.07.001.

[19] C. Fox. (2023) A Little CNC History. Accessed: 2023-11-09. [Online]. Available: https://tormach.com/articles/cnc-history

[20] K. Latif, A. Adam, Y. Yusof, and A. Z. A. Kadir, "A review of G code, STEP, STEP-NC, and open architecture control technologies based embedded CNC systems," *The International Journal of Advanced Manufacturing Technology*, Jun 2021, doi:10.1007/s00170-021-06741-z.

[21] P. Smid, *CNC control setup for milling and turning: mastering CNC control systems*. Industrial Press, 2010.

[22] jbrazio. (2024) What is Marlin? Accessed: 2024-02-06. [Online]. Available: https://marlinfw.org/docs/basics/introduction.html

[23] GNU Lesser General Public License v3.0 — GNU Project — Free Software Foundation. [Online]. Available: https://www.gnu.org/licenses/lgpl-3.0.en.html

[24] MANUFACTUR3D. (2019) Ultimaker Moves To New Headquarters As Company Expands On Global Scale — Manufactur3D. Accessed: 2024-01-27. [Online]. Available: https://manufactur3dmag.com/ultimaker-moves-to-new-headquarters-as-company-expands-on-global-scale/

[25] Prusa Research. General info — Prusa Knowledge Base. Accessed: 2024-01-27. [Online]. Available: https://help.prusa3d.com/article/general-info_1910

[26] RepRap. List of Firmware. Accessed: 2024-02-06. [Online]. Available: https://reprap.org/wiki/List_of_Firmware

[27] grbl, "An open source, embedded, high performance g-code-parser and CNC milling controller written in optimized C that will run on a straight Arduino," original-date: 2009-01-24T23:47:13Z. [Online]. Available: https://github.com/grbl/grbl

[28] L. Li, Q. Sun, C. Bellehumeur, and P. Gu, "Investigation of Bond Formation in FDM Process," in *International Solid Freeform Symposium*, 2002, doi:10.26153/tsw/4500.

[29] R. P. Wool, B.-L. Yuan, and O. J. McGarel, "Welding of polymer interfaces," *Polymer Engineering & Science*, 1989, doi:10.1002/pen.760291906.

[30] T. Lodge and P. C. Hiemenz, *Polymer Chemistry*, 3rd ed. CRC Press, 2020. doi:10.1201/9780429190810.

[31] T. Abbas, F. M. Othman, and H. B. Ali, "Effect of infill Parameter on compression property in FDM Process," *International Journal of Engineering Research and Application*, 2017.

[32] P. Yadav, A. Sahai, and R. S. Sharma, "Strength and Surface Characteristics of FDM-Based 3D Printed PLA Parts for Multiple Infill Design Patterns," *Journal of The Institution of Engineers (India): Series C*, 2021, doi:10.1007/s40032-020-00625-z.

[33] A. Pandzic, D. Hodzic, and A. Milovanovic, "Effect of Infill Type and Density on Tensile Properties of PLA Material for FDM Process," in *DAAAM Proceedings*, 1st ed., B. Katalinic, Ed. DAAAM International Vienna, 2019, vol. 1, doi:10.2507/30th.daaam.proceedings.074.

[34] M. Naik, D. Thakur, and S. Chandel, "An insight into the effect of printing orientation on tensile strength of multi-infill pattern 3D printed specimen: Experimental study," *Materials Today: Proceedings*, 2022, doi:10.1016/j.matpr.2022.02.305.

[35] H. K. Dave, B. H. Patel, S. R. Rajpurohit, A. R. Prajapati, and D. Nedelcu, "Effect of multi-infill patterns on tensile behavior of FDM printed parts," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 2021, doi:10.1007/s40430-020-02742-3.

[36] I. Mustafa and T.-H. Kwok, "Development of Intertwined Infills to Improve Multi-Material Interfacial Bond Strength," *Journal of Manufacturing Science and Engineering*, 2022, doi:10.1115/1.4051884.

[37] L. Xia, S. Lin, and G. Ma, "Stress-based tool-path planning methodology for fused filament fabrication," *Additive Manufacturing*, 2020, doi:10.1016/j.addma.2019.101020.

[38] S. Shaikh, N. Kumar, P. K. Jain, and P. Tandon, "Hilbert Curve Based Toolpath for FDM Process," in *CAD/CAM, Robotics and Factories of the Future*, D. K. Mandal and C. S. Syan, Eds. Springer India, 2016, series Title: Lecture Notes in Mechanical Engineering, doi:10.1007/978-81-322-2740-3_72.

[39] C. Yoo, S. Lensgraf, R. Fitch, L. M. Clemon, and R. Mettu, "Toward Optimal FDM Toolpath Planning with Monte Carlo Tree Search," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020. doi:10.1109/ICRA40945.2020.9196945.

[40] M. Borish, A. Roschli, C. Wade, B. Post, L. White, and C. Adkins, "Single Path Generation for Closed Contours via Graph Theory and Topological Hierarchy," in *International Solid Freeform Fabrication Symposium*, 2023.

[41] A. Roschli, A. Messing, M. Borish, B. K. Post, and L. J. Love, "ORNL Slicer 2: A Novel Approach for Additive Manufacturing Tool Path Planning," in *International Solid Freeform Fabrication Symposium*, 2017. [Online]. Available: https://hdl.handle.net/2152/89889

[42] Oak Ridge National Laboratory, "ORNL Slicer 2 User's Guide," Accessed: 2024-02-12. [Online]. Available: https://ornlslicer.github.io/Slicer-2/doc/Slicer_2_User_Guide.pdf

[43] L. R. Foulds, *Graph Theory Applications*. Springer New York, 1992. doi:10.1007/978-1-4612-0933-1.

[44] J. A. McHugh, *Algorithmic Graph Theory*. Prentice Hall, 1990.

[45] S. Mathew, J. N. Mordeson, and D. S. Malik, *Fuzzy Graph Theory*. Springer International Publishing, 2018. doi:10.1007/978-3-319-71407-3.

[46] D. B. West, *Introduction to Graph Theory*, 2nd ed. Prentice Hall, 2001.

[47] T. Imaizumi, A. Nakayama, S. Yokoyama, and A. Okada, Eds., *Advanced Studies in Behaviormetrics and Data Science: Essays in Honor of Akinori Okada*. Springer, 2020.

[48] D. Legland, "MatGeom library user manual," MATLAB Central File Exchange.

[49] R. Tamassia, Ed., *Handbook of Graph Drawing and Visualization*, 1st ed. CRC Press, Taylor & Francis Group, 2013. doi:10.1201/b15385.

[50] J. Pach, *Towards a Theory of Geometric Graphs*. Providence, R.I.: American Mathematical Society, 2004.

[51] Tradesman Series™ P3-44. Accessed: 2024-02-03. [Online]. Available: https://juggerbot3d.com/products/tradesman-series-p3-44/

[52] Autodesk. Help | STL Export Settings. Accessed: 2024-02-17. [Online]. Available: https://help.autodesk.com/view/RVT/2023/ENU/?guid=GUID-01504202-EF80-4815-9675-ADB8802592BD

[53] S. Bhandari, "slice_stl_create_path(triangles,slice_height)," MATLAB Central File Exchange. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/62113-slice_stl_create_path-triangles-slice_height

[54] MathWorks. Intersection points for lines or polygon edges — MATLAB polyxpoly. R2023b. Accessed: 2023-10-10. [Online]. Available: https://www.mathworks.com/help/map/ref/polyxpoly.html

[55] Dlegland, Oqilipo, J. P. Carbajal, Gaturo, Roozbeh Geraili Mikola, M. Schappler, C. Gorman, Drdadr, Robingeorg, Zubiao Xiong, Hamdan Al-Musaibeli, and S. Holcombe, "mattools/matGeom: MatGeom 1.2.6, doi:10.5281/ZENODO.7799184."

[56] Dijkstra's shortest path algorithm - a detailed and visual introduction. [Online]. Available: https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

[57] thinkyhead. Linear Move. Accessed: 2024-01-16. [Online]. Available: https://marlinfw.org/docs/gcode/G000-G001.html

[58] FARO. (2016) FARO Edge and FARO Laser ScanArmEdge Manual. [Online]. Available: https://downloads.faro.com/index.php/s/N95m3Hi8LZQw4FP

[59] J. D. Valenti, J. Bartolai, and M. A. Yukish, "Experimental Study of Wing Structure Geometry to Mitigate Process-Induced Deformation," in *International Solid Freeform Fabrication Symposium*, 2022, doi:10.26153/tsw/44322.

[60] D20 Committee, "Test Method for Tensile Properties of Plastics," ASTM International, Tech. Rep., 2022, doi:10.1520/D0638-22.

[61] J. Bartolai, A. E. Wilson-Heid, J. R. Kruse, A. M. Beese, and T. W. Simpson, "Full Field Strain Measurement of Material Extrusion Additive Manufacturing Parts with Solid and Sparse Infill Geometries," *JOM*, 2019, doi:10.1007/s11837-018-3217-1.