4-2022

# Thermal Simulation of Additive Manufacturing from G-Code

Noah Foster
jwarr28@lsu.edu

# Thermal Simulation of Additive Manufacturing from G-Code

by

Noah Foster

Undergraduate honors thesis under the direction of

Dr.  Dimitris Nikitopoulos

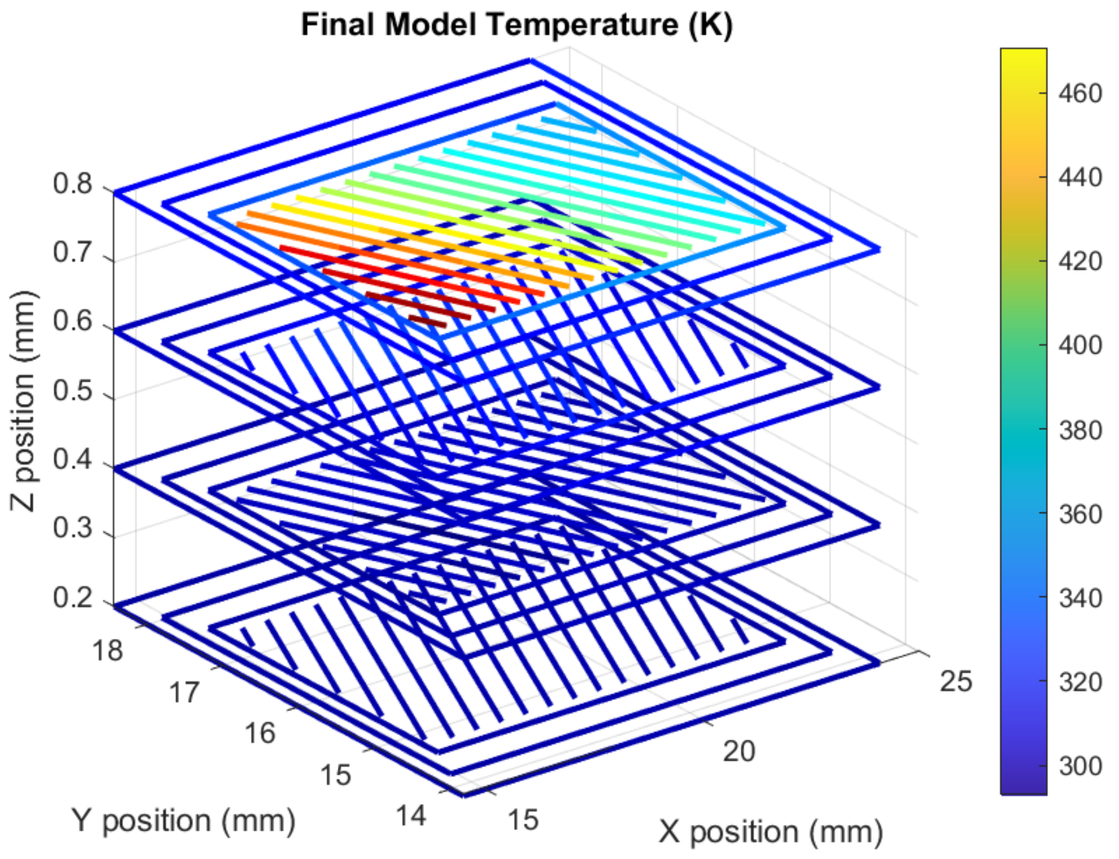
Department of Mechanical Engineering

Submitted to the LSU Roger Hadfield Ogden Honors College in partial fulfillment of
the Upper Division Honors Program.

April, 2022

Louisiana State University
& Agricultural and Mechanical College
Baton Rouge, Louisiana

# Thermal Simulation of Additive Manufacturing



## Abstract

Additive Manufacturing (AM) is a rapidly expanding technique that is being adopted by industries and universities across the world. Accurate thermal simulation of the AM process can lead to less build failures and identify parts with high residual stresses that may benefit from annealing. Traditional FEA simulation of AM parts lacks the capability to simulate the part as built by the G-Code and cannot account for the anisotropic properties that result from the AM process. This work seeks to address this gap by introducing an as-built simulation process which takes the G-Code output from the slicer to dictate the model geometry. By calculating the contact between elements and applying a finite difference approximation, the simulation accurately models the AM process. Because empirical thermal properties were not available, the simulation results in this work should not be used for design decisions. But with proper study of the user's FDM process, this model will provide an accurate approximation of the thermal history of the AM part. Future research should focus on addressing the gaps in the thermal understanding of FDM thermoplastics and consider integrating thermal simulation capabilities into existing slicer software.

## Table of Contents

## Table of Figures

# Introduction

Additive manufacturing (AM) is quickly becoming the industry standard for rapid prototyping and small batch production. However, its growth beyond these applications is limited by its relatively poor repeatability and non-uniform physical properties [1]. For thermoplastic parts manufactured by extrusion-based AM, also known as fused deposition modeling (FDM), the properties are highly dependent on their thermal history throughout the printing process [2]. High thermal gradients can cause warping, insufficient cooling can cause drooping, and rapid heating and cooling can leave high residual stresses and cracks. Simulating the AM process is important for predicting and mitigating these defects, however, due to the complex internal geometry and time dependent manufacturing process, traditional finite element simulation (FEA) is untenable. This thesis details an alternative simulation approach that directly utilizes the G-Code produced by the slicing software to accurately capture the exact manufacturing process.

To accomplish this, the G-Code was interpreted and stored as discrete elements, then the contact between the elements was determined, and finally, the thermal simulation was performed using a forward finite difference approximation. As long as the thermal properties of the FDM process are carefully determined beforehand, this simulation method is capable of returning a part's complete thermal history without the need for computationally expensive FEA.

# Chapter 1: Thermal Simulation of Additive Manufacturing

## 1.1 Background

Since its inception in 1989, FDM has grown rapidly, becoming a staple for industries, labs, and hobbyists [3]. FDM is particularly useful for rapidly manufacturing prototypes and complex parts. Because AM allows users to precisely control both the external and internal geometry of their parts, it is particularly useful to the aerospace and automotive industries for its weight saving potential as illustrated in Figure 1 [4].

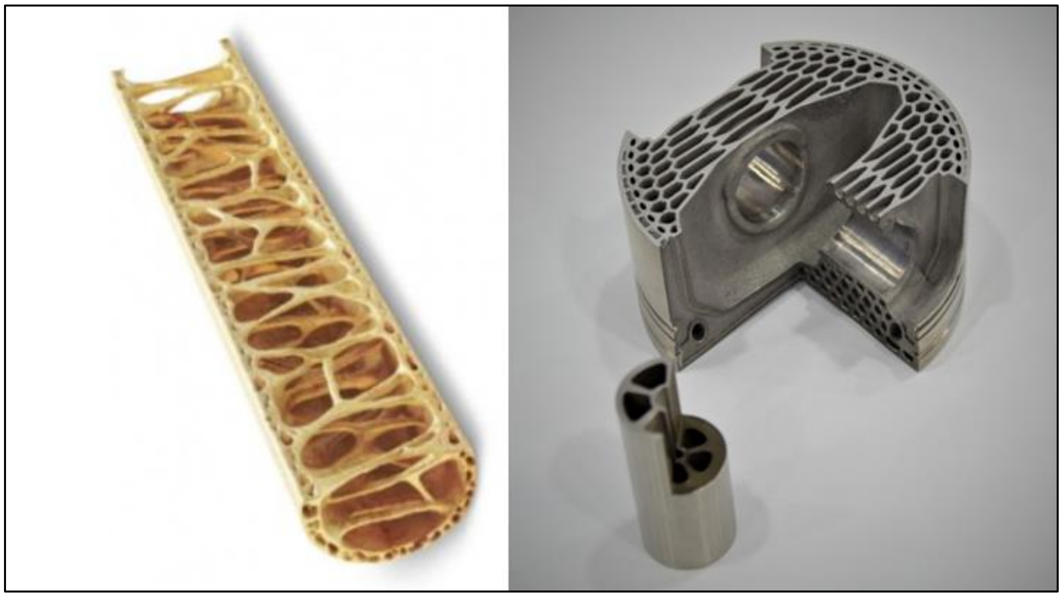


*Figure 1: Bird bone vs. AM part internal geometry [5] [6]*

AM, however, has yet to see widespread adoption in consumer products due to its poor repeatability and difficulty in accurately simulating the manufacturing process [1]. Accurate thermal simulation of AM parts is critical to predicting the properties of AM parts, but traditional FEA is too computationally complex and time intensive for rapid iterations.

## 1.2 LSAM at LSU

The Advanced Manufacturing and Machining Facility (AMMF) at Louisiana State University (LSU) currently lacks the ability to produce Large Scale Additively Manufactured (LSAM) parts. This shortcoming is being addressed by 2021-22 Capstone Team 55, which has been tasked with building a LSAM system using a robotic arm positioning system. The ability to simulate the thermal history of the parts produced by the LSAM system will ensure that prints have the highest chances of success, because high thermal gradients can be caught and avoided, and that the produced parts can be safely used, because the temperature dependent physical properties are known. While such simulations are outside of the scope of Project 55, the simulation method developed for this thesis can be used to perform the necessary simulations.

## 1.3 Problem Definition

To ensure that the developed simulation method accurately and efficiently predicts the thermal history of FDM parts, it is important to fully define the physical problem that will be simulated. Though AM parts are created directly from Computer-Aided Design (CAD) models, the final geometry is not accurately captured by the CAD model, because each AM part requires the CAD model to be processed by a slicing software. Slicing software takes a 3D CAD model and divides it into individual layers to be built sequentially by the 3D printer. Furthermore, the software instructs the printer to build each layer in a specific order: outer walls, inner walls, and then infill according to the geometric pattern specified in the slicer. This process is illustrated in Figure 2.

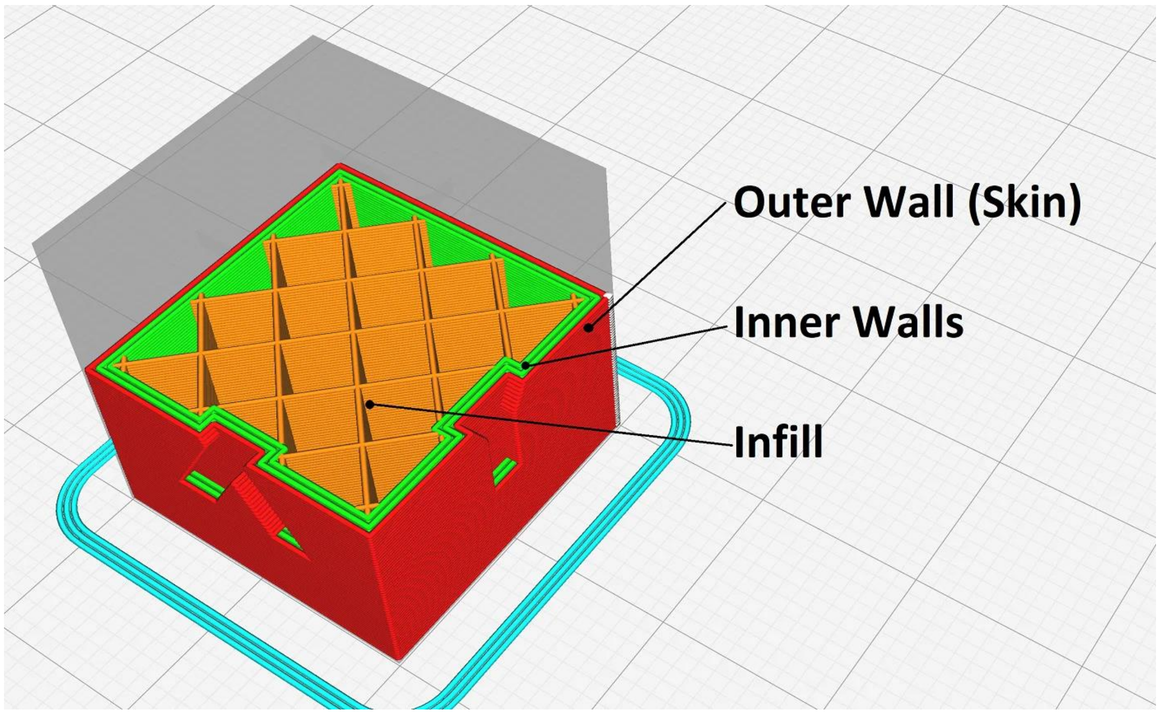


*Figure 2: Sliced geometry for an AM part with grid infill [7]*

For accurate simulation, the exact physical geometry of the AM part must be extracted from the G-Code output of the slicing software.

Conduction, convection, and radiation heat transfer are all non-negligible during the FDM process. The simplified model used to describe the heat transfer process is illustrated in Figure 3 below.

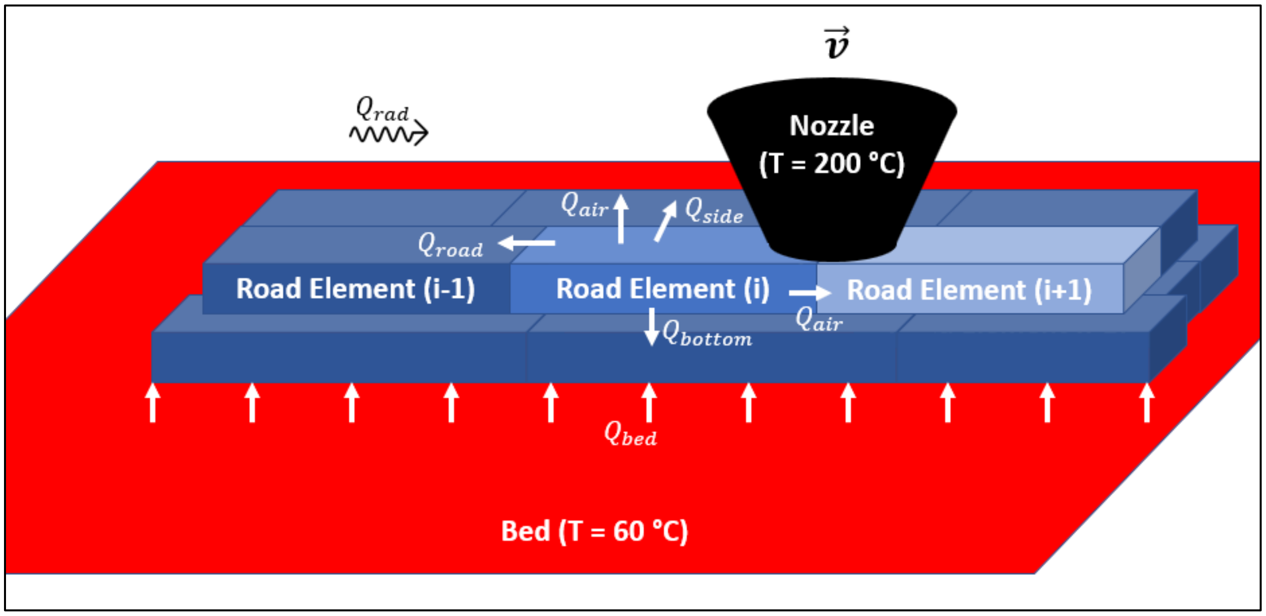


*Figure 3: Element heat transfer diagram*

The heat transfer modes can be broken into two primary categories: element-to-element heat transfer and heat transfer to surroundings. For element-to-element transfer, it is important to distinguish between heat conduction through consecutive elements – $Q_{road}$ – where there is no additional thermal resistance and heat transfer between non-consecutive elements – $Q_{side}, Q_{top}, Q_{bottom}$ – where layer lines and small air bubbles prevent pure conduction. The elements in the bottom layer experience significant heat transfer from the heated bed – $Q_{bed}$ – which is in turn transfered to the elements in subsequent layers. Because the thermoplastic material is extruded at high temperatures, typically 200 °C for PLA, radiation heat transfer – $Q_{rad}$ – is non-negligible. Finally, the amount of area not in direct contact with other elements or the bed, experiences convective heat transfer – $Q_{air}$ – with the air, this is further complicated if a cooling fan is used to provide forced convection around the extrusion area.

## 1.4 Literature Survey

As AM has grown in application and adoption, accurate thermal simulation is an increasingly important research topic. Rapid thermal simulation techniques for Selective Laser Sintering (SLS) with metal powders have been developed by Nachiket et al. in 2013 and Michaleris in 2014 [8] [9]. Both methods utilize the isotropic thermal properties of the metal powder to simplify the physical problem, and Michaleris introduced an additional simplification by categorizing elements as active or inactive based on their proximity to the heat source and time since activation. FDM simulation presents additional challenges not addressed by these methods, due to the anisotropic properties of the parts and lack of a dominating heat transfer form.

Bhandari et al. describes the progression of FDM simulation techniques

> "Compton et al. discussed a 1D transient thermal model to describe a build process and analyze warping and cracking in thin-walled structures. The thermal model was solved by using a finite difference method that calculated the temperature at the nodes at each time step. Zhang et al. used an adaptable, boundary adjusting finite difference method to simulate the thermal history of a 3D-printed polylactic acid (PLA) part. Stockman et al. presented a thermal model tailored for additive manufacturing that was based on the 3D finite difference method. The researchers used coarse meshing in time and space along with simplifying assumptions about the solidification process. The finite difference scheme-based models work well for simple geometries such as thin-layered walls and rectangular cuboids. However, for more complex geometry parts that extrusion-based 3D printing usually produces, a method that can account for changes in geometry is necessary" [2].

While the sophistication of the models has increased, the computational complexity has increased as well. Additionally, these simulation methods often assume the model has a 100% infill and each layer is identically extruded. Zhang and Shapiro addressed this issue by introducing active and inactive elements for the FDM thermal simulation [10]. This method is important for simulating larger and more complex parts and is implemented in this thesis.

# Chapter 2: Thermal Simulation from G-Code Approach

## 2.1 General Approach

The simulation process implemented in this project is structurally similar to the approach of Bhandari et al. shown in Figure 4 below.

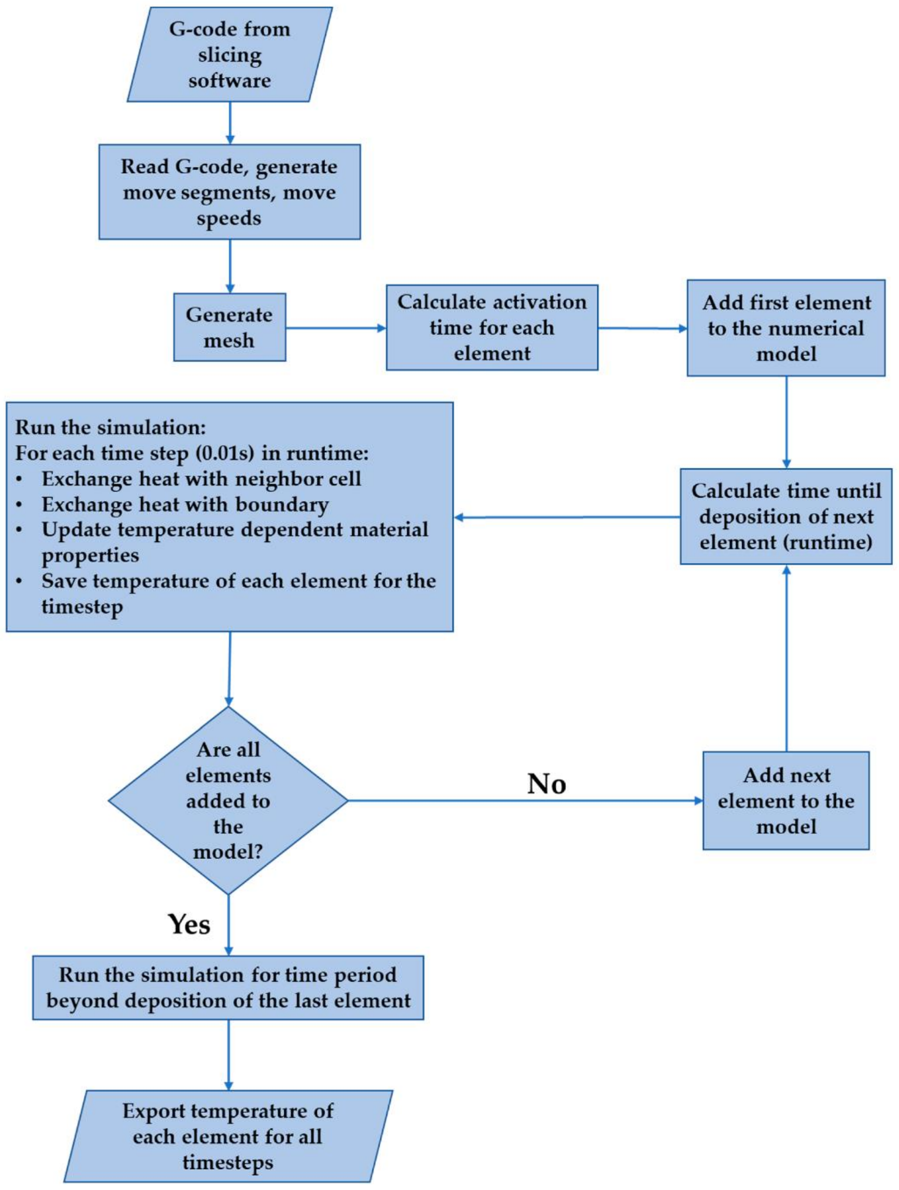

G-code from slicing software
Read G-code, generate move segments, move speeds
Generate mesh
Calculate activation time for each element
Add first element to the numerical model
Run the simulation:
For each time step (0.01s) in runtime:
• Exchange heat with neighbor cell
• Exchange heat with boundary
• Update temperature dependent material properties
• Save temperature of each element for the timestep
Calculate time until deposition of next element (runtime)
Are all elements added to the model?
No
Add next element to the model
Yes
Run the simulation for time period beyond deposition of the last element
Export temperature of each element for all timesteps

*Figure 4: Code flowchart for FDM thermal simulation*

In addition to updating the temperatures, the active elements are updated before each timestep to reduce the simulation time. The simulation can be broken into five key steps which can be coded independently of one another and then run sequentially.

1. Read and reformat G-Code: The G-Code should be accessed by MATLAB and the relevant lines stored in an accessible data structure such as a cell array.
2. Define simulation elements from stored G-Code: The stored G-Code lines should be translated into extruder velocity, element XYZ positions, and element start and end times.
3. Create a contact matrix to calculate and store element interactions: The area of overlap between each element should be calculated and stored.
4. Perform the thermal simulation: The thermal simulation should be run through each timestep, updating the number of extruded elements and active elements for each interval.
5. Display the results: The stored temperature data and final temperature for each element should be displayed as requested by the user.

## 2.2 G-Code Processing

Any G-Code can be divided into three sections: starting G-Code, main G-Code, and end G-Code. The starting G-Code prepares the machine for the main building operations and the ending G-Code contains the relevant shutdown codes to prepare the machine for the next operation. While important to the FDM process, these codes do not contain relevant information for the thermal simulation and the simulation program should ignore these lines.

The main G-Code can contain millions of lines of code depending on the complexity and scale of the model, however, each of these lines is structurally similar. An excerpt of G-Code from a simple rectangle is listed below.

### Test Rectangle G-Code

```
G1 F775.6 X15.447 Y16.578 E12.72979
G0 F12000 X15.447 Y17.144
G1 F775.6 X15.781 Y17.478 E12.74551
G0 F12000 X15.781 Y17.83
;MESH:NONMESH
G0 F300 X15.781 Y17.83 Z0.8
G0 F12000 X15.781 Y17.92
G0 X23.698 Y18.03
;TIME_ELAPSED:39.255089
;LAYER:3
M106 S255
;TYPE:WALL-INNER
;MESH:TestBlock v1.stl
G1 F774.2 X14.898 Y18.03 E13.03819
G1 X14.898 Y14.23 E13.16458
G1 X23.698 Y14.23 E13.45727
G1 X23.698 Y18.03 E13.58366
G0 F12000 X24.098 Y18.43
;TYPE:WALL-OUTER
G1 F774.2 X14.498 Y18.43 E13.90296
G1 X14.498 Y13.83 E14.05595
```

All comment lines are identified by a semicolon at the start of the line. While helpful for understanding the code, these lines do not contain needed information and should be discarded. Similarly, lines that begin with an M-Code can be discarded as these are machine codes and do not impact any of the thermal simulation parameters. Though information like the line width and height, nozzle and bed temperature, maximum nozzle travel speed, etc. could be recovered directly from the G-Code, it is far quicker and easier to input these parameters manually from the slicer settings.

The relevant G-Code lines all begin with G0 or G1. G0 lines contain XYZ information and are used for rapid positioning of the nozzle without extrusion. G1 lines contain XYZ information as well as extruder – E – position and feedrate – F – speed. Traditionally, XYZE are in mm and F is in mm/min. For each line, any of these five indexes may be referenced in FXYZE order, if one is not listed, its value is inherited from the preceding line. To store the information in each line, a cell array with 6 columns can be used, because, while unlikely, a line could potentially include a G command and FXYZE commands. The basic program structure for storing the G-Code data should follow the pseudo-code below.

**G-Code Processing Program Pseudo-Code**

```
1: Open G-Code file in MATLAB
2: Discard starting G-Code lines
3: Loop through main code
  A: Locate G0 and G1 commands
  B: Split commands by spaces and store in cell array
  C: Discard other lines
4: Discard ending G-Code and close file
```

## 2.3 Element Definition

To effectively simulate the FDM process, the continuous lines of extruded material must be discretized into small elements. These elements are defined by seven variables: their starting and ending X and Y positions, their starting and ending time, and their layer. This information can be easily recovered from the processed G-Code. The elements are also defined by their height, width, and material; however, these parameters are assumed to remain constant throughout the FDM process.

Each G1 command could be directly stored as a single element, but elements created this way would be arbitrarily long. In order to approximate the elements as being laid instantaneously, the elements must be sufficiently short. For thermoplastics, a maximum timestep of 0.1 seconds is reasonable for simulation stability and speed [10]. To cut longer elements such that they are divided into equal parts with $t_{max} < 0.1s$, the following pseudo-code is used.

**Element Division Pseudo-Code**

```
If element_time > 0.1s
```

```
   Number of splits = element_time rounded up then divided by 0.1
   For each split
         Element_time_split = element_time divided by the number of splits
         Repeat for XYZ position
```

With the above splitting algorithm in place, the element definition program can be created following the pseudo-code below.

### Element Definition Pseudo-Code

```
 For all lines processed lines of G-Code
   For columns 2-6
         If the first character is “F”
               Update current speed
         If the first character is “X”
               Update the current X position
         If the first character is “Y”
               Update the current Y position
         If the first character is “Z”
               Update the current Z position
               Update the current layer
         If the first character is “E”
               Update the current E position
   Calculate distance traveled
   Update current time based on distance travel and speed
   If the line is a G1 command and material was extruded
         If element_time > 0.1s
               Split element and store split elements
         Else
               Store element
   Set old XYZE position to equal current XYZE position
```

## 2.4 Contact Matrix

Convective and conductive heat transfer between two objects are both proportional to the contact area between those objects. It is therefore necessary to accurately determine the contact area between each element in the FDM thermal simulation. This, however, is a non-trivial problem as illustrated in Figure 5. Each side of each element could have contact with any number of other elements. Additionally, this overlap has multiple fringe cases which appear quiet frequently in the G-Code.

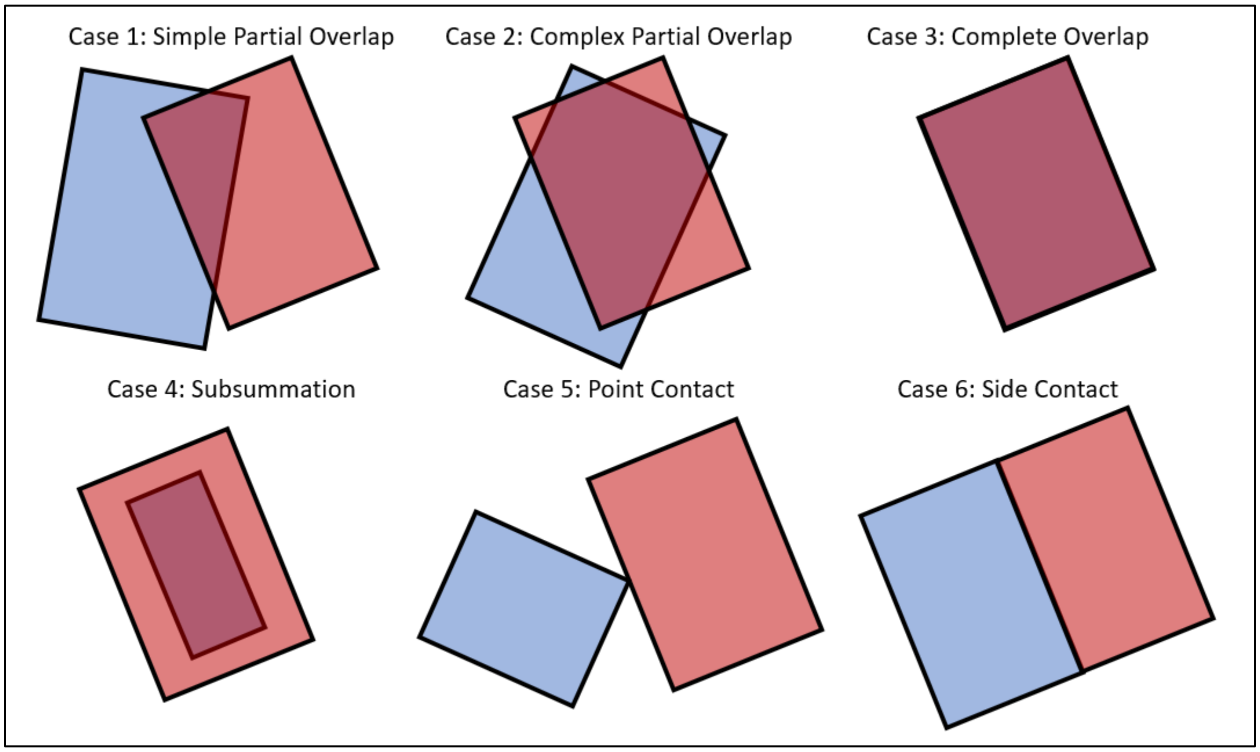


*Figure 5: Possible contact areas*

Algorithms to determine the overlapping area of two rectangles are quite common when the rectangles can both be assumed to align with the X and Y axes, however, for the case of arbitrary orientation, no simple solution was found. A workaround to this issue was found by applying the Sutherland-Hodgman algorithm, an algorithm that returns the polygon of overlap between two arbitrary polygons, and MATLAB's built-in function for returning the area of an arbitrary polygon. The original Sutherland-Hodgman algorithm was coded in 1974 and has since been translated into MATLAB, as well as many other programming languages, by Rosetta Code [11].

To reduce simulation time, all easily recovered contact areas are first determined before applying the Sutherland-Hodgman algorithm to the remaining elements. Additionally, the model is divided by an XY grid with side lengths defined by the maximum distance the nozzle can travel in 0.1 seconds. Because no element is larger than this distance, only elements in the same section or neighboring sections must be checked for contact.

While these conditions are sufficient for contact between elements in adjacent layers, determining the side contact requires a slight modification and another step. Because the area of overlap in Case 6 is zero, the algorithm cannot properly capture the contact between such elements. To remedy this, one element is expanded a very small amount in all directions before applying the Sutherland-Hodgman algorithm. The longest side of the returned polygon is then recovered and its length multiplied by the height of the element to calculate the total area of overlap.

### Contact Matrix Pseudo-Code

```
For all layers
  For all elements in layer
        Calculate element bounding box
        For all other close elements in layer
              Calculate expanded bounding box
              Calculate polygon of overlap
              Calculate longest side of overlap
              Store contact area in matrix
        For all other close elements in above layer
              Calculate bounding box
              Calculate polygon of overlap
              Calculate area of overlap
              Store contact area in matrix
Store top contact area as bottom contact area shifted up a layer
```

## 2.5 Thermal Simulation

Once the G-Code has been processed and the model has been reconstructed, the thermal simulation can begin. For an arbitrary element, five modes of heat transfer must be calculated. These modes of heat transfer are defined in Table 1 below, all variables used in equations are listed in Appendix B.

*Table 1: Element heat transfer modes*

| Type | Equation [10] |
|---|---|
| Radiation to surroundings | $Q_i^{\mathrm{rad}} = \int_{\Delta t} \varepsilon\sigma A_i^{\mathrm{free}}(T_i(t)^4 - T_\infty^4)\cdot dt \quad (1)$ |
| Convection to surroundings | $Q_i^{\mathrm{conv}} = \int_{\Delta t} h_{\mathrm{conv}} A_i^{\mathrm{free}}(T_i(t) - T_\infty)\cdot dt \quad (2)$ |
| Conduction to consecutive elements | $Q_i^{\mathrm{cond}} = \int_{\Delta t} \lambda\left(A_{i,i-1}\frac{T_i(t) - T_{i-1}(t)}{0.5(L_i + L_{i-1})} + A_{i,i+1}\frac{T_i(t) - T_{i+1}(t)}{0.5(L_i + L_{i+1})}\right)\cdot dt \quad (3)$ |
| Conduction to contacting elements | $Q_i^{\mathrm{elem}} = \sum_{j\in\mathbb{S}_i}\int_{\Delta t} h_r A_{i,j}^c\left(T_i(t) - T_j(t)\right)\cdot dt \quad (4)$ |
| Conduction to heated bed | $Q_i^{\mathrm{bed}} = \int_{\Delta t} h_c A_i^{\mathrm{bed}}(T_i(t) - T_{\mathrm{bed}})\cdot dt \quad (5)$ |

Each element can be approximated as having a spatially uniform temperature distribution due to their small Biot numbers. Using this approximation, the conservation of energy can be applied as seen in Equation 6.

$$-\rho cV\Delta T = Q^{rad} + Q^{conv} + Q^{cond} + Q^{elem} + Q^{bed} \tag{6}$$

Finally, Equation 6 is discretized in time using a forward Euler approximation to form the finite difference approximation shown in Equation 7 below [10].

$$\rho cV_i(T_i^{n+1} - T_i^n) = -\Delta t^n\left(\lambda\left(A_{i,i-1}\frac{T_i^n - T_{i-1}^n}{0.5(L_i + L_{i-1})} + A_{i,i+1}\frac{T_i^n - T_{i+1}^n}{0.5(L_i + L_{i+1})}\right) + h_{\mathrm{conv}}A_i^{\mathrm{free}}(T_i^n - T_\infty)\right.$$
$$\left. + \varepsilon\sigma A_i^{\mathrm{free}}((T_i^n)^4 - T_\infty^4) + h_c A_i^{\mathrm{bed}}(T_i^n - T_{\mathrm{bed}}) + \sum_{j\in\mathbb{S}_i} h_r A_{i,j}^c(T_i^n - T_j^n)\right) \tag{7}$$

With these equations defined, the pseudo-code proceeds as seen below.

### Thermal Simulation Pseudo-Code

```
Define parameters
For all layers
  For all elements
        Calculate timestep
        For all active layers
              For all active elements
                    Calculate Qs
                    Update temperature
                    Store temperature and current time
```

## 2.6 Results and Validation

After performing the simulation, the data must be stored and processed appropriately to highlight potential problems and inform design decisions. The temperature of each element in the simulation is recorded for each timestep that element is active. From this data, the heating and cooling of each element can be graphed and the thermal gradients throughout the model can be recovered.

A simple check for the numerical model is to compare the results of a known case to the analytical solution. For a line of filament extruded at a constant temperature – $T_0$ – and velocity – v, with uniform width – W – and height – H, the heat transfer equation, considering only conduction along the extrusion direction and uniform convection around the perimeter of the extruded line, reduces to Equation 8 [12].

$$\rho c \mathrm{A}_{\mathrm{cross}} \frac{\partial T}{\partial t} = A_{cross} \lambda \frac{\partial^2 T}{\partial x^2} - h P_{cross}(T - T_\infty) \tag{8}$$

This equation can be reduced to the ordinary differential equation in Equation 9 by substituting $\frac{\partial T}{\partial t} = \frac{\partial T}{\partial x} v$ and $T^* = (T - T_\infty)$ into Equation 8.

$$\frac{\lambda}{\rho c}\frac{\partial^2 T^*}{\partial x^2} - \frac{\partial T^*}{\partial x} - \frac{hP_{cross}}{\rho c A_{cross}} T^* = 0 \tag{9}$$

By applying the boundary conditions

$$T = \begin{cases} T_0 & x = 0 \quad \text{and} \quad t \geq 0 \\ T_\infty & x = \infty \quad \text{and} \quad t \geq 0 \end{cases} \tag{10}$$

Equation 9 has the known solution shown in Equation 11.

$$T = T_\infty + (T_0 - T_\infty)\mathrm{e}^{-\mathrm{mx}} \tag{11}$$

$$m = (\sqrt{(1 + 4\alpha\beta)} - 1)/2\alpha, x = vt, \alpha = (\lambda/\rho c v), \beta = (hP/\rho \mathrm{c} \mathrm{A} v)$$

For a given set of thermal and geometric parameters, the numerical simulation and analytical solution can be compared to check the accuracy of the simulation.

For more complex models, no analytical solution exists, however, the entire model can be rendered visually and various sanity checks can be performed. For example, at $t_\infty$ it is expected that $T_{elem} \approx T_{air}$ for all elements in the top layer. By reconstructing the model and coloring the elements to reflect their temperature at various simulation times, the model can be visually checked to ensure it aligns with expectations.

To perform the operations discussed, the following pseudo-code was used.

### Data Analysis and Validation Pseudo-Code

```
Read element data
Render 3D model with temperature coloration
Read maximum and minimum temperature
Normalize temperature data to range [min, max]
Load colorjet map
Plot all elements using colorjet color range
Compare analytical solution to numerical simulation
  Calculate analytical solution using the defined thermal parameters
  Plot the temperature data of the middle element used in the simulation
  Plot the analytical solution
```

# Chapter 3: Thermal Simulation Implementation

## 3.1 G-Code Processing Program

The full MATLAB programs are listed Appendix A; however, it is also important to discuss the implementation methods used in each so that the programs can be expanded upon by other researchers and modified as needed by other users. The G-Code processing file will require slight modifications depending on the slicer software used and the name of the stored G-Code file.

### G-Code Processing Code Requiring Modification

```
% Select file to open
fid = fopen('YOUR_FILE_HERE.gcode','r');

% Initialize line variables
tlines = cell(0,1);
check = false;
% Loop until beginning of relavent Gcode
while check == false
    line = fgetl(fid);
    if length(line) >= 5
        if line(1:5) == ';MESH'
            check = true;
        end
    end
end
```

The G-Code file on which the simulation is to be performed must be selected by modifying the first field in the *fopen* function with the appropriate filename. Additionally, the identifier used to discard the starting G-Code must be updated depending on the slicer used. Here, the Cura slicer is used, and ';MESH' is selected as the break before the relevant G-Code. This is because, as seen in the code below, all Cura G-Code files use this identifier as the first 5 characters in the line before the model file is built.

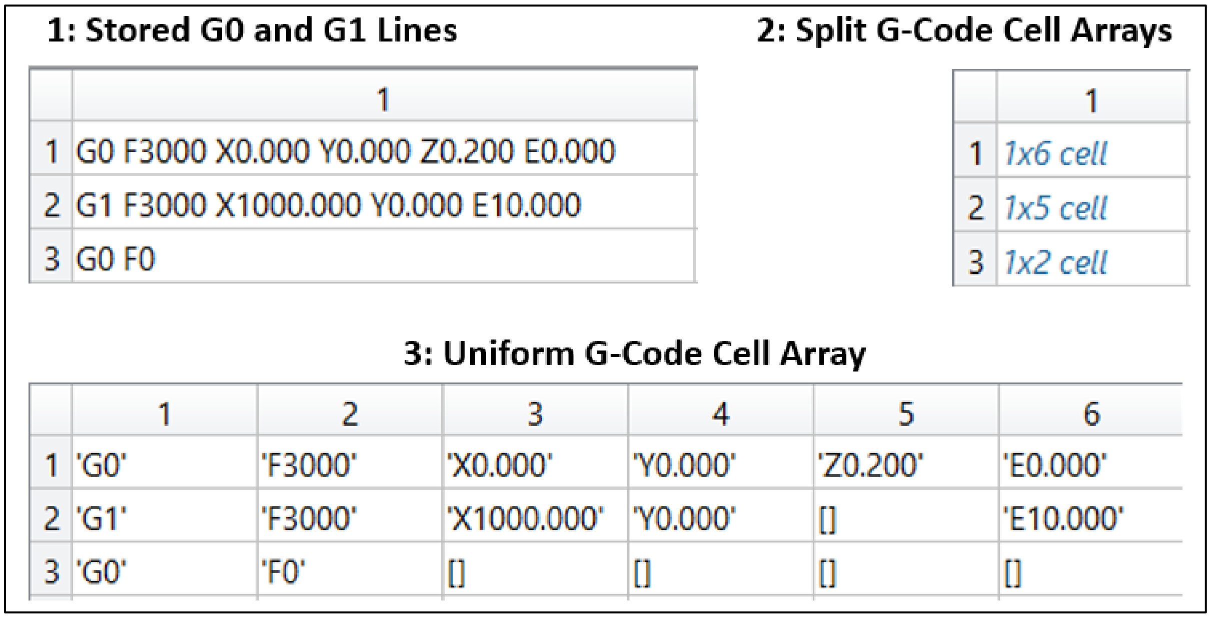
### Start-up G-Code Excerpt from Cura Slicer

```
G28 ;Home

G92 E0 ;Reset Extruder
G1 Z2.0 F3000 ;Move Z Axis up
G1 X10.1 Y20 Z0.28 F5000.0 ;Move to start position
G1 X10.1 Y200.0 Z0.28 F1500.0 E15 ;Draw the first line
G1 X10.4 Y200.0 Z0.28 F5000.0 ;Move to side a little
G1 X10.4 Y20 Z0.28 F1500.0 E30 ;Draw the second line
G92 E0 ;Reset Extruder
G1 Z2.0 F3000 ;Move Z Axis up

G92 E0
G92 E0
G1 F2700 E-5
;LAYER_COUNT:4
;LAYER:0
M107
;MESH:TestBlock v1.stl
G0 F6000 X23.698 Y18.03 Z0.2
;TYPE:WALL-INNER
G1 F2700 E0
G1 F802.2 X14.898 Y18.03 E0.29269
```

While this is true for the Cura slicer, this comment line may not be included in other slicing software. If the ';MESH' identifier is not present, a different identifier should be selected, or the identifier should be inserted into the G-Code file before the start of the first layer.

After discarding the starting G-Code, the remaining G-Code should be stored until the end identifier is found. For the Cura slicer, ';End' was chosen as the identifier. If a different slicer is used, this identifier should be updated as well. Because the G-Code can contain any number of comment lines or M codes, only lines with G commands should be stored in a cell array. This is accomplished by checking whether or not the first character of each line is a 'G', and, if so, storing the line.

Once each G command line is stored, the lines must be split using the space key – ‘ ‘ – as the delimiter. The *strsplit* function can be used on each cell to split each parameter into a separate cell. Because the resulting cell arrays have non-uniform dimensions, they should be standardized by segmenting all the G commands in the first cell, all the F commands in the second cell, all the X commands in the third cell, etc. This division process is illustrated in Figure 6 for a G-Code program with 3 lines.


**1: Stored G0 and G1 Lines**

| | 1 |
|---|---|
| 1 | G0 F3000 X0.000 Y0.000 Z0.200 E0.000 |
| 2 | G1 F3000 X1000.000 Y0.000 E10.000 |
| 3 | G0 F0 |

**2: Split G-Code Cell Arrays**

| | 1 |
|---|---|
| 1 | *1x6 cell* |
| 2 | *1x5 cell* |
| 3 | *1x2 cell* |

**3: Uniform G-Code Cell Array**

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 'G0' | 'F3000' | 'X0.000' | 'Y0.000' | 'Z0.200' | 'E0.000' |
| 2 | 'G1' | 'F3000' | 'X1000.000' | 'Y0.000' | [] | 'E10.000' |
| 3 | 'G0' | 'F0' | [] | [] | [] | [] |

*Figure 6: G-Code cell division process*

With this segmentation complete, the G-Code information can be quickly and conveniently accessed throughout the rest of the simulation process.

## 3.2 Element Definition Loop

Using the processed G-Code, the model can be broken into small elements that can be approximated as having uniform properties. While the G0 codes contain important position information, only G1 codes are used for extrusion. However, not all G1 codes produce model elements, as G1 codes can also be used for filament retraction before rapid movement. Therefore, for each line of G-Code, the time and position information should be updated but model elements should only be created if it is a G1 command and the new extruder position is larger than the previous extruder position.

For a valid approximation, the Biot number of each element should be less than 0.1 [10]. In general, the heat transfer coefficient of thermoplastics in air can be up to four orders of magnitude larger than its thermal conductivity: e.g., $h_{PLA} = 101\frac{W}{mK}, \lambda_{PLA} = 0.13\frac{W}{m^2K}$ [13] [14]. This requires that the characteristic length – $L_c$ – be approximately less than or equal to 0.0001 m or 0.1 mm. The characteristic length is defined for a rectangular prism by Equation 12.

$$L_c = \frac{V}{A_{surface}} = \frac{lwh}{2lw + 2lh + 2wh} \tag{12}$$

This equation can be solved for the element length – l – in terms of the element width – w – and height – h. A script was then created to validate whether a maximum timestep of 0.1 seconds would result in elements with valid Biot numbers.

### Valid Element Lengths Function

```
   % Function to determine if 0.1 timestep is valid
   function [L_min, L_max] = Length_Determine (w,h,max_speed)
   L_min = -w*h/(w+h-5*w*h); % Calculate characteristic length that satisfies
Lc < 0.1 mm
   L_max = max_speed*0.1; % Calculate maximum length for 0.1 sec timestep

   if L_max <L_min % Throw error if Biot number would be too large
       error('Biot Number too large!');
   end

   % Print valid element length range
   fprintf('Element  length  must  be  greater  than  %f  and  less  than  %f
\n',L_min,L_max);
   end

   >> [L_min, L_max] = Length_Determine (0.4,0.2,50);
   Element length must be greater than -0.400000 and less than 5.000000
```

For the standard line width of 0.4 mm, line height of 0.2 mm, and max speed of 50 mm/s, the maximum Biot number possible for the elements is 0.05, therefore, a uniform temperature can be assumed for each element. Once this assumption is confirmed, the element definition code can be run without issue.

## 3.3 Contact Graph and Clipping Algorithm

The contact graph code depends on the line width and height defined in the slicing software. To ensure the clipping algorithm runs correctly, these values must first be defined at the start of the code. To accurately simulate the anisotropic properties of the FDM part, four forms of contact must be calculated and stored: the top contact, bottom contact, side contact, and consecutive element contact. This can be simplified to three forms of contact – because the bottom contact area of the layer(k+1) is equivalent to the top contact area of layer(k) – by running the following line of code.

```
% Set bottom contact based on top contact with shifted layers
bottomcontact(:,:,2:h) = topcontact(:,:,1:h-1);
```

To calculate the top and side contact between each of the elements, the Sutherland-Hodgman algorithm was applied to all element pairs. To loop through element pairs the following structure was used.

### For Loop Structure for Contact Matrix Program

```
for i = 1:l
   % Skip iteration if Elem row is empty
   if sum(Elem(i,:,k)) ~= 0
       % Calculate subject rectangle for side overlap
       subside = Box_Over(Elem(i,1:4,k),W);
       % Caclulate subject rectangle for top overlap
       sub = Box(Elem(i,1:4,k),W);
       % Loop through all other elements in row
       for j = 1:l
        % Apply condition checks
```

```
            % Calculate and store overlap
```

To reduce computation time, before applying this algorithm the following conditions were checked and the algorithm was skipped if any of the conditions were met.

1. If elements i and j are not in neighboring grid squares
2. If i = j
3. If i and j are consecutive
    a. Store consecutive data

If none of the conditions were true, the Sutherland-Hodgman algorithm was applied to the elements. If the algorithm detected overlap, the longest side was stored as the side contact and the total area was stored as the top contact.

## 3.4 Finite Difference Approximation Thermal Simulation

The thermal simulation program requires the user to define a large number of thermal parameters. These parameters are listed in the code below.

### Thermal Parameters for Thermal Simulation

```
W = 0.4; % Width in mm
H = 0.2; % Height in mm
K = 273.15; % Conversion to Kelvin
h_bed = 50; %W/m2*K Thermal contact coefficient to bed
h_elem = 50; %W/m2*K Thermal contact coefficient to touching elements
h_air = 50; %W/m2*K Heat transfer coefficient to air
lam = 0.13; %W/m*K Thermal conductivity
T_air = 25+K; % K Temperature of air
T_bed = 25+K; % K Temperature of bed
p = 1300; % kg/m^3 Density of PLA
c = 1800; % J/kg*K Specific heat capacity of PLA
E = 0.9; % Emissivity of PLA
```

These thermal properties of PLA can be easily found in literature, however, no accepted values for the thermal contact coefficients and heat transfer coefficient were found. These parameters should be determined experimentally for the thermoplastic and bed material being used and the room in which the print is taking place.

To save computational resources, the storage for the temperature data collected during the simulation should be preallocated. This is accomplished using the following for loop.

### Preallocation For Loop for Thermal Simulation

```
% Define temperature collection cell array
Elem_Temp_time = cell(l,h);
% Preallocate memory for stored temperature data
for k = 1:h
    for i = 1:l
        Elem_Temp_time{i,k} = NaN(2,(h-k)*(l)+(l+1-i));
        %Store temperature and time data
        Elem_sim(i,w+1,k) = Elem_sim(i,w+1,k) + 1;
        Elem_Temp_time{i,k}(1,1) = Elem_sim(i,7,k);
        Elem_Temp_time{i,k}(2,1) = Elem_sim(i,11,k);
    end
end
```

First, a cell array is created with length equal to the number of elements per layer – l – and height equal to the number of layers – h. Then each cell is filled with a NaN matrix with dimensions equal to the number elements remaining in the model.  Finally, the first column of each matrix is filled with that element’s starting time and temperature.

The thermal simulation then proceeds following the logic shown in the Pseudo-Code in Section 2.5. For this simulation to work properly, the free area – $A_{free}$ – must be calculated accurately or the simulation becomes unstable. To do this, the total possible free area is defined as the entire surface area of the element. Then, as each form of heat transfer is calculated, the area used in that calculation is subtracted from the free area. Finally, $Q_{rad}$ and $Q_{conv}$ can be calculated over the remaining free area.

To reduce computation time in large models, only the active elements are included in the simulation. To calculate which elements are active at each time step, the *Active* function was built. This function returns 1 for all active elements and 0 for all inactive elements. An element is active if any of the following conditions are met.

1. The element was extruded less than 8 seconds ago
2. The current element being extruded is in the same or a neighboring grid square and the current element is less than 3 layers away

For elements with width and height much larger than those used here, the inactive time of 8 seconds should be redetermined.

## 3.5 Results Collection and Graphics

Thermal simulation results are typically returned as graphics with a temperature dependent color gradient and graphs of a particular element or nodes temperature plotted against time. To graph the model as built, each element can be plotted using the *plot3* function in MATLAB. To color each element based on its temperature, the jet colormap, as seen in Figure 7, was used.

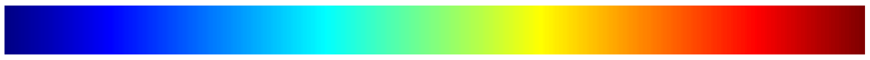


*Figure 7: Jet colormap*

The jet colormap has 256 possible colors, so the temperature data collected must be normalized to a range of 1-256. To do this, the maximum and minimum temperature are first identified, then the temperatures are normalized from 0-1. Next, they are multiplied by 255, rounded down, and then 1 is added.

Plotting a single element's temperature data is straightforward, first, that elements cell should be accessed and then the first row plotted along the x-axis and the second row plotted along the y-axis. To do so, the position element position – I – and layer – K – should be input in the code below.

## Element Temperature Graph Code

```
   figure(2)
   plot((Elem_Temp_time{100,1}(1,:)-
Elem_Temp_time{100,1}(1,1)),Elem_Temp_time{I,K}(2,:))
   title('Element 100 temperature data')
   xlabel('Time (sec)')
   ylabel('Temperature (K)')
```

# Chapter 4: Results and Improvements

## 4.1 Simple Accuracy Test Results

A simple accuracy test was performed using the known analytical solution to a single line of extruded filament at constant velocity – v. The results from the analytical solution and numerical solution with an 0.1 sec maximum timestep are shown in Figure 8 below.

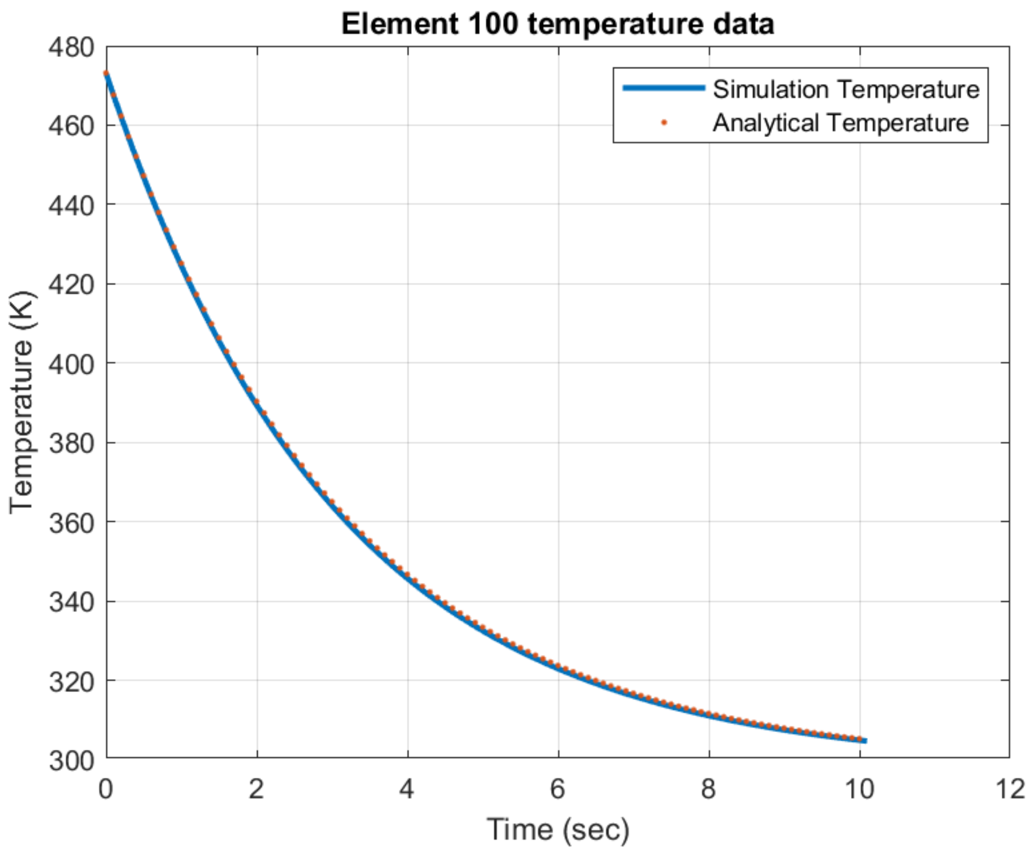


*Figure 8: Simulation temperature vs. analytical temperature*

The numerical and analytical solutions closely align with a maximum temperature error of 0.28% at 3.1 seconds and an average error of 0.21% from 0-10 seconds. This suggests a high degree of accuracy in the numerical simulation. For additional verification, this check was run with the varying parameters shown in Table 2 below.

*Table 2: Numerical and Analytical Solution Comparison*

| $h\left(\frac{W}{m^2K}\right)$ | $\lambda\left(\frac{W}{mK}\right)$ | $c\left(\frac{J}{kgK}\right)$ | $T_\infty\ (C)$ | $W * H(mm^2)$ | Max Error | Avg. Error |
|---|---|---|---|---|---|---|
| 50 | 0.13 | 1800 | 25 | 0.4x0.2 | 0.3% | 0.2% |
| 5 | 0.26 | 600 | 25 | 0.8x0.4 | 0.03% | 0.01% |
| 25 | 0.05 | 3600 | 50 | 0.4x0.2 | 0.06% | 0.03% |

## 4.2 Simple Rectangle Simulation Results

To allow for rapid testing and modifications, a small rectangular model was used for testing. When sliced, this model had 4 layers with 190 elements per layer. Each layer was comprised of three outer wall lines and was filled with a diagonal pattern that alternated direction with each layer. The final timestep in the thermal simulation can be seen in Figure 9 below.

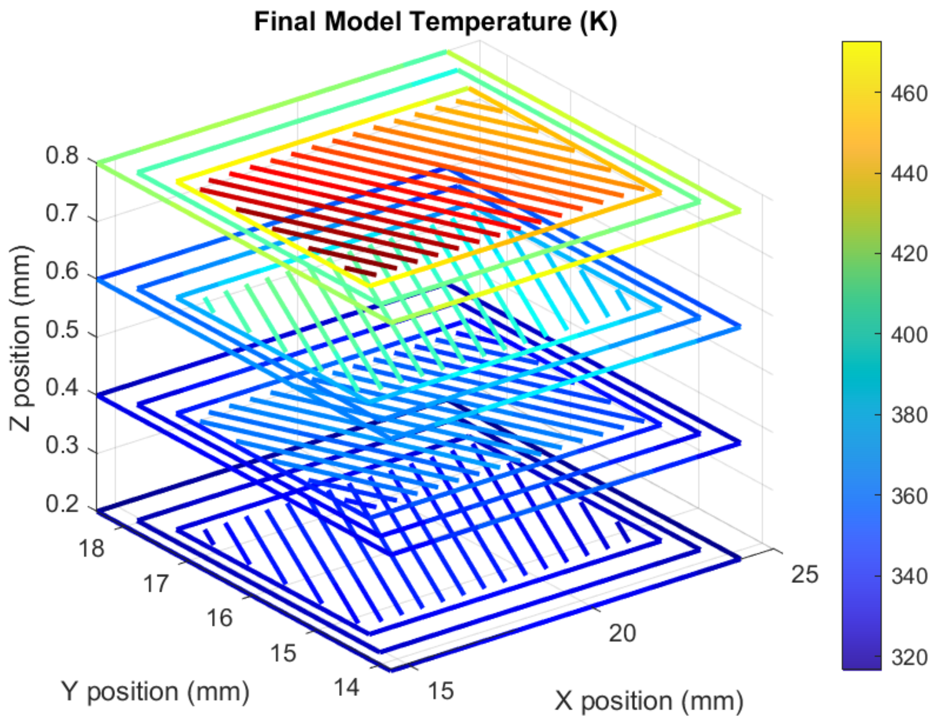


*Figure 9: Rectangular model with low heat transfer*

This model cooled very slowly due to the low heat transfer coefficient used – 5 $\frac{W}{mK}$. By increasing this coefficient to 50 $\frac{W}{mK}$, the model cools more rapidly as seen in Figure 10.

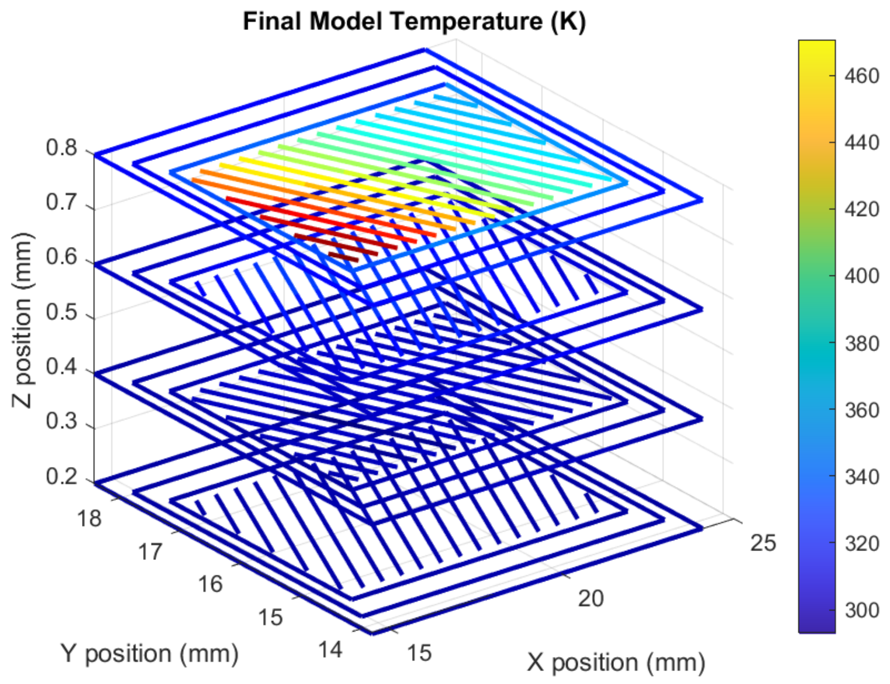


*Figure 10: Rectangular model with rapid heat transfer*

For the rapid heat transfer mode, the temperature of the elements initially followed the analytical trajectory, and then, as expected, they experienced slight reheating and recooling as neighboring elements were laid, Figure 11.

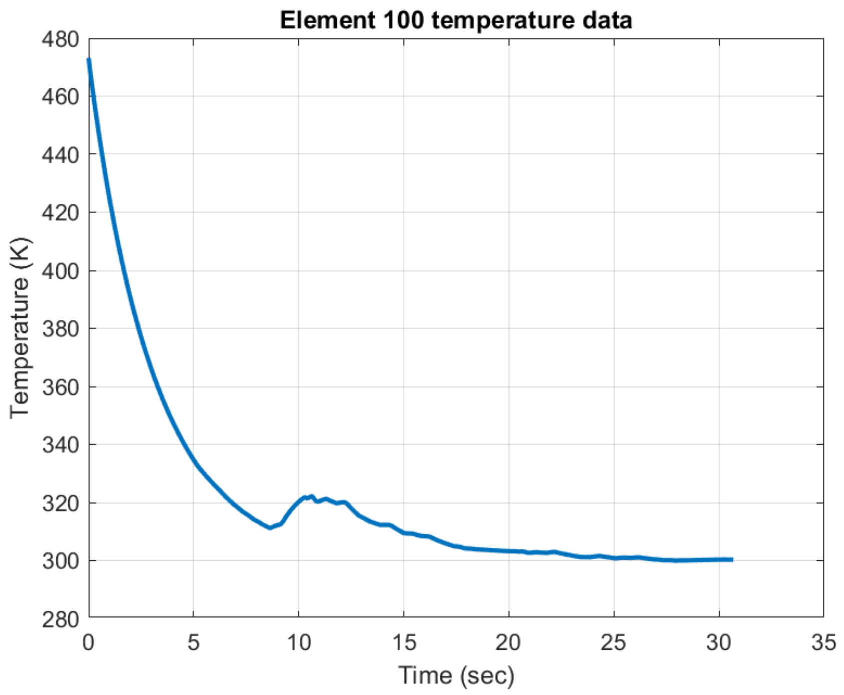


*Figure 11: Element 100 temperature with rapid heating and cooling*

This simulation data aligns closely with expectations; however, no empirical data is available to validate if the model can accurately predict a real world FDM process.

## 4.3 Physical Parameter Testing and Empirical Data Collection

While some of the thermal parameters used in the simulation are well documented in literature [14], the thermal contact coefficients and heat transfer coefficients used were not found in literature or from testing. Though these values do not impact the validity of the simulation model, it prevents useful engineering data from being determined from the results. To remedy this, future research should be conducted to determine empirical values for these parameters over a wide range of printing conditions. In particular, the anisotropic properties of thermoplastics used in FDM should be investigated. It is likely that heat transfer occurs more rapidly along the extrusion path than from layer to layer and line to line, however, the exact effects are unknown.

The heat transfer coefficient between the filament and the air is the dominating heat transfer mode for the outer wall as well as a significant mode for every element until the element is covered by the next layer. It is well documented that the heat transfer coefficient varies widely with air speed and humidity. Therefore, the use of a cooling fan and the specific composition of air surrounding the print volume are also important parameters for accurate simulation results.

## 4.4 Space-based Computational Approach

The elements in the simulation were stored in the order that they are extruded. This is effective computationally and programmatically for MATLAB; however, a space-based storage method may allow for more rapid simulations in other languages. A space-based method would store elements based on their location, allowing for rapid identification of nearby elements and element to element interactions. This approach would require storing the time information non-natively and would likely require more time to process the simulation step, however, the computational load is likely less because the space information is accessed once for every element per time step while the time information is only accessed once per timestep. Future work should consider this implementation method to further reduce computation times.

## 4.5 Expanded G-Code Capabilities and Slicer Integration

While this simulation method is capable of identifying poor print settings and points of possible failure, it is likely that most users forgo this step in favor of trial-and-error printing and their own experience with FDM printers. To address this shortcoming, this simulation approach should be integrated with a common slicer such as Cura. This integration would allow users to see the thermal simulation of their part as it is sliced, allowing for an informed choice to be made on settings such as the cooling fan, bed temperature, nozzle temperature, and print speed. Additionally, the software could recommend adding a larger brim or raft if warping is expected from the simulation data.

## Conclusion

As additive manufacturing becomes more prevalent, accurate simulation techniques must be developed as well. For LSAM, material costs can be thousands of dollars per part and prints can last days. Simulating the process before starting the print can be used to avoid critical failures and provide helpful information about possible areas with high thermal gradients. Traditional FEA is not capable of simulating the parts as built by the FDM machines in a reasonable amount of time.

Extracting the model directly from the G-Code allows the simulation to accurately account for the parts internal geometry as built. Additionally, creating elements directly from the G-Code allows for more rapid simulation without sacrificing accuracy. This numerical simulation technique accounts for the anisotropic properties of the as-built FDM part. By storing the top contact, bottom contact, side contact, and consecutive contact separately, variations in thermal contact resistance and thermal conductance can be included. These additional considerations require empirical data not yet available, but future works and research will likely fill these gaps.

The finite difference approximation used in the thermal simulation is valid for elements with Biot number less than 0.1. To satisfy this condition, a maximum element length should be computed and translated into a maximum allowable timestep based on the travel speed of the extruder. By comparing the simulation to a known analytical case, a simple accuracy check was performed. While the numerical model aligned closely with the analytical model and the complex simulation results matched expectations, without more accurate thermal parameters no engineering conclusions can be made from the data.

## References

[1] L. Dowling, J. Kennedy, S. O'Shaugnessy and D. Trimble, "A review of critical repeatability and reproducibility issues in powder bed fusion," *Materials & Design*, vol. 186, no. 15, 2020.

[2] S. Bhandari and R. Lopez-Anido, "Discrete-Event Simulation Thermal Model for Extrusion-Based Additive Manufacturing of PLA and ABS," *Additive Manufacturing Methods and Modeling Approaches*, 2020.

[3] Milwaukee School of Engineering, "History of 3D Printing: The Free Beginner’s Guide," 2013.

[4] J. Najmon, S. Raeisi and A. Tovar, Review of additive manufacturing technologies and applications in the aerospace industry, Cambridge: Elsevier, 2019.

[5] Pletsch, "Class Aves," 29 March 2017. [Online]. Available: https://blogs.ubc.ca/mrpletsch/2017/03/29/class-aves/.

[6] T. Murphy, "IAV Sees Huge Potential With 3D-Printed Pistons," 12 April 2018. [Online]. Available: https://www.wardsauto.com/engines/iav-sees-huge-potential-3d-printed-pistons.

[7] Martin, "Cura Wall Thickness & Line Count | How to Get Perfect Walls," The 3D Printer Bee, [Online]. Available: https://the3dprinterbee.com/cura-wall-thickness-line-count/.

[8] P. Nachiket, P. Deepankar and S. Brent, "A New Finite Element Solver using Numerical Eigen Modes for Fast Simulation of Additive Manufacturing Processes," *International Solid Freeform Fabrication Symposium*, 2013.

[9] P. Michaleris, "Modeling metal deposition in heat transfer analyses of additive manufacturing processes," *Finite Elements in Analysis and Design*, 2014.

[10] Y. Zhang and V. Shapiro, "Linear-Time Thermal Simulation of As-Manufactured Fused Deposition Modeling Components," *J. Manuf. Sci. Eng.*, 2018.

[11] Rosetta Code, "Sutherland-Hodgman polygon clipping," 2020. [Online]. Available: https://rosettacode.org/wiki/Sutherland-Hodgman_polygon_clipping#MATLAB_.2F_Octave.

[12] C. Bellehumuer, L. Li, Q. Sun and P. Gu, "Modeling of Bond Formation Between Polymer Filaments in the Fused Deposition Modeling Process," *Journal of Manufacturing Processes*, 2004.

[13] L. Pereira, "The Effects of 3D Printing P ects of 3D Printing Parameters and Sur ameters and Surface Treatments on eatments on," *Electronic Theses and Dissertations*, 2019.

[14] 3DPrinting.com, "Thermally Conductive Polymer Materials for 3D Printing," July 2020. [Online]. Available: https://3dprinting.com/3d-printing-use-cases/thermally-conductive-polymer-materials/#:~:text=PLA%20has%20a%20thermal%20conductivity,W%2F(m*K)..

# Appendix A: Code Listing

## A.1 G-Code Processing

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Noah Foster
% Process Gcode file
% Code a
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Clear and reset
clf;
clear;
clc;

% Select file to open
fid = fopen('Small_Test_Block_H02_W04.gcode','r');

% Initialize line variables
tlines = cell(0,1);
check = false;
% Loop until beginning of relavent Gcode
while check == false
    line = fgetl(fid);
    if length(line) >= 5
        if line(1:5) == ';MESH'
            check = true;
        end
    end
end

% Reset line variables
check = false;
tline = fgetl(fid);
n=1;
Codes = cell(0,1);

% Loop until end of relavent Gcode and store G0 and G1 lines in Codes cell
% array
while check == false
    if length(tline) > 0
        if tline(1) == 'G'
            Codes{end+1,1} = tline;
        end
    end
    tline = fgetl(fid);
    n=n+1;
    if length(tline) >= 4
        if tline(1:4) == ';End'
            check = true;
        end
    end
end

% Close Gcode file
fclose(fid);
%%

% Split code cells by spaces in between codes
CodeSplit = cellfun(@(x) strsplit(x, ' '), Codes, 'UniformOutput', false);

% Store split codes into cell with unifrom dimensions
CodeCells = cell(length(CodeSplit),6);
for i = 1:length(CodeSplit)
    CodeCells{i,1} = CodeSplit{i,1}{1,1};
    j_max = length(CodeSplit{i,1});
    j = 2;
    if CodeSplit{i,1}{1,j}(1) == 'F'
        CodeCells{i,2} = CodeSplit{i,1}{1,j};
        j = j+1;
    end

    if j > j_max
        continue;
    end

    if CodeSplit{i,1}{1,j}(1) == 'X'
        CodeCells{i,3} = CodeSplit{i,1}{1,j};
        j = j+1;
    end

    if j > j_max
        continue;
    end

    if CodeSplit{i,1}{1,j}(1) == 'Y'
        CodeCells{i,4} = CodeSplit{i,1}{1,j};
        j = j+1;
    end
```

```
        if j > j_max
            continue;
        end

        if CodeSplit{i,1}{1,j}(1) == 'Z'
            CodeCells{i,5} = CodeSplit{i,1}{1,j};
            j = j+1;
        end

        if j > j_max
            continue;
        end

        if CodeSplit{i,1}{1,j}(1) == 'E'
            CodeCells{i,6} = CodeSplit{i,1}{1,j};
            j = j+1;
        end
    end
```

## A.2 Element Chopping

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Build model from Gcode, MUST RUN GCODE_PROCESS_LIVE FIRST
    % Noah Foster
    % Code b
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Initialize potion and time vectors
    posE = 0;
    posX = str2double(CodeCells{1,3}(2:end));
    posY = str2double(CodeCells{1,4}(2:end));
    time = 0;
    nposX = posX;
    nposY = posY;
    nposE = 0;
    K = 273.15; % Conversion to Kelvin
    Temp = 200+K; % Initial filament temperature in Kelvin
    Layer = str2double(CodeCells{1,5}(2:end));
    %Initialize distance and speed vector
    d = zeros([length(CodeCells)-1,1]);
    Speed = zeros([length(CodeCells),1]);
    max_speed = 0;

    %Initialize results vector
    Elem = zeros([0,11,1]);
```

```
k = 1;
lay = 1;

% Calculate length of CodeCells
[code_length, ~] = size(CodeCells);
% for each Gcode command store distance, postion and speed data
for i = 2:code_length

    % Store F codes as Speed data
    if ~isempty(CodeCells{i,2})
        Speed(i) = str2double(CodeCells{i,2}(2:end))/60; %mm/s
    else
        Speed(i) = Speed(i-1);
    end

    % Store X Y and Z codes as positions
    if ~isempty(CodeCells{i,3})
        nposX = str2double(CodeCells{i,3}(2:end));
    end
    if ~isempty(CodeCells{i,4})
        nposY = str2double(CodeCells{i,4}(2:end));
    end
    if ~isempty(CodeCells{i,5})
        Layer(k+1,1) = str2double(CodeCells{i,5}(2:end));

        % Update to layer K and reset lay variable
        k = k+1;
        lay = 1;
    end

    % Update extruder position for E codes
    if ~isempty(CodeCells{i,6})
        nposE = str2double(CodeCells{i,6}(2:end));
    end

    % Calculate total distance travel for the line of Code
    d(i-1) = sqrt((nposX-posX)^2+(nposY-posY)^2);
    if d(i-1) == 0
        d(i-1) = abs(nposE-posE);
    end

    % Calculate time step based on distance/speed
    ntime = time + d(i-1)/Speed(i);
```

```
        % Store G1 codes with extrusion as elements
        if CodeCells{i,1}(1:2) == 'G1' & nposE-posE > 0
            % Update max extrusion speed
            if Speed(i) > max_speed
                max_speed = Speed(i);
            end
            % Split elements if timestep is shorter than 0.1 seconds
            if (ntime-time)*100 > 10
                split = ceil((ntime-time)*10);
                iposX = (nposX-posX)/split;
                iposY = (nposY-posY)/split;
                iposE = (nposE-posE)/split;
                itime = (ntime-time)/split;
                for j = 1:split
                    Elem(lay,:,k)                                                   =
[posX,posX+iposX,posY,posY+iposY,posE,posE+iposE,time,time+itime,d(i-
1)/split,Temp,Temp];
                    posX = posX+iposX;
                    posY = posY+iposY;
                    posE = posE+iposE;
                    time = time + itime;
                    lay = lay+1;
                end
            else
                Elem(lay,:,k)                                                       =
[posX,nposX,posY,nposY,posE,nposE,time,ntime,d(i-1),Temp,Temp];

                % Increase lay to move to next element in layer k
                lay = lay+1;
            end

        end
        % Reset pos and time variables for next loop iteration
        posX = nposX;
        posY = nposY;
        posE = nposE;
        time = ntime;

    end

    % Reset Layer to appropriate number of elements
    [l,w,h] = size(Elem);
    Layer = Layer(1:h);
```

```
% Uncomment to plot
% for k = 1:h
%     for i = 1:l
%         plot3(Elem(i,1:2,k),Elem(i,3:4,k),[Layer(k); Layer(k)]);
%         hold on;
%         grid on;
%     end
% end
```

## A.3 Contact Graph

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Noah Foster
% Contact Graph
% Code C
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Set Width and Height based on slicing parameters
W = 0.4; % Width in mm
H = 0.2; % H in mm

% Calculate size of Elem matrix
[l,w,h] = size(Elem);

% Create contact matrices
sidecontact = zeros(l,l,h);
topcontact = zeros(l,l,h);
bottomcontact = zeros(l,l,h);

% Create consecutive elements and non consecutive elements matrices
consec = zeros(l,l,h);
free = 2*ones(l,h);

% Calculate grid_divide with grid_divide function
[grid_x,grid_y,Elem_grid] = grid_divide(Elem,max_speed);

% Loop through all elements not in top layer
for k = 1:h-1
    for i = 1:l
        % Skip iteration if Elem row is empty
        if sum(Elem(i,:,k)) ~= 0
            % Calculate subject rectangle for side overlap
            subside = Box_Over(Elem(i,1:4,k),W);
```

```
               % Caclulate subject rectangle for top overlap
               sub = Box(Elem(i,1:4,k),W);
               % Loop through all other elements in row
               for j = 1:l
                   % Skip if side element is not in same or neighboring
                   % grid_divide
                   if   abs(Elem_grid(i,1,k)-Elem_grid(j,1,k))   <   2   &&
abs(Elem_grid(i,2,k)-Elem_grid(j,2,k)) < 2

                       % Skip if identical element
                       if i == j

                           % Skip if consectutive but store consecutive data
                       elseif Elem(i,2,k)  ==  Elem(j,1,k)  &&  Elem(i,4,k)  ==
Elem(j,3,k)

                           consec(i,j,k) = 1;
                           free(i,k) = free(i,k) - 1;

                       elseif Elem(i,1,k)  ==  Elem(j,2,k)  &&  Elem(i,3,k)  ==
Elem(j,4,k)

                           consec(i,j,k) = 1;
                           free(i,k) = free(i,k) - 1;
                           %  Else,  calculate  overlap  and  longest  side  and
store contact area
                       else

                           % Calculate clip rectangle
                           clip = Box(Elem(j,1:4,k),W);

                           % Calculate top overlap polygon
                           over = sutherlandHodgman(subside,clip);
                           % Skip if no overlap
                           if isempty(over)

                           else

                               sidecontact(i,j,k) = long_side(over)*H;
                               % Uncomment to visualize process
                               %              sidecontact(i,j,k)
                               %              figure(1)
                               %              daspect([1 1 1])
                               %                              plot([clip(:,1);
clip(1,1)],[clip(:,2); clip(1,2)]);
                               %              daspect([1 1 1])
```

```
                              %              hold on;
                              %                           plot([subside(:,1);
subside(1,1)],[subside(:,2); subside(1,2)]);
                              %              daspect([1 1 1])
                              %              hold on;
                              %                              plot([over(:,1);
over(1,1)],[over(:,2); over(1,2)],'LineWidth',2);
                              %              daspect([1 1 1])
                              %              pause(0.3);
                              %              hold off;
                          end
                      end
                  end

                  % Skip if top element is not in same or neighboring
                  % grid_divide
                  if   abs(Elem_grid(i,1,k)-Elem_grid(j,1,k+1))   <   2   &&
abs(Elem_grid(i,2,k)-Elem_grid(j,2,k+1)) < 2

                      % Calculate top overlap for layer k + 1
                      cliptop = Box(Elem(j,1:4,k+1),W);
                      overtop = sutherlandHodgman(sub,cliptop);

                      % Skip if no overlap
                      if isempty(overtop)

                          % Else, calculate overlap area
                      elseif polyarea(overtop(:,1),overtop(:,2)) > 0
                          topcontact(i,j,k)                                =
polyarea(overtop(:,1),overtop(:,2));
                          % Uncomment to visualize and plot
                          %              topcontact(i,j,k)
                          %              figure(1)
                          %              daspect([1 1 1])
                          %                                plot([cliptop(:,1);
cliptop(1,1)],[cliptop(:,2); cliptop(1,2)]);
                          %              daspect([1 1 1])
                          %              hold on;
                          %              plot([sub(:,1); sub(1,1)],[sub(:,2);
sub(1,2)]);
                          %              daspect([1 1 1])
                          %              hold on;
                          %                                plot([overtop(:,1);
overtop(1,1)],[overtop(:,2); overtop(1,2)],'LineWidth',2);
```

```
                            %             daspect([1 1 1])
                            %             pause(0.5);
                            %             hold off;
                        end
                    end
                end
            end
        end
        k
    end
    k = k+1

    if h == 1
        k = 1
    end
    % Repeat without top contact for last layer
    for i = 1:l
        if sum(Elem(i,:,k)) ~= 0
            subside = Box_Over(Elem(i,1:4,k),W);
            for j = 1:l
                if i == j
                    sidecontact(i,j,k) = 0;
                elseif Elem(i,2,k) == Elem(j,1,k) && Elem(i,4,k) == Elem(j,3,k)
                    consec(i,j,k) = 1;
                    free(i,k) = free(i,k) - 1;

                elseif Elem(i,1,k) == Elem(j,2,k) && Elem(i,3,k) == Elem(j,4,k)
                    consec(i,j,k) = 1;
                    free(i,k) = free(i,k) - 1;
                else
                    clip = Box(Elem(j,1:4,k),W);
                    over = sutherlandHodgman(subside,clip);
                    if isempty(over)

                    else
                        sidecontact(i,j,k) = long_side(over)*H;
                    end
                end
            end
        end
    end

    % Normalize data to account for slight over estimations of
    % contact area
    for k = 1:h
        for i = 1:l

            % Skip if row is empty or total area is less than available area
            if sum(Elem(i,:,k)) ~= 0
                % Skip if overlapping area is less than total area
                if sum(topcontact(i,:,k)) > W*Elem(i,9,k)
                    % Normalize based on actual top surface area
                    topcontact(i,:,k)                                  =
topcontact(i,:,k)*W*Elem(i,9,k)/sum(topcontact(i,:,k));
                end
                % Skip if overlapping area is less than total area
                if sum(sidecontact(i,:,k)) > (Elem(i,9,k)*H+free(i,k)*W*H)
                    % Normalize based on non-consectutive side surface area
                    sidecontact(i,:,k)                                 =
sidecontact(i,:,k)*(Elem(i,9,k)*H+free(i,k)*W*H)/sum(sidecontact(i,:,k));
                end
            end
        end
    end
    % Convert Contact to m^2 from mm^2
    sidecontact = sidecontact/1000^2;
    topcontact = topcontact/1000^2;

    % Set bottom contact based on top contact with shifted layers
    bottomcontact(:,:,2:h) = topcontact(:,:,1:h-1);
```

## A.4 Thermal Simulation

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Thermal Simulation using contact matrices
    % Noah Foster
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Calculate size of Elem
    [l,w,h] = size(Elem);

    % Define Editable Elem
    Elem_sim = zeros(l,w+1,h);
    Elem_sim(:,1:w,:) = Elem; % Store Elem data in new variable

    Elem_sim(:,9,:) = Elem_sim(:,9,:)/1000; %Distance converted to meters.
    W = 0.4/1000; % Width in m
    H = 0.2/1000; % Height in m
```

```
K = 273.15; % Conversion to Kelvin
h_bed = 5; %W/m2*K heat transfer coefficient to bed
h_elem = 5; %W/m2*K heat transfer coefficient to touching elements
h_air = 5; %W/m2*K heat transfer coefficient to air
lam = 0.13; %W/m*K thermal conductivity
T_air = 25+K; % K temperature of air
T_bed = 25+K; % K temperature of bed
p = 1300; % kg/m^3 density of PLA
c = 1800; % J/kg*K specific heat capacity of PLA
E = 0.9; % Emissivity of PLA

t = 0; % sec Current print time
o = 5.67*10^-8; % W/m2*K4 Boltzman constant



% Define temperature collection cell array
Elem_Temp_time = cell(l,h);

% Preallocate memory for stored temperature data
for k = 1:h
    for i = 1:l
        Elem_Temp_time{i,k} = NaN(2,(h-k)*(l)+(l+1-i));
        %Store temperature and time data
        Elem_sim(i,w+1,k) = Elem_sim(i,w+1,k) + 1;
        Elem_Temp_time{i,k}(1,1) = Elem_sim(i,7,k);
        Elem_Temp_time{i,k}(2,1) = Elem_sim(i,11,k);
    end
end

% Iteration count
iter = 1;
% Loop through all layers
for ko = 1:h
    ko
    % Loop through all elements in each layer
    for io = 1:l
        % Skip if element row is empty
        if sum(Elem_sim(io,1:11,ko)) ~= 0
            % Calculate simulation timestep
            dt = Elem_sim(io,8,ko)-t;

            % Update temperatures if extra time passed between elements
            if dt <= 0 || dt > 0.11
```

```
                    % If time between last element and next element > 0.01 sec
                    if abs(Elem_sim(io,7,ko)-t) > 0.01
                        % Calculate number of addition timesteps
                        split = ceil((Elem_sim(io,7,ko)-t)/0.1);
                        % Split into equal dt timesteps
                        dt =(Elem_sim(io,7,ko)-t)/split;
                        % Define active elements = to last simulation step
                        if io == 1
                            ksplit = ko - 1;
                            isplit = 1;
                        else
                            ksplit = ko;
                            isplit = io - 1;
                        end
                        % Run simulation update for each needed split
                        for split_time = 1:split
                            t = t + dt;
                            % Loop through all active layers
                            for k = 1:ksplit
                                % Check if on top layer
                                if k<ksplit
                                    icheck = 1;
                                else
                                    icheck = isplit;
                                end

                                % Loop through all elements if not on top layer
                                %  Loop  through  deposited  elements  if  on  top
layer

                                for i = 1:icheck

                                    % Skip if row is empty
                                    if sum(Elem_sim(i,1:11,k)) ~= 0
                                        grid = Elem_grid(isplit,1:2,ksplit);
                                        [Act]                               =
Active(t,grid,k,Elem_sim,Elem_grid);

                                        % Initialize Q_elem
                                        Q_elem = 0;

                                        % Reset free area to total surface area
                                        A_free                              =
2*(W*H+W*Elem_sim(i,9,k)+H*Elem_sim(i,9,k));
                                        % Conduction Heat Transfer
                                        % If forward element is consectutive
calculate Q
                                        if i ~= l && consec(i,i+1,k) == 1
                                            Q_cond_f                        =
lam*W*H*(Elem_sim(i,10,k)-
Elem_sim(i+1,10,k))/(0.5*(Elem_sim(i,9,k)+Elem_sim(i+1,9,k)));
                                            A_free = A_free - W*H;
                                        else
                                            Q_cond_f = 0;
                                        end

                                        % If previous element is consectutive
calculate Q
                                        if i ~= 1 && consec(i,i-1,k)
                                            Q_cond_b                        =
lam*W*H*(Elem_sim(i,10,k)-Elem_sim(i-
1,10,k))/(0.5*(Elem_sim(i,9,k)+Elem_sim(i-1,9,k)));
                                            A_free = A_free - W*H;
                                        else
                                            Q_cond_b = 0;
                                        end


                                        %  Platform  Heat  Convection  for  bed
layer
                                        if k == 1
                                            Q_bed                           =
h_bed*W*Elem_sim(i,9,k)*(Elem_sim(i,10,k)-T_bed);
                                            A_free      =       A_free      -
W*Elem_sim(i,9,k);
                                        else
                                            Q_bed = 0;
                                        end

                                        % Element to Element Convection
                                        % Loop through all elements
                                        for j = 1:l
                                            % Skip if element is inactive
                                            if Act(j,k) == 1
                                                %  If  active  calculate  side
convection
                                                Q_elem     =     Q_elem     +
h_elem*sidecontact(i,j,k)*(Elem_sim(i,10,k)-Elem_sim(j,10,k));
```

```
                                                A_free      =      A_free      -
sidecontact(i,j,k);
                                                if k < h && Act(j,k+1) == 1
                                                    %  If  active  and  not  top
layer
                                                    % Calculate top convection
                                                    Q_elem     =     Q_elem     +
h_elem*topcontact(i,j,k)*(Elem_sim(i,10,k)-Elem_sim(j,10,k+1));
                                                    A_free     =     A_free     -
topcontact(i,j,k);
                                                end
                                                if k ~= 1 && Act(j,k-1) == 1
                                                    % If active and not bottom
layer
                                                    %     Calculate     bottom
convection
                                                    Q_elem     =     Q_elem     +
h_elem*bottomcontact(i,j,k)*(Elem_sim(i,10,k)-Elem_sim(j,10,k-1));
                                                    A_free     =     A_free     -
bottomcontact(i,j,k);
                                                end
                                            end
                                        end


                                        % Air Convection and Radiation
                                        % Calculate based on non contact area
                                        Q_conv                                  =
h_air*A_free*(Elem_sim(i,10,k)-T_air);


                                        Q_rad                                   =
E*o*A_free*(Elem_sim(i,10,k)^4-T_air^4);

                                        %  %  Uncomment  if  running  analytical
comparison
                                        % Q_rad = 0;

                                        %  Calculate  new  temperature  based  on
conservation
                                        % of energy
                                        Elem_sim(i,11,k)  = Elem_sim(i,10,k)  -
dt*(Q_cond_f+Q_cond_b+Q_conv+Q_rad+Q_bed+Q_elem)/p/c/W/H/Elem_sim(i,9,k);
```

```
                                       % Throw error if temperature data is
NaN
                                       if isnan(Elem_sim(i,11,k))
                                           error('NaN');
                                       end

                                       if Elem_sim(i,11,k) < 0
                                           error('Temperature    less    than
Zero!');
                                       end
                                   end
                               end
                           end
                       end
                   else
                       error('Timestep Error')
                   end
               end

               % Update simulation time
               t = Elem_sim(io,8,ko);

               % Loop through all active layers
               for k = 1:ko
                   % Check if on top layer
                   if k<ko
                       icheck = l;
                   else
                       icheck = io;
                   end

                   % Loop through all elements if not on top layer
                   % Loop through deposited elements if on top layer
                   for i = 1:icheck

                       % Skip if row is empty
                       if sum(Elem_sim(i,1:11,k)) ~= 0
                           grid = Elem_grid(io,1:2,ko);
                           [Act] = Active(t,grid,k,Elem_sim,Elem_grid);

                           % Initialize Q_elem
                           Q_elem = 0;

                           % Reset free area to total surface area
                           A_free                                                   =
2*(W*H+W*Elem_sim(i,9,k)+H*Elem_sim(i,9,k));
                           % Conduction Heat Transfer
                           % If forward element is consectutive calculate Q
                           if i ~= l && consec(i,i+1,k) == 1
                               Q_cond_f      =      lam*W*H*(Elem_sim(i,10,k)-
Elem_sim(i+1,10,k))/(0.5*(Elem_sim(i,9,k)+Elem_sim(i+1,9,k)));
                               A_free = A_free - W*H;
                           else
                               Q_cond_f = 0;
                           end

                           % If previous element is consectutive calculate Q
                           if i ~= 1 && consec(i,i-1,k)
                               Q_cond_b      =      lam*W*H*(Elem_sim(i,10,k)-
Elem_sim(i-1,10,k))/(0.5*(Elem_sim(i,9,k)+Elem_sim(i-1,9,k)));
                               A_free = A_free - W*H;
                           else
                               Q_cond_b = 0;
                           end


                           % Platform Heat Convection for bed layer
                           if k == 1
                               Q_bed                                                =
h_bed*W*Elem_sim(i,9,k)*(Elem_sim(i,10,k)-T_bed);
                               A_free = A_free - W*Elem_sim(i,9,k);
                           else
                               Q_bed = 0;
                           end

                           % Element to Element Convection
                           % Loop through all elements
                           for j = 1:l
                               % Skip if element is inactive
                               if Act(j,k) == 1
                                   % If active calculate side convection
                                   Q_elem          =          Q_elem          +
h_elem*sidecontact(i,j,k)*(Elem_sim(i,10,k)-Elem_sim(j,10,k));
                                   A_free = A_free - sidecontact(i,j,k);
                                   if k < h && Act(j,k+1) == 1
                                       % If active and not top layer
                                       % Calculate top convection
```

```
                                        Q_elem        =        Q_elem        +
h_elem*topcontact(i,j,k)*(Elem_sim(i,10,k)-Elem_sim(j,10,k+1));
                                        A_free = A_free - topcontact(i,j,k);
                                    end
                                    if k ~= 1 && Act(j,k-1) == 1
                                        % If active and not bottom layer
                                        % Calculate bottom convection
                                        Q_elem        =        Q_elem        +
h_elem*bottomcontact(i,j,k)*(Elem_sim(i,10,k)-Elem_sim(j,10,k-1));
                                        A_free        =        A_free        -
bottomcontact(i,j,k);
                                    end
                                end
                            end


                            % Air Convection and Radiation
                            % Calculate based on non contact area
                            Q_conv = h_air*A_free*(Elem_sim(i,10,k)-T_air);


                            Q_rad = E*o*A_free*(Elem_sim(i,10,k)^4-T_air^4);

                            % % Uncomment if running analytical comparison
                            % Q_rad = 0;

                            % Calculate new temperature based on conservation
                            % of energy
                            Elem_sim(i,11,k)      =      Elem_sim(i,10,k)      -
dt*(Q_cond_f+Q_cond_b+Q_conv+Q_rad+Q_bed+Q_elem)/p/c/W/H/Elem_sim(i,9,k);

                            % Throw error if temperature data is NaN
                            if isnan(Elem_sim(i,11,k))
                                error('NaN');
                            end

                            if Elem_sim(i,11,k) < 0
                                error('Temperature less than Zero!');
                            end

                            %Store temperature and time data
                            Elem_sim(i,w+1,k) = Elem_sim(i,w+1,k) + 1;
                            iter = Elem_sim(i,w+1,k);
                            Elem_Temp_time{i,k}(1,iter) = t;
                            Elem_Temp_time{i,k}(2,iter) = Elem_sim(i,11,k);

                        end
                    end
                end
            end
            % Update old temperature to new temperatue
            Elem_sim(:,10,:) = Elem_sim(:,11,:);
        end
    end
```

## A.5 Results Graphing

```
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Noah Foster
   % Plot Thermal temp
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Store Elem_sim data
   Elem_sim(Elem_sim == 0) = NaN;
   Elem_Plot = Elem_sim;

   % Store normalized temperature data as nearest interger
   ca = [min(min(Elem_Plot(:,11,:))) max(max(Elem_Plot(:,11,:)))]; % Saved for
color axis
   Elem_Plot(:,11,:)              =              1+floor((Elem_Plot(:,11,:)-
min(min(Elem_Plot(:,11,:))))/(max(max(Elem_Plot(:,11,:)))-
min(min(Elem_Plot(:,11,:))))*254);
   [l,w,h] = size(Elem_Plot);

   % Store jet color map
   clr = jet;

   figure(1)
   % Plot all elements that are not empty
   for k = 1:h
       for i = 1:l
           if isnan(Elem_sim(i,1:11,k))
```

```
            else
                plot3(Elem_Plot(i,1:2,k),Elem_Plot(i,3:4,k),[Layer(k);
Layer(k)],'Color', clr(Elem_Plot(i,11,k),:),'LineWidth',2);
            end
            hold on;
            grid on;
        end
    end

    colorbar;
    caxis(ca);
    title('Final Model Temperature (K)')
    xlabel('X position (mm)')
    ylabel('Y position (mm)')
    zlabel('Z position (mm)')

    % Plot Node Temperature with adjusted time
    figure(2)
    node = 100;
    grid on;
    plot((Elem_Temp_time{node,1}(1,:)-
Elem_Temp_time{node,1}(1,1)),Elem_Temp_time{node,1}(2,:),'LineWidth',2)
    title('Element 100 temperature data')
    xlabel('Time (sec)')
    ylabel('Temperature (K)')

    % % Uncomment if running analytical comparison

    % % Validation test graph
    % v = max_speed/1000; % Velocity in meters per second
    % P = (2*W+2*H); % Perimeter in meters
    % A = W*H ; % Cross-sectional area
    % a = lam/p/c/v; % alpha parameter
    % b = h_air*P/p/c/A/v; % beta parameter
    % m = (sqrt(1+4*a*b)-1)/2/a; % m parameter
    % time = 0:0.1:10; % Time in sec
    % analytical_Temp = T_air + (Temp-T_air)*exp(-m*v*time);
    %
    % hold on;
    % plot(time,analytical_Temp,'.','LineWidth',3);
    % grid on;
    % legend('Simulation Temperature','Analytical Temperature')
    %
    % % Calculate percentage error
```

```
   %     [max_dif     pos]     =     max(abs(Elem_Temp_time{node,1}(2,1:100)-
analytical_Temp(1:100)));
   % max_perc_error = max_dif/analytical_Temp(pos)*100
   % time_of_error = time(pos)
   %       ave_error       =       sum(abs(Elem_Temp_time{node,1}(2,1:100)-
analytical_Temp(1:100))./analytical_Temp(1:100))
```

## A.6 Supplemental Scripts

### Grid Division Function

```
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % XY grid_divide
   % Noah Foster
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Function to split model elements into XY grid
   function [grid_x,grid_y,Elem_grid] = grid_divide(Elem,max_speed)
   [l,~,h] = size(Elem); % Calculate the size of Elem
   max_travel = max_speed*0.1; % Maximum travel distance in 0.1s in mm
   NaN_Elem = Elem; % Store Elem data in editable matrix
   NaN_Elem(NaN_Elem == 0) = NaN; % Change all zeros to NaNs
   % Calculate max and min X and Y values
   xmin = min(min(NaN_Elem(:,1,:)));
   xmax = max(max(NaN_Elem(:,1,:)));
   ymin = min(min(NaN_Elem(:,3,:)));
   ymax = max(max(NaN_Elem(:,3,:)));
   xlen = xmax-xmin; % Calculate X length
   ylen = ymax-ymin; % Calculate Y length
   % Calculate number of allowable splits such that no interval is smaller
than max_travel
   split_grid_x = floor(xlen/max_travel)-1;
   split_grid_y = floor(ylen/max_travel)-1;
   % Define X and Y grid boundaries
   grid_x = linspace(xmin,xmax,2+split_grid_x);
   grid_y = linspace(ymin,ymax,2+split_grid_y);

   % Initialize Elem_grid matrix
   Elem_grid = ones(l,2,h);

   for k = 1:h % Loop through layers
```

```
        for i = 1:l % Loop through elements
            j = 1; % Reset j
            while j <= length(grid_x) % loop through X grid
                % If the X coordinate of the midway point is less than the
                % boundary put it in that grid square
                if (Elem(i,1,k)+Elem(i,2,k))/2 < grid_x(j)
                    Elem_grid(i,1,k) = j;
                    j = length(grid_x);
                end
                j = j+1;
            end
            j = 1;
            while j <= length(grid_y) % loop through Y grid
                % If the Y coordinate of the midway point is less than the
                % boundary put it in that grid square
                if (Elem(i,3,k)+Elem(i,4,k))/2 < grid_y(j)
                    Elem_grid(i,2,k) = j;
                    j = length(grid_y);
                end
                j = j+1;
            end
        end
    end
    end
```

### Active Element Function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Active Function
% Noah Foster
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Function to determine which elements are thermally active in the
% simulation
function [Act] = Active(time,grid,layer,Elem,Elem_grid)
[l,~,h] = size(Elem); % Calculate size of Elem
Act = ones(l,h); % Initialize Act matrix
inactive_time = 8; % Amount of time until 99% of heat has dissapated in sec
for k = 1:h % Loop through layers
    for i = 1:l % Loop through elements
        if sum(Elem(i,:,k)) ~= 0 % If row is not empty
            if time < Elem(i,8,k)
                Act(i,k) = 0; % Set inactive if element has not been layed
                elseif time - Elem(i,8,k) > inactive_time && abs(layer-k)>2
                    Act(i,k) = 0; % Set inactive if time has passed and element
is greater than 2 layers away
                elseif time - Elem(i,8,k) > inactive_time && (abs(grid(1)-
Elem_grid(i,1,k))>2 || abs(grid(2)-Elem_grid(i,2,k))>2)
                    Act(i,k) = 0; % Set inactive if time has passed and element
is more than 2 grid squares away in X or Y direction
                end
            end
        end
   end
   end
```

### Box and Box_over functions

```
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   % Box function
   % Noah Foster
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   function [XY] = Box(AB,W)
   A = [AB(1),AB(3)];
   B = [AB(2),AB(4)];
   q = atan2(B(2)-A(2),B(1)-A(1));
   C = [0,0];
   C(1) = A(1)+sin(q)*W/2;
   C(2) = A(2)-cos(q)*W/2;
   D = [0,0];
   D(1) = A(1)-sin(q)*W/2;
   D(2) = A(2)+cos(q)*W/2;
   E = [0,0];
   E(1) = B(1)+sin(q)*W/2;
   C(2) = A(2)-cos(q)*W/2;
   E(2) = B(2)-cos(q)*W/2;
   F = [0,0];
   F(1) = B(1)-sin(q)*W/2;
   F(2) = B(2)+cos(q)*W/2;
   X = [D(1),C(1),E(1),F(1)]';
   Y = [D(2),C(2),E(2),F(2)]';
   XY = [X,Y];
   end
```

```
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Box_over function
% Noah Foster
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [XY] = Box(AB,W)
W = W + 0.02;
if AB(1) > AB(2)
    AB(1) = AB(1)+0.01;
    AB(2) = AB(2)-0.01;
elseif AB(2) > AB(1)
    AB(1) = AB(1)-0.01;
    AB(2) = AB(2)+0.01;
end
if AB(3) > AB(4)
    AB(3) = AB(3)+0.01;
    AB(4) = AB(4)-0.01;
elseif AB(4) > AB(3)
    AB(3) = AB(3)-0.01;
    AB(4) = AB(4)+0.01;
end
A = [AB(1),AB(3)];
B = [AB(2),AB(4)];
q = atan2(B(2)-A(2),B(1)-A(1));
C = [0,0];
C(1) = A(1)+sin(q)*W/2;
C(2) = A(2)-cos(q)*W/2;
D = [0,0];
D(1) = A(1)-sin(q)*W/2;
D(2) = A(2)+cos(q)*W/2;
E = [0,0];
E(1) = B(1)+sin(q)*W/2;
C(2) = A(2)-cos(q)*W/2;
E(2) = B(2)-cos(q)*W/2;
F = [0,0];
F(1) = B(1)-sin(q)*W/2;
F(2) = B(2)+cos(q)*W/2;
X = [D(1),C(1),E(1),F(1)]';
Y = [D(2),C(2),E(2),F(2)]';
XY = [X,Y];
end
```

### Long side function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Noah Foster
% Longest side Function
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function len = long_side(poly)
polyX = poly(:,1);
polyY = poly(:,2);
[r, ~] = size(poly);
d = 0;
for i = 1:r-1
    dnew = sqrt((polyX(i)-polyX(i+1))^2+(polyY(i)-polyY(i+1))^2);
    if dnew > d
        len = dnew;
        d = dnew;
    end
end
    dnew = sqrt((polyX(1)-polyX(i+1))^2+(polyY(1)-polyY(i+1))^2);
    if dnew > d
        len = dnew;
    end
```

### Sutherland-Hodgman Algorithm [11]

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PolyClip
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The inputs are a table of x-y pairs for the verticies of the subject
%polygon and boundary polygon. (x values in column 1 and y values in column
%2) The output is a table of x-y pairs for the clipped version of the
%subject polygon.
function clippedPolygon = sutherlandHodgman(subjectPolygon,clipPolygon)

%% Helper Functions

    %computerIntersection() assumes the two lines intersect
    function intersection = computeIntersection(line1,line2)

        %this is an implementation of
        %http://en.wikipedia.org/wiki/Line-line_intersection

        intersection = zeros(1,2);

        detL1 = det(line1);
        detL2 = det(line2);

        detL1x = det([line1(:,1),[1;1]]);
        detL1y = det([line1(:,2),[1;1]]);
        detL2x = det([line2(:,1),[1;1]]);
        detL2y = det([line2(:,2),[1;1]]);

        denominator = det([detL1x detL1y;detL2x detL2y]);

        intersection(1) = det([detL1 detL1x;detL2 detL2x]) / denominator;
        intersection(2) = det([detL1 detL1y;detL2 detL2y]) / denominator;

    end %computeIntersection

    %inside() assumes the boundary is oriented counter-clockwise
    function in = inside(point,boundary)

        pointPositionVector = [diff([point;boundary(1,:)]) 0];
        boundaryVector = [diff(boundary) 0];
        crossVector = cross(pointPositionVector,boundaryVector);

        if ( crossVector(3) <= 0 )
            in = true;
        else
            in = false;
        end

    end %inside

%% Sutherland-Hodgman Algorithm

    clippedPolygon = subjectPolygon;
    numVerticies = size(clipPolygon,1);
    clipVertexPrevious = clipPolygon(end,:);

    for clipVertex = (1:numVerticies)

        clipBoundary = [clipPolygon(clipVertex,:) ; clipVertexPrevious];

        inputList = clippedPolygon;

        clippedPolygon = [];
        if ~isempty(inputList),
            previousVertex = inputList(end,:);
        end

        for subjectVertex = (1:size(inputList,1))
```

```
               if ( inside(inputList(subjectVertex,:),clipBoundary) )

                   if( not(inside(previousVertex,clipBoundary)) )
                       subjectLineSegment                                 =
[previousVertex;inputList(subjectVertex,:)];
                       clippedPolygon(end+1,1:2)                          =
computeIntersection(clipBoundary,subjectLineSegment);
                   end

                   clippedPolygon(end+1,1:2) = inputList(subjectVertex,:);

               elseif( inside(previousVertex,clipBoundary) )
                       subjectLineSegment                                 =
[previousVertex;inputList(subjectVertex,:)];
                       clippedPolygon(end+1,1:2)                          =
computeIntersection(clipBoundary,subjectLineSegment);
               end

               previousVertex = inputList(subjectVertex,:);
               clipVertexPrevious = clipPolygon(clipVertex,:);

           end %for subject verticies
       end %for boundary verticies
   end %sutherlandHodgman
```

# Appendix B: Variables

Variables used in Equations listed in order of appearance.

| Symbol | Units | Description |
|---|---|---|
| $Q^{rad}$ | W | Radiation to surroundings |
| t | sec | Simulation time |
| $\varepsilon$ | ~ | Emissivity of PLA |
| $\sigma$ | $\frac{W}{m^2K^4}$ | Stefan-Boltzmann constant |
| $A^{free}$ | $m^2$ | Area of element open to surroundings |
| T | K | Temperature of element |
| $T_\infty$ | K | Temperature of surrounding air |
| $Q^{conv}$ | W | Convection to surroundings |
| $h_{conv}$ | $\frac{W}{m^2K}$ | Surrounding air convection coefficient |
| $Q^{cond}$ | W | Conduction to consecutive elements |
| $\lambda$ | $\frac{W}{mK}$ | Thermal conductivity of PLA |
| $A_{i,i-1}$ | $m^2$ | Cross sectional area of road element |
| L | m | Length of road element |
| $Q^{elem}$ | W | Conduction to neighboring elements |
| $h_r$ | $\frac{W}{m^2K}$ | Thermal contact conductance coefficient between elements |
| $A^c_{i,j}$ | $m^2$ | Contact area between neighboring elements |

| | | |
|---|---|---|
| $Q^{bed}$ | W | Conduction to heated bed |
| $h_c$ | $\frac{W}{m^2K}$ | Thermal contact coefficient between elements and bed |
| $A^{bed}$ | $m^2$ | Contact area between elements and bed |
| $T_{bed}$ | K | Temperature of heated bed |
| $\rho$ | $\frac{kg}{m^3}$ | Density of PLA |
| c | $\frac{J}{kgK}$ | Specific heat capacity of PLA |
| V | $m^3$ | Volume of road element |
| $A_{cross}$ | $m^2$ | Cross sectional area |
| $P_{cross}$ | m | Perimeter of cross section |
| $T_0$ | K | Deposition temperature of filament |