LOW-COST OPEN-SOURCE GMAW-BASED METAL 3-D PRINTING: MONITORING, SLICER, OPTIMIZATION, AND APPLICATIONS

Yuenyong Nilsiam
Michigan Technological University, ynilsiam@mtu.edu

LOW-COST OPEN-SOURCE GMAW-BASED METAL 3-D PRINTING: MONITORING, SLICER, OPTIMIZATION, AND APPLICATIONS

By

Yuenyong Nilsiam

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Computer Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2017

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computer Engineering.

Department of Electrical and Computer Engineering

Dissertation Advisor:	*Dr. Joshua M Pearce*
Committee Member:	*Dr. Timothy C Havens*
Committee Member:	*Dr. John L Irwin*
Committee Member:	*Dr. Chee-Wooi Ten*
Department Chair:	*Dr. Daniel R Fuhrmann*

Table of Contents

List of Figures

List of Tables

Preface

This dissertation contains published, submitted, or work completed by the author of this dissertation. The contributions of the author are detailed in the following paragraphs.

Chapter 2: Yuenyong Nilsiam, Amberlee Haselhuhn, Bas Wijnen, Paul Sanders, and Joshua M. Pearce. Integrated Voltage—Current Monitoring and Control of Gas Metal Arc Weld Magnetic Ball-Jointed Open Source 3-D Printer. Machines 3, no. 4 (2015): 339-351. Y.N. wrote the algorithm, helped with data analysis and took the lead on writing, A.H. performed the metal printing and analysis, B.W. wrote the firmware and assisted with experiments, P.S. and J.M.P. formulated the project and assisted on the analysis. All authors co-wrote and edited the manuscript.

Chapter 3: Yuenyong Nilsiam, Paul Sanders, and Joshua M. Pearce. Slicer and Optimization for Open-source GMAW-based Metal 3-D Printing (to be published). Y.N. customized the software, performed the metal 3-D printing, and optimized the settings. P.S. and J.M.P. formulated the project and assisted on the analysis. All authors co-wrote and edited the manuscript.

Chapter 4: Yuenyong Nilsiam, Paul Sanders, and Joshua M. Pearce. Applications of Open Source GMAW-based Metal 3-D Printing (to be published). Y.N. designed or customized the 3-D models, performed steel 3-D printing, and machining. P.S. and J.M.P. formulated the project and assisted on the analysis. All authors co-wrote and edited the manuscript.

Acknowledgement

I would like to express my special thanks to my advisor, Dr. Joshua M. Pearce for his inspiration, encouragement, support, trust, patience, and guidance throughout the research. I am really thankful for all of his time and effort he had put into being my advisor.

I would like to thank Dr. Timothy C. Havens, Dr. John L. Irwin, and Dr. Chee-Wooi Ten for serving as my committee members.

I would also like to thank Dr. Paul G. Sanders who provided supports and suggestions during the research and the helps for review the papers.

I am grateful for my family and my wife who is everything that I need. Without her, I would not be able to come this far.

I would also like to express my gratitude and appreciation to Dr. Amberlee Haselhuhn, Dr. Bas Wijnen, and Gerald Anzalone for their kindly helps and supports. Also, I would like to thank all the members of the Michigan Tech's Open Sustainability Technology lab for their friendship and supports.

I am also thankful for the financial support from Royal Thai Scholarship and the John Wesley James Jones Memorial Scholarship.

Finally, I would like to thank God and my brothers and sisters at Evangel Baptist Church for their prayers and supports.

Abstract

Low-cost and open-source gas metal arc welding (GMAW)-based 3-D printing has been demonstrated yet the electrical design and software was not developed enough to enable wide-spread adoption. This thesis provides three novel technical improvements based on the application of mechatronic and software theory that when combined demonstrate the ability for distributed digital manufacturing at the small and medium enterprise scale of steel and aluminum parts. First, low cost metal inert gas welders contain no power monitoring needed to tune GMAW 3-D printers. To obtain this data about power and energy usage during the printing, an integrated monitoring system was developed to measure current (I) and voltage (V) in real-time. The new design of this monitoring system integrates an open source microcontroller and free and open source software on the open-source metal 3-D printer to record the data. Second, the primary obstacle to the diffusion of this technology was that existing slicing software, which determines the toolpath of the printhead was optimized for polymer 3-D printing and inappropriate for printed parts made from metal due to their mechanical strength. Previous prints were accomplished by manually designing the toolpath, which was not practical for real use by an extended userbase. To overcome the problem, the free and open-source slicing software, CuraEngine, was forked to MOSTMetalCura, which supports the needs of GMAW-based metal 3-D printing. The optimized setting for wire feed rate is calculated by the new slicer based on printing speed, bead width, layer height, and material diameter. Previous studies have shown that GMAW-based metal 3-D printing is capable of fabricating parts with good layer adhesion and porosity. However, this preliminary work lacked demonstrations of real-world applications. Finally, in this work, the practical applications of open-source GMAW-based metal 3-D printing are well demonstrated for both developing world and developed world applications including: 1) fixing an existing part by adding on a 3-D metal feature, 2) creating a product using the substrate as part of the component, 3) 3-D printing useful objects in high resolution, 4) near net shape objects and 5) making an integrated product using a combination of steel and polymer 3-D printing. The results prove that low-cost and open-source GMAW-based metal 3-D printing is ready for distributed manufacturing by SMEs and adequate for a wide range of applications.

Chapter 1: Introduction

1.1 Motivation

The targets of this research were to improve and optimize the process of the low-cost open-source gas metal arc welding (GMAW) based metal 3-D printing (Anzalone, et al. 2013) and to demonstrate its practical applications. To gain more insightful understanding of the metal 3-D printing process, a system that can measure current (I) and voltage (V) of the welder in real-time during the printing process was needed. The previous successful printing of the low-cost GMAW-based metal 3-D printing was based on hand-writing G-code which is not practical for wider adoption of the technology. Therefore, software was needed to convert a 3-D model into G-code or so called a slicer was vital for the success of this type of additive manufacturing. There was also a need for an optimization of settings for the slicer and the welder including printing speed, bead width, layer height, filament diameter, voltage, and wire feed speed in order to obtain useful parts. Many researchers have investigated material and mechanical properties of GMAW 3-D printing, however; they did not demonstrate significant real applications. Accordingly, this work utilized a new slicer and optimization functions to fabricate useful products with GMAW-based metal 3-D printing.

1.2 Dissertation Outline

The accomplished research will be in three main chapters (Chapters 2 to 4). First, Chapter 2 describes an open-source integrated system to record real-time current and voltage of the welder used for the GMAW-based metal 3-D printer during the printing process and its data analysis based on each selected alloy and layer number. Second, Chapter 3 presents software for slicing 3-D model and optimization settings for open-source GMAW-based metal 3-D printing. A customized version of the CuraEngine (Ultimaker 2017) named MOSTMetalCura (MOST Metal Cura 2017) and its usage is explained in this Chapter. The necessary settings for the slicer and the welder is also discussed here. Last, Chapter 4, then demonstrates practical applications of the GMAW-based metal 3-D printing and discusses the potential of the technology in the context of distributed manufacturing. Finally, Chapter

5 draws conclusions on all of the components of this thesis and provides guidance for future work.

1.3 References

Anzalone, Gerald C, Chenlong Zhang, Bas Wijnen, Paul G Sanders, and Joshua M Pearce. 2013. “A Low-Cost Open-Source Metal 3-D Printer.” *IEEE Access* 1: 803–10. doi:10.1109/ACCESS.2013.2293018.

Ultimaker. "Ultimaker/CuraEngine." GitHub. March 15, 2017. Accessed March 16, 2017. https://github.com/Ultimaker/CuraEngine.

"MOST Metal Cura." MOST Metal Cura - Appropedia: The sustainability wiki. Accessed March 16, 2017. http://www.appropedia.org/MOST_Metal_Cura.

Chapter 2: Integrated Voltage—Current Monitoring and Control of Gas Metal Arc Weld Magnetic Ball-Jointed Open Source 3-D Printer[1]

2.1 Abstract

To provide process optimization of metal fabricating self-replicating rapid prototyper (RepRap) 3-D printers requires a low-cost sensor and data logger system to measure current (I) and voltage (V) of the gas metal arc welders (GMAW). This paper builds on previous open-source hardware development to provide a real-time measurement of welder I-V where the measuring circuit is connected to two analog inputs of the Arduino that is used to control the 3-D printer itself. Franklin firmware accessed through a web interface that is used to control the printer allows storing the measured values and downloading those stored readings to the user's computer. To test this custom current and voltage monitoring device this study reports on its use on an upgraded all metal RepRap during the printing of aluminum alloy (ER1100, ER4043, ER4943, ER4047, and ER5356). The voltage and current data were analyzed on a per alloy basis and also layer-by-layer in order to evaluate the device's efficacy as a monitoring device for 3-D printing and the results of the integrated design are discussed.

2.2 Introduction

There has been a sustained technological development in the global community of makers of low-cost self-replicating rapid prototypers, which started with polymer 3-D printers that could fabricate approximately half of their components [1–3]. Today these RepRap platforms have evolved to machines capable of manufacturing using subtractive [4,5] as well as additive methods in a wide variety of polymers [5–7], composites [8], ceramics [9], and metals [10–15]. Of perhaps the most widespread interest in industry is the potential for a low-cost metal 3-D printer capable of printing both steel [10] and aluminum parts [11].

[1] This chapter has been published as an article in *Machines*. Citation: Nilsiam Y, Haselhuhn A, Wijnen B, Sanders P, & Pearce J (2015). Integrated Voltage—Current Monitoring and Control of Gas Metal Arc Weld Magnetic Ball-Jointed Open Source 3-D Printer. *Machines*, 3(4):339–351. Available online 3 November 2015 http://dx.doi.org/10.3390/machines3040339.

These open source metal 3-D printers can be fabricated for as little as $1200 [10] using a conventional metal inert gas (MIG) welder and controlled with open-source Arduino electronic boards [16], which effectively cuts the costs of metal 3-D printing by two orders of magnitude and make the technology far more accessible for a wide range of applications, perhaps even those in the developing world [17–19]. The low-cost consumer-grade MIG welders used for RepRap 3-D metal printing contain minimal controls. To provide process optimization of these RepRap 3-D printers requires a low-cost sensor and data logger system to measure current and voltage of the gas metal arc welders (GMAW) and previous work has developed an open source method for real-time measurement of welder voltage or current at the expense of adding another Arduino microcontroller to the system [20]. This data is critical for both gaining a fundamental understanding of the material processing technique in order to begin to optimize deposition predictively, but also in process monitoring is important for enabling feedback control and error detection.

In this study this extra cost is overcome as a new design is provided where the measurement of the current and voltage is done by attaching a measuring circuit to two analog inputs of the Arduino that is used to control the 3-D printer. The Franklin firmware, detailed extensively in Wijnen et al. [21] continuously sends the data that it reads from those pins to the host computer, which converts the raw ADC readings into voltage and current, and allows storing them on the file system. The web interface that is used to control the printer allows downloading those stored readings to the user's computer. To test this custom current and voltage monitoring device this study reports on its use on an upgraded all metal RepRap during the printing of aluminum alloy mechanical test specimens [22]. Common aluminum weld alloys include ER1100, ER4043, ER4047, and ER5356 (Table 2.1). ER4943 is a new welding alloy that was designed to eliminate the need for chemical dilution required for traditional weld alloys in order to obtain a quality weld [23]. Since ER4943 does not require chemical dilution, it may serve as an ideal 3-D printing alloy. Voltage and current were monitored during the printing of ER1100, ER4043, ER4943, ER4047, and ER5356. The voltage and current data were analyzed to provide monitoring on a per alloy basis and also layer-by-layer for 3-D printing metal process and property optimization.

Table 2.1 Aluminum Weld Alloys and their Major Alloying Elements [23,24].

Alloy	**Main Alloying Element**
ER1100	None; ≥99% Aluminum
ER4043	4.5%–6% Silicon
ER4943	5%–6% Silicon + 0.1%–0.5% Magnesium
ER4047	11%–13% Silicon
ER5356	4.5%–5.5% Magnesium

2.3 Experimental Section

A low-cost, open-source, metal 3-D printer and an open-source software tool chain were used to print all test specimens. This metal 3-D printer utilized GMAW technology to weld aluminum parts 3-dimensionally. A Miller Spoolmate 100 weld gun supplied the feedstock material which was melted by a Millermatic 190 GMAW. The 3-D printer design described by Anzalone, et al., [10] and Haselhuhn, et al. [11] was further refined to the new machine design (Figure 2.1). It was originally inspired by a Rostock self-replicating rapid - prototyper (RepRap) but was modified such that the weld gun print head remained stationary while the print substrate build plate moved on a 3-axis stage [25]. Both the last version [11] and this all-metal device have 304 mm long, 8 mm diameter guide rods on a 340 mm diameter circle. Following the Open Source Hardware Associations definition of open hardware [26], the bill of materials and the open source blue prints for the magnetic bearing-based 3-D printer are available in the Open Science Framework [27]. As can be seen in Figure 1, the open-source controller and relay board are mounted to a leg with polymer RepRap 3-D printed parts that electrically isolate the electronics from the frame to minimize the potential of damaging electronics should the frame become electrified during GMAW printing.

This original design has been further developed with the replacement of mechanical rotary bearings with magnetic bearings to allow for an increased range of motion, smoother motion, and a larger build volume (Figure 2.2). The range of motion in the x–y plane is approximately 26 cm in each direction, 10 cm more than the previous version of the robot with conventional tie rod ends. Motion in the z-direction is roughly equivalent between the two machines at 76 mm. The modification also reduced backlash, but highlighted other potential deficiencies in the design. The most notable of which is temporarily moving away from true RepRap potential until the low-cost metal printing precision is improved. Thus, this device should be viewed as a research 3-D printer, which in the future can be converted back to a true RepRap.

Each magnetic ball joint consists of a 19.05 mm (3/4") G25 chrome plated steel ball bearing, a 19.05 mm (3/4") diameter × 12.7 mm (1/2") thick high-strength neodymium ring magnet with countersinks accommodating #8 or #10 screws, and a 19.05 mm (3/4") inner diameter metal sleeve epoxied to the outside diameter of the magnet. The joint is effected by the spherical ball bearing seating in the inner diameter of the ring magnet, where it is held in place by magnetic force. The 19.05 mm (3/4") id sleeve acts to provide additional support in the magnet's radial direction, reducing the potential for disengagement as the end effector approaches the end of the printable radius when the tie rods approach a horizontal orientation. A close-up of one of the carriages is shown in Figure 2.2, which also shows the magnetic ball joints with sleeves. Earlier work found that in the absence of these sleeves, joints were prone to disengage under high acceleration or as the end effector approached the outer end of the printable radius, resulting in the entire end effector/substrate assembly falling off the machine.

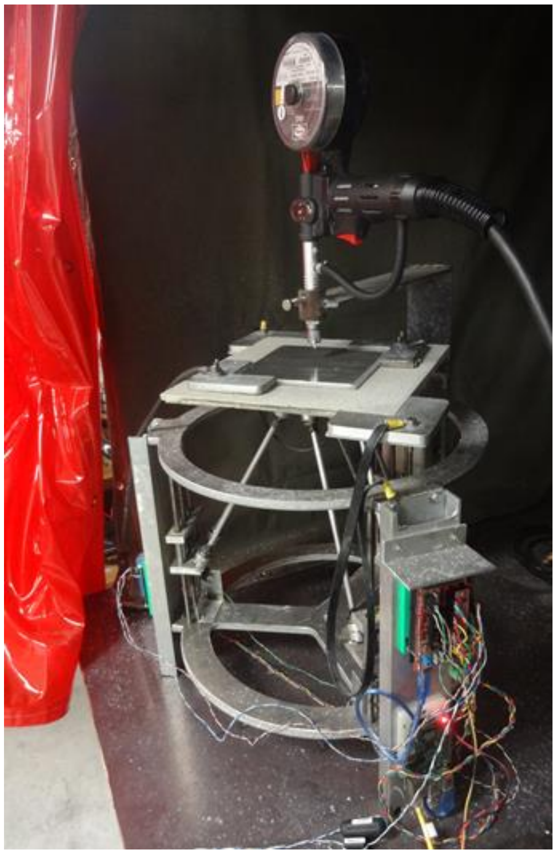


Figure 2.1 Photograph of magnetic ball joint metal RepRap 3-D printer used in this study. A Miller Spoolmate 100 weld gun supplied the feedstock material which was melted by a Millermatic 190 gas metal arc welders (GMAW) (not shown). In the photograph the open source Raspbery Pi and Arduino based electronics are visible on the right support column.

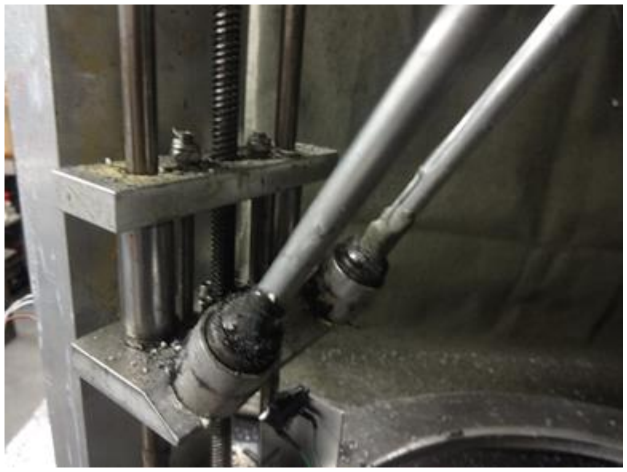


Figure 2.2 Detail of carriage assembly and magnetic ball joint with sleeves on the open source metal RepRap 3-D printer. The lead screw from the stepper motor driving that carriage is also visible in the center of the image.

As can be seen in Figure 2.1, there are 12 ball joints, one at each end of the six tie rods that connect the end effector to the carriages. The six guide rods (precision-ground 8 mm diameter A2 tool steel.) are grouped in pairs with each pair on 6 cm centers using a single axis. The all-aluminum frame consists of a pair of circular ends cut from 10.16 mm (0.4") thick plate (alloy 1100). The upper plate allows the tie rods to pass through it to hold the substrate. Vertical support legs are provided by three pieces of 25.4 mm × 76.2 mm (1" × 3") rectangular aluminum tubing 400 mm long (6063 T52 aluminum), which prevent the three stepper motors from becoming a path to ground should the frame become electrified during welding.

Motion is provided by three stepper motors with integrated lead screw shafts. Lead screws are four-start and have an 8 mm pitch. The stepper motors are 200 step bi-polar driven with 1/16 microstepping. The combination of lead screw and motor yields movement precision in the z-direction (vertical) of 2.5 microns. Movement precision in the x–y plane is 4.6 micron in the center of the bed, and varies with the location of the end effector. Linear bearings (LM8UU) ride on the guide rods and are clamped into a pair of 0.400" aluminum housings using bolts, forming carriages to which tie rods are connected. Tie rods are constructed from 5/16" rigid aluminum tubing. The end effector is triangular and incorporates means for attaching a platform upon which insulation is mounted as seen in Figure 2.1. The welding gun support is a welded "L" made with 9.525 mm × 76.2 mm (3/8" × 3") mild steel. The gun nozzle is held in place directly over the axial center of the robot by a piece of 25.4 mm (1") mild steel pipe having a set screw in one side to secure the nozzle. The support can be moved vertically to set the location of the gun relative to the substrate.

2.3.1 Electronics

The IV measurement board from [20] is connected to a power supply. It has three wires which in the previous version were connected to an Arduino: the ground, and two analog signals. In the new version, they need to be connected to the RAMPS board that is operating the printer. The ground must again be shared with a ground pin on the RAMPS, and the analog signals must be connected to any available analog inputs, such as A3 and A4. Those three pins are all located on the AUX1 header [28]. For protection it is housed in a 3-D printed polymer case [27] as shown in Figure2. 3.

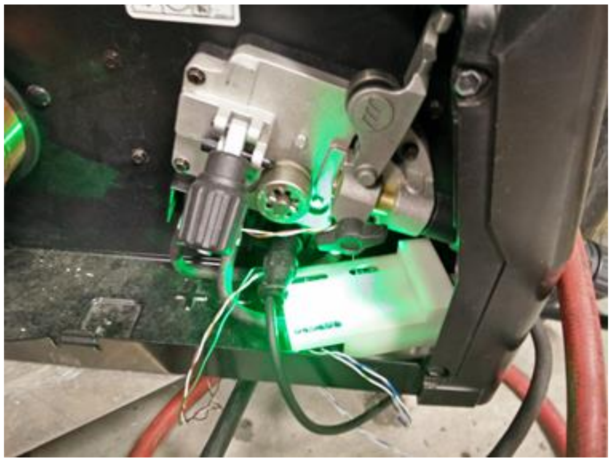


Figure 2.3 Installed current and voltage measurement circuit in the Millermatic 190 gas metal arc welder with 3-D printed cover (white).

In Franklin's interface (Figure 2.4) all analog inputs are handled as if they are temperature controls. Therefore, two new temperature controls must be added, and their pins must be set to correspond to the analog pins receiving the IV measurements. Franklin can use temperature controls to either use a thermistor or a linear relation between the ADC reading and the reported value. β is a property of the thermistor and β needs to be set to NaN (which is not a valid value for a thermistor), so a linear relation, $ax+b$, is used for its signal. Setting a to 1 and b to 0 will output the analog reading in a range from 0 to 1024. To make the output display voltage and current, a different value for a will be required, and possibly for b as well. For the electronics that were used, the formulas for the conversion from output voltage to measured values are as follows.

$$I_{real} = 2000 \times I_{in} / 15 \tag{2.1}$$

$$V_{real} = 27636 \times V_{in} / 2636 \tag{2.2}$$

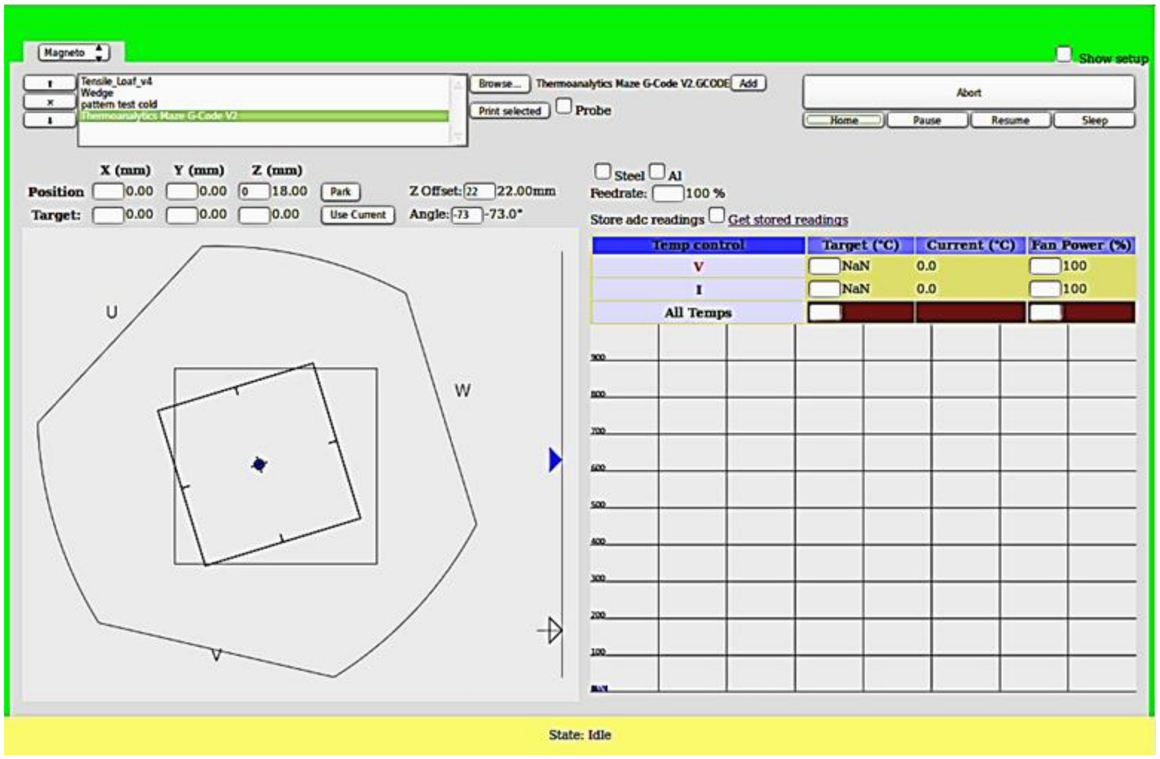
where I_{real} and V_{real} are the values at the welder, in ampere and volt respectively, and I_{in} and V_{in} are the values at the Arduino, both in volt.

R_0 needs to be set to the slope on the ADC reading. Using the fact that the maximum value for the values at the Arduino is 5V and the maximum ADC value is 1024, this means for I

$$\frac{dy}{dx} = \frac{2000 \cdot 5/15 - 0}{1024 - 0} = 0.651 \tag{2.3}$$

And for V

$$\frac{dy}{dx} = \frac{27636 \cdot 5/2636 - 0}{1024 - 0} = 0.0512 \tag{2.4}$$



Tensile_Loaf_v4
Wedge
pattern test cold
Thermoanalytics Maze G-Code V2

Thermoanalytics Maze G-Code V2.GCODE

	X (mm)	Y (mm)	Z (mm)
Position	0.00	0.00	0 18.00
Target:	0.00	0.00	0.00

Z Offset: 22 22.00mm
Angle: -73 -73.0°
Feedrate: 100 %
Store adc readings

Temp control	Target (°C)	Current (°C)	Fan Power (%)
V	NaN	0.0	100
I	NaN	0.0	100
All Temps			

Figure 2.4 Screenshot of the interface for the open source Franklin software.

The firmware continuously sends the data that it reads from those pins to the host computer, which converts the raw ADC readings into voltage and current using the given values for a and b, and allows storing them on the file system. The web interface that is used to control the printer allows downloading those stored readings to the individual user's computer.

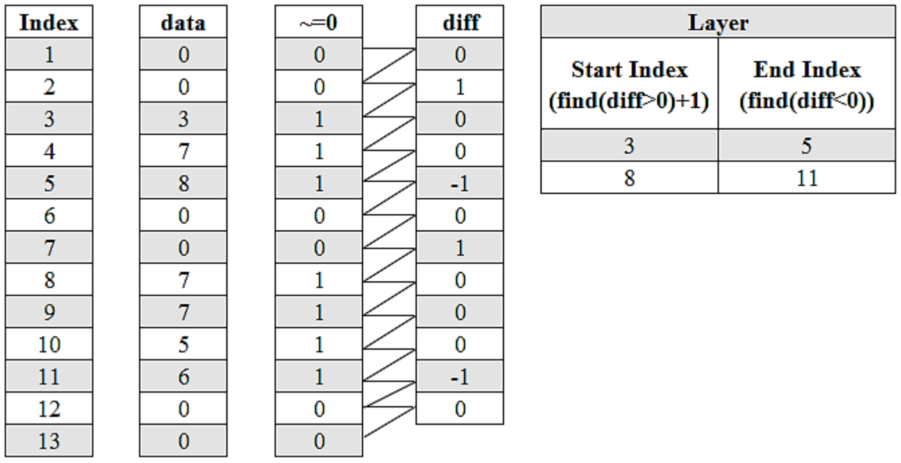
Using the Arduino of the printer comes at the cost of sampling speed: the controller has other tasks that take time, and the serial connection is also used for other traffic. A dedicated controller for the readings, as was used in the previous version, can provide more readings per second. The measurements presented here did not require the extra speed, so this was sacrificed for the benefit of reduced hardware complexity and cost.

2.3.2 Algorithm

From the recorded data file, the timestamp in the first column is in a negative millisecond format which is not intuitive. To convert the negative millisecond (t_{mil}) to the positive second ($t_{\sec}$) can be done using Equation (2.5) where $N$is the number of lines in the input data file.

$$\begin{aligned} &t_{\sec}(i) = \left(t_{mil}(i) - t_{mil}(0)\right)/1000 \\ &i = 0,1,2,\ldots,N-1 \end{aligned} \tag{2.5}$$

There can be two types of noise in the data: (1) zero-noise and (2) non-zero-noise. Zero-noise is the zero values that occur during the layer that need to be replaced with a very small value so the layer algorithm does not misinterpret them as the layer separation points (In this experiment, 10^{-7} is used). Non-zero-noise is non-zero values that occur between the layers that need to be replaced by 0. After that data is separated into layers of non-zero data by the following concept (Figure 2.5). First, the logical operator “not equal” (~=) is used to find non-zero data. Then the “diff” function is used to find the difference between the current cell and the previous cell in each row of the result from the previous step. Finally, the start index and the end index of each layer are found using “find” function which find the positive value for the start index and the negative value for the end index.



Index	data	~=0	diff
1	0	0	0
2	0	0	1
3	3	1	0
4	7	1	0
5	8	1	-1
6	0	0	0
7	0	0	1
8	7	1	0
9	7	1	0
10	5	1	0
11	6	1	-1
12	0	0	0
13	0	0	

Layer	
Start Index (find(diff>0)+1)	End Index (find(diff<0))
3	5
8	11

Figure 2.5 Layers Algorithm.

The two standard error (2SE) of each layer is calculated in Equations (2.6)–(2.8) where *d* is the data, μ is the mean or the average of the data layer length *n*, and *SD* is the standard deviation.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{2.6}$$

$$SD = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} \left| d_i - \mu \right|^2} \tag{2.7}$$

$$2SE = \frac{2 \times SD}{\sqrt{n}} \tag{2.8}$$

The average voltage and current for each layer were calculated on a per alloy basis.

2.3.3 Printing of Test Specimens

Standard 0.035 inch (0.89 mm) diameter ER1100 and ER4047 wire (AlcoTec , Traverse, MI, USA) in addition to ER4043, ER4943, and ER5356 wire (Hobart) were used as feedstock material to print test blocks on clean and degreased ASTM A36 low carbon steel print substrates. The print test blocks were each 108 mm × 31.75 mm × 25.4 mm whereas the print substrates were 127 mm × 127 mm × 6.35 mm in size. Low carbon steel was utilized as a print substrate because this was shown in previous work to encourage a weak interface between printed part and print substrate, thus allowing printed parts to be removed with minimal energy [11,12]. Print settings and print path were the same for all alloys (Table 2.2, Figure 2.6). Five blocks per alloy were printed (Figure 2.7). Voltage and current data were collected for all specimens during each print cycle.

Table 2.2 Print Settings Utilized for Test Specimens.

Parameter	Value
Welder Setting (unitless)	1
Wire Feed Rate (mm/s)	124.6
Print Speed (mm/s)	10
Wire Stick-Out (mm)	10
Shield Gas Flow Rate (L/s)	0.24
G-Code Layer Height (mm)	2.5
G-Code Lateral Bead Spacing (mm)	3.3
Pause After Each Layer (s)	60
Number of Print Layers	15

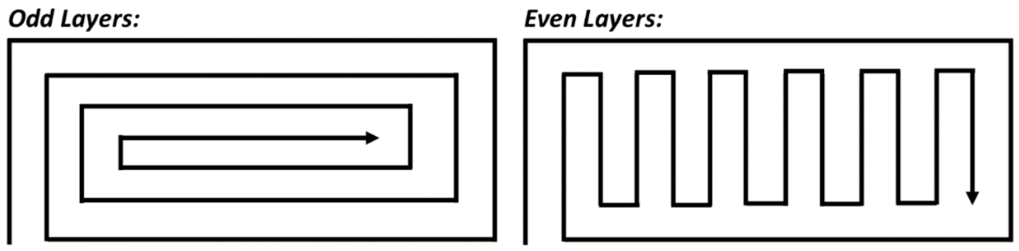


Figure 2.6 Alternating print paths viewed in the direction of the z-axis.

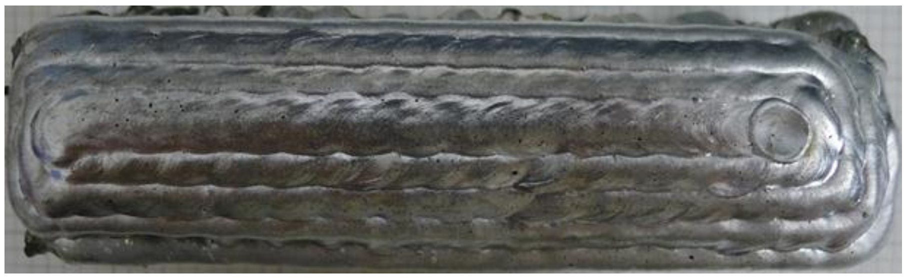
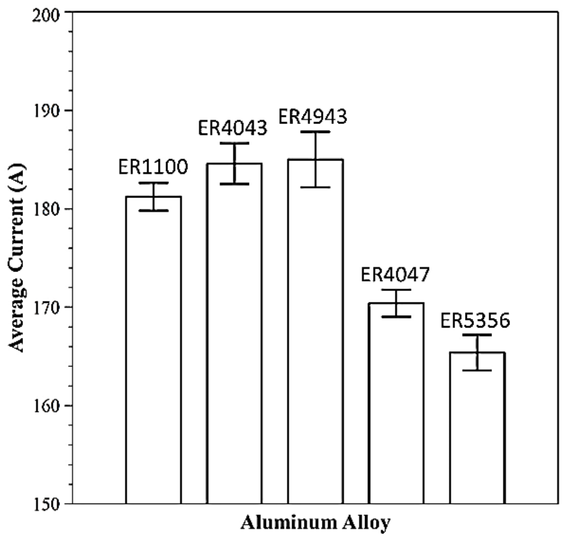


Figure 2.7 Example top surface of a 4943 printed specimen viewed in the direction of the z-axis.

2.4 Result and Discussion

The currents, averaged from more than 200,000 data points, of ER1100, ER4043, and ER4943 specimens were statistically equivalent and greater in value than that of ER4047 and ER5356 (Figure 2.8). By contrast, ER4047 exhibited the largest voltage on average, followed by ER1100, and finally with ER4043, ER4943, and ER5356 exhibiting statistically equivalent voltages.



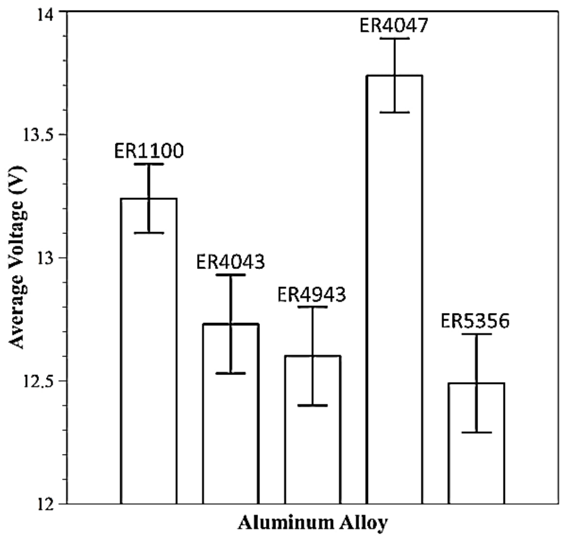


Figure 2.8 Average current (**left**) and average voltage (**right**) of all five aluminum alloys. Error bars represent ±2 standard error (≈95% confidence).

On a per-layer basis, there is a significant difference on average between the first layer and subsequent print layers. The first print layer typically exhibited significantly lower current and higher voltage compared with other print layers (Figure 2.9). Odd layers appeared to exhibit lower current and voltage, compared with even layers although this trend was not statistically significant; this may be due to differences in print paths (Figure 2.6). More scatter in the current and voltage data, and thus more error, was observed for initial layers as opposed to the final layers. This is because the initial weld is purposely poor to allow for substrate release [10,11].

On a per-alloy basis, the differences may be explained by the electrical resistivity of each alloy (Table 2.3). The commercially pure aluminum alloy, ER1100, had the smallest electrical resistivity and it exhibited the highest voltages and currents. The three 4000 series aluminum alloys, 4043, 4943, and 4047, all had very similar electrical resistivities and the current-voltages they exhibited were all statistically equivalent and between those of 1100 and 5356. The 5356 aluminum-magnesium alloy had the largest electrical resistivity and also exhibited the smallest currents and voltages. As electrical resistivity decreases, electrical conductivity increases and more current can be supplied at a given voltage. Since the welder control strategy is unknown, patterns in voltage variation cannot be fully explained.

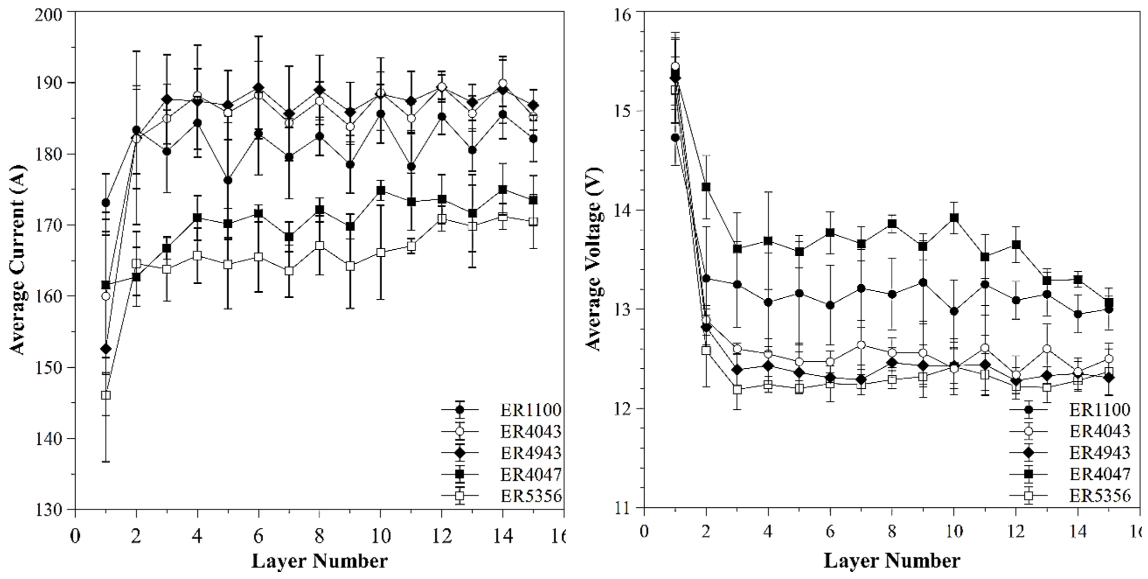


Figure 2.9 Average current (**left**) and voltage (**right**) of all five alloys on a per-layer basis. Error bars represent ±2 standard error.

Table 2.3 Electrical Resistivity of Common Aluminum Weld Alloys.

Alloy	Electrical Resistivity ($\times 10^{-6}$ Ω·cm)
1100	2.99
4043	4.16
4943	4.21
4047	4.31
5356	5.98

A difference in electrical resistivity may be insufficient to fully explain the differences in weld currents and voltages between the five alloys studied. Specifically, the large difference in 5356 values compared with the other four alloys does not follow the same scaled difference in electrical resistivity. During welding it was observed that there was more spatter of the 5356 alloy than of the other printed alloys. A contributing factor to the spatter may have been wire and arc travel. The wire was sufficiently stiff such that, even after traveling through the weld gun, the wire continued to curve in the direction it was spooled. At sample edges, the curved wire compounded with arc travel to produce more spatter and shorts in the arc. These electrical shorts were recorded as zero values by the measurement device which would lower an average of the data.

The current and voltage of the first layer were different than subsequent layers due to differences in substrate and print materials. The first weld layer attempted to weld the aluminum print material to the low carbon steel substrate whereas all subsequent layers directly welded aluminum to aluminum. Due to differences in the steel and aluminum melting temperatures, there was insufficient arc energy to melt the steel. This resulted in lack of penetration of aluminum into the steel substrate and lack of fusion to the surface of the steel (which in beneficial in substrate release). When aluminum was printed on aluminum, there was sufficient energy to melt the aluminum to provide weld penetration and fusion.

A byproduct of printing aluminum on a steel substrate is a first layer with significant topographical variation [10]. As subsequent layers are printed, the distance between the first layer and the weld gun changes. As distance increases, the arc length also increases, and this increases the voltage [29]. Longer arcs are less focused and can result in more spatter of metal [29]. This erratic behavior of the arc thus translates to more scatter in the data. As more layers are printed, the topography of the sample becomes more uniform, stabilizing the arc length and behavior.

Both the specific results for the alloys presented here as well as the open-source integration scheme provided under open hardware licenses here are applicable to other similar projects such as wire + arc additive manufacturing (WAAM) for aluminum [30,31].

2.5 Conclusion

This paper has provided the new design for an integrated mechanically improved delta-style GMAW 3-D printer with current and voltage measurement of welder without adding an extra controller. The improved mechanical design increased the build radius by 10 cm and improved print quality. In addition, by connecting the IV measuring board back to the RAMPS board that is controlling the printer, the measurement can be recorded in real-time through the RAMPS board. This new design helps reduce the cost and the complexity of hardware. The measurement provides the current and voltage aspect for each type of alloy during welding. The alloys were successfully monitored and had measurements consistent with their electrical resistivities. The ability to monitor the voltage and current of GMAW provides more data related to the energy input for modeling and printing process and property optimization.

Acknowledgements

The authors would like to thank Anthony Pinar, James De Clerk, Jerry Anzalone and Tim Havens for the help in the designs and technical support. The authors would like to acknowledge helpful discussions with America Makes team members. The authors would also like to acknowledge support and helpful discussions with C. Hsu and technical assistance from the Miller Electric Manufacturing Company. This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8650-12-2-7230. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.

2.6 References

1 Jones, R.; Haufe, P.; Sells, E.; Iravani, P.; Olliver, V.; Palmer, C.; Bowyer, A. Reprap—The replicating rapid prototyper. *Robotica* **2011**, *29*, 177–191.

2 Sells, E.; Bailard, S.; Smith, Z.; Bowyer, A.; Olliver, V. Reprap: The replicating rapid prototyper: Maximizing customizability by breeding the means of production. In *Handbook of Research in Mass Customization and Personalization*; World Scientific: Singapore, 2009.

3 Bowyer, A. 3D printing and humanity's first imperfect replicator. *3D Print. Addit. Manuf.* **2014**, *1*, 4–5.

4 Kostakis, V.; Papachristou, M. Commons-based peer production and digital fabrication: The case of a reprap-based, lego-built 3D printing-milling machine. *Telemat. Inform.* **2014**, *31*, 434–443.

5 Corbett, J. *Reprap Colour Mixing Project*; Faculty of Engineering and Design, Final Year MEng Project; Department of Mechanical Engineering, University of Bath: Bath, UK, 2012.

6 Baechler, C.; DeVuono, M.; Pearce, J.M. Distributed recycling of waste polymer into reprap feedstock. *Rapid Prototyp. J.* **2013**, *19*, 118–125.

7 Hunt, E.J.; Zhang, C.; Anzalone, N.; Pearce, J.M. Polymer recycling codes for distributed manufacturing with 3-D printers. *Resour. Conserv. Recycl.* **2015**, *97*, 24–30.

8 Leigh, S.J.; Bradley, R.J.; Purssell, C.P.; Billson, D.R.; Hutchins, D.A. A simple, low-cost conductive composite material for 3D printing of electronic sensors. *PLoS ONE* **2012**, *7*, e49365, doi:10.1371/journal.pone.0049365.

9 Anzalone, G.C.; Wijnen, B.; Pearce, J.M. Multi-material additive and subtractive prosumer digital fabrication with a free and open-source convertible delta RepRap 3-D printer. *Rapid Prototyp. J.* **2015**, *21*, 506–519.

10 Anzalone, G.C.; Zhang, C.; Wijnen, B.; Sanders, P.G.; Pearce, J.M. A low-cost open-source metal 3-D printer. *IEEE Access* **2013**, *1*, 803–810.

11 Haselhuhn, A.S.; Gooding, E.J.; Glover, A.G.; Anzalone, G.C.; Wijnen, B.; Sanders, P.G.; Pearce, J.M. Substrate release mechanisms for gas metal arc weld 3D aluminum metal printing. *3D Print. Addit. Manuf.* **2014**, *1*, 204–209.

12 Haselhuhn, A.S.; Wijnen, B.; Anzalone, G.C.; Sanders, P.G.; Pearce, J.M. *In situ* formation of substrate release mechanisms for gas metal arc weld metal 3-D printing. *J. Mater. Process. Technol.* **2015**, *226*, 50–59.

13 Kading, B.; Kegley, M.; Delzer, T.; Straub, J.; Kerlin, S. Development of a Metal-Printing 3D Printer at the University of North Dakota. In Proceedings of the University

of North Dakota School of Graduate Studies Scholarly Forum, Grand Forks, ND, USA, 10–11 March 2015.

14 Torabi, P.; Petros, M.; Khoshnevis, B. Selective inhibition sintering: The process for consumer metal additive manufacturing. *3D Print. Addit. Manuf.* **2014**, *1*, 152–155.

15 Teles, G.; Duke, T.; Coleman, K.; Zhao, Y.; Kim, J.; Shafai, C.; Shafai, L. 3D metal-plastic printer for fabrication of antennas on custom and flexible surfaces. University of Manitoba: Winnipeg, MB, Canada, 2015.

16 Arduino. Available online: https://www.arduino.cc (accessed on 18 October 2014).

17 Pearce, J.M.; Blair, C.M.; Laciak, K.J.; Andrews, R.; Nosrat, A.; Zelenika-Zovko, I. 3-D printing of open source appropriate technologies for self-directed sustainable development. *J. Sustain. Dev.* **2010**, *3*, 17–29.

18 Canessa, E.; Fonda, C.; Zennaro, M. Low-cost 3D printing for science, education and sustainable development. The Abdus Salam International Centre for Theoretical Physics (ICTP): Trieste, Italy, 2013; Volume 11.

19 Birtchnell, T.; Hoyle, W. *3D Printing for Development in the Global South: The 3D4D Challenge*; Palgrave Macmillan: London, UK, 2014.

20 Pinar, A.; Wijnen, B.; Anzalone, G.; Havens, T.; Sanders, P.; Pearce, J. Low-cost open-source voltage and current monitor for gas metal arc weld 3D printing. *J. Sens.* **2015**, *2015*, 1–8.

21 Wijnen, B.; Anzalone, G.C.; Haselhuhn, A.S.; Sanders, P.G.; Pearce, J.M. Free and Open Source Control Software for 3-D Motion and Processing. Available online: https://github.com/mtu-most/franklin (accessed on 29 October 2015).

22 Haselhuhn, A.S.; Buhr, M.B.; Wijnen, B.; Wood, T.D.; Anzalone, G.C.; Sanders, P.G.; Pearce, J.M. Structure-property relationships of common aluminum weld alloys utilized as feedstock for gmaw-based metal 3-D printing. 2015, under review.

23 Anderson, B.E.; Hsu, C. Aluminum Alloy Welding Wire. U.S. Patent Application No. 13/023,158, 11 August 2011.

24 Dickerson, P.B. Welding of aluminum alloys. In *ASM Handbook*; ASM International: Materials Park, OH, USA, 1993; Volume 6, pp. 722–739.

25 Rostock. Available online: http://reprap.org/wiki/Rostock (accessed on 23 October 2015).

26 Open Source Hardware Association. Open Source Hardware (OSHW) Statement of Principles 1.0. Available online: http://www.oshwa.org/definition/ (accessed on 25 September 2015).

27 Magneto: Open Source Metal 3-D Printer. Open Science Framework. Available online: https://osf.io/ytvgm/ (accessed on 25 September 2015).

28 Ramps. Available online: http://reprap.org/wiki/RAMPS_1.4 (accessed on 23 October 2015).

29 Mandal, N.R. *Aluminum Welding*; Narosa Publishing House: New Delhi, India, 2002.

30 Gu, J.L.; Ding, J.L.; Cong, B.Q.; Bai, J.; Gu, H.M.; Williams, S.W.; Zhai, Y.C. The Influence of Wire Properties on the Quality and Performance of Wire + Arc Additive Manufactured Aluminium Parts. *Adv. Mater. Res.* **2015**, *1081*, 210–214.

31 Ding, J.; Colegrove, P.; Martina, F.; Williams, S.; Wiktorowicz, R.; Palt, M.R. Development of a laminar flow local shielding device for wire + arc additive manufacture. *J. Mater. Process. Technol.* **2015**, *226*, 99–105.

Chapter 3: Slicer and Optimization for Open-source GMAW-based Metal 3-D Printing[2]

3.1 Abstract

Low-cost gas metal arc welding (GMAW)-based 3-D printing has proven effective at additive manufacturing steel and aluminum parts. Early success was based on hand-writing G-code. To be functional for a wide array of users, software must be capable of slicing a 3-D model and generating G-code for the path of each layer automatically. In order for a free slicer program to support the open-source metal 3-D printer, this paper reports on the upgrading of the free and open source *CuraEngine* into *MOSTMetalCura*, which provides the abilities to: i) change the perimeter metric from width to track count, ii) avoid movement that overlaps previous weld beads, iii) have infill start immediately after the perimeter finished and in the direction that eliminates translations, iv) add a variable pause between layers to allow for substrate cooling, v) configure GPIO pins to turn on/off the welder, and vi) set optimized wire feed speed and voltage of the welder based on printing speed, layer height, filament diameter, and tool track width. The process for doing this and the changes made are first detailed and then used to help optimize the function of the printer for ER70S-6 steel. To find the optimized function based on volume of material, the line width, layer height, and printing speed are varied to provide wire feed speed calculated by *MOSTMetalCura*, then the settings are used to optimally print 3-D models. The results of 3-D printing three case study objects of increasing geometric complexity using the process optimization are presented, which show resolution of 1mm bead widths.

3.2 Introduction

Additive manufacturing with 3-D printing has matured beyond simple rapid prototyping [1-5] to small-batch production [6-8] and distributed manufacturing [9-13]. Some of the most industrially interesting 3-D printing is that of metals, which include laser sintering and melting [14-18] and electron beam melting [19-21]. These systems are mature and such industrial-grade additive manufacturing machines can be prohibitively expensive (e.g. >

[2] This chapter has been completed as an article to submit. Citation: Nilsiam Y, Sanders P, & Pearce J (2017). Slicer and Optimization for Open-source GMAW-based Metal 3-D Printing.

US$500,000 - US$1.5 million), which is beyond the reach of consumers and small and medium sized enterprises (SMEs) [22]. Recently progress has been made in upgrading the popular open-source self-replicating rapid prototyper (RepRap) 3-D printer designs [23-25] into a low-cost open-source metal 3-D printer [26]. This metal 3-D printer uses a low-cost gas-metal arc welder as a fixed printer head and is controlled by an open-source Arduino micro-controller [27,28]. Opposite the motion of most polymer 3-D printers where the printer head moves, the stage of the metal 3-D printer holds a re-usable substrate and moves during the printing [29,30]. The power of the welder can also be monitored with open source hardware and software [31,32]. An open-source firmware called *Franklin* [33] is used to control the metal 3-D printer by translating G-code into controlling signals. G-code is a numerical control programming language that is commonly used for controlling automated machine tools and it can be manually written or generated by slicer programs. Low-cost GMAW-based 3-D printing has proven effective at both steel and aluminum [26, 34].

This early success was based largely on hand-coding G-code for relatively simple geometries. To be functional for a wide array of users, a slicer software must be capable of slicing a 3-D model into layers and then generating G-code for the path of each layer. Expensive metal 3-D printing systems come with their own proprietary slicers. Unfortunately, the free and open source slicers such as *Cura* [35] and Slic3r [36] are made primarily for polymer based 3-D printing. In order for a slicer program to support the open-source metal 3-D printer, some functions need to be added to existing polymer based slicers including: i) the ability to change the perimeter metric from width of the perimeter to track count because the line width is constant, ii) the ability to avoid movement that will run over the previously laid weld bead (polymer printers can handle this contact but this is not possible for metal as the solidified metal surface does not have the give of warm polymer layer), iii) have infill starts immediately where the last segment of the perimeter finished and in the direction that eliminates translations to reach previously unfilled areas, iv) add an option to pause between layers to allow for substrate cooling and ability to set the pause time, v) the capability to configure the General-Purpose Input/Output (GPIO) pins (to turn on and off the welder), and vi) set the optimized wire feed speed and voltage of the welder based on the printing speed, layer height, filament diameter, and the width tool track. To provide this new functionality for open-source metal 3-D printing, the open source *CuraEngine* [37], has been upgraded here to *MOSTMetalCura*. The process for doing this and the changes made are first detailed and then used to enable better control over the printer to help optimize the function of the printer for steel using printing speed, the wire feed speed, and the voltage of the welder. The optimized function would result in an accurately printed part with good observational resolution and surface quality. To find the optimized function, the following experiments are done: the line width, layer height, and printing speed are varied to provide wire feed speed calculated by the slicer

MOSTMetalCura, then the settings would be used to optimally print 3-D models. Other 3-D printing quality factors are assessed for optimization of the slicing algorithm as well including shield gas parameters, temperature and humidity. Even though, these factors are not used directly by the slicer, they are crucial for metal 3-D printing. The 3-D printing results of three case study objects of increasing geometric complexity are presented and discussed.

3.3 Background

Today, metal 3-D printing is a popular topic, there is a rapid growth of journal papers about metal 3-D printing in search results from Google Scholar [38]. Even with the high costs of commercial metal 3-D printers and their maintenance, many companies still pay the price due to their ability to perform rapid prototyping [38,39]. A low-cost open-source metal 3-D printer will lower the barrier for the technology to be accessible to individuals and small and medium enterprises (SMEs). This will allow more people to fabricate customized 3-D objects and will be rapid growth and improvements by open-source communities around the world. The metal 3-D printer RepRap [26] is open-source hardware inspired by the *Rostock*, which is a *Deltabot RepRap* and is controlled by open-source software, *Franklin* [33]. *Franklin* is the control system for the low-cost metal 3-D printer which contains two parts. First, the firmware that controls the printer and exchanges information with the web-based server on a host computer in the same network. Second, the web interface that handles the G-code from a slicer then communicates with the *Franklin* Firmware, which can control the motors, temperature, and GPIO pins (see Figure 3.1). One of the GPIO pins is used in the low-cost metal 3-D printer to turn the welder on or off. Opposite from the original *RepRap* printers, the metal 3-D printer has the welder as the printer head fixed in a single position and the three-axis stage moving according to the printing path. Weld-based 3-D printing is relatively inexpensive and produces good adhesion between layers with low porosity, but there are constraints in resolution and surface quality [29]. To solve the problem, gas metal arc welding (GMAW) is used for the low-cost metal 3-D printer.

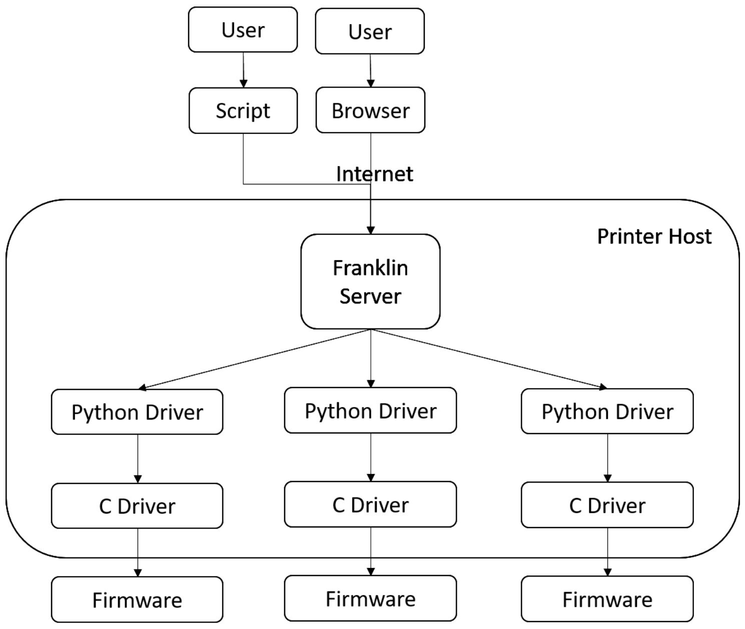


Figure 3.1 The architecture of *Franklin* and schematic of the workflow from users to the printer.

Cura, one of the most popular open source slicing programs, was developed by Daid (David Braam) and was released on the AGPLv3 license. Two main components of *Cura* are the GUI (Graphic User Interface) and the slicing engine called *CuraEngine* [40,41]. The GUI part is written in Python and features a 3-D model file viewer, the ability to send a .STL file to *CuraEngine* as well as receive G-code from, and connecting with a 3-D printer and sending the G-code to it. The *CuraEngine* is written in C++ and is the core for the slicing process. Working directly with the *CuraEngine* also gives us the capability to have more control over the slicing process through settings in the configuration file (*fdmprinter.json*).

3.4 Methods

3.4.1 Open Source Cura

Open-source software, such as *Cura,* provides access to all the source code and the freedom to modify it. However, often, as in this case, is only available with the limited technical details and support. *Cura* has two main parts: 1) the GUI and 2) the slicing engine called *CuraEngine*. The *CuraEngine* is written in C++ and can be used as a separate program that handles slicing and generating G-code. The CuraEngine starting code was taken from *GitHub* with the version number 15.06 [37]. In order to modify the source code, the structure and the processes of the program need to be known and understood. The process flow of the *CuraEngine* can be found in Figure 3.2.

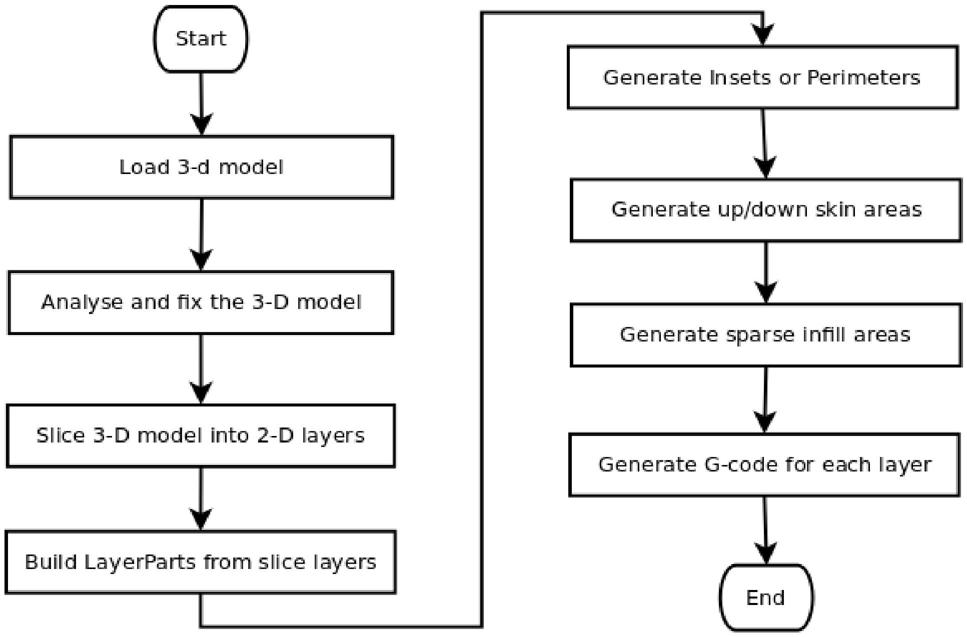

Start
Load 3-d model
Analyse and fix the 3-D model
Slice 3-D model into 2-D layers
Build LayerParts from slice layers
Generate Insets or Perimeters
Generate up/down skin areas
Generate sparse infill areas
Generate G-code for each layer
End

Figure 3.2 The process flow of *CuraEngine* based on information from [37].

LayerParts are isolated parts in a layer. For example, a simple table with 4 legs is sliced into layers. The lowest layer would have 4 parts in the layer (see Figure 3.3.). Skins are areas that supposed to be fully filled with material. Usually the top and bottom of a model would be fully filled. Infill are areas inside the model and either can be fully filled, partially filled or left empty. However, in metal 3-D printing both skin and infill are always fully filled.

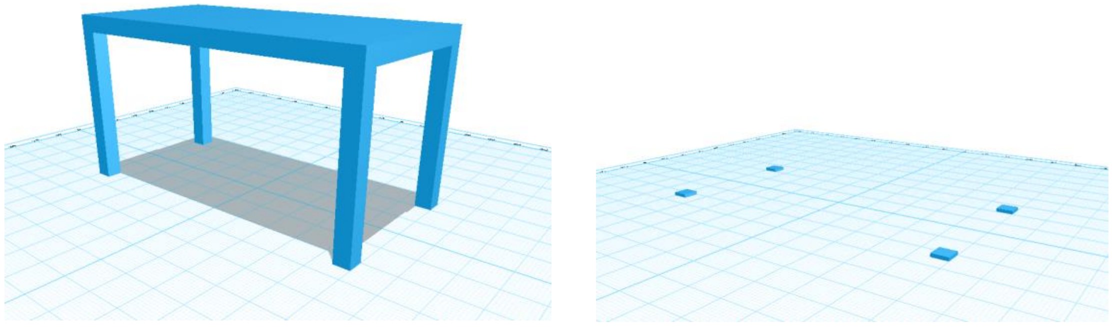


Figure 3.3 A table 3-D model and its *LayerParts* in the first layer.

3.4.2 Altering Cura

Working directly with *CuraEngine* provides more control compared to using *Cura* because more settings can changed via the configuration file, *fdmprinter.json*. First a setting called *machine_metal_printing* is added to the configuration file. This setting is a boolean data type. If it is set to true, then the generated G-code would be appropriate for the open-source metal 3-D printer. If it is set to false, then the generated G-code would be the same as using original *CuraEngine* without using other additional settings for MOST's metal 3-D printer. A boolean member called *isMetalPrinting* is added to class *gcodeExport* in order to keep the setting value of *machine_metal_printing*. Next, the perimeter metric can be changed from width to track count as the width is not controllable as in FFF of polymers by changing the setting named *wall_line_count* in the configuration file. This number would define how many track count for the perimeter. To avoid running over the previous laid weld bead and also to start the infill immediately after the last segment of the perimeter is done, the parameters *top_bottom_pattern* and *fill_pattern* should be set to *Concentric*, so all printing would be in concentric pattern (see Figure 3.4). The concentric pattern would make the printing go around either form the outside to the inside or vice versa. The pattern is switched and repeated every other layer.

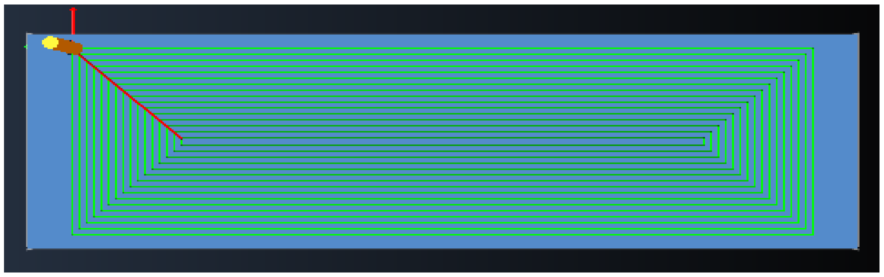


Figure 3.4 Concentric pattern of a layer of block (101.6 mm × 31.75 mm × 31.75 mm).

Next, for a GMAW metal 3-D printer, the welder can be turned on or off via G-code, so the settings *machine_welder_on_gcode* and *machine_welder_off_gcode* were added to the configuration file. The data type of these settings is string and the value of each setting is G-code commands to turn on and turn off welder through GPIO pin [33]. A default value of *machine_welder_on_gcode* is *G4 P0\nM42 P1 S1\n*. *G4* is G-code command for the printer to be still doing nothing and *P0* means for zero millisecond. *\n* is newline command. *M42* is switching general purpose I/O pin command and *M42 P1 S1* means set pin 1 to value 1 [42]. A default value of *machine_welder_off_gocde* is *G4 P0\nM42 P1 S0\n* and can be interpreted in the same way as previous example. A boolean member called *isWelding* is added to class *gcodeExport* to keep a status of the welder. Two string variables are added to the class, *welder_on* to store setting value from *machine_welder_on_gocde* and *welder_off* to store setting value from *machine_welder_off_gcode*. In function *writeMove()* of *gcodeExport* class, if *extrusion_mm3_per_mm* is greater than 0.000001, *isMetalPrinting* is true, and *isWelding* is not true, then the *welder_on* would be inserted into the generated G-code. If *extrusion_mm3_per_mm* is lesser than 0.000001, then it is only a traveling without printing and the *welder_off* would be inserted into the generated G-code.

In order to set the values of *welder_on* and *welder_off*, two methods or functions called *setWelderOn()* and *setWelderOff()* were added to *gcodeExport* class according to the object-oriented programming concept. These methods would be the way to set the values of the attributes of *gcodeExport* object or class. *fffProcessor* class reads values of *machine_welder_on_gcode* and *machine_welder_off_gocde* from the configuration file then assigns those values to the attributes *welder_on* and *welder_off* of *gcodeExport* class through the methods, *setWelderOn()* and *setWelderOff()*. This is how objects interacts with each other. In the same way, method *setIsMetalPrinting()* and *setIsWelding()* are for setting values of *isMetalPrinting* and *isWelding* attributes. These methods are implemented in *gcodeExport.cpp*. The initial value of *isWelding* attribute is set to false.

When a movement of the substrate is for traveling, not printing or welding, actually the welder should be turned off. However, to avoid turning off and on the welder too often or unnecessary, the setting named *machine_min_dist_welder_off* is added to the configuration file. The data type of the setting is double type and the unit is in millimeter. If a traveling distance is less than the setting value (10 millimeters by default), then it will travel without turning the welder off. A double data type member called *min_dist_welder_off* is added to *gcodeExport* class to store value from *machine_min_dist_welder_off* setting. In function *writeMove()* of *gocdeExport* class, before *welder_off* willbe inserted into the generated G-code, the travel distance has to be greater than *min_dist_welder_off* value. The method *setMinDistWelderOff()* was added to *gcodeExport* class, so other classes can set the value of attribute *min_dist_welder_off*.

The pause between layers for substrate cooling is added as an option in the configuration file. The parameter named *machine_layer_pause* is a boolean data type. If it is set to true, then the *machine_layer_pause_gcode, machine_layer_pause_time, and machine_layer_increase* values would be inserted in a generated G-code between each layer. The *machine_layer_pause_gcode* is added to the configuration file as well and would be used only if the value of *machine_layer_pause* is true. In function *writeGCode()* of *fffProcessor* class, if *machine_layer_pause* value is true, then at the end of each layer the *machine_welder_off_gocde* and *machine_layer_pause_gcode* values would be added to the generated G-code. A default value of *machine_layer_pause_gocde* is *G4 P*, of *machine_layer_pause_time* is 60000, and of *machine_layer_pause_increase* is 20 which tells the printer to do nothing for 60,000 milliseconds and increase this pause time by 20 percent every layer. Similarly, future work could utilize this flexibility in the code to use IR sensor feedback to begin a new layer when the part has reached an acceptably low temperature.

In addition, there are some functions (or options) that are not available for GMAW-based 3-D metal printers, so there is no need to include those G-code commands. In function *writeTemperatureCommand()* of *gcodeExport* class, before inserting G-code commands about extruder temperature, the checking of metal printing would be checked. If it is the metal printing, then those extruder temperature G-code commands would not be inserted.

Finally, the version name string is defined in *settings* class at the top of the file. So it was changed to *MOSTMetalCura*. In *CmakeLists.txt*, the name of the software and library links are set in that file, so they were changed from *CuraEngine* to *MOSTMetalCura*. At the top of a generated G-code file, this version text would be added as a comment. Likewise, important information are added as comments after the version text which include line width, layer height, printing speed, material diameter, material volume per second, recommended voltage and wire feed speed for the welder (Millermatic 190 MIG welder). This information was added in the function *writeGCode()* of *fffProcessor* class.

3.4.3 GMAW 3-D Printing with MOSTMetalCura

The new *MOSTMetalCura* slicing program is tested on a *RepRap* GMAW-based 3-D printer previously described [32]. Based on the previous works [26,29], ER70S-6 steel welding wire with 0.024 inches or 6 mm diameter was used in the experiment. The welder was Millermatic 190 MIG welder. To optimize the wire feed speed, the expected and actual volume of material are calculated, so that they are matched. The expected volume of steel can be calculated in Equation (3.1) and the actual volume that coming out from the welder is calculated by Equation (3.2).

$$E_v = L_w \times L_h \times P_s \tag{3.1}$$

Where E_v is expected volume in cubic millimeters per second (mm^3/s). L_w is the welding bead width in mm, L_h is the height of each layer in mm, and P_s is the printing speed or the movement speed of the substrate in mm per second in this case.

$$C_{aw} = \pi \times \left(\frac{D_w}{2}\right)^2$$

$$W_s = \frac{(202.3 - 25.0)}{(100 - 20)} \times W_n - 19 = (2.216 \times W_n) - 19 \qquad (3.2)$$

$$A_v = C_{aw} \times W_s$$

Where C_{aw} is the cross-sectional area of the wire in square millimeters (mm^2). D_w is the diameter of the steel welding wire in mm. W_s is the actual wire feed speed in millimeters per second (mm/s) and W_n is a unitless of the wire speed setting on the welder. A_v is the actual volume of steel wire from the welder in cubic millimeters per second (mm^3/s). According to Miller website [43], the wire feed speed is between 60 to 600 inches per minute. On a scale from 10 to 100 of the wire speed setting, each for 10 seconds of running and the measurement of the wire feed rate can be shown in Figure 3.5. The measurement of the wire length at scale 10 is discarded because it is given the same speed as scale 20 and the speed is only changed after scale 20. Then the W_s is derived from the length values in Table 3.1. The optimized wire feed speed is found if an expected volume equals to an actual volume and the calculation is done by *MOSTMetalCura*.

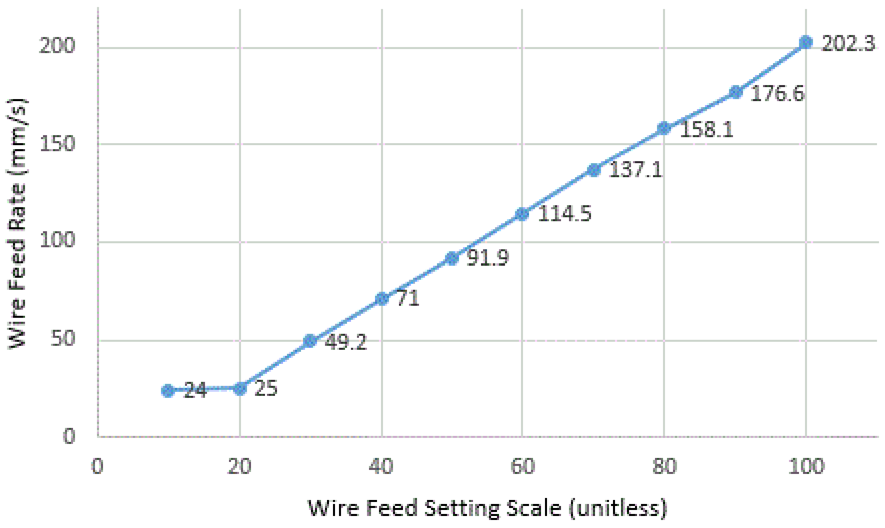


Figure 3.5 The wire feed rate for each scale setting.

To test this approach the 3-D prints were made with the following 3-D models, demonstrating increased geometric complexity of extruded shapes (see Figure 3.6-3.8). The simple block (101.6 mm × 31.75 mm × 31.75 mm), chisel (140 mm × 20 mm × 20 mm) and gear (60 mm diameter × 10 mm) were used because their geometries do not include any bridges or overhangs. All STL files for these 3-D models are available [44].

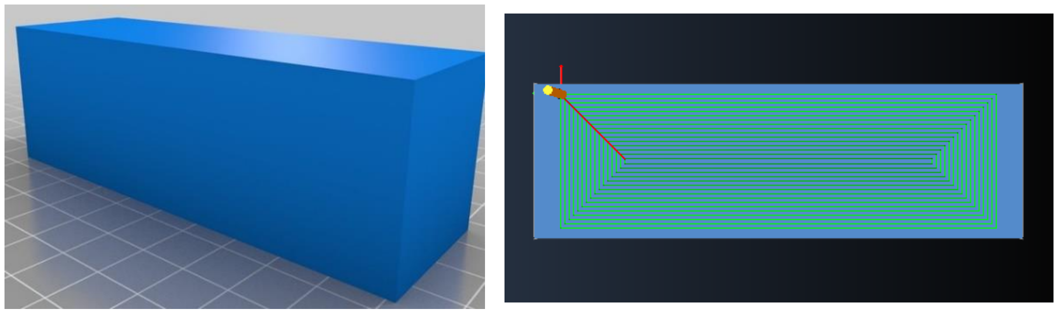


Figure 3.6 a) 3-D model of block (grid lines are every 10mm) and b) a slice showing the path for 3-D printing using *MOSTMetalCura*.

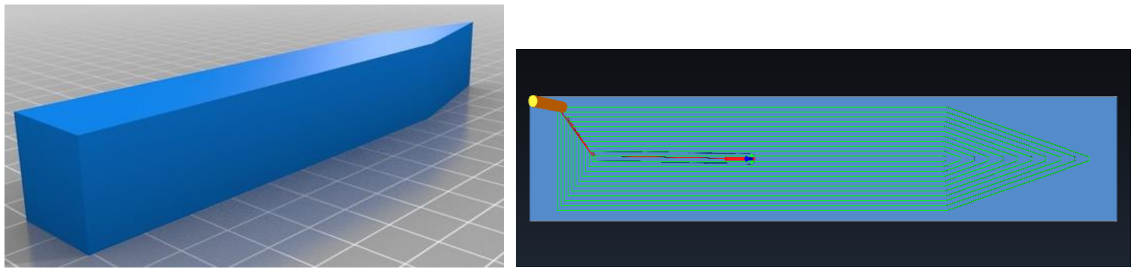


Figure 3.7 a) 3-D model of chisel (grid lines are every 10mm) and b) a slice showing the path for 3-D printing using *MOSTMetalCura*.

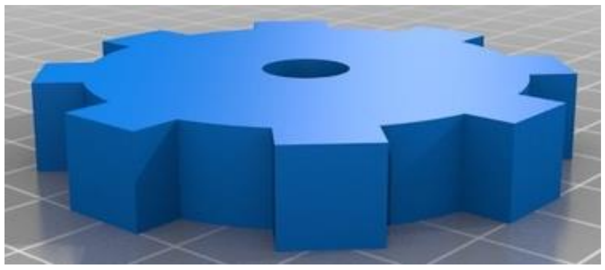
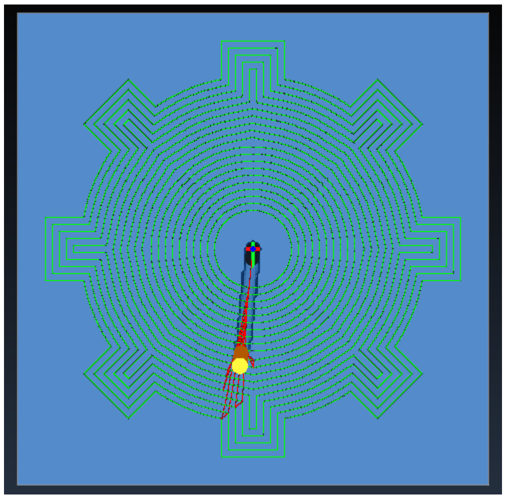



Figure 3.8 a) 3-D model of gear (grid lines are every 10mm) and b) a slice showing the path for 3-D printing using *MOSTMetalCura*.

All experiments are conducted using Millimatic 190 MIG welder. The distance between the nozzle and the substrate should be around 8 to 10 millimeters. The use of RC25 shield gas, which is a mix of 25% carbon dioxide (CO_2) and 75% argon (Ar), would be tested at different level of pressures (20-60 CFH). Temperature and humidity would be measured using digital temperature and humidity monitor (±0.05 Celsius and ±0.5 Percent) and assessed. When the printing speed is set in the configuration file, then the wire feed speed and voltage setting would be calculated and put in comment of a generated G-code file by the *MOSTMetalCura*. The ER70S-6 steel welding wire with 0.024 inches or 6 millimeters diameter is used in all experiments. According to preliminary experiments [45], the recommended voltage setting for the welder is 5. The quality of the 3-D printed objects was quantified with digital calipers (±0.1 mm).

3.5 Results and Discussion

The output of the slicer program is a G-code file. The header would contain important information as in the following text:

```
;Generated with Cura_SteamEngine MOSTMetalCura
;////////////////////////////////////////
;Infill line width: 0.98 mm.
;Layer height: 1 mm.
;Printing speed: 7 mm/s
```

```
;Material diameter: 0.6 mm.
;Expected Material: 6.86 mm3/s.
;/////////////////////////////////////
;Recommended welder (Millermatic 190) settings
;Voltage setting: 5
;Wire speed: 19.61 - 21.61
;/////////////////////////////////////
```

These are comments in the G-code file which gives information about line width, layer height, printing speed, material diameter, expected material in mm^3 per second, and the recommended voltage and wire feed speed settings for the welder, specifically for Millermatic 190 MIG welder.

The line width is set in the configuration file and is set close to one millimeter, but a little smaller or a little bigger line width, ±0.02 range, might help fit all paths in one layer better. The paths can be viewed by one of the G-code viewer, such as the open source CAMOtics [46].

With the setting of 1 millimeter layer height, 0.98 millimeter bead width, and the printing speed of 7 millimeters per second, the calculated wire feed scale is 25.3 mm/s. The printed results of the gear model were poor (see Figure 3.9). The height of the printed dots was measured by digital caliper to about 2 millimeters (±0.1). However, the diameter of the printed gear was measured to about 60 millimeters as expected. According to these results, the layer height was change to 2 millimeters and the rest of the parameters remain the same, the new calculated wire feed scale was about 30 and the printed results were good as shown in Figure 3.10.

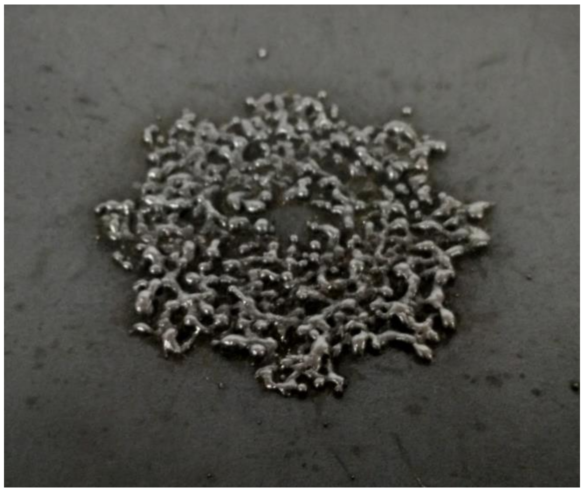


Figure 3.9 The result of printed gear (60 mm diameter) with 1 mm layer height.

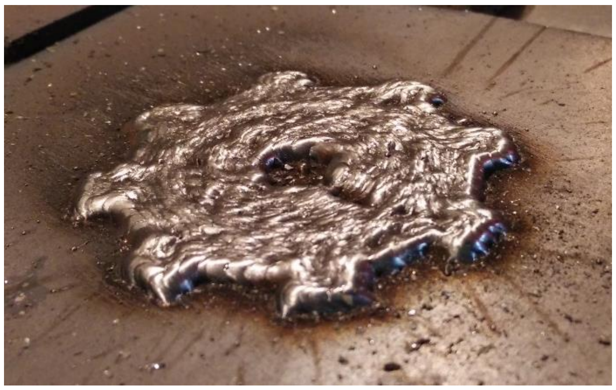


Figure 3.10 The result of printed gear (60 mm diameter) with 2 mm layer height.

A problem with higher speed is rough printed surface for higher layers. The printing speed at 10 mm per second causes a surface non-uniformity after the third layer (see Figure 3.11). Another problem is that it is not easy to set wire speed on the welder to suit every printing speed. For example, if the printing speed is changed to 8 mm per second, then the wire speed number would need to be around 34.1 which is hard to set it to be the same every time because the wire feed speed setting on the welder does not provide scale details between the tens (10-100). It should be noted that users can add a finer physical scale to the welder to more accurately set the wire speed.

The default configuration of slicing for plastic 3-D printing is using *Lines* pattern which would cause more turning off the welder and traveling in each layer for metal 3-D printing.

In the default slicing, there is no pause between layers, so the heat would be accumulated as the printing is ongoing and that would cause an unexpected shape and poor quality of surface.

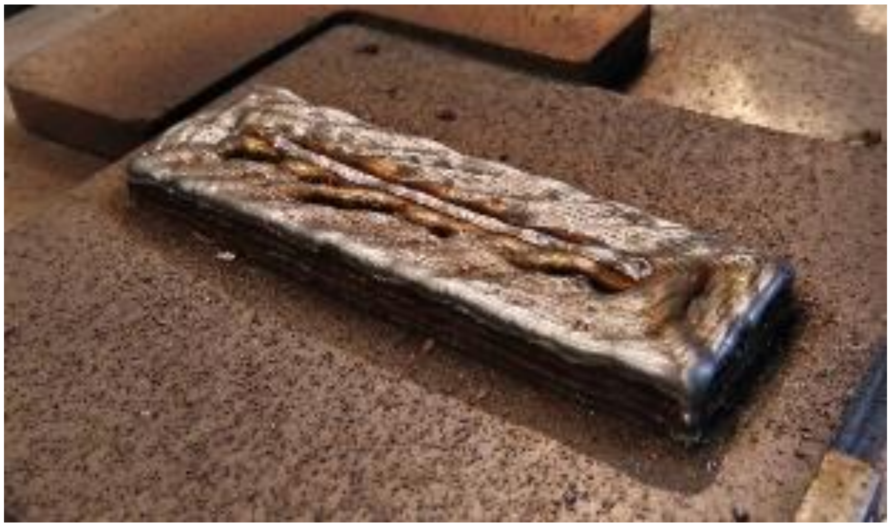


Figure 3.11 Poor surface quality after third layer of printing speed 10 mm/s on block (101.6 mm × 31.75 mm).

When the optimization of settings was found: a layer height of 2 millimeters, line width of 1 millimeter, printing speed of 7 millimeters per second, voltage setting of 5, and wire feed scale of 30. The steel 3-D printed of a few layers of block, chisel and gear are shown in Figure 3.12-3.14. The few layers of printed objects are shown here to demonstrate visual beads/ line widths. The finished 3-D printed gear is shown in Figure 3.15 and the machined gear is shown in Figure 3.16.

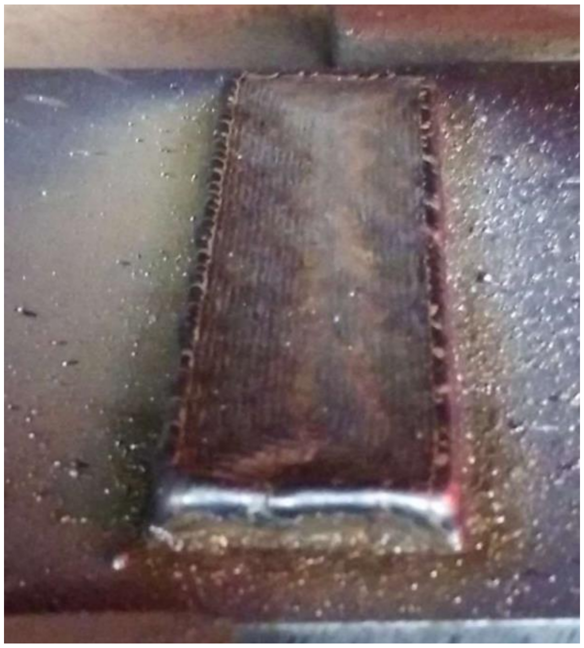


Figure 3.12 3 layers of 3-D printed block (101.6 mm × 31.75 mm).

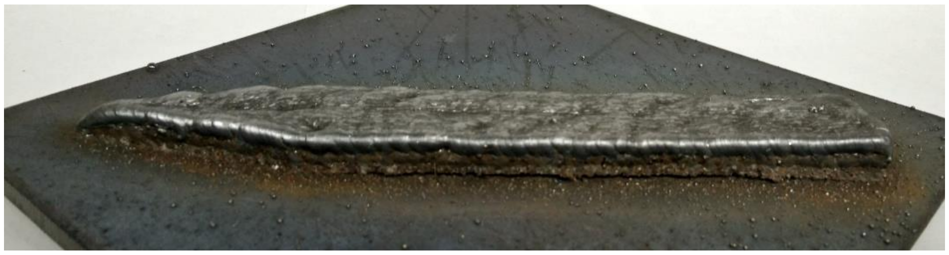


Figure 3.13 3 layers of 3-D printed chisel (140 mm × 20 mm × 20 mm).

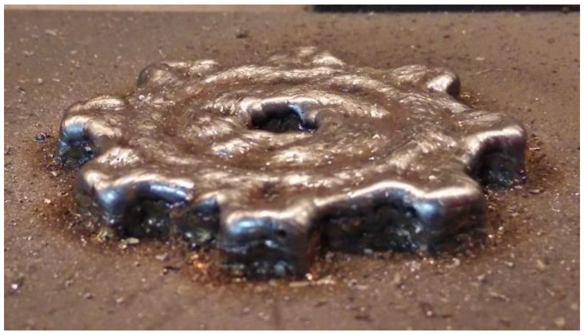


Figure 3.14 3 layers of 3-D printed gear (60 mm diameter).

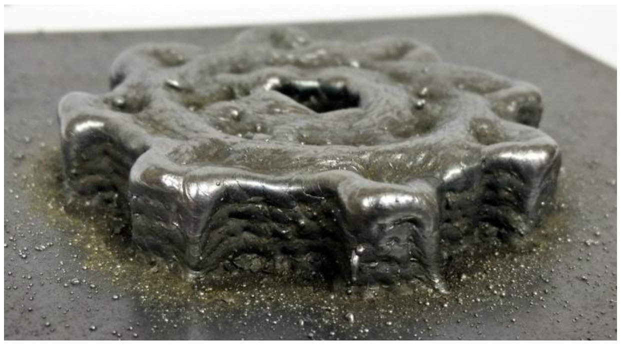


Figure 3.15 The finished 3-D printed gear (60 mm diameter) (5 layers).

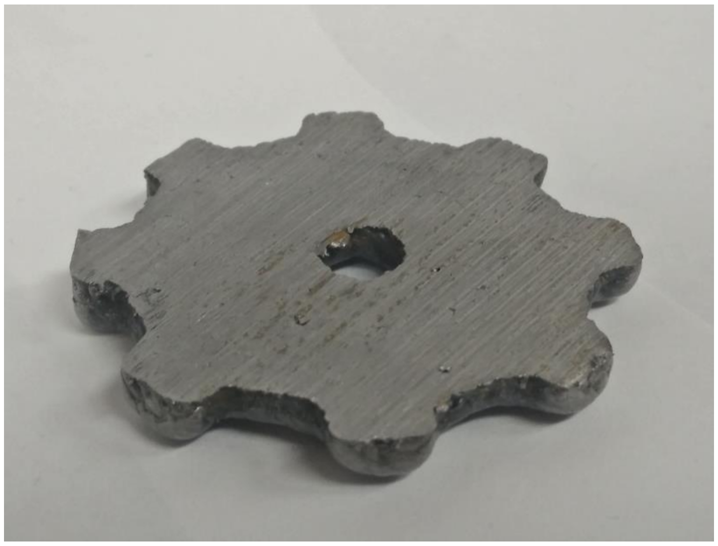


Figure 3.16 The finished machined gear (60 mm diameter).

From Figure 3.14 and 3.15, the surface of the printed part gets rougher as the layer number is increased, because of the substrate effect. Specifically, any imperfection on the lower layer will be continuously propagated in all the following layers often this imperfection is aggravated. This still enables near net shape printing as seen in Figure 3.16, which shows the gear after machining to be functional.

Several issues can arise in GMAW-based 3-D printing, which can be overcome with understanding of the shield gas settings, printer geometries, and heat loss from the substrates as shown below.

The shield gas flow rate was found to be optimized at 25 CFH (cubic foot per hour). If the shield gas flow rate is too low it will affect the quality of welding bead and adhesion due to not enough shield gas to cover the welding area from exposure to oxygen and nitrogen in the air. On the other hand, too high shield gas flow rate will cause a rough surface of the printed part because the melting metal can be blown away from the intended location [47]. The correct shield gas flow rate will help with spatter control, heat control, and adhesion quality [48]. The weld gun needs to be perpendicular to the substrate, so the shield gas can cover around the welding area. If the weld gun is inclined to one side of the nozzle, then some of shield gas will be blocked by the nozzle and cause brown oxide problem on one side of the printed part as shown in Figure 3.17. This error can be caused by the weight of the cord that connected to the weld gun pulling the weld gun down to lean to one side of the nozzle. In order to fix this problem, a mechanical support was added to hold the cord of the weld gun or a more rigid gun holding assembly can be used.

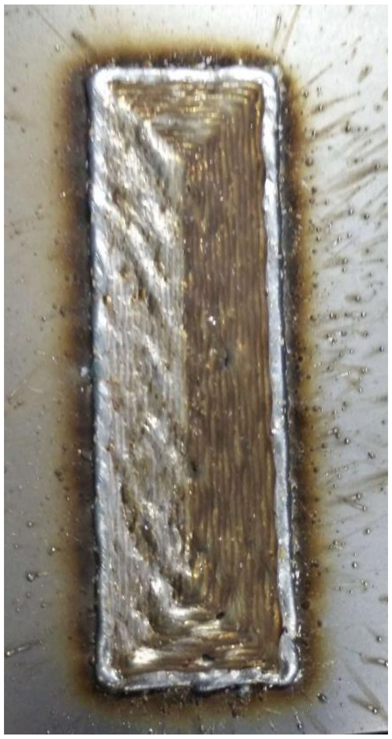


Figure 3.17 With optimal settings, brown oxide due to one side blocked shield gas (101.6 mm × 31.75 mm).

Brown oxidation during the printing can also occur from high humidity printing environments, leaking gas lines, or old gas tanks. Dehumidifcation of the printing room, keep the room at certain and suitable temperature (25 degree Celsius), non-porous gas lines and new RC25 gas tanks can reduce oxidation to next to nothing.

Substrate deformation can also be a problem if heat is built up during printing as shown in Figure 3.18. When a substrate is bending during GMAW 3-D printing, it can deform the printed part and in extreme cases destroy the print. To help reduce the heat from the substrate, the aluminum plate (thickness of 0.249 inches or 6.325 mm) was place on the cement board under the substrate. The results of experiments show that adding aluminum plate helped reduce and eliminate substrate deformation. However, the larger the part being printed, the longer the delay in-between layers to dissipate heat built up in the part from GMAW deposition.

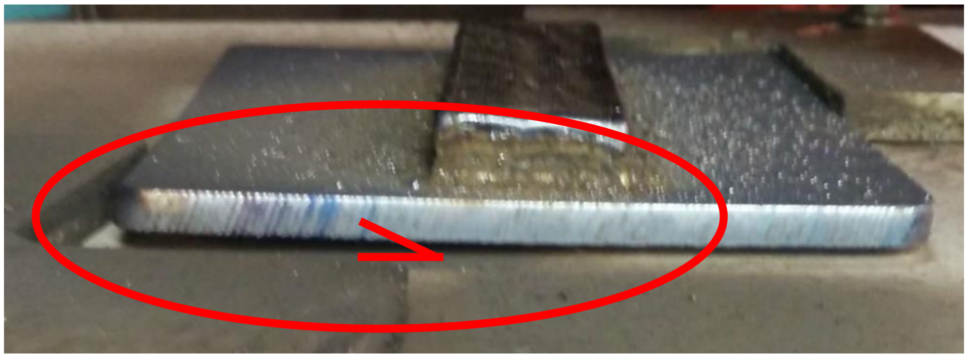


Figure 3.18 Thermal induced substrate deformation which is indicated by the angle symbol (101.6 mm × 31.75 mm).

If the temperature of the part is too high during printing it will increase the surface roughness of the next layer. As more layers are printed, the heat transfer from the printed part will become slower, so the pause after the layer needs to be longer. The heat is trapped in the center of the printed part, so the effectiveness of the aluminum plate is reduced. In *MOSTMetalCura*, the pause time can be set in the configuration file, but the needed pause time is variable and dependent on the size of a printed part and is not known for the first time of printing and object. So the pause time between layers was set to 60 seconds in *MOSTMetalCura*, then it was manually paused and resumed in *Franklin* between layers. During the pause time, the nozzle can be checked and cleaned to make sure it is not clogged with spatter that can block the shield gas.

According to the fact that the wire feed speed cannot be automatically controlled by the metal 3-D printer due to the use of a low-cost welder, the fixed width tool track or fixed bead width is used for metal 3-D printing. The change of the speed can change the width of the bead, but the shape of the bead cannot be controlled to be repeatable. To be able to print the wanted shape the bead width is minimized, which in these experiments was found to be about 1 millimeter. However, the smaller the bead width the more tool tracks are necessary to cover a given area. In addition, to increasing print time this also adds more heat to the part during the welding process. Thus, printing large models is still challenging due to the heat problem inside a printed part when the surface and volume ratio becomes less and less as more layers added. A 3-D steel printed part keeps the heat inside itself and slowly releases the heat. It is much slower compared to the heat release rate of the carbon steel substrate that is on an aluminum plate. It takes about 5 minutes for a substrate to release the heat, but it took about 20 minutes for a layer of the printed block to cool to about 30^{o}C. Future work could utilize an active cooling system to reduce the pause time between layers and accelerate the part production process.

A complexity print geometry has a significant impact to the printing quality with GMAW 3-D printing. A 3-D model with a lot of small details, a lot of small different curves or patterns in each layer or parts that smaller than 1 millimeter results in low quality of surface and shape because the limitation of the smallest bead width is 1 millimeter. The diameter of the welding wire (0.6 mm) limits the bead width as the welding is melting the wire on to a substrate so there would always be some spreading out of that melting steel. The smallest of good welding beads in experiments using the setup and materials described is about 1 millimeter. For an overhang or a bridge in a model, a support can be added, but it might require machining to remove the support or the use of different metals that can be preferentially removed by a chemical or high temperature process. Thus, many of the free and open-source 3-D models available on-line are not suitable for metal 3-D printing. The size of the welding bead also causes the smoothness between layers is also deficient resulting in a staircase effect would be seen on a printed part of complicated 3-D models with a shallow slope of less than 45 degrees. For example, consider a beveled gear [49]. When printed at 30 degrees it has an obvious staircase effect and shape problem when it was metal 3-D printed (see Figure 3.19). When the angle is changed to 45 degrees the stair case effect is reduced but it still has a shape problem at the top part due to the gear teeth being to small (see Figure 3.20). Both examples have gap problem on the surface because the size of the models are not in full millimeter. However, the author provided an SCAD file, that the user can edit to increase the size of the beveled gear and generate a new model appropriate for GMAW metal 3-D printing. This can be seen in Figure 3.21. After the beveled gear is printed, it would require more machining until the gear is appropriate to use for a high tolerance application. Another option would be design your own 3-D model for metal 3-D printing with an open-source software 3-D CAD modeler, such as OpenSCAD [50]. The 3-D model for metal 3-D printing should be in full millimeter of length and if there is a slope it should not be less than 45 degrees if this is possible option. For an overhang or a bridge in a model, a support can be added, but it might require machining to remove the support.

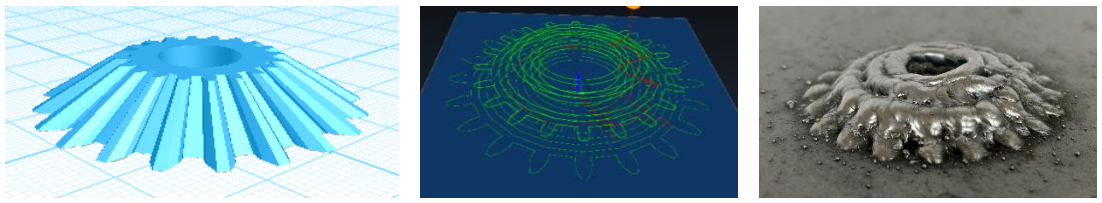


Figure 3.19 A beveled gear 3-D model with 30 degrees angle (grid lines are every 10mm), its generated path and metal 3-D printed.

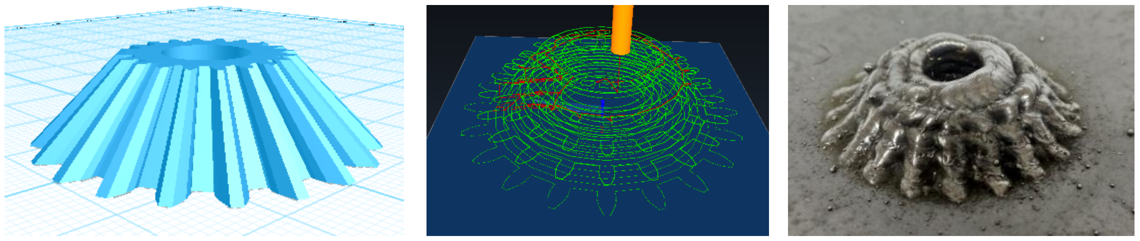


Figure 3.20 A beveled gear 3-D model with 45 degrees angle (grid lines are every 10mm), its generated path and metal 3-D printed.

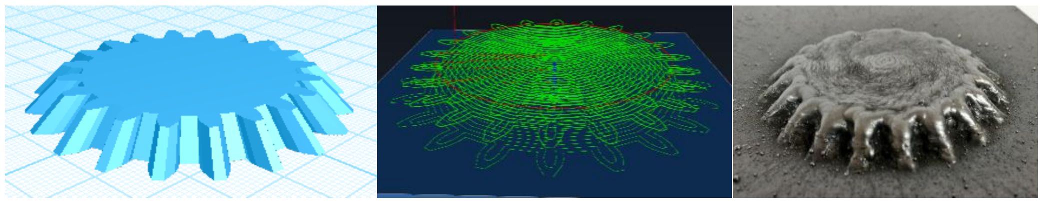


Figure 3.21 A beveled gear 3-D model (grid lines are every 10mm), its generated path and metal 3-D printed.

For the future work, it would be great if the layer height of each layer can be reduced. One option is to mill out the surplus layer height [51-55], which helps reduce the layer height and also get rid of the rough surface which results in a better quality. However, this will increase the printing time in total and create more waste. Methods to extract heat inside a steel printed part during printing needs to be researched more in order to mitigate or solve the problem. The most obvious solution to this problem is to use this technology primarily for larger parts.

3.6 Conclusions

The open-source slicer software called *MOSTMetalCura* for metal 3-D printing has been implemented successfully and reported in this paper. It was customized from the open-source slicing engine named *CuraEngine*. Many customizable settings were added to the new slicer to support the metal 3-D printing. The output G-code is specifically for the low-cost open-source metal 3-D printer developed by MOST, however, it can be adjusted to support other metal 3-D printers as well. The paper also reports how to optimize the settings for the metal 3-D printing. The optimized settings for the low-cost open-source metal 3-D printer are 7 millimeters per second printing speed, 2 millimeters of layer height, about 1 millimeter of bead width, 5 of welding voltage setting, 28-31 on scale of wire feed speed, and 25 CFH of RC25 shield gas. The result shows the high resolution of 3-D printed steel simple gear.

3.7 References

1 Gebhardt, Andreas, (2003) Rapid Prototyping. Hanser, Munich.

2 Noorani R, (2006) Rapid prototyping: principles and applications. John Wiley & Sons Incorporated.

3 Cooper K, (2001) Rapid prototyping technology: selection and application. CRC press.

4 Islam MN, Pramanik A, Slamka S, (2016) Errors in different geometric aspects of common engineering parts during rapid prototyping using a Z Corp 3D printer. Progress in Additive Manufacturing, 1-9. doi: 10.1007/s40964-016-0006-7

5 Boparai KS, Singh R, Singh H (2016) Modeling and optimization of extrusion process parameters for the development of Nylon6–Al–Al2O3 alternative FDM filament. Progress in Additive Manufacturing, 1-14. doi: 10.1007/s40964-016-0011-x

6 Bak D (2003) Rapid prototyping or rapid production? 3D printing processes move industry towards the latter. Assembly Automation, 23(4), 340-345.

7 Petrick IJ, Simpson TW (2013) 3D printing disrupts manufacturing: how economies of one create new rules of competition. Research-Technology Management, 56(6), 12-16.

8 Lasagni F, Vilanova J, Periñán A, Zorrilla A, Tudela S, Gómez-Molinero V (2016) Getting confidence for flying additive manufactured hardware. Progress in Additive Manufacturing, 1-11. doi: 10.1007/s40964-016-0014-7

9 Lipson H, Kurman M (2013) Fabricated: The new world of 3D printing. John Wiley & Sons.

10 Wittbrodt BT, Glover AG, Laureto J, Anzalone GC, Oppliger D, Irwin JL, Pearce JM (2013) Life-cycle economic analysis of distributed manufacturing with open-source 3-D printers. Mechatronics, 23(6), 713-726.

11 Mota C (2011, November) The rise of personal fabrication. In Proceedings of the 8th ACM conference on Creativity and cognition, 279-288. ACM.

12 Gwamuri J, Wittbrodt BT, Anzalone NC, Pearce JM (2014) Reversing the Trend of Large Scale and Centralization in Manufacturing: The Case of Distributed Manufacturing of Customizable 3-D-Printable Self-Adjustable Glasses. Challenges in Sustainability, 2(1), 30-40.

13 King DL, Babasola A, Rozario J, Pearce JM (2014) Development of mobile solar photovoltaic powered open-source 3-D printers for distributed customized manufacturing in off-grid communities. Challenges in Sustainability, 2(1), 18-27.

14 Laeng J, Stewart JG, Liou FW (2000) Laser metal forming processes for rapid prototyping-A review. International Journal of Production Research, 38(16), 3973-3996.

15 Lewis GK, Schlienger E (2000) Practical considerations and capabilities for laser assisted direct metal deposition. Materials & Design, 21(4), 417-423.

16 Santos EC, Shiomi M, Osakada K, Laoui T (2006) Rapid manufacturing of metal components by laser forming. International Journal of Machine Tools and Manufacture, 46(12), 1459-1468.

17 Delgado J, Ciurana J, Serenó L (2011) Comparison of forming manufacturing

processes and selective laser melting technology based on the mechanical properties of products: In this work, the superior property of the selective laser melting technology is presented by comparing four real parts manufactured using forming processes and selective laser melting technology and analysed for tension, compression and flexural. Virtual and Physical Prototyping, 6(3), 167-178.

18 Cooper F (2016) Sintering and additive manufacturing:"additive manufacturing and the new paradigm for the jewellery manufacturer". Progress in Additive Manufacturing, 1-15. doi: 10.1007/s40964-015-0003-2

19 Heinl P, Rottmair A, Körner C, Singer RF (2007) Cellular titanium by selective electron beam melting. Advanced Engineering Materials, 9(5), 360-364.

20 Gaytan SM, Murr LE, Medina F, Martinez E, Lopez MI, Wicker RB (2009) Advanced metal powder based manufacturing of complex components by electron beam melting. Materials technology, 24(3), 180-190.

21 Murr LE, Gaytan SM, Ramirez DA, Martinez E, Hernandez J, Amato KN, Wicker RB (2012) Metal fabrication by additive manufacturing using laser and electron beam melting technologies. Journal of Materials Science & Technology, 28(1), 1-14.

22 Peels J (2014) Metal 3D printing: From lab to fab. Inside 3DP. http://www.inside3dp.com/metal-3d-printing-lab-fab/. Accessed 23 May 2014

23 Sells E, Smith Z, Bailard S, Bowyer A, Olliver V (2009) RepRap: The Replicating Rapid Prototyper: Maximizing Customizability by Breeding the Means of Production. in F. T. Piller, M. M. Tseng (Eds.), Handbook of Research in Mass Customization and Personalization: Strategies and concepts. World Scientific, 1:568-580.

24 Jones R, Haufe P, Sells E, Iravani P, Olliver V, Palmer C, Bowyer A (2011) RepRap–the replicating rapid prototyper. Robotica, 29(01), 177-191.

25 Bowyer A (2014) 3D printing and humanity's first imperfect replicator. 3D printing and additive manufacturing, 1(1), 4-5.

26 Anzalone GC, Zhang C, Wijnen B, Sanders PG, Pearce JM (2013) A low-cost open-source metal 3-D printer. IEEE Access, 1, 803-810.

27 Banzi M, Shiloh M (2014) Getting Started with Arduino: The Open Source Electronics Prototyping Platform. Maker Media, Inc..

28 Arduino - Home (2015) https://www.arduino.cc/. Accessed 1 November 2015

29 Haselhuhn AS, Gooding EJ, Glover AG, Anzalone GC, Wijnen B, Sanders PG, Pearce JM (2014) Substrate release mechanisms for gas metal arc weld 3D aluminum metal printing. 3D Printing and Additive Manufacturing, 1(4), 204-209.

30 Haselhuhn AS, Wijnen B, Anzalone GC, Sanders PG, Pearce JM (2015) In situ formation of substrate release mechanisms for gas metal arc weld metal 3-D printing. Journal of Materials Processing Technology, 226, 50-59.

31 Pinar A, Wijnen B, Anzalone GC, Havens TC, Sanders PG, Pearce JM (2015) Low-cost open-source voltage and current monitor for gas metal arc weld 3D printing. Journal of Sensors. doi:10.1155/2015/876714

32 Nilsiam Y, Haselhuhn A, Wijnen B, Sanders P, Pearce JM (2015) Integrated Voltage—Current Monitoring and Control of Gas Metal Arc Weld Magnetic Ball-Jointed Open Source 3-D Printer. Machines, 3(4), 339-351.

33 Wijnen B, Anzalone GC, Haselhuhn AS, Sanders PG, Pearce JM (2016) Free and open-source control software for 3-D motion and processing. Journal of Open

Research Software, 4(1).
34 Haselhuhn AS, Buhr MW, Wijnen B, Sanders PG, Pearce JM (2016) Structure-property relationships of common aluminum weld alloys utilized as feedstock for GMAW-based 3-D metal printing. Materials Science and Engineering: A, 673, 511-523.
35 Cura 3D Printing Slicing Software (2015) https://ultimaker.com/en/products/cura-software. Accessed 1 November 2015
36 Slic3r (2015) http://slic3r.org/. Accessed 1 November 2015
37 Ultimaker/CuraEngine (2015) https://github.com/Ultimaker/CuraEngine. Accessed 1 November 2015
38 Laplume AO, Petersen B, Pearce JM (2016) Global value chains from a 3D printing perspective. Journal of International Business Studies. doi:10.1057/jibs.2015.47
39 Berman B (2012) 3-D printing: The new industrial revolution. Business horizons, 55(2), 155-162. https://doi.org/10.1016/j.bushor.2011.11.003
40 Cura - RepRapWiki (2016) http://reprap.org/wiki/Cura. Accessed 4 March 2016
41 daid/Cura (2016) https://github.com/daid/Cura. Accessed 4 March 2016
42 G-code - RepRapWiki (2015) http://reprap.org/wiki/G-code. Accessed 12 October 2015
43 Miller MIG Welders - MIG Welding and GMAW Welding Machines - MillerWelds (2015) https://www.millerwelds.com/equipment/welders/mig-gmaw/millermatic-190-mig-welder-m00487. Accessed 18 October 2015.
44 MOST metal slicing examples https://osf.io/6u5sp/ Accessed 26 January 2017.
45 Clark, R., Lund, P., Sanders, P., Pearce, JM. Weld bead performance metrics for arc weld-based additive manufacturing of ER70S-2 steel and 316L stainless steel (to be published).
46 CAMotics (2015) http://camotics.org/. Accessed 5 December 2015
47 Lancaster JF (1984) The physics of welding. Physics in technology, 15(2), 73.
48 Suban M, Tušek J (2001) Dependence of melting rate in MIG/MAG welding on the type of shielding gas used. Journal of Materials Processing Technology, 119(1), 185-192.
49 Thornburg, D. (2014). https://www.youmagine.com/designs/beveled-gear. Accessed 20 September 2016
50 OpenSCAD (2016) http://www.openscad.org/. Accessed 4 March 2016
51 Choi DS, Lee SH, Shin BS, Whang KH, Song YA, Park SH, Jee HS (2001) Development of a direct metal freeform fabrication technique using CO 2 laser welding and milling technology. Journal of Materials Processing Technology, 113(1), 273-279.
52 Xiong X, Haiou Z, Guilan W (2008) A new method of direct metal prototyping: hybrid plasma deposition and milling. Rapid Prototyping Journal, 14(1), 53-56.
53 Krassenstein B (2014) VDK6000, Incredible 6-axis Metal 3D Printer, Milling Machine, Laser Scanner Unveiled. https://3dprint.com/10079/vdk6000-robotic-work-environment/. Accessed 14 November 2016
54 Anderton J (2014) 3D Printing and 5-Axis Machining Combined in One Machine ENGINEERING.com. http://www.engineering.com/AdvancedManufacturing/ArticleID/8778/3D-Printing-and-5-Axis-Machining-Combined-in-One-Machine.aspx. Accessed 14 November

2016
55 Sodick Inc (2016) One-Process Metal 3D printing & milling machine - Today's Motor Vehicles. http://www.todaysmotorvehicles.com/product/sodick-3d-print-milling-machine-092916/.

3.8 Pseudocode of the Core Functions of MOSTMetalCura

```
FUNCTION main
    IF machine is Linux or Mac THEN
        Lower the process priority on Linux and Mac machine
    ENDIF
    Display copyright information
    FOR each input argument from command line
        IF argument start with '-' THEN
            IF argument is "--connect" THEN
                Connect to command socket
            ELSE
                SWITCH argument of
                    'h': Print usage information
                    'j': Load JSON configuration file
                    'p': Enable progress logging
                    'o': Set target or output file
                    's': Save next argument as setting
                    default: Unknow option
                ENDSWITCH
            ENDIF
        ELSE
            file = 3-D model file name
        ENDIF
    IF file is existed THEN
        CALL fffProcessor.processFIles to process the file
    ENDIF
    CALL fffProcessor.finalize to add end.gcode and report statistics
    RETURN 0
ENDFUNCTION

FUNCTION fffProcessor.processFiles
    CALL modelFile.loadMeshFromFile to load 3-D model file as mesh file
    CALL fffProcessor.processModel
ENDFUNCTION

FUNCTION fffProcessor.processModel
    CALL fffProcessor.preSetup to set up extrusion offset, code for switch extruder,
    filament diameter, G-code format, retraction amount
    CALL fffProcessor.prepareModel to slice model into layerparts
    CALL fffProcessor.processSliceData to generate insets, supports, skins, skirt, raft
    CALL fffProcessor.writeGCode
ENDFUNCTION

FUNCTION fffProcessor.preSetup
```

```
    Set machine extruder offset
    Set code for before and after switch extruder
    Set filament diameter
    Set G-code flavor format
    Set retraction amount
ENDFUNCTION


FUNCTION fffProcessor.prepareModel
    Create slicer object to slice model, Slicer::Slicer
    CALL layerPart.createlayerParts to create layerParts
ENDFUNCTION


FUNCTION Slicer::Slicer
    Find all segment in each layer, face and mesh points
    CALL SlicerLayer::makePolygons to generate polygons for each layer
ENDFUNCTION


FUNCTION SlicerLayer::makePolygons
    Generate polygons from face and mesh
    Connecting any open polygons
    Link up all the missing ends
    Close the gaps
    Remove the tiny polygons or still open polygons
    Optimize the polygons by removing unnecessary points
    CALL polygonOptimizer.optimizePolygons
ENDFUNCTION


FUNCTION layerPart.createLayerParts
    CALL layerPart.createlayerWithParts
ENDFUNCTION


FUNCTION layerPart.createLayerWithParts
    Split polygons in to part or island
    Store all parts
ENDFUNCTION


FUNCTION fffProcessor.processSliceData
    Generate insets or perimeters by CALL inset.generateInsets
    Generate support area
    Generate skins by CALL skin.generateSkins
    Combine all layerparts by CALL skin.combineSparseLayers
    Generate skirt
    Generate raft
ENDFUNCTION
```

```
FUNCTION inset.generateInsets

        Create inset for each layerpart, inset width is wall line width

        Remove parts with no inset because they are too small

ENDFUNCTION


FUNCTION skin.generateSkins

        CALL skin.generateSkinAreas to generate skin

        CALL skin.generateSkinInsets to generate inset for each skin

ENDFUNCTION


FUNCTION skin.generateSkinAreas

        Generate skin by union all polygons in the layer

ENDFUNCTION


FUNCTION skin.generateSkinInsets

        Generate insets for skin

ENDFUNCTION


FUNCTION skin.combineSparseLayers

        Combine all parts together for the layer, 100% filled

ENDFUNCTION
```

```
FUNCTION fffProcessor.writeGCode
    Reset total print time and filament
    Setup retraction parameters
    IF RepRap style G-code THEN
        Write bed temperature command G-code
        Write temperature command for material
        Write start G-code
        IF metal 3-D printing THEN
            Load welder on G-code
            Load welder off G-code
            Set minimum distance welder off
            Initiate welding status to false
        ENDIF
        Write comment for software version, printing line width, layer height,
        printing speed, material diameter
        Calculate wire feed speed based on line width, layer height, print speed,
        material diameter.
        Write comment for voltage setting on the welder
        Write comment for recommend wire feed speed range
        IF raft is true THEN
            Load configures for the raft, speed, line width, layer height, material
            flow
            Generate raft and its path
            Write G-code for raft
        ENDIF
    ENDIF
    Load pause time between layer setting
    Load increase time for each layer setting
    Load move up height at the end of each layer setting
    Load welder off G-code setting
    Load layer pause Boolean setting
    FOR each layer
        Load layer height setting
        Load skirt setting, speed, line width, material flow, thickness
        Load support setting, speed, line width, material flow, thickness
        FOR each mesh
            Load mesh inset setting, line width, speed, material flow, layer
            height
            Load mesh skin setting, line width, speed, material flow, layer
            height
            Load mesh infill setting, line width, speed, material flow, layer
            height
        ENDFOR
        Write comment for layer number
        Create GCodePlanner object
```

```
Set z value for the object
Reset the start position
IF layer number == 0 THEN
    IF skirt size > 0 THEN
        Add travel to print skirt at the closest point
        Add polygons to the object to print
    ENDIF
ENDIF
IF print support first THEN
    Add support to the object to print
ENDIF
CALL fffProcessor.calculateMeshOrder to sort the meshes, start from the
closest mesh to the extruder
FOR each mesh in the sorted meshes
    CALL fffProcessor.addMeshLayerToGCode
ENDFOR
IF not print support first THEN
    CALL fffProcessor.addSupportToGCode
ENDIF
Force minimum layer time
Set fan speed
IF layer pause is true THEN
```

```
                Write G-code to turn off welder
                Write G-code to move the print head up
                Calculate the pause time
                Write G-code to set the pause time
                Set welding status to false
            ENDIF
        ENDFOR
        Write G-code retraction
        Write G-code fan command
ENDFUNCTION


FUNCTION fffProcessor.calculateMeshOrder
        Add closest meshes first then the next closest
ENDFUNCTION


FUNCTION fffProcessor.addMeshLayerToGCode
        Create partOrderOptimization object
        FOR each layerpart
            Add layerpart to the object by CALL partOrderOptimizer.addPlygon
        ENDFOR
        CALL partOrderOptimizer.optimize to optimize part order
        FOR each layerpart
```

```
Set fill angle
IF infill line distance > 0 THEN
    SWITCH fill pattern
        Fill_Grid:      CALL infill.generateGridInfill
                        CALL gcodePlanner.addLinesByOptimizer
        Fill_Lines:     CALL infill.generateLineInfill
                        CALL gcodePlanner.addLinesByOptimizer
        Fill_Triangles: CALL infill.generateTriangleInfill
                        CALL gcodePlanner.addLinesByOptimizer
        Fill_Concentric: CALL infill.generateConcentricInfill
                CALL gcodePlanner.addPolygonsByOptimizer
        Fill_ZigZag:    CALL infill.generateZigZagInfill
                CALL gcodePlanner.addPolygonsByOptimizer
        Default:        break
    ENDSWITCH
    CALL fffProcessor.sendPolygons
ENDIF
IF infill line distance > 0 and layer part outline size > 0 THEN
    Combine infill and top/bottom skin together
    SWITCH fill pattern
        Fill_Grid:      CALL infill.generateGridInfill
                        CALL gcodePlanner.addLinesByOptimizer
```

```
            Fill_Lines:     CALL infill.generateLineInfill
                            CALL gcodePlanner.addLinesByOptimizer
            Fill_Triangles: CALL infill.generateTriangleInfill
                            CALL gcodePlanner.addLinesByOptimizer
            Fill_Concentric: CALL infill.generateConcentricInfill
                  CALL gcodePlanner.addPolygonsByOptimizer
            Fill_ZigZag:    CALL infill.generateZigZagInfill
                  CALL gcodePlanner.addPolygonsByOptimizer
            Default:        break
      ENDSWITCH
      CALL fffProcessor.sendPolygons
ENDIF
CALL gcodePlanner.addPolygonsByOptimizer
CALL gcodePlanner.addLinesByOptimizer
IF wall line count > 0 THEN
      FOR each inset
            IF inset == 0 THEN
                  CALL gcodePlanner.addPolygonsByOptimizer with
                  inset0
            ELSE
                  CALL  gcodePlanner.addPolygonByOptimizer  with
                  insetX
            ENDIF
```

```
        ENDFOR
    ENDIF
    FOR each layerpart in skin
        SWITCH top and bottom pattern
            Fill_Lines:
                FOR each inset in layerpart
                    gcodePlanner.addPlygonsByOptimizaer
                    infill.generateLineInfill
                ENDFOR
            Fill_Concentric:
                infill.generateConcentricInfillDense
            default: break
        ENDSWITCH
    ENDFOR
    IF fill perimeter gaps != “Nowhere” THEN
        CALL infill.generateLineInfill to fill the gap
    ENDIF
    CALL gcodePlanner.addPolygonsByOptimizer
    CALL gcodePlanner.addLinesByOptimizer
    Check to make sure that the nozzle is inside comb boundary
ENDFOR
```

```
ENDFUNCTION

FUNCTION partOrderOptimizer.addPolygon
    Add polygon to the vector
ENDFUNCTION

FUNCTION partOrderOptimizer.optimize
    FOR each polygon
        Find the closest point to start in each polygon
    ENDFOR
    FOR each polygon
        Find the closest polygon to continue
    ENDFOR
    FOR each polygon
        CALL partOrderOptimizer.getClosestPointInPlolygon to find starting point
        in each polygon
    ENDFOR
ENDFUNCTION

FUNCTION infill.generateGridInfill
    CALL infill.generateLineInfill
    CALL infill.generateLineInfill with rotation of 90 degree
```

```
ENDFUNCTION


FUNCTION gcodePlanner.addLinesByOptimizer
    CALL LineOrderOptimizer to set start point
    FOR each polygon
        CALL LineOrderOptimizer.addPolygon
    ENDFOR
    FOR each polygon
        CALL gcodePlanner.addPolygon to add optimized polygon
    ENDFOR
ENDFUNCTION


FUNCTION infill.generateLineInfill
    FOR each polygon
        FOR each point in polygon
            Find the next point in scanline
        ENDFOR
    ENDFOR
    CALL infill.addLineInfill to fill the polygon
ENDFUNCTION


FUNCTION infill.generateTriangleInfill
```

```
    CALL infill.generateLineInfill
    CALL infill.generateLineInfill with rotation of 60 degree
    CALL infill.generateLineInfill with rotation of 120 degree
ENDFUNCTION

FUNCTION infill.generateConcentricInfill
    WHILE outline.size > 0
        FOR each outline
            Add outline to plolygon
        ENDFOR
    ENDWHILE
ENDFUNCTION

FUNCTION gcodePlanner.addPolygonsByOptimizer
    Create PathOrderOptimizer object
    FOR each polygon
        CALL pathOrderOptimizer.addPolygon
    ENDFOR
    CALL pathOrderOptimizer.optimize
    FOR each polygon
        CALL gcodePlanner.addPolygon
    ENDFOR
```

```
ENDFUNCTION

FUNCTION infill.generateZigZagInfill
    IF endPieces THEN
        Return infill.generateZigZagInfill_endPieces
    ELSE
        Return infill.generateZigZagInfill_noEndPieces
    ENDIF
ENDFUNCTION

FUNCTION partOrderOptimizer.getClosestPointInPlolygon
    FOR each point in polygon
        Find the closest point based on orientation
    ENDFOR
ENDFUNCTION

FUNCTION LineOrderOptimizer.addPolygon
    Add polygon to the object
ENDFUNCTION

FUNCTION gcodePlanner.addPolygon
    FOR each point in the polygon
```

```
        CALL gcodePlanner.addExtrusionMove
    ENDFOR
ENDFUNCTION
FUNCTION infill.addLineInfill
    FOR each line
        Add line to fill the polygon
    ENDFOR
ENDFUNCTION

FUNCTION gcodePlanner.addExtrusionMove
    CALL gcodePlanner.getLatestPathWithConfig
    Set the last positon
ENDFUNCTION

FUNCTION gcodePlanner.getLatestPathWithConfig
    IF path size > 0 THEN
        Return the last path
    ENDIF
    Create GCodePath object and set value
    Return the object
ENDFUNCTION
```

```
FUNCTION infill.generateZigZagInfill_endPieces
    Add line to fill polygon
    Connect lines
    CALL infill.addLineInfill
ENDFUNCTION


FUNCTION infill.generateZigZagInfill_noEndPieces
    Add line to fill polygon
    Connect lines
    CALL infill.addLineInfill
ENDFUNCTION


FUNCTION infill.generateConcentricInfillDense
    WHILE polygon size > 0
        FOR each polygon
            Add polygon to result
        ENDFOR
        CALL polygonUtils.offsetExtrusionWidth
        Next polygon
    ENDWHILE
ENDFUNCTION
```

```
FUNCTION fffProcessor.addSupportToGCode
    Generate support
    CALL pathOrderOptimizer.islandOrderOptimizer
    IF support line distance > 0 THEN
        SWITCH support pattern
            Fill_Grid:
                IF support line distance > extrusion width * 4 THEN
                    CALL infill.generateGridInfill
                ELSE
                    CALL infill.generateLineInfill
                ENDIF
            Fill_Lines:
                IF layer number == 0 THEN
                    CALL infill.generateGridInfill
                ELSE
                    CALL infill.gemerateLineInfill
                ENDIF
            Fill_ZigZag:
                IF layer number == 0 THEN
                    CALL infill.generateGridInfill
                ELSE
                    CALL infill.gemerateZigZagInfill
                ENDIF

        ENDSWITCH
    ENDIF
ENDFUNCTION

FUNCTION polygonOptimizer.optimizePolygons
    FOR each polygon
        CALL polygonOptimizer.optimizePolygon
    ENDFOR
ENDFUNCTION

FUNCTION polygonUtils.offsetExtrusionWidth
    Distance negative for inward, positive otherwise
    Add distance to polygon offset
ENDFUNCTION

FUNCTION polygonOptimizer.optimizePolygon
    FOR each point in polygon
        Remove point that too shorts
    ENDFOR
ENDFUNCTION
```

```
FUNCTION fffProcessor.finalize
    Load end G-code
    CALL gcodeExport.finalize passing end G-code
ENDFUNCTION

FUNCTION gcodeExport.finalize
    Turn off the fan
    Set Z
    gcodeExport.writeMove to move print head away from the printed part
    gcodeExport.writeCode to write the end G-code
ENDFUNCTION

FUNCTION gcodeExport.writeMove
    Get extrusion amount
    IF flavor == GCODE_FLAVOR_BFB
        Calculate speed
        Calculate rpm
        Calculate mm per rpm
        IF rpm >0 THEN
            Calculate extrusion amount
        ELSE
```

```
            Retraction
        ENDIF
    ELSE
        IF extrusion mm3 per mm > 0.000001 THEN
            IF metal printing THEN
                IF not welding THEN
                    Write G-code welder on
                    Set welding status to true
                ENDIF
            ENDIF
            Calculate extrusion amount
            Write G1 to output
        ELSE
            Travel only
            IF metal printing THEN
                IF welding is true and distance > min distance THEN
                    Set welding status to false
                    Write G-code to turn off the welder
                ENDIF
            ENDIF
        ENDIF
        IF current speed != speed THEN
```

```
                Update current speed with speed
            ENDIF
            Write output G-code
            Update current position
            Update start positon
            Calculate the time left
        ENDIF
ENDFUNCTION

FUNCTION gcodeExport.writeCode
        Wirte G-code to file
ENDFUNCTION

FUNCTION gcodeExport.setWelderOn
        Load welder on G-code
ENDFUNCTION

FUNCTION gcodeExport.setWelderOff
        Load welder off G-cdoe
ENDFUNCTION

FUNCTION gcodeExport.setMinDistWelderOff
```

```
    Load minimum distance welder off
ENDFUNCTION


FUNCTION gcodeExport.setIsMetalPrinting
    Set metal printing Boolean
ENDFUNCTION


FUNCTION gcodeExport.setIsWelding
    Set welding status
ENDFUNCTION


FUNCTION gcodePlanner.writeGCode
    FOR every path
        Change extruder
        Set last configuration
        Load speed setting
        gcodeExport.writeMove
    ENDFOR
    Update the total print time
    IF print head need lifted and there is extra time THEN
        gcodeExport.writeMove to move the print head up
    ENDIF
```

```
ENDFUNCTION


FUNCTION gcodePlanner.addTravel
    Create a pathOrderOptimizer object
    Add travel destination point to the object
ENDFUNCTION


FUNCTION gcodePlanner.moveInsideCombBoundary
    Check that the last position is inside the boundary then return
    IF last position is not inside the boundary THEN
        gcodePlanner.addTravel to move the nozzle inside the boundary
        gcodePlanner.forceNewpathStart
    ENDIF
ENDFUNCTION


FUNCTION gcodePlanner.forceNewpathStart
    IF path size > 0 THEN
        Set the current path done to be true
    ENDIF
ENDFUNCTION


FUNCTION fffProcessor.setTargetFile
```

```
    IF output file is already open THEN
        gcodeExport.setOutputStream with filename from the input argument
    ENDIF
ENDFUNCTION


FUNCTION fffProcessor.setTargetStream
    gcodeExport.setOutputStream with the variable name
ENDFUNCTION


FUNCTION LineOrderOptimizer.optimize
    FOR each polygon
        Find the closest point to initial starting point in each polygon
    ENDFOR
    FOR each polygon
        FOR each line in polygon
            LineOrderOptimizer.checkIfLineIsBest
        ENDFOR
        IF no best single line found THEN
            Skip to next line
        ENDIF
        IF best single line found THEN
            Add the line to the polygon order
        ENDIF
    ENDFOR
    FOR each polygon in polygon order
        Find the best starting point (minimum distance) in each polygon
    ENDFOR
ENDFUNCTION

FUNCTION LineOrderOptimizer.checkIfLineIsBest
    IF distance to the first point is better THEN
        Start fill polygon from the first point
    ENDIF
    IF distance to the second point is better THEN
        Strat fill polygon from the second point
    ENDIF
ENDFUNCTION

FUNCTION skin.generateSparse
    FOR each layerpart
        Create Polygons object as sparse
        Add layerpart inset to the sparse
    ENDFOR
ENDFUNCTION
```

```
FUNCTION skin.generatePerimeterGaps
    FOR each layerpart
        IF down skin count and up skin count > 0 and layer number > down skin
        count and Layer number < layer size – up skin count THEN
            Remove gaps inside the print
        ENDIF
        Remove small area gaps
    ENDFOR
ENDFUNCTION


FUNCTION mesh.addFace
    IF two vertices are the same THEN
        Return, there is no face
    ENDIF
    Create MeshFace object
    Add the three vertices to the MeshFace object
    Add the faces to vertices as well
ENDFUNCTION


FUNCTION mesh.clear
    Clear the relationship between faces and vertices
```

```
ENDFUNCTION


FUNCTION mesh.finish
    FOR each face
        Store other three faces that connected to the face
    ENDFOR
ENDFUNCTION
FUNCTION mesh.min
    Return the minimum point in the vertices
ENDFUNCTION


FUNCTION mesh.max
    Return the maximum point in the vertices
ENDFUNCTION


FUNCTION mesh.findIndexOfVertex
    FOR each point in vertex map
        Return the index of the vertex
    ENDFOR
ENDFUNCTION


FUNCTION mesh.getFaceIdxWithPoints
```

```
    Find the faces that connected to the point
    Return error if no connected face is found
    If only one face connected, then return the face
    If more than two faces connected, then return the counter-clockwise face.
ENDFUNCTION


FUNCTION settingRegistry.loadJSON
    Create rapidjson document object
    Open the specified file
    Check if there is any error, return otherwise continue
    Load all the settings and add them to category
ENDFUNCTION


FUNCTION SettingsBase.setSetting
    IF setting is already existed THEN
        Set new value to the setting
    ELSE
        Warning that it is unregister setting
        Set new value to the setting
    ENDIF
ENDFUNCTION
```

```
FUNCTION SettingsBase.getSettingString
    IF the key of the setting is found THEN
        Return the value of the setting
    ENDIF
    IF the key is the existed key THEN
        Load the default value and return
    ELSE
        Return empty string with error message
    ENDIF
ENDFUNCTION


FUNCTION SettingsBase.getSettingAsIndex
    Convert the value setting to integer and return
ENDFUNCTION


FUNCTION SettingsBase.getSettingAsCount
    Convert the value setting to integer and return
ENDFUNCTION


FUNCTION SettingsBase.getSettingInMicrons
    Convert the setting to float and multiple 1000 then return
ENDFUNCTION
```

```
FUNCTION SettingsBase.getSettingInAngleRadians

    Convert the setting to float and divided by 180.0 then time PI value and return

ENDFUNCTION


FUNCTION SettingsBase.getSettingBoolean

    Convert value “on”, “yes”, “true” or “True” to true and return

    Return true if value is other number than zero

ENDFUNCTION


FUNCTION SettingsBase.getSettingInDegreeCelsius

    Return converted value as float

ENDFUNCTION


FUNCTION SettingsBase.getSettingInMilimetersPerSecond

    Return the maximum between 1 and the value

ENDFUNCTION


FUNCTION SettingsBase.getSettingInPercentage

    Return the maximum between 0 and the value

ENDFUNCTION
```

```
FUNCTION SettingsBase.getSettingInSeconds

    Return the maximum between 0 and the value

ENDFUNCTION


FUNCTION SettingsBase.getSettingAsGCodeFlavor

    IF value == “RepRap” THEN

        Return GCODE_FLAVOR_REPRAP

    ELSEIF value == “UltiGCode” THEN

        Return GCODE_FLAVOR_ULTIGCODE

    ELSEIF value == “Makerbot” THEN

        Return GCODE_FLAVOR_MAKERBOT

    ELSEIF value == “BFB” THEN

        Return GCODE_FLAVOR_BFB

    ELSEIF value == “MACH3” THEN

        Return GCODE_FLAVOR_MACH3

    ELSEIF value == “RepRap (Volumatric)” THEN

        Return GCODE_FLAVOR_REPRAP_VOLYMATRIC

    ENDIF

    Return GCODE_FLAVOR_REPRAP

ENDFUNCTION


FUNCTION SettingsBase.getsettingAsFillMethod
```

```
        IF value == “Lines” THEN
                Return Fill_Lines
        ELSEIF value == “Grid” THEN
                Return Fill_Grid
        ELSEIF value == “Triangles” THEN
                Return Fill_Triangles
        ELSEIF value == “Concentric” THEN
                Return Fill_Concentric
        ELSEIF value == “ZigZag” THEN
                Return Fill_ZigZag
        ENDIF
        Return Fill_None
ENDFUNCTION


FUNCTION SettingsBase.getSettingAsPlatformAdhesion
        IF value == “Brim” THEN
                Return Adhesion_Brim
        ENDIF
        IF value == “Raft” THEN
                Return Adhesion_Raft
        ENDIF
        Return Adhesion_None
```

```
ENDFUNCTION


FUNCTION SettingsBase.getSettingAsSupportType
    IF value == “Everywhere” THEN
        Return Support_Everywhere
    ENDIF
    IF value == “Touching Buildplace” THEN
        Return Support_PlatformOnly
    ENDIF
    Return Support_None
ENDFUNCTION


FUNCTION modelFile.loadModelSTL_ascii
    Open ASCII mode STL file
    WHILE not the end of line
        Extract three vertices
        Add them to mesh’s face
    ENDWHILE
    Finish the mesh
    Return
ENDFUNCTION
```

```
FUNCTION modelFile.loadModelSTL_binary
    Open the binary mode STL file
    Skip the header
    Read the face count
    FOR each face
        Extract the three vertices
        Add them to mesh’s face
    ENDFOR
    Finish the mesh
    Return
ENDFUNCTION


FUNCTION modelFile.loadModelSTL
    Open model file
    Read first line of the file
    IF first line is “solid” THEN
        CALL modelFile.loadModelSTL_ascii
        IF mesh’s face size < 1 THEN
            Clear mesh
            CALL modelFile.loadModelSTL_binary
            Return
        ENDIF
    ENDIF
    Return modelFile.loadModelSTL_binary
ENDFUNCTION

FUNCTION modelFile.loadMeshFromFile
    IF extension of file is “.stl or “.STL” THEN
        Return modelFile.loadModelSTL to the passing object
    ENDIF
ENDFUNCTION

FUNCTION gocdeExport.getFilamentArea
    R is Radius of filament which is half of diameter
    Filament area is PI*r*r
    Return filament area
ENDFUNCTION

FUNCTION gcodeExport.getExtrusionAmountMM3
    IF it is volumetric THEN
        Return extrusion amount
    ELSE
        Return extrusion amount * gcodeExport.getFilamentArea
    ENDIF
```

```
ENDFUNCTION


FUNCTION gcodeExport.writeComment
        Add input string to the output stream
ENDFUNCTION


FUNCTION gcodeExport.writeLayerComment
        Add layer number to the output stream
ENDFUNCTION


FUNCTION gcodeExport.writeLine
        Add line of string to the output stream
ENDFUNCTION
```

Chapter 4: Applications of Open Source GMAW-based Metal 3-D Printing[3]

4.1 Abstract

The metal 3-D printing market is currently dominated by high-end applications, which make it is inaccessible for small and medium enterprises (SMEs), fablabs, and individual makers who are interested in the ability to prototype and make final products in metal using additive manufacturing technology. Recent progress on the open source self-Replicating Rapid-prototyper (RepRap) project led to a low-cost open-source metal 3-D printers using gas metal arc welding (GMAW) based print head. This reduced the cost of metal 3-D printers into the range of desktop prosumer polymer fused filament fabrication printers. Consequent research established good material properties of metal 3-D printed parts with readily-available weld filler wire, reusable substrates, thermal and stress, toolpath planning, bead-width control, mechanical properties, and support for overhangs. This previous works showed that GMAW-based metal 3-D printing has a good adhesion between layers and is not porous inside the printed parts, but it did not proceed far enough to demonstrate applications. In this study the utility of the GMAW approach to 3-D printing is investigated using a low-cost open-source metal 3-D printer and a converted Computer Numerical Control (CNC) router machine to make useful parts over a range of applications including: 1) fixing an existing part by adding on a 3-D metal feature, 2) creating a product using the substrate as part of the component, 3) 3-D printing in high resolution of useful objects, 4) near net objects and 5) making an integrated product using a combination of steel and polymer 3-D printing. The results show that GMAW based 3-D printing is capable of distributed manufacturing useful products by SMEs for a wide variety of applications.

[3] This chapter has been completed as an article to submit. Citation: Nilsiam Y, Sanders P, & Pearce J (2017). Applications of Open Source GMAW-based Metal 3-D Printing.

4.2 Introduction

Most of the metal 3-D printers available on the market are for high-end applications, which require expensive equipment and use relatively dangerous fine metal powders [1]. Due to the cost and the complicity of the technology, it is inaccessible for small and medium enterprises (SMEs), fablabs, and individual makers who are interested in the ability to prototype and make final products in metal using additive manufacturing technology. Following the tradition of the self-Replicating Rapid-prototyper (RepRap) [2-4], a low-cost open-source metal 3-D printers was developed with a gas metal arc welding (GMAW) based print head, which radically reduces the costs of metal 3-D printers to less than $1,200 [5]. The open source metal 3-D printer uses readily available weld filler wire as the filament and the initial designs have been improved upon with integrated monitoring [6] of the welding system [7]. In addition, recent work has shown approaches to reuse substrates which help to reduce costs, energy, time and the environment impact of manufacturing [8-9]. This previous works showed that GMAW-based metal 3-D printing has a good adhesion between layers and is not porous inside the printed parts, but it did not proceed far enough to demonstrate its applications, e.g. only test cubes and dog bones were printed for mechanical testing. Many studies have been done on 3-D weld deposit based process [10-13] and investigated thermal properties and stresses [14-17], toolpath planning [18-21], bead-width control [22-23], mechanical properties [24-25], and support for overhangs [26].

In this paper the utility of the GMAW approach to 3-D printing will be investigated using a low-cost open-source metal 3-D printer and a converted Computer Numerical Control (CNC) router machine to make useful parts over a range of applications including 1) fixing an existing part by adding on a 3-D metal feature, 2) creating a product using the substrate as part of the component, 3) 3-D printing in high resolution of useful objects, 4) near net objects and 5) making an integrated product using a combination of steel and polymer 3-D printing.

4.3 Materials and Methods

The design of the low-cost open-source metal 3-D printer [5, 7] is inspired by the Rostock style delta RepRap [27]. However, it uses a stage printing setup allowing for stationary heavy toolheads [28-29] while automatically controlling movement of a substrate with 3-axis control under a fixed perpendicular weld gun printer head (Figure 4.1c). The motions are managed by an Arduino-based microcontroller and the free and open source 3-D motion control software called Franklin [30]. Franklin also controls the welder (e.g. on for printing and off for traveling). A Millermatic 190 welder, ER70S-6 steel wire and shield gas of RC25 (75% Ar and 25% CO_2) was used for the experiments. Printing is performed on a re-useable substrate of low carbon steel with dimensions of 127 x 127 x 6 mm. The stage that holds a substrate is covered with cement board and then an aluminum plate (Figure 4.1c) to accelerate the transfer out a heat from a substrate and a printed part.

For a 3-D model larger than 127 mm in any dimension, a CNC Router Parts machine was adapted as a GMAW 3-D printer [31]. In Figure 4.4d, a Benchtop PRO CNC Machine Kit is used in this research [32]. The work area is 25” × 25” and the Z clearance is 7”. The resolution or repeatability is ± 0.001” or ± 0.0254 mm. With the capable of 3D motion managed by the control unit, CNC machine is almost ready for metal 3-D printing. The Millermatic 190 GMAW is used for the filament deposition tool. The weld gun is mounted to the tool holder of the machine as the printer head and modified to accept a control signal. The control unit is modified to add output signal wires to the weld gun connection to turn the welder on and off. Substrates of the same material with dimensions of 254 x 254 x 6 mm were then used. The aluminum plate with the same size of the substrate is placed under the substrate. Here also the substrate is held down at four corners during the printing and the moving weld gun is mounted to the tool holder. The welder and the shield gas are the same as in the delta printer above. The packaged Mach3 CNC [33] software was used to communicate to the control unit via an Ethernet cable.

MOSTMetalCura [34] is a customized version of CuraEngine for metal 3-D printing. It slices a 3-D model into 2-D layers and generates toolpaths for each layer. The produced toolpaths are recorded as G-Code. Franklin and Mach3 use G-Code instructions to control the movements of the printers. MOSTMetalCura has added the abilities to turn on and turn off the welder through G-Code, to keep the status of printing or welding, to set how long to pause between layers for the printed part to cool down, and to recommend the wire feed speed setting on the welder (specific for Millermatic 190, for other welders an equation for wire feed speed in MOSTMetalCura would need to be edited). The important settings for MOSTMetalCura are infill line width or bead width, layer height, printing speed, and material diameter.

From a 3-D model, which can be downloaded from free design repositories or created by open-source CAD software (e.g., OpenSCAD [35]), MOSTMetalCura generates a G-Code file from the 3-D model. The settings for open-source GMAW-based metal 3-D printing is shown in Table 4.1. Connecting to Franklin through a browser via web service, Franklin loads the G-Code file and verifies instructions inside the file. When the printing is started by the user, Franklin translates each G-Code instruction and controls the stepper motors on the MOST’s open-source 3-D metal printer to move the substrate as commanded. On the CNC converted printer, Mach3 is acting in similar way to Franklin, excepting that the substrate is stationary and the weld gun is moving as directed by the G-Code. The printing is continued with pausing between layers until the whole model is printed.

Table 4.1 Settings for open-source GMAW-based steel 3-D printing.

Settings	Value (unit)
Voltage on the welder	5 (unitless)
Wire feed rate on the welder	30 (unitless)
Distance between nozzle and substrate	8 (mm)
Wire sticking out from contact tip	5 (mm)
Printing speed	7 (mm/s)
Layer height	2 (mm)
Line or bead width (±0.03)	~1 (mm)
Shield gas	25 (CFH)

4.4 Results and Discussion

Applications of GMAW-based metal 3-D printing are successfully demonstrated by the following printed parts as seen in Figure 4.1-4.5. Parts and products, which would be of interest to SMEs or those in developing regions are focused on here because of the low-cost of the system. The bracket, the hoe, and the chisel were printed on the open-source delta-style metal 3-D printer and the horseshoe and the axe head were printed on the CNC machine. The handle of the axe was polymer 3-D printed on a larger CNC machine [36] converted to use Franklin. These 3-D models as STL files can be found at https://osf.io/bbbtd [37].

1. The system can be used for fixing or printing onto an existing part. A bracket is an example used here where it was printed on the substrate as an existing part. Then some holes can be drilled or printed on the open end of the bracket, so another part can be attached and secured with bolts and nuts. This can be utilized in fixing broken equipment. A different design of a bracket can optimize its strength, stiffness, size, and weight. For example, General Electric (GE) held a contest for such a bracket design for jet engine in 2013 [38]. Similar bracket fixes can be useful for a wide range of applications including solar photovoltaic racking [39].

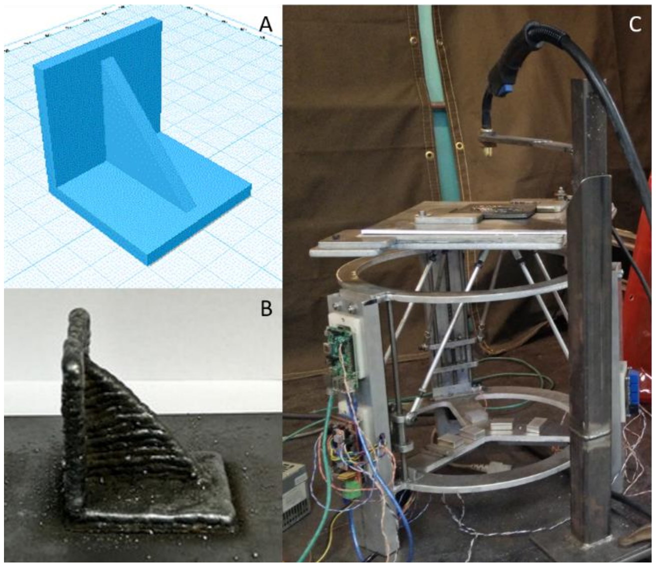


Figure 4.1 A bracket and metal 3-D printer a) 3-D model, b) metal 3-D printed part on substrate, where the substrate is a model for an existing part, and c) the set-up of open-source GMAW-based metal 3-D printer

2. The system can be used to create a product using metal 3-D printing and a substrate as an integral part of the product. For example, a hoe can be made by 3-D printing a cylinder on the substrate (Figure 4.2b). Then the substrate is cut into a shape of a hoe and sharpened on the edge opposite the printed cylinder (Figure 4.2c). A wood or a polymer 3-D printed stick can be used as a handle for the hoe. Being able to manufacture such a product in an isolated rural community can be considered appropriate technology and can foster sustainable development [40-41]. The ability to manufacture metal objects significantly expands the utility of 3-D printing for small farmers in the developing world [42].

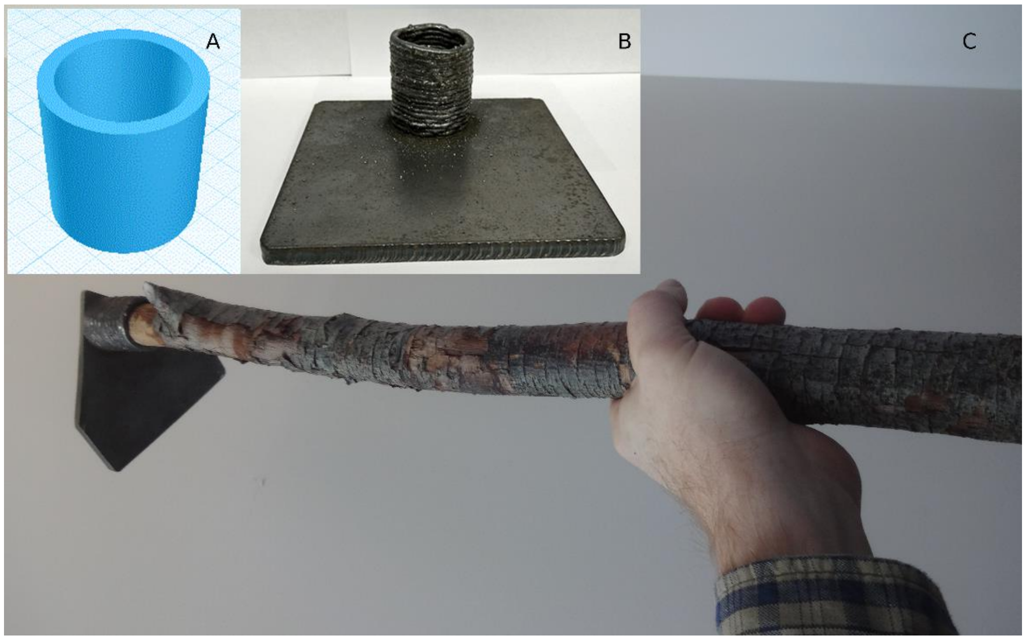


Figure 4.2 A hoe a) 3-D model of handle hold, b) metal 3-D printed part on substrate, and c) finished hoe, cut and mounted to wooden handle

3. The system is capable of higher resolution that previous attempts at GMAW-3-D printing [5]. A high resolution chisel model (Figure 4.3a) is used to demonstrate this capability. The printed part is ready to use with a minimal machining (Figure 4.3c). A model with small details can be printed as long as they are not smaller than 1 mm.

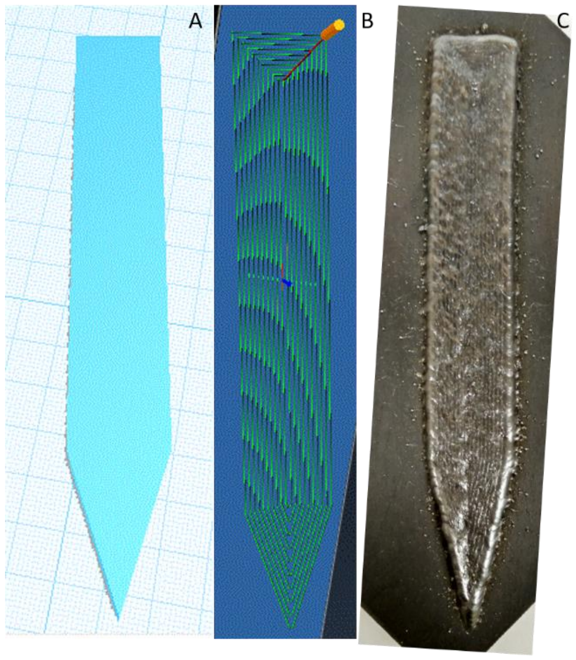


Figure 4.3. a) 3-D model, b) toolpath, and c) metal 3-D printed part on substrate

4. Near-net shape objects can be fabricated with the system. An example of this is a horseshoe (Figure 4.4), which needs to be customized for specific horses, so it is suitable to be metal 3-D printed. The printed part is near-net shape, so it needs finish machining. This technique can be applied in similar situations that require a custom part. For example, in the design of open source scientific equipment [43-45] a custom size of a ring support or a vial holder for a hot plate can be easily designed and printed.

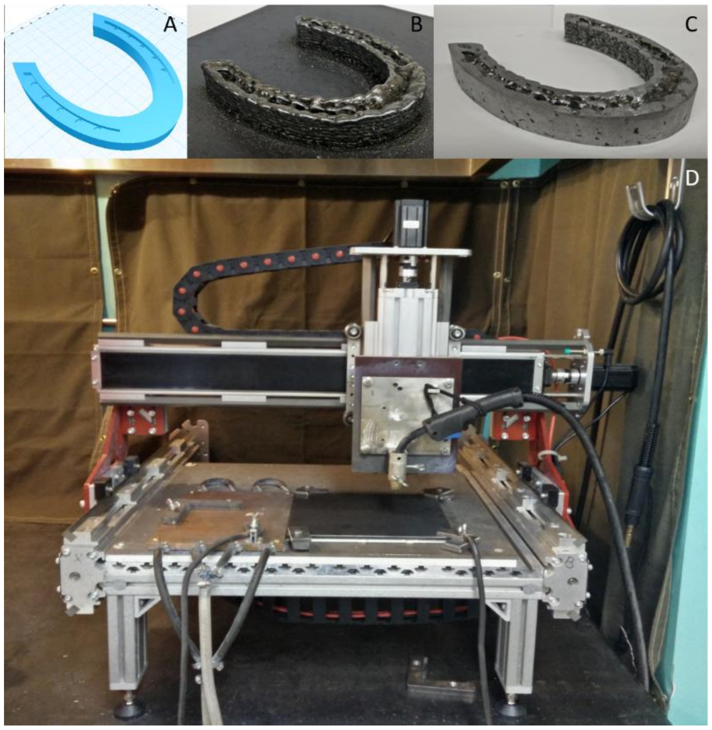


Figure 4.4. A horseshoe and CNC Router Parts a) 3-D model, b) metal 3-D printed part on substrate, c) finished part, and d) a converted CNC Router Parts metal 3-D printer

5. Finally, fully functional integrated products can be fabricated using a combination of metal and polymer 3-D printing. Here, an axe head was 3-D printed in steel (Figure 4.5) and a handle was 3-D printed in polymer. A combination process like this can be used to remotely manufacture similar open source instruments such as a hammer or other hand tools [46-47].

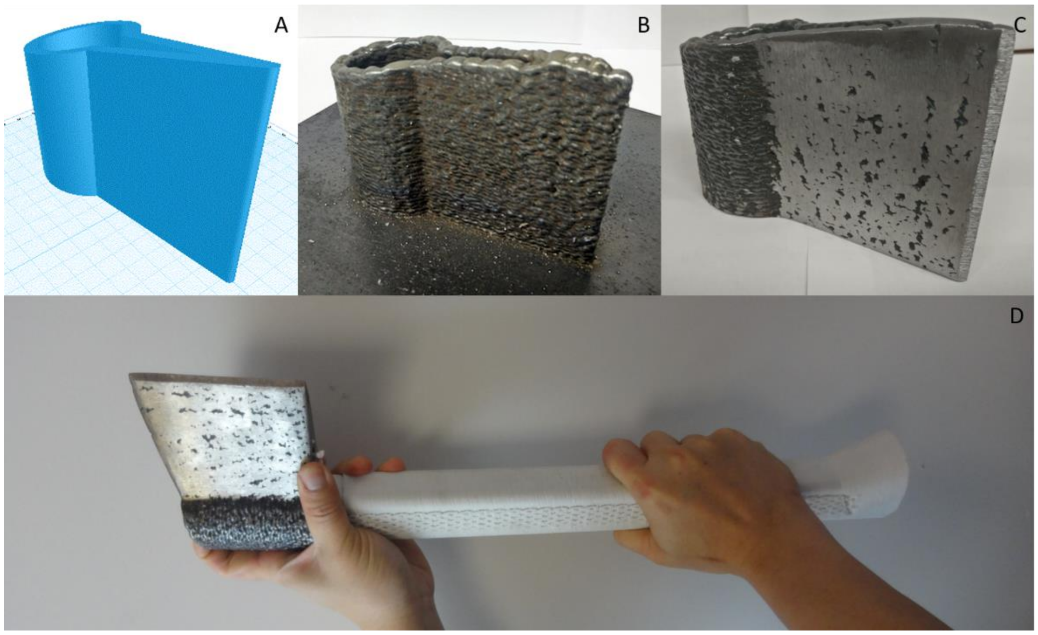


Figure 4.5. An axe a) 3-D model, b) metal 3-D printed part on substrate, c) finished part, and d) integrated product of metal and polymer 3-D printing

From the results, it is clear that GMAW-based metal 3-D printing can be applied to many real-life problems. First, the technique can be used to repair or add functionality to an existing steel product. As in the case with the bracket the settings can be adapted to leave the part on the substrate of an existing part. Thus if, for example, a bracket were to break off a tractor part, the tractor could be repaired by replacing the broken bracket on the main component or a new bracket to be added to a part to improve the mechanical assembly of the assembly. There are many applications for this functionality, which include; repairing damaged parts [48-50] and customizing or adding on an existing object [51]. This is particularly important in the field in isolated regions (e.g. for development or military personnel).

A close application to this functionality is to use the substrate and a 3-D printed design to create a new product as is shown with the hoe (Figure 4.2). The printed metal has a good adhesion to the substrate, so they became as one part. This kind of application is useful whenever the end product can be primarily manufactured from a plate of steel. Although the entire hoe could have been printed without using the steel substrate, the manufacturing time is reduced considerably (roughly two hours) by incorporating the substrate. Other applications of this method include similar products, such as a rake, a flag pole stand, a flute stand, etc.

As can be seen in Figure 4.3 the process is capable of printing in relatively high resolution for the cost of the process – down to 1 mm lines. This functionality is useful for making high detail steel parts such as a gear.

The most useful current application of open source GMAW-based 3-D printing, however, is to manufacture near net shape objects. This is demonstrated in Figure 4.4 with the horseshoe. The near net shape is seen in many industries and has many applications. For example, in the auto industry, when a needed part is no longer available or short supply, the part can be 3-D printed [52-54].

The combination of metal and polymer 3-D printing as shown in Figure 4.5 can be applied to produce many things that need both metal and plastic. Many tools have metal part with plastic handles, such as screwdriver, knife, gardening tools, etc.

Many 3-D models that are available but commonly printed only in plastic would have improved performance if metal printing technology such as this were employed. However, as the resolution of printing is constant at 1 mm if there are details in a 3-D model that smaller than that they will be lost. So only near-net shape functionality is available for the majority of readily available 3-D models. The smaller details would need to be post machined to the print. A 3-D model that is not in full millimeter in any dimension will also result in a little bit smaller or bigger printed part (e.g. 0.5 mm designed wall will result in 1mm print). If there is an angle less than 45 degree of z-axis in a 3-D model, the staircase effect will appear at the angle in the model.

A big 3-D model with a lot of area to be filled will result in a long pause time between layers to let the printed layer to cool down before printing the next layer. Otherwise, the heat inside the printed layers can cause defective surfaces for the next layer. If a model can be hollow, it will reduce the pause time by half. For example, the axe head would require 30 minutes of pause time between layers if it were 100% filled, but it is hollow so only 15 minutes needed.

The CNC machine in this experiment does not have a consistency of moving speed between moving along x- or y-axis and moving diagonal. When moving along x- or y- axis, it has a little faster speed than moving diagonal, which caused a rougher surface and layer height is higher. To avoid the different speed, a 3-D model can be rotated or using a shape (e.g., cylinder) that requires diagonal moving.

For the future work, the finer resolution would provide the ability to achieve a detail of smaller than 1 mm and the thinner layer height would diminish the staircase effect. A better method to release the heat from the printed part, such as a water-cooled chill plate, would cut down the waiting time between layers.

4.5 Conclusions

This paper has successfully shown applications of open-source GMAW-based metal 3-D printing. The results show that GMAW based 3-D printing is capable of distributed manufacturing useful products by SMEs for a wide variety of applications. Metal products and parts can be designed and created using this technology and the low-cost and open-source makes it available to everyone. This also gives user the flexibility to customize the hardware and software for other uses.

4.6 References

1 Wohlers TT and Caffrey T. Wohlers report 2015: 3D printing and additive manufacturing state of the industry annual worldwide progress report. Wohlers Associates. 2015.

2 Sells E, Smith Z, Bailard S, et al. RepRap: The Replicating Rapid Prototyper: Maximizing Customizability by Breeding the Means of Production. In Piller, F. T., Tseng, M. M. (Eds.). Handbook of Research in Mass Customization and Personalization: Strategies and concepts. World Scientific 2010;1:568-580.

3 Jones R, Haufe P, Sells E, et al. RepRap – the replicating rapid prototype. Robotica 2011;29(1):177-191.

4 Bowyer A. 3D printing and humanity's first imperfect replicator. 3D printing and additive manufacturing 2014;1(1):4-5.

5 Anzalone G, Zhang C, Wijnen B, et al. A Low-Cost Open-Source Metal 3-D Printer. IEEE Access 2013;1:803-810. https://doi.org/10.1109/ACCESS.2013.2293018

6 Pinar A, Wijnen B, Anzalone GC, et al. Low-cost open-source voltage and current monitor for gas metal arc weld 3D printing. Journal of Sensors 2015. 876714, https://doi.org/10.1155/2015/876714

7 Nilsiam, Y., Haselhuhn, A., Wijnen, B., Sanders, P. and Pearce, J.M., 2015. Integrated Voltage—Current Monitoring and Control of Gas Metal Arc Weld Magnetic Ball-Jointed Open Source 3-D Printer. Machines, 3(4), pp.339-351.

8 Haselhuhn A, Gooding E, Glover A, et al. Substrate Release Mechanisms for Gas Metal Arc Weld 3D Aluminum Metal Printing. 3D Printing and Additive Manufacturing 2014;1(4):204-209. https://doi.org/10.1089/3dp.2014.0015

9 Haselhuhn A, Wijnen B, Anzalone G, et al. In situ formation of substrate release mechanisms for gas metal arc weld metal 3-D printing. Journal of Materials Processing Technology 2015;226:50-59. https://doi.org/10.1016/j.jmatprotec.2015.06.038

10 Zhang YM, Chen Y, LI P, et al. Weld deposition-based rapid prototyping: a preliminary study. Journal of Materials Processing Technology 2003;135:347–357.

11 Song YA, Park S, Choi D, Jee H. 3D welding and milling: Part I–a direct approach for freeform fabrication of metallic prototypes. International Journal of Machine Tools and Manufacture 2005 Jul 31;45(9):1057-62.

12 Song YA, Park S, Chae SW. 3D welding and milling: part II—optimization of the 3D welding process using an experimental design approach. International Journal of Machine Tools and Manufacture 2005 Jul 31;45(9):1063-9.

13 Ding D, Pan Z, Cuiuri D, Li H. Wire-feed additive manufacturing of metal components: technologies, developments and future interests. The International Journal of Advanced Manufacturing Technology 2015 Oct 1;81(1-4):465-81.

14 Spencer J, Dickens P, and Wykes C. Rapid prototyping of metal parts by three-dimensional welding. Proceedings of the Institution of Mechanical Engineers 1998; 212(3):175-182.

15 Kwak YM, Doumanidis CC. Geometry regulation of material deposition in near-net shape manufacturing by thermally scanned welding. Journal of Manufacturing Processes 2002 Jan 1;4(1):28-41.

16 Zhao H and Zhang G. A 3D dynamic analysis of thermal behavior during single-pass multi-layer weld-based rapid prototyping. Journal of Materials Processing Technology 2011;211:488-495.

17 Zhang G, Yin Z, and Wu L. Effects of Interpass Idle Time on Thermal Stresses in Multipass Multilayer Weld-Based Rapid Prototyping. Journal of Manufacturing Science and Engineering 2013;135.

18 Dwivedi R, and Kovacevic R. Automated torch path planning using polygon subdivision for solid freeform fabrication based on welding. Journal of Manufacturing Systems 2004;23(4):278-91.

19 Ding D, Pan ZS, Cuiuri D, Li H. A tool-path generation strategy for wire and arc

additive manufacturing. The international journal of advanced manufacturing technology 2014 Jul 1;73(1-4):173-83.

20 Ding D, Pan Z, Cuiuri D, Li H. A multi-bead overlapping model for robotic wire and arc additive manufacturing (WAAM). Robotics and Computer-Integrated Manufacturing 2015 Feb 28;31:101-10.

21 Ding D, Pan Z, Cuiuri D, Li H. A practical path planning methodology for wire and arc additive manufacturing of thin-walled structures. Robotics and Computer-Integrated Manufacturing 2015 Aug 31;34:8-19.

22 Xiong J, Zhang G, Qiu Z, Li Y. Vision-sensing and bead width control of a single-bead multi-layer part: material and energy savings in GMAW-based rapid manufacturing. Journal of Cleaner Production 2013 Feb 28;41:82-8.

23 Xiong J, Zhang G, Gao H, Wu L. Modeling of bead section profile and overlapping beads with experimental validation for robotic GMAW-based rapid manufacturing. Robotics and Computer-Integrated Manufacturing 2013 Apr 30;29(2):417-23.

24 Ding J, Colegrove P, Mehnen J, et al. Thermo-mechanical analysis of wire and arc additive layer manufacturing process on large multi-layer parts. Computational Materials Science 2011 Dec 31;50(12):3315-22.

25 Hildreth OJ, Nassar AR, Chasse KR, et al. Dissolvable Metal Supports for 3D Direct Metal Printing. 3D Printing and Additive Manufacturing 2016 Jun 1;3(2):90-7.

26 Das S, Bourell DL, Babu SS. Metallic materials for 3D printing. MRS Bulletin 2016 Oct;41(10):729-41.

27 RepRap. Rostock. http://reprap.org/wiki/Rostock. Last accessed date 2017 Mar 2.

28 Anzalone GC, Wijnen B and Pearce JM. Multi-material additive and subtractive prosumer digital fabrication with a free and open-source convertible delta RepRap 3-D printer. Rapid Prototyping Journal 2015;21(5):506-519.

29 Zhang C, Wijnen B and Pearce JM. Open-source 3-D platform for low-cost scientific instrument ecosystem. Journal of laboratory automation 2016;21(4):517-525.

30 Wijnen B, Anzalone GC, Haselhuhn AS, Sanders PG, Pearce JM. Free and open-source control software for 3-D motion and processing. Journal of Open Research Software 2016 Jan 27;4(1).

31 Pearce JM and Nilsiam Y. CNC Router Parts metal 3D printer. http://www.appropedia.org/CNC_Router_Parts_metal_3D_printer. Last accessed date 2017 Mar 13.

32 CNCRouterParts. Benchtop PRO CNC Machine Kit. http://www.cncrouterparts.com/benchtop-pro-cnc-machine-kit-p-314.html. Last accessed date 2017 Mar 14.

33 Newfrangled Solutions. Mach3. http://www.machsupport.com/software/mach3. Last accessed date 2017 Mar 13.

34 Nilsiam Y, Sanders P, and Pearce J. Slicer and Optimization for Open-Source GMAW-based Metal 3-D Printing. to be published (2017).

35 OpenSCAD. http://www.openscad.org. Last accessed date 2017 Mar 6.

36 Chandra H, Skalsky N, Laureto J, et al. Large Form Factor Open Source FFF-based 3-D Printer for Fabrication of Multi-Cubic Meter Models. to be published (2017).

37 Nilsiam Y and Pearce JM. MOST Metal Application Models. https://osf.io/bbbtd/. Last accessed date 2017 Mar 14.

38 GrabCAD. GE jet engine bracket challenge. https://grabcad.com/challenges/ge-jet-engine-bracket-challenge. Last accessed date 2017 Mar 14.

39 Wittbrodt B and Pearce JM. 3-D printing solar photovoltaic racking in developing world. Energy for Sustainable Development 2017;36:1-5.

40 Hazeltine B and Bull C. Appropriate Technology; Tools, Choices, and Implications. Academic Press, Inc. 1998.

41 Smith A. Transforming technological regimes for sustainable development: a role for Appropriate Technology niches?. University of Sussex, SPRU. 2002.

42 Pearce JM. Applications of open source 3-D printing on small farms. Organic Farming 2015;1(1): 19-35.

43 Pearce J.M. Building research equipment with free, open-source hardware. Science 2012;337(6100):1303-1304.

44 Pearce, Open-Source Lab: How to Build Your Own Hardware and Reduce Research Costs, Elsevier, 2014.

45 Baden T, Chagas AM, Gage G, et al. Open Labware: 3-D printing your own lab equipment. PLoS Biol 2015;13(3):1002086.

46 Pearce JM, Blair CM, Laciak KJ, et al. 3-D printing of open source appropriate technologies for self-directed sustainable development. Journal of Sustainable Development 2010;3(4):17.

47 Heyer S and Seliger G. Open manufacturing for value creation cycles. In Design for Innovative Value Towards a Sustainable Society. Springer Netherlands 2012;110-115

48 Optomec. Components Repair. https://www.optomec.com/3d-printed-metals/lens-core-applications/component-repair/. Last accessed date 2017 Mar 14.

49 Kira. BeAM repairs more than 800 aerospace parts with industrial metal 3D printers. http://www.3ders.org/articles/20160204-beam-repairs-more-than-800-aerospace-parts-with-industrial-metal-3d-printers.html. Last accessed date 2017 Mar 14.

50 Langnau L. Using 3D printing to repair metal parts. http://www.makepartsfast.com/using-3d-printing-repair-metal-parts/. Last accessed date 2017 Mar 14.

51 Matisons M. Sustainable 3D Printing Methods Add to or Subtract from Existing Objects. https://3dprint.com/105562/3d-print-for-existing-objects/. Last accessed date 2017 Mar 14.

52 Norfolk M. Maintenance and Repair – 3D Printing Metal parts. Fabrisonic. http://fabrisonic.com/maintenance-repair-3d-printing-metal-parts/. Last accessed date 2017 Mar 14.

53 Leno J. Jay Leno's 3D Printer Replaces Rusty Old Parts. Popular mechanics

http://www.popularmechanics.com/cars/a4354/4320759/. Last accessed date 2017 Mar 14.

54 Petrova M. Your car's parts could one day be made by a 3D printer. PC World. http://www.pcworld.com/article/3159056/hardware/your-cars-parts-could-one-day-be-made-by-a-printer.html . Last accessed date 2017 Mar 14.

Chapter 5: Conclusions and Future Work

5.1 Overview

The work in this dissertation has shown the development of open-source toolchains and the applications of open-source GMAW-based metal 3-D printing. The integrated voltage-current monitoring system provides more data about energy usage and how it affects to the printing. The MOSTMetalCura gives the capability to generate G-code from a 3-D model instead of manual input. The slicer also provides the optimization wire feed rate setting for the welder based on other settings. Finally, the applicability of open-source GMAW-based metal 3-D printing demonstrates the usefulness in a wide range of industries. Overall, the completed work helps improve the process and utility of the open-source GMAW-based metal 3-D printing. The low-cost and open-source of the technology made it accessible to all who want to use it or customize it for improvement and for other uses.

5.2 Conclusions

5.2.1 Integrated Voltage-Current Monitoring System

- The design of system reduces the need of an additional controller board which efficiently utilizes the existing software and hardware.
- The design also cuts down the cost and the complication of electric equipment.
- The voltage and current data was logged in real-time as raw measurement and need to be processed.
- The open-source script for data processing was developed to clean noise and to calculate the two standard error, average voltage, and average current for each layer and per alloy.
- The system provides more data for optimization purpose.

5.2.2 Slicer and Optimization for Open-Source GMAW-based Metal 3-D Printing

- The concentric pattern is chosen to avoid overrun the previous welded bead.
- The ability to add G-code that pauses the printing between layers and be able to set how long it is and even increase the time as the layers are higher was added.

- The capability to config which GPIO pins to be used for turning on/off the welder.
- Based on printing speed, layer height, wire diameter, and bead width, the optimized wire feed rate setting is calculated by the MOSTMetalCura.
- The result demonstrates that the calculated setting is optimized for the open-source GMAW-based metal 3-D printing with a resolution of 1 millimeter.

5.2.3 Applications of Open-Source GMAW-based Metal 3-D Printing

- GMAW-based metal 3-D printing has been studied by many researchers and the results indicated that the printed parts by the technique are good in both material and mechanical properties.
- The existing CNC Router Parts was converted to a metal 3-D printer by adding the welder to it and the ability for the control unit of the machine to control the welder to be on/off.
- Metal 3-D printing can be applied to print on the existing part in order for fixing or customizing purpose.
- A product can be created from a merger of a substrate and metal 3-D printing.
- A combination of polymer and metal 3-D printing can be used to produced many useful products.
- The ability to print near net shape objects reduces material waste if the object was made by subtractive manufacturing.
- The results demonstrate that it can be applied to a wide range of applications and it is ready for distributed manufacturing.

5.3 Future Work

5.3.1 Integrated Voltage-Current Monitoring System

- The data should be used for feedback control in the real-time. Franklin would need to be customized to send control signal to the welder based on the current and voltage data. However, the welder need to be controllable by electric signal.
- From the real-time measurement of current and voltage, using those data to automatically adjust the printing speed in order to achieve the optimal settings.

- A model for calculating energy usage based on the current and voltage data should be developed.
- There should be a further study about the relationship between printing pattern and energy consuming.

5.3.2 Slicer and Optimization for Open-Source GMAW-based Metal 3-D Printing

- MOSTMetalCura is currently a command-line control software. To make it more user friendly, a graphical user interface can be implemented or open-source Cura software can be customized to support metal 3-D printing.
- Bead width is manually adjusted so that the model is filled evenly. This might be able to be dynamically adapted based on the dimension of the model.
- Toolpath planning for a complex geometry should be adjusted to suit metal 3-D printing. There is a lot of stops and travelling if the model has holes inside.
- With sensor to measure the temperature of the printed part, thermal model of 3-D printed part should be created to optimize the pause time between layers. The thermocouple or infrared (IR) sensor could be used to measure the temperature at the surface of the printed part then using the thermal model to predict the heat in the part.
- Thermal control during printing should be developed for a better quality finished surface and cooling system during pausing time between layers would help cut down a lot of waiting time.
- Another option to mitigate the surface roughness is using milling between layers which is not hard to do on the CNC Router Parts system.

5.3.3 Applications of Open-Source GMAW-based Metal 3-D Printing

- There is limited open-source 3-D models for metal printing. A community website to share 3-D models and designs should be started. The collaboration would bring more applications of the technology.
- To print a model with small details, the finer printing resolution is needed. The experiments with different welders and smaller weld wire would need to be

conducted for optimizing the print. If smaller weld wire does not exist, then it should be developed to suit metal 3-D printing in size and material properties.

- The current CNC Router Parts setup has different moving velocity between moving by one and two motors. This can be hardware or software problem. Franklin, free and open-source software, should be tried to control the machine for testing.

Appendix

A CuraEngine information for Chapter 3

A.1 Data flow map of the core functions of CuraEngine

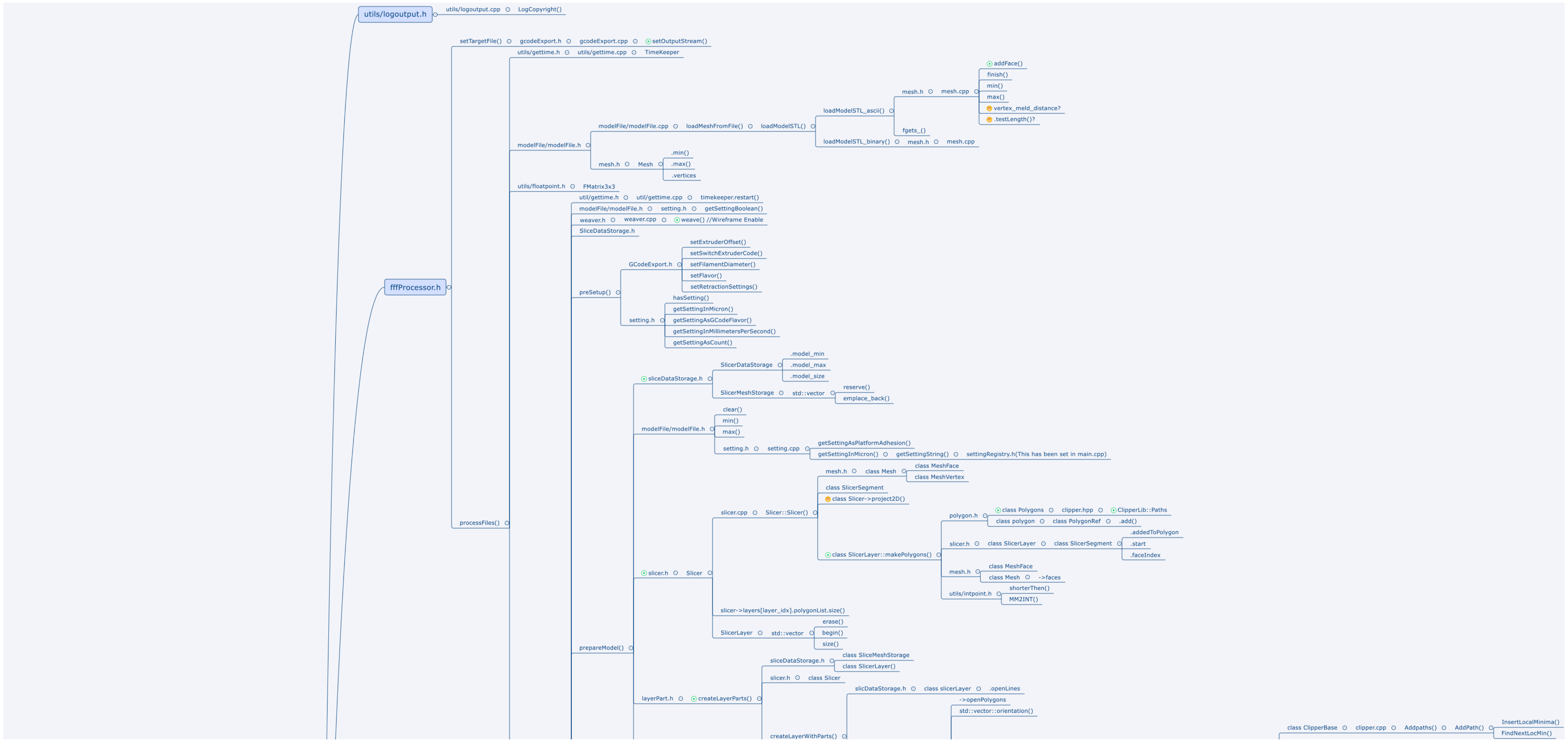


Figure A.1 Data flow map of the core functions of CuraEngine (part 1)

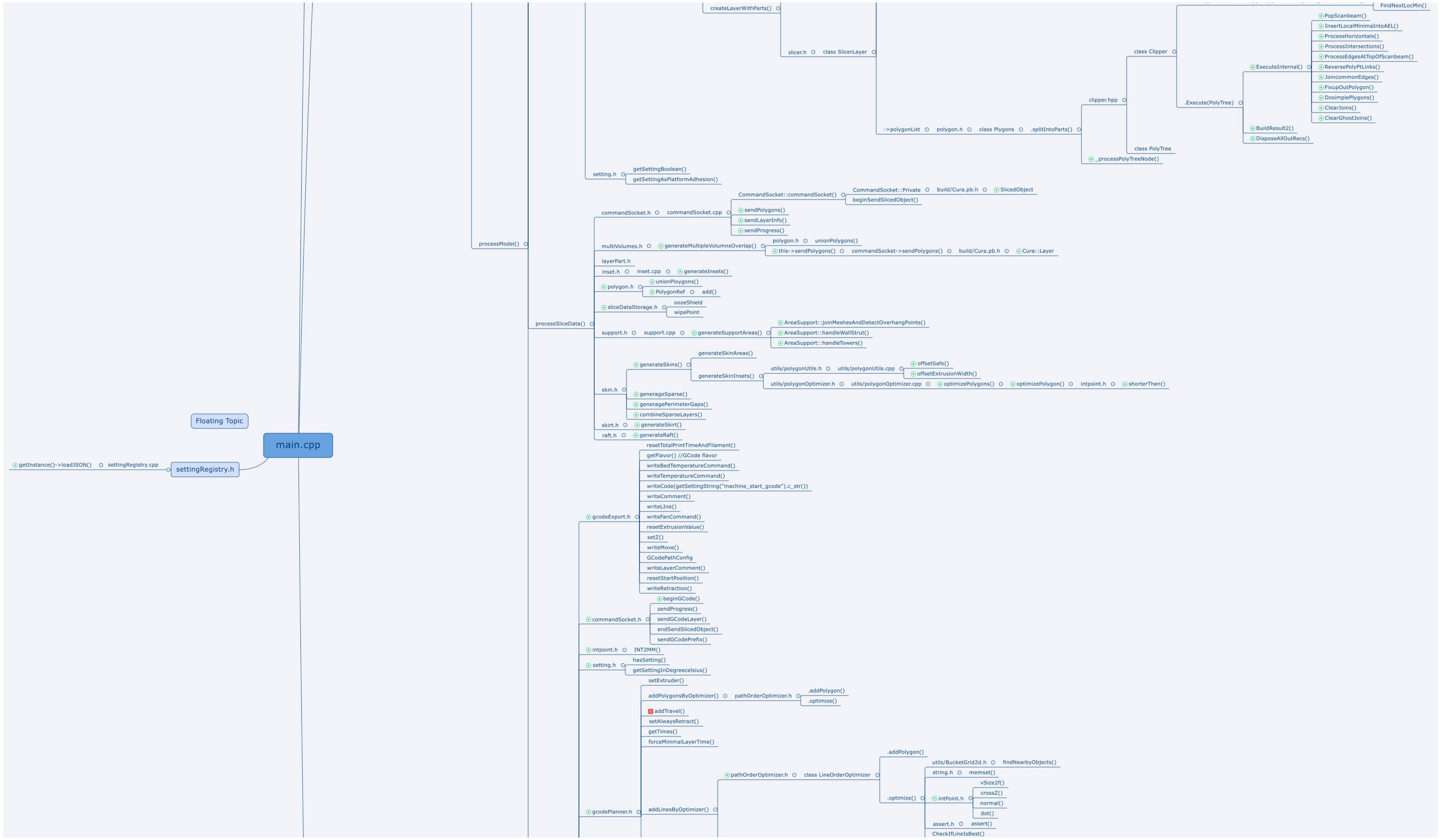


Figure A.2 Data flow map of the core functions of CuraEngine (part 2)

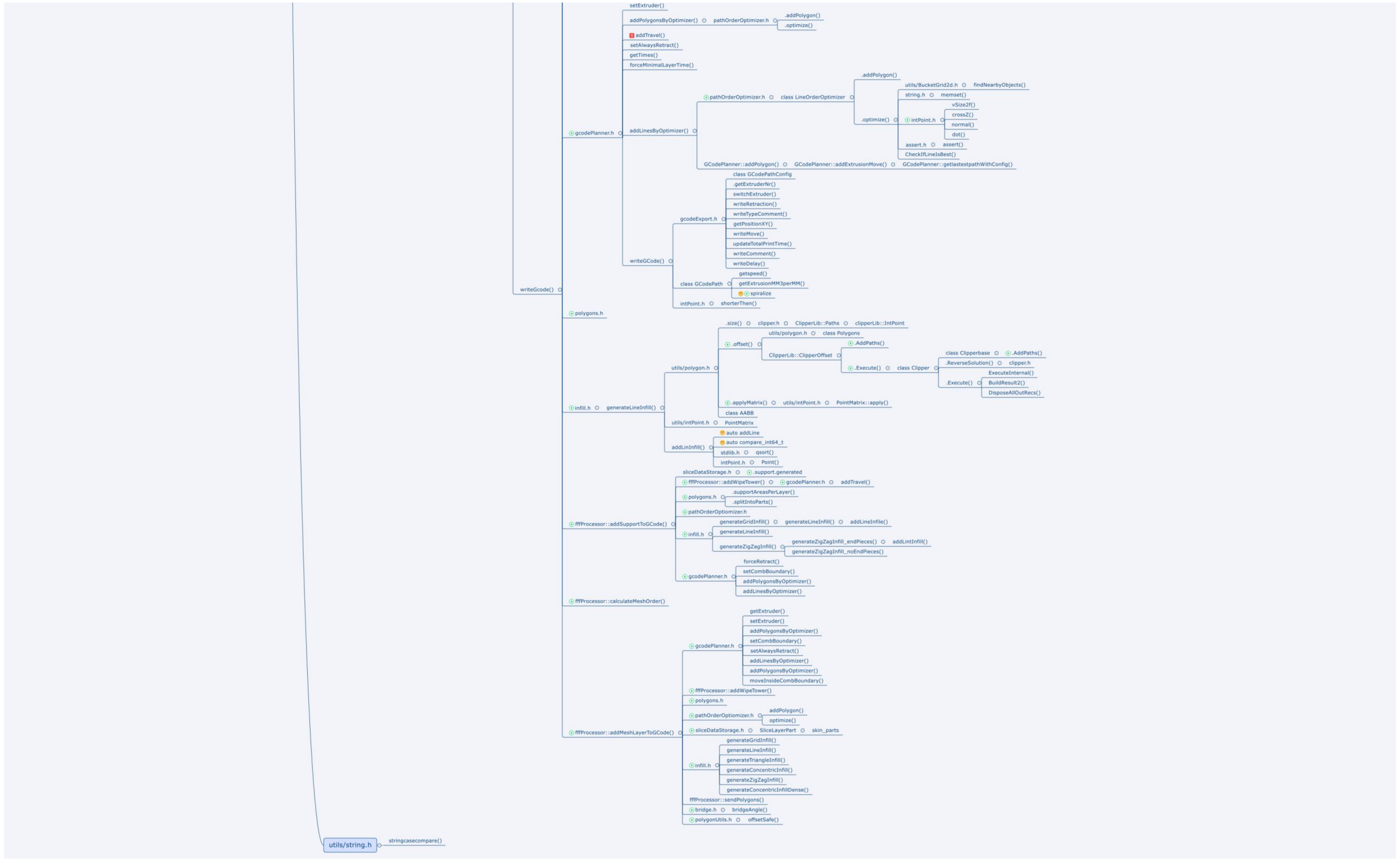


Figure A.3 Data flow map of the core functions of CuraEngine (part 3)

A.2 Source code of the core functions of CuraEngine

```
#CMakeLists.txt
project(MOSTMetalCura)
cmake_minimum_required(VERSION 2.8.12)
find_package(Arcus REQUIRED)
if(NOT ${CMAKE_VERSION} VERSION_LESS 3.1)
   set(CMAKE_CXX_STANDARD 11)
else()
   set(CMAKE_CXX_FLAGS "-std=c++11")
endif()
# Add warnings
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall")
if(NOT APPLE AND NOT WIN32)
   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -static-libstdc++")
elseif(APPLE)
   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -stdlib=libc++")
endif()
include_directories(${CMAKE_CURRENT_BINARY_DIR} libs)
add_library(clipper STATIC libs/clipper/clipper.cpp)
set(engine_SRCS
   src/bridge.cpp
   src/comb.cpp
   src/commandSocket.cpp
   src/gcodeExport.cpp
   src/gcodePlanner.cpp
   src/infill.cpp
   src/inset.cpp
   src/layerPart.cpp
   src/main.cpp
   src/mesh.cpp
   src/multiVolumes.cpp
   src/pathOrderOptimizer.cpp
   src/polygonOptimizer.cpp
   src/raft.cpp
   src/settingRegistry.cpp
   src/settings.cpp
   src/skin.cpp
   src/skirt.cpp
   src/slicer.cpp
   src/support.cpp
   src/timeEstimate.cpp
   src/Weaver.cpp
   src/Wireframe2gcode.cpp
   src/modelFile/modelFile.cpp
   src/utils/gettime.cpp
```

```
    src/utils/logoutput.cpp
    src/utils/polygonUtils.cpp
)
protobuf_generate_cpp(engine_PB_SRCS engine_PB_HEADERS Cura.proto)
add_executable(MOSTMetalCura ${engine_SRCS} ${engine_PB_SRCS})
target_link_libraries(MOSTMetalCura clipper Arcus)
add_executable(Test src/test.cpp)
target_link_libraries(Test clipper)
if (UNIX)
    target_link_libraries(MOSTMetalCura pthread)
endif()
include(GNUInstallDirs)
install(TARGETS MOSTMetalCura DESTINATION ${CMAKE_INSTALL_BINDIR})
```

fdmprinter.json

```
{
    "visible": false,
    "machine_settings": {
        "machine_start_gcode": {
            "default": ";mm units\nG21\n;Feed per minute mode\nG94 \n;Absolute Distance
mode\nG90\n"
        },
        "machine_end_gcode": {
            "default": ""
        },
        "machine_metal_printing": { "default": true },
        "machine_welder_on_gcode": {
          "default": ";Turn welder on\nG4 P.001\nM101\n"
        },
        "machine_welder_off_gcode": {
          "default": ";Turn welder off\nG4 P.001\nM102\n"
        },
        "machine_min_dist_welder_off": {
          "unit": "mm",
          "default": 5.0
        },
        "machine_up_layer_end": {
          "unit": "mm",
          "default": 10.0
        },
        "machine_layer_pause": { "default": true },
        "machine_layer_pause_gcode": { "default": ";Layer pause\nG4 P" },
        "machine_layer_pause_time": { "description": "millisecond for franklin, second for
cnc", "default": 300 },
        "machine_layer_pause_increase": { "default": 0 },
        "meshfix_union_all_remove_holes": {"default": false},
        "meshfix_union_all":{"default": false},
        "machine_width": { "default": 230 },
        "machine_depth": { "default": 225 },
        "machine_height": { "default": 205 },
        "machine_heated_bed": { "default": false },
        "machine_center_is_zero": { "default": false },
        "machine_nozzle_size": { "default": 2.5 },
        "machine_head_shape_min_x": { "default": 40 },
        "machine_head_shape_min_y": { "default": 10 },
        "machine_head_shape_max_x": { "default": 60 },
        "machine_head_shape_max_y": { "default": 30 },
        "machine_nozzle_gantry_distance": { "default": 55 },
        "machine_nozzle_offset_x_1": { "default": 18.0 },
        "machine_nozzle_offset_y_1": { "default": 0.0 },
        "machine_gcode_flavor": { "default": "RepRap" },
        "machine_disallowed_areas": { "default": [
            [[-115.0,  112.5], [ -82.0,  112.5], [ -84.0,  104.5], [-115.0,  104.5]],
            [[ 115.0,  112.5], [ 115.0,  104.5], [ 110.0,  104.5], [ 108.0,  112.5]],
            [[-115.0, -112.5], [-115.0, -104.5], [ -84.0, -104.5], [ -82.0, -112.5]],
            [[ 115.0, -112.5], [ 108.0, -112.5], [ 110.0, -104.5], [ 115.0, -104.5]]
        ]},
        "machine_platform_offset": { "default": [9.0, 0.0, 0.0] },

        "machine_nozzle_tip_outer_diameter": { "default": 1.0 },
        "machine_nozzle_head_distance": { "default": 3.0 },
        "machine_nozzle_expansion_angle": { "default": 45 }
    },
    "categories": {
        "resolution": {
            "label": "Quality",
            "visible": true,
            "icon": "category_quality",
            "settings": {
                "layer_height": {
                    "label": "Layer Height",
                    "description": "The height of each layer, in mm. Normal quality prints are
0.1mm, high quality is 0.06mm. You can go up to 0.25mm with an Ultimaker for very
fast prints at low quality. For most purposes, layer heights between 0.1 and 0.2mm give a
good tradeoff of speed and surface finish.",
                    "unit": "mm",
                    "type": "float",
                    "default": 2.00,
                    "min_value": 0.06,
                    "max_value": 2.0,
                    "always_visible": true,
                    "children": {
                        "layer_height_0": {
                            "label": "Initial Layer Thickness",
                            "description": "The layer thickness of the bottom layer. A thicker
bottom layer makes sticking to the bed easier.",
                            "unit": "mm",
                            "type": "float",
                            "default": 2.00,
                            "min_value": 0.06,
                            "max_value": 2.0,
                            "visible": false
                        }
                    }
                },
                "shell_thickness": {
```

```
                "label": "Shell Thickness",
                "description": "The thickness of the outside shell in the horizontal and
vertical direction. This is used in combination with the nozzle size to define the number
of perimeter lines and the thickness of those perimeter lines. This is also used to define
the number of solid top and bottom layers.",
                "unit": "mm",
                "type": "float",
                "default": 2.0,
                "min_value": 0.0,
                "max_value": 5.0,
                "children": {
                   "wall_thickness": {
                      "label": "Wall Thickness",
                      "description": "The thickness of the outside walls in the horizontal
direction. This is used in combination with the nozzle size to define the number of
perimeter lines and the thickness of those perimeter lines.",
                      "unit": "mm",
                      "default": 0.99,
                      "min_value": 0.0,
                      "max_value": 5.0,
                      "min_value_warning": 0.4,
                      "max_value_warning": 2.0,
                      "type": "float",
                      "visible": false,
                      "children": {
                         "wall_line_count": {
                            "label": "Wall Line Count",
                            "description": "Number of shell lines. This these lines are called
perimeter lines in other tools and impact the strength and structural integrity of your
print.",
                            "default": 100,
                            "type": "int",
                            "visible": false,
                            "inherit_function": "max(1, (int(parent_value /
(machine_nozzle_size - 0.0001) + 1) if (parent_value / max(1, int(parent_value /
(machine_nozzle_size - 0.0001))) > machine_nozzle_size) * 1.5 else int(parent_value /
(machine_nozzle_size - 0.0001))))"
                         },
                         "wall_line_width": {
                            "label": "Wall Line Width",
                            "description": "Width of a single shell line. Each  line of the shell
will be printed with this width in mind.",
                            "unit": "mm",
                            "default": 0.99,
                            "type": "float",
                            "visible": false,
                                "inherit_function": "max(machine_nozzle_size, (parent_value /
(int(parent_value / (machine_nozzle_size - 0.0001) + 1))) if (parent_value /
(int(parent_value / (machine_nozzle_size - 0.0001))) > machine_nozzle_size * 1.5) else
(parent_value / int(parent_value / (machine_nozzle_size - 0.0001))))",
                                "children": {
                                  "wall_line_width_0": {
                                    "label": "First Wall Line Width",
                                    "description": "Width of the outermost shell line. By printing
a thinner outermost wall line you can print higher details with a larger nozzle.",
                                    "unit": "mm",
                                    "default": 0.99,
                                    "type": "float",
                                    "visible": false
                                  },
                                  "wall_line_width_x": {
                                    "label": "Other Walls Line Width",
                                    "description": "Width of a single shell line for all shell lines
except the outermost one.",
                                    "unit": "mm",
                                    "default": 0.99,
                                    "type": "float",
                                    "visible": false
                                  },
                                  "skirt_line_width": {
                                    "label": "Skirt line width",
                                    "description": "Width of a single skirt line.",
                                    "unit": "mm",
                                    "default": 0.99,
                                    "type": "float",
                                    "visible": false
                                  },
                                  "skin_line_width": {
                                    "label": "Top/bottom line width",
                                    "description": "Width of a single top/bottom printed line.
Which are used to fill up the top/bottom areas of a print.",
                                    "unit": "mm",
                                    "default": 0.99,
                                    "type": "float",
                                    "visible": false
                                  },
                                  "infill_line_width": {
                                    "label": "Infill line width",
                                    "description": "Width of the inner infill printed lines.",
                                    "unit": "mm",
                                    "default": 0.99,
                                    "type": "float",
                                    "visible": false
                                  },
                                  "support_line_width": {
                                    "label": "Support line width",
                                    "description": "Width of the printed support structures lines.",
                                    "unit": "mm",
                                    "default": 0.99,
                                    "type": "float",
                                    "visible": false
                                  }
                                }
                             }
                          }
                       },
                       "alternate_extra_perimeter": {
                          "label": "Alternate Extra Wall",
                          "description": "Make an extra wall at every second layer, so that infill
will be caught between an extra wall above and one below. This results in a better
cohesion between infill and walls, but might have an impact on the surface quality.",
                          "type": "boolean",
                          "default": false,
                          "visible": false
                       },
                       "top_bottom_thickness": {
                          "label": "Bottom/Top Thickness",
                          "description": "This controls the thickness of the bottom and top layers,
the amount of solid layers put down is calculated by the layer thickness and this value.
Having this value a multiple of the layer thickness makes sense. And keep it near your
wall thickness to make an evenly strong part.",
                          "unit": "mm",
                          "default": 2.0,
                          "min_value": 0.0,
                          "max_value": 5.0,
                          "min_value_warning": 0.4,
                          "max_value_warning": 1.0,
                          "type": "float",
                          "visible": false,
                          "children": {
                             "top_thickness": {
                                "label": "Top Thickness",
                                "description": "This controls the thickness of the top layers. The
number of solid layers printed is calculated from the layer thickness and this value.
Having this value be a multiple of the layer thickness makes sense. And keep it nearto
your wall thickness to make an evenly strong part.",
                                "unit": "mm",
                                "default": 2.0,
                             "type": "float",
                             "visible": false,
                             "children": {
                                 "top_layers": {
                                     "label": "Top Layers",
                                     "description": "This controls the amount of top layers.",
                                     "default": 100,
                                     "type": "int",
                                     "visible": false,
                                     "inherit_function": "math.ceil(parent_value / layer_height)"
                                 }
                             }
                         },
                         "bottom_thickness": {
                             "label": "Bottom Thickness",
                             "description": "This controls the thickness of the bottom layers.
The number of solid layers printed is calculated from the layer thickness and this value.
Having this value be a multiple of the layer thickness makes sense. And keep it near to
your wall thickness to make an evenly strong part.",
                             "unit": "mm",
                             "default": 2.0,
                             "type": "float",
                             "visible": false,

                             "children": {
                                 "bottom_layers": {
                                     "label": "Bottom Layers",
                                     "description": "This controls the amount of bottom layers.",
                                     "default": 100,
                                     "type": "int",
                                     "visible": false,
                                     "inherit_function": "math.ceil(parent_value / layer_height)"
                                 }
                             }
                         }
                     }
                 }
             }
         },
         "wall_overlap_avoid_enabled": {
             "label": "Avoid Overlapping Walls",
             "description": "Remove parts of a wall which share an overlap which would
result in overextrusion in some places. These overlaps occur in thin pieces in a model and
sharp corners.",
             "type": "boolean",
             "default": true,
             "visible": false
         },
         "fill_perimeter_gaps":{
             "label": "Fill Gaps Between Walls",
             "description": "Fill the gaps created by walls where they would otherwise be
overlapping. This will also fill thin walls. Optionally only the gaps occurring within the
top and bottom skin can be filled.",
             "type": "enum",
             "options": [
                 "Nowhere",
                 "Everywhere",
                 "Skin"
             ],
             "default": "Nowhere",
             "visible": false,
             "active_if": {
                 "setting": "wall_overlap_avoid_enabled",
                 "value": true
             }
         },
         "top_bottom_pattern": {
             "label": "Bottom/Top Pattern",
             "description": "Pattern of the top/bottom solid fill. This normally is done
with lines to get the best possible finish, but in some cases a concentric fill gives a nicer
end result.",
             "type": "enum",
             "options": [
                 "Lines",
                 "Concentric"
             ],
             "default": "Concentric",
             "visible": false
         },
         "skin_outline_count": {
             "label": "Skin Perimeter Line Count",
             "description": "Number of lines around skin regions. Using one or two skin
perimeter lines can greatly improve on roofs which would start in the middle of infill
cells.",
             "default": 20,
             "type": "int",
             "visible": false,
             "active_if": {
                 "setting": "top_bottom_pattern",
                 "value": "Lines"
             }
         },
         "xy_offset": {
             "label": "Horizontal expansion",
             "description": "Amount of offset applied all polygons in each layer. Positive
values can compensate for too big holes; negative values can compensate for too small
holes.",
             "unit": "mm",
             "type": "float",
             "default": 0.0,
             "visible": false

         }
      }
   },

   "material": {
      "label": "Material",
      "visible": true,
      "icon": "category_material",
      "settings": {
         "material_print_temperature": {
             "label": "Printing Temperature",
             "description": "The temperature used for printing. Set at 0 to pre-heat
yourself. For PLA a value of 210C is usually used.\nFor ABS a value of 230C or higher
is required.",
             "unit": "°C",
             "type": "float",
             "default": 0,
             "min_value": 10,
             "max_value": 340
         },
         "material_bed_temperature": {
             "label": "Bed Temperature",
             "description": "The temperature used for the heated printer bed. Set at 0 to
pre-heat it yourself.",
             "unit": "°C",
             "type": "float",
             "default": 0,
             "min_value": 0,
             "max_value": 340
         },
         "material_diameter": {
             "label": "Diameter",
             "description": "The diameter of your filament needs to be measured as
accurately as possible.\nIf you cannot measure this value you will have to calibrate it, a
higher number means less extrusion, a smaller number generates more extrusion.",
             "unit": "mm",
             "type": "float",
             "default": 0.6,
             "min_value": 0.4,
             "max_value": 5.0
         },
         "material_flow": {
             "label": "Flow",
             "description": "Flow compensation: the amount of material extruded is
multiplied by this value.",
             "unit": "%",
             "default": 100.0,
             "type": "float",
             "min_value": 5.0,
             "max_value": 300.0
         },
         "retraction_enable": {
             "label": "Enable Retraction",
             "description": "Retract the filament when the nozzle is moving over a non-
printed area. Details about the retraction can be configured in the advanced tab.",
             "type": "boolean",
             "default": false,

             "children": {
                 "retraction_speed": {
                     "label": "Retraction Speed",
                     "description": "The speed at which the filament is retracted. A higher
retraction speed works better, but a very high retraction speed can lead to filament
grinding.",
                     "unit": "mm/s",
                     "type": "float",
                     "default": 20.0,
                     "visible": false,
                     "inherit": false,

                     "children": {
                         "retraction_retract_speed": {
                             "label": "Retraction Retract Speed",
                             "description": "The speed at which the filament is retracted. A
higher retraction speed works better, but a very high retraction speed can lead to filament
grinding.",
                             "unit": "mm/s",
                             "type": "float",
                             "default": 20.0,
                             "visible": false
                         },
                         "retraction_prime_speed": {
```

```
                    "label": "Retraction Prime Speed",
                    "description": "The speed at which the filament is pushed back
after retraction.",
                    "unit": "mm/s",
                    "type": "float",
                    "default": 20.0,
                    "visible": false
                  }
                }
              },
              "retraction_amount": {
                "label": "Retraction Distance",
                "description": "The amount of retraction: Set at 0 for no retraction at
all,4 defalut. A value of 4.5mm seems to generate good results for 3mm filament in
Bowden-tube fed printers.",
                "unit": "mm",
                "type": "float",
                "default": 0.0,
                "visible": false,
                "inherit": false
              },
              "retraction_min_travel": {
                "label": "Retraction Minimum Travel",
                "description": "The minimum distance of travel needed for a retraction
to happen at all. This helps ensure you do not get a lot of retractions in a small area.",
                "unit": "mm",
                "type": "float",
                "default": 1.5,
                "visible": false,
                "inherit": false
              },
              "retraction_combing": {
                "label": "Enable Combing",
                "description": "Combing keeps the head within the interior of the print
whenever possible when traveling from one part of the print to another, and does not use
retraction. If combing is disabled the printer head moves straight from the start point to
the end point and it will always retract.",
                "type": "boolean",
                "default": false,
                "visible": false,
                "inherit": false
              },
              "retraction_count_max": {
                "label": "Maximal Retraction Count",
                "description": "This settings limits the number of retractions occuring
within the Minimal Extrusion Distance Window. Further retractions within this window
```

```
will be ignored. This avoids retracting repeatedly on the same piece of filament as that
can flatten the filament and cause grinding issues.",
                        "default": 6,
                        "type": "int",
                        "visible": false,
                        "inherit": false
                    },
                    "retraction_extrusion_window": {
                        "label": "Minimal Extrusion Distance Window",
                        "description": "The window in which the Maximal Retraction Count is
enforced. This window should be approximately the size of the Retraction distance, so
that effectively the number of times a retraction passes the same patch of material is
limited.",
                        "unit": "mm",
                        "type": "float",
                        "default": 0.5,
                        "visible": false,
                        "inherit_function": "retraction_amount"
                    },
                    "retraction_hop": {
                        "label": "Z Hop when Retracting",
                        "description": "Whenever a retraction is done, the head is lifted by this
amount to travel over the print. A value of 0.075 works well. This feature has a lot of
positive effect on delta towers.",
                        "unit": "mm",
                        "type": "float",
                        "default": 0.0,
                        "visible": false,
                        "inherit": false
                    }
                }
            }
        }
    },
    "speed": {
        "label": "Speed",
        "visible": true,
        "icon": "category_speed",
        "settings": {
            "speed_print": {
                "label": "Print Speed",
                "description": "The speed at which printing happens. A well-adjusted
Ultimaker can reach 150mm/s, but for good quality prints you will want to print slower.
Printing speed depends on a lot of factors, so you will need to experiment with optimal
settings for this.",
                "unit": "mm/s",
          "type": "float",
          "default": 7.0,

          "children": {
            "speed_infill": {
              "label": "Infill Speed",
              "description": "The speed at which infill parts are printed. Printing the
infill faster can greatly reduce printing time, but this can negatively affect print quality.",
              "unit": "mm/s",
              "type": "float",
              "default": 7.0,
              "visible": false
            },
            "speed_wall": {
              "label": "Shell Speed",
              "description": "The speed at which shell is printed. Printing the outer
shell at a lower speed improves the final skin quality.",
              "unit": "mm/s",
              "type": "float",
              "default": 7.0,
              "visible": false,

              "children": {
                "speed_wall_0": {
                  "label": "Outer Shell Speed",
                  "description": "The speed at which outer shell is printed. Printing
the outer shell at a lower speed improves the final skin quality. However, having a large
difference between the inner shell speed and the outer shell speed will effect quality in a
negative way.",
                  "unit": "mm/s",
                  "type": "float",
                  "default": 7.0,
                  "visible": false
                },
                "speed_wall_x": {
                  "label": "Inner Shell Speed",
                  "description": "The speed at which  all inner shells are printed.
Printing the inner shell fasster than the outer shell will reduce printing time. It is good to
set this in between the outer shell speed and the infill speed.",
                  "unit": "mm/s",
                  "type": "float",
                  "default": 7.0,
                  "visible": false
                }
              }
            },
                        "speed_topbottom": {
                            "label": "Top/Bottom Speed",
                            "description": "Speed at which top/bottom parts are printed. Printing the
top/bottom faster can greatly reduce printing time, but this can negatively affect print
quality.",
                            "unit": "mm/s",
                            "type": "float",
                            "default": 7.0,
                            "visible": false
                        },
                        "speed_support": {
                            "label": "Support Speed",
                            "description": "The speed at which exterior support is printed. Printing
exterior supports at higher speeds can greatly improve printing time. And the surface
quality of exterior support is usually not important, so higher speeds can be used.",
                            "unit": "mm/s",
                            "type": "float",
                            "default": 7.0,
                            "visible": false,
                            "inherit_function": "speed_wall_0"
                        }
                    }
                },
                "speed_travel": {
                    "label": "Travel Speed",
                    "description": "The speed at which travel moves are done. A well-built
Ultimaker can reach speeds of 250mm/s. But some machines might have misaligned
layers then.",
                    "unit": "mm/s",
                    "type": "float",
                    "default": 7.0
                },
                "speed_layer_0": {
                    "label": "Bottom Layer Speed",
                    "description": "The print speed for the bottom layer: You want to print the
first layer slower so it sticks to the printer bed better.",
                    "unit": "mm/s",
                    "type": "float",
                    "default": 7.0,
                    "visible": false,

                    "children": {
                        "skirt_speed": {
                            "label": "Skirt Speed",
                    "description": "The speed at which the skirt and brim are printed.
Normally this is done at the initial layer speed. But sometimes you want to print the skirt
at a different speed.",
                    "unit": "mm/s",
                    "type": "float",
                    "default": 7.0,
                    "visible": false
                  }
                }
              },
              "speed_slowdown_layers": {
                "label": "Amount of Slower Layers",
                "description": "The first few layers are printed slower then the rest of the
object, this to get better adhesion to the printer bed and improve the overall success rate
of prints. The speed is gradually increased over these layers. 4 layers of speed-up is
generally right for most materials and printers.",
                "type": "int",
                "default": 0,
                "visible": false
              }
            }
          },
          "infill": {
            "label": "Infill",
            "visible": true,
            "icon": "category_infill",
            "settings": {
              "fill_sparse_density": {
                "label": "Infill Density",
                "description": "This controls how densely filled the insides of your print will
be. For a solid part use 100%, for an hollow part use 0%. A value around 20% is usually
enough. This won't affect the outside of the print and only adjusts how strong the part
becomes.",
                "unit": "%",
                "type": "float",
                "default": 100.0,

                "children": {
                  "fill_pattern": {
                    "label": "Infill Pattern",
                    "description": "Cura defaults to switching between grid and line infill.
But with this setting visible you can control this yourself. The line infill swaps direction
on alternate layers of infill, while the grid prints the full cross-hatching on each layer of
infill.",
                    "type": "enum",
                    "visible": false,
                        "options": [
                            "Grid",
                            "Lines",
                            "Concentric",
                            "ZigZag"
                        ],
                        "default": "Concentric",
                        "inherit_function": "'Lines' if parent_value > 25 else 'Grid'"
                    },
                                                        "infill_line_distance": {
                                                                "label": "Line distance",
                                                                "description": "Distance between the
printed infill lines.",
                                                                "unit": "mm",
                                                                "type": "float",
                                                                "default": 0.0,
                                                                "visible": false,
                                                                "inherit_function":
"(infill_line_width * 100) / parent_value"
                                                        }
                }
            },
            "fill_overlap": {
                "label": "Infill Overlap",
                "description": "The amount of overlap between the infill and the walls. A
slight overlap allows the walls to connect firmly to the infill.",
                "unit": "%",
                "type": "float",
                "default": 0.0,
                "visible": false
            },
            "fill_sparse_thickness": {
                "label": "Infill Thickness",
                "description": "The thickness of the sparse infill. This is rounded to a
multiple of the layerheight and used to print the sparse-infill in fewer, thicker layers to
save printing time.",
                "unit": "mm",
                "type": "float",
                "default": 2.0,
                "visible": false,

                "children": {
                    "fill_sparse_combine": {
                        "label": "Infill Layers",
                        "description": "Amount of layers that are combined together to form
sparse infill.",
                    "type": "int",
                    "default": 0,
                    "visible": false,
                    "inherit_function": "math.floor((parent_value + 0.001) / layer_height)"
                  }
                }
              }
            }
          },
          "cooling": {
            "label": "Cooling",
            "visible": true,
            "icon": "category_cool",
            "settings": {
              "cool_fan_enabled": {
                "label": "Enable Cooling Fan",
                "description": "Enable the cooling fan during the print. The extra cooling
from the cooling fan helps parts with small cross sections that print each layer quickly.",
                "type": "boolean",
                "default": false,

                "children": {
                  "cool_fan_speed": {
                    "label": "Fan Speed",
                    "description": "Fan speed used for the print cooling fan on the printer
head.,100 default",
                    "unit": "%",
                    "type": "float",
                    "default": 0.0,
                    "visible": false,
                    "inherit_function": "100.0 if parent_value else 0.0",

                    "children": {
                      "cool_fan_speed_min": {
                        "label": "Minimum Fan Speed",
                        "description": "Normally the fan runs at the minimum fan speed. If
the layer is slowed down due to minimum layer time, the fan speed adjusts between
minimum and maximum fan speed.",
                        "unit": "%",
                        "type": "float",
                        "default": 0.0,
                        "visible": false
                      },
                      "cool_fan_speed_max": {
                        "label": "Maximum Fan Speed",
                                "description": "Normally the fan runs at the minimum fan speed. If
the layer is slowed down due to minimum layer time, the fan speed adjusts between
minimum and maximum fan speed.",
                                "unit": "%",
                                "type": "float",
                                "default": 0.0,
                                "visible": false
                            }
                        }
                    }
                }
            },
            "cool_fan_full_at_height": {
                "label": "Fan Full on at Height",
                "description": "The height at which the fan is turned on completely. For the
layers below this the fan speed is scaled linearly with the fan off for the first layer.",
                "unit": "mm",
                "type": "float",
                "default": 0.5,
                "visible": false,

                "children": {
                    "cool_fan_full_layer": {
                        "label": "Fan Full on at Layer",
                        "description": "The layer number at which the fan is turned on
completely. For the layers below this the fan speed is scaled linearly with the fan off for
the first layer.",
                        "type": "int",
                        "default": 0,
                        "visible": false,
                        "inherit_function": "int((parent_value - layer_height_0 + 0.001) /
layer_height)"
                    }
                }
            },
            "cool_min_layer_time": {
                "label": "Minimal Layer Time",
                "description": "The minimum time spent in a layer: Gives the layer time to
cool down before the next one is put on top. If a layer would print in less time, then the
printer will slow down to make sure it has spent at least this many seconds printing the
layer.",
                "unit": "sec",
                "type": "float",
                "default": 5.0,
                "visible": false
            },
            "cool_min_layer_time_fan_speed_max": {
                "label": "Minimal Layer Time Full Fan Speed",
                "description": "The minimum time spent in a layer which will cause the fan
to be at minmum speed. The fan speed increases linearly from maximal fan speed for
layers taking minimal layer time to minimal fan speed for layers taking the time specified
here.",
                "unit": "sec",
                "type": "float",
                "default": 10.0,
                "visible": false
            },
            "cool_min_speed": {
                "label": "Minimum Speed",
                "description": "The minimum layer time can cause the print to slow down so
much it starts to droop. The minimum feedrate protects against this. Even if a print gets
slowed down it will never be slower than this minimum speed.",
                "unit": "mm/s",
                "type": "float",
                "default": 10.0,
                "visible": false
            },
            "cool_lift_head": {
                "label": "Lift Head",
                "description": "Lift the head away from the print if the minimum speed is hit
because of cool slowdown, and wait the extra time away from the print surface until the
minimum layer time is used up.",
                "type": "boolean",
                "default": false,
                "visible": false
            }
        }
    },
    "support": {
        "label": "Support",
        "visible": true,
        "icon": "category_support",
        "settings": {
            "support_enable": {
                "label": "Enable Support",
                "description": "Enable exterior support structures. This will build up
supporting structures below the model to prevent the model from sagging or printing in
mid air.",
                "type": "boolean",
                "default": false
            },
            "support_type": {
                "label": "Placement",
                "description": "Where to place support structures. The placement can be
restricted such that the support structures won't rest on the model, which could otherwise
cause scarring.",
                "type": "enum",
                "options": [
                   "Touching Buildplate",
                   "Everywhere"
                ],
                "default": "Touching Buildplate",
                "visible": true,
                "inherit_function": "'Everywhere' if support_enable else 'None'",
                "active_if": {
                   "setting": "support_enable",
                   "value": true
                }
             },
             "support_angle": {
                "label": "Overhang Angle",
                "description": "The maximum angle of overhangs for which support will be
added. With 0 degrees being horizontal, and 90 degrees being vertical.",
                "unit": "°",
                "type": "float",
                "default": 60.0,
                "visible": false,
                "active_if": {
                   "setting": "support_enable",
                   "value": true
                }
             },
             "support_xy_distance": {
                "label": "X/Y Distance",
                "description": "Distance of the support structure from the print, in the X/Y
directions. 0.7mm typically gives a nice distance from the print so the support does not
stick to the surface.",
                "unit": "mm",
                "type": "float",
                "default": 0.7,
                "visible": false,
                "active_if": {
                   "setting": "support_enable",
                   "value": true
                }
             },
             "support_z_distance": {
                "label": "Z Distance",
                "description": "Distance from the top/bottom of the support to the print. A
small gap here makes it easier to remove the support but makes the print a bit uglier.
0.15mm allows for easier separation of the support structure.",
                "unit": "mm",
                "type": "float",
                "default": 0.15,
                "visible": false,
                "active_if": {
                    "setting": "support_enable",
                    "value": true
                },

                "children": {
                    "support_top_distance": {
                        "label": "Top Distance",
                        "description": "Distance from the top of the support to the print.",
                        "unit": "mm",
                        "default": 0.15,
                        "type": "float",
                        "visible": false
                    },
                    "support_bottom_distance": {
                        "label": "Bottom Distance",
                        "description": "Distance from the print to the bottom of the support.",
                        "unit": "mm",
                        "default": 0.15,
                        "type": "float",
                        "visible": false
                    }
                }
            },
            "support_bottom_stair_step_height": {
                "label": "Stair Step Height",
                "description": "The height of the steps of the stair-like bottom of support
resting on the model. Small steps can cause the support to be hard to remove from the top
of the model.",
                "unit": "mm",
                "type": "float",
                "default": 0.5,
                "visible": false,
                "active_if": {
                    "setting": "support_type",
                    "value": "Everywhere"
                }
            },
            "support_join_distance": {
                "label": "Join Distance",
                "description": "The maximum distance between support blocks, in the X/Y
directions, such that the blocks will merge into a single block.",
                "unit": "mm",
                "type": "float",
                "default": 0.7,
                "visible": false
            },
            "support_area_smoothing": {
                "label": "Area Smoothing",
                "description": "Maximal distance in the X/Y directions of a line segment
which is to be smoothed out. Ragged lines are introduced by the join distance and support
bridge, which cause the machine to resonate. Smoothing the support areas won't cause
them to break with the constraints, except it might change the overhang.",
                "unit": "mm",
                "type": "float",
                "default": 0.6,
                "visible": false
            },
            "support_use_towers": {
                "label": "Use towers.",
                "description": "Use specialized towers to support tiny overhang areas. These
towers have a larger diameter than the region they support. Near the overhang the towers'
diameter decreases, forming a roof.",
                "type": "boolean",
                "default": true,
                "visible": true
            },
            "support_minimal_diameter": {
                "label": "Minimal Diameter",
                "description": "Maximal diameter in the X/Y directions of a small area
which is to be supported by a specialized support tower. ",
                "unit": "mm",
                "type": "float",
                "default": 1.0,
                "visible": false,
                "active_if": {
                    "setting": "support_use_towers",
                    "value": true
                }
            },
            "support_tower_diameter": {
                "label": "Tower Diameter",
                "description": "The diameter of a special tower. ",
                "unit": "mm",
                "type": "float",
                "default": 1.0,
                "visible": false,
                "active_if": {
                    "setting": "support_use_towers",
                    "value": true
                }
            },
            "support_tower_roof_angle": {
                "label": "Tower Roof Angle",
                "description": "The angle of the rooftop of a tower. Larger angles mean more
pointy towers. ",
                "unit": "°",
                "type": "int",
                "default": 65,
                "visible": false,
                "active_if": {
                    "setting": "support_use_towers",
                    "value": true
                }
            },
            "support_pattern": {
                "label": "Pattern",
                "description": "Cura supports 3 distinct types of support structure. First is a
grid based support structure which is quite solid and can be removed as 1 piece. The
second is a line based support structure which has to be peeled off line by line. The third
is a structure in between the other two; it consists of lines which are connected in an
accordeon fashion.",
                "type": "enum",
                "options": [
                    "Grid",
                    "Lines",
                    "ZigZag"
                ],
                "default": "ZigZag",
                "visible": true,
                "active_if": {
                    "setting": "support_enable",
                    "value": true
                }
            },
            "support_connect_zigzags": {
                "label": "Connect ZigZags",
                "description": "Connect the ZigZags. Makes them harder to remove, but
prevents stringing of disconnected zigzags.",
                "type": "boolean",
                "default": true,
                "visible": false,
                "active_if": {
                    "setting": "support_pattern",
                    "value": "ZigZag"
                }
            },
            "support_fill_rate": {
                "label": "Fill Amount",
                "description": "The amount of infill structure in the support, less infill gives
weaker support which is easier to remove.",
                "unit": "%",
                "type": "float",
                "default": 20,
                "visible": false,
                "active_if": {
                    "setting": "support_enable",
                    "value": true
                },

                                "children": {
                                        "support_line_distance": {
                                                "label": "Line distance",
                                                "description": "Distance between the
printed support lines.",
                                                "unit": "mm",
                                                "type": "float",
                                                "default": 2.66,
                                                "visible": false,
                                                "active_if": {
                                                        "setting": "support_enable",
                                                        "value": true
                                                },
                                                "inherit_function":
"(support_line_width * 100) / parent_value"
                                        }
                                }
            }
        }
    },
    "platform_adhesion": {
        "label": "Platform Adhesion",
        "visible": true,
        "icon": "category_adhesion",
        "settings": {
            "adhesion_type": {
                "label": "Type",
          "description": "Different options that help in preventing corners from lifting due to warping. Brim adds a single-layer-thick flat area around your object which is easy to cut off afterwards, and it is the recommended option. Raft adds a thick grid below the object and a thin interface between this and your object. (Note that enabling the brim or raft disables the skirt.)",
          "type": "enum",
          "options": [
            "None",
            "Brim",
            "Raft"
          ],
          "default": "None"
        },
        "skirt_line_count": {
          "label": "Skirt Line Count",
          "description": "The skirt is a line drawn around the first layer of the. This helps to prime your extruder, and to see if the object fits on your platform. Setting this to 0 will disable the skirt. Multiple skirt lines can help to prime your extruder better for small objects.",
          "type": "int",
          "default": 0,
          "active_if": {
            "setting": "adhesion_type",
            "value": "None"
          }
        },
        "skirt_gap": {
          "label": "Skirt Distance",
          "description": "The horizontal distance between the skirt and the first layer of the print.\nThis is the minimum distance, multiple skirt lines will extend outwards from this distance.",
          "unit": "mm",
          "type": "float",
          "default": 6.0,
          "active_if": {
            "setting": "adhesion_type",
            "value": "None"
          }
        },
        "skirt_minimal_length": {
          "label": "Skirt Minimum Length",
          "description": "The minimum length of the skirt. If this minimum length is not reached, more skirt lines will be added to reach this minimum length. Note: If the line count is set to 0 this is ignored.",
          "unit": "mm",
          "type": "float",
          "default": 150,
          "active_if": {
            "setting": "adhesion_type",
            "value": "None"
          }
        },
        "brim_line_count": {
          "label": "Brim Line Count",
          "description": "The amount of lines used for a brim: More lines means a
  larger brim which sticks better, but this also makes your effective print area smaller.",
          "type": "int",
          "default": 20,
          "active_if": {
            "setting": "adhesion_type",
            "value": "Brim"
          }
        },
        "raft_margin": {
          "label": "Raft Extra Margin",
          "description": "If the raft is enabled, this is the extra raft area around the
  object which is also given a raft. Increasing this margin will create a stronger raft while
  using more material and leaving less area for your print.",
          "unit": "mm",
          "type": "float",
          "default": 5.0,
          "active_if": {
            "setting": "adhesion_type",
            "value": "Raft"
          }
        },
        "raft_line_spacing": {
          "label": "Raft Line Spacing",
          "description": "The distance between the raft lines. The first 2 layers of the
  raft have this amount of spacing between the raft lines.",
          "unit": "mm",
          "type": "float",
          "default": 1.0,
          "active_if": {
            "setting": "adhesion_type",
            "value": "Raft"
          }
        },
        "raft_base_thickness": {
          "label": "Raft Base Thickness",
          "description": "Layer thickness of the first raft layer. This should be a thick
  layer which sticks firmly to the printer bed.",
          "unit": "mm",
          "type": "float",
          "default": 0.3,
          "active_if": {
            "setting": "adhesion_type",
            "value": "Raft"
          }
        },
        "raft_base_linewidth": {
          "label": "Raft Base Line Width",
          "description": "Width of the lines in the first raft layer. These should be thick
  lines to assist in bed adhesion.",
          "unit": "mm",
          "type": "float",
          "default": 0.7,
          "active_if": {
            "setting": "adhesion_type",
            "value": "Raft"
          }
        },
        "raft_base_speed": {
          "label": "Raft Base Print Speed",
          "description": "The speed at which the first raft layer is printed. This should
  be printed quite slowly, as the amount of material coming out of the nozzle is quite
  high.",
          "unit": "mm/s",
          "type": "float",
          "default": 15.0,
          "active_if": {
            "setting": "adhesion_type",
            "value": "Raft"
          }
        },
        "raft_interface_thickness": {
          "label": "Raft Interface Thickness",
          "description": "Thickness of the 2nd raft layer.",
          "unit": "mm",
          "type": "float",
          "default": 0.2,
          "active_if": {
            "setting": "adhesion_type",
            "value": "Raft"
          }
        },
        "raft_interface_linewidth": {
          "label": "Raft Interface Line Width",
                "description": "Width of the 2nd raft layer lines. These lines should be
thinner than the first layer, but strong enough to attach the object to.",
                "unit": "mm",
                "type": "float",
                "default": 0.2,
                "active_if": {
                    "setting": "adhesion_type",
                    "value": "Raft"
                }
            },
            "raft_airgap": {
                "label": "Raft Air-gap",
                "description": "The gap between the final raft layer and the first layer of the
object. Only the first layer is raised by this amount to lower the bonding between the raft
layer and the object. Makes it easier to peel off the raft.",
                "unit": "mm",
                "type": "float",
                "default": 0.22,
                "active_if": {
                    "setting": "adhesion_type",
                    "value": "Raft"
                }
            },
            "raft_surface_layers": {
                "label": "Raft Surface Layers",
                "description": "The number of surface layers on top of the 2nd raft layer.
These are fully filled layers that the object sits on. 2 layers usually works fine.",
                "type": "int",
                "default": 2,
                "active_if": {
                    "setting": "adhesion_type",
                    "value": "Raft"
                }
            }
        }
    },
    "blackmagic": {
        "label": "Fixes",
        "visible": true,
        "icon": "category_fixes",
        "settings": {
            "magic_spiralize": {
                "label": "Spiralize the Outer Contour",
                "description": "Spiralize smooths out the Z move of the outer edge. This will
create a steady Z increase over the whole print. This feature turns a solid object into a
```

```
single walled print with a solid bottom. This feature used to be called ‘Joris’ in older
versions.",
          "type": "boolean",
          "default": true,
          "visible": false
        },
        "wireframe_enabled": {
          "label": "Wireframe Printing",
          "description": "Print only the outside surface with a sparse webbed structure,
printing 'in thin air'. This is realized by horizontally printing the contours of the model at
given Z intervals which are connected via upward and diagonally downward lines.",
          "type": "boolean",
          "default": false,
          "visible": false
        },
        "wireframe_printspeed": {
          "label": "Printing speed",
          "description": "Speed at which the nozzle moves when extruding material.",
          "unit": "mm/s",
          "type": "float",
          "default": 7.0,
          "visible": false,
          "active_if": {
            "setting": "wireframe",
            "value": true
          },
          "children": {
            "wireframe_printspeed_bottom": {
              "label": "Bottom printing speed",
              "description": "Speed of printing the first layer, which is the only layer
touching the bluidplatform.",
              "unit": "mm/s",
              "type": "float",
              "default": 7.0,
              "visible": false,
              "inherit":true
            },
            "wireframe_printspeed_up": {
              "label": "Upward printing speed",
              "description": "Speed of printing a line upward 'in thin air'.",
              "unit": "mm/s",
              "type": "float",
              "default": 7.0,
              "visible": false,
              "inherit":true
            },
            "wireframe_printspeed_down": {
              "label": "Downward printing speed",
              "description": "Speed of printing a line diagonally downward.",
              "unit": "mm/s",
              "type": "float",
              "default": 7.0,
              "visible": false,
              "inherit":true
            },
            "wireframe_printspeed_flat": {
              "label": "Horizontal printing speed",
              "description": "Speed of printing the horizontal contours of the object.",
              "unit": "mm/s",
              "type": "float",
              "default": 7.0,
              "visible": false,
              "inherit":true
            }
          }
        },
        "wireframe_flow": {
          "label": "Flow",
          "description": "Flow compensation: the amount of material extruded is
multiplied by this value.",
          "unit": "%",
          "default": 100.0,
          "type": "float",
          "visible": false,
          "active_if": {
            "setting": "wireframe",
            "value": true
          },
          "children": {
            "wireframe_flow_connection": {
              "label": "Connection flow",
              "description": "Flow compensation when going up or down.",
              "unit": "%",
              "default": 100.0,
              "type": "float",
              "visible": false
            },
            "wireframe_flow_flat": {
              "label": "Flat flow",
              "description": "Flow compensation when printing flat lines.",
              "unit": "%",
              "default": 100.0,
                "type": "float",
                "visible": false
            }
        }
    },
    "wireframe_top_delay": {
        "label": "Top delay",
        "description": "Delay time after an upward move, so that the upward line can
harden.",
        "unit": "sec",
        "type": "float",
        "default": 0.0,
        "visible": false,
        "active_if": {
            "setting": "wireframe",
            "value": true
        }
    },
    "wireframe_bottom_delay": {
        "label": "Bottom delay",
        "description": "Delay time after a downward move.",
        "unit": "sec",
        "type": "float",
        "default": 0.0,
        "visible": false,
        "active_if": {
            "setting": "wireframe",
            "value": true
        }
    },
    "wireframe_flat_delay": {
        "label": "Flat delay",
        "description": "Delay time between two horizontal segments. Introducing
such a delay can cause better adhesion to previous layers at the connection points, while
too large delay times cause sagging.",
        "unit": "sec",
        "type": "float",
        "default": 0.1,
        "visible": false,
        "active_if": {
            "setting": "wireframe",
            "value": true
        }
    },
    "wireframe_up_half_speed": {
        "label": "Ease upward",
        "description": "Distance of an upward move which is extruded with half
speed.\nThis can cause better adhesion to previous layers, while not heating the material
in those layers too much.",
        "type": "float",
        "unit": "mm",
        "default": 0.3,
        "visible": false
    },
    "wireframe_top_jump": {
        "label": "Knot size",
        "description": "Creates a small knot at the top of an upward line, so that the
consecutive horizontal layer has a better chance to connect to it.",
        "type": "float",
        "unit": "mm",
        "default": 0.6,
        "visible": false,
        "active_if": {
            "setting": "wireframe",
            "value": true
        }
    },
    "wireframe_fall_down": {
        "label": "Fall down",
        "description": "Distance with which the material falls down after an upward
extrusion. This distance is compensated for.",
        "type": "float",
        "unit": "mm",
        "default": 0.5,
        "visible": false,
        "active_if": {
            "setting": "wireframe",
            "value": true
        }
    },
    "wireframe_drag_along": {
        "label": "Drag along",
        "description": "Distance with which the material of an upward extrusion is
dragged along with the diagonally downward extrusion. This distance is compensated
for.",
        "type": "float",
        "unit": "mm",
        "default": 0.6,
        "visible": false,
        "active_if": {
            "setting": "wireframe",
            "value": true
          }
        },
        "wireframe_strategy": {
          "label": "Strategy",
          "description": "Strategy for making sure two consecutive layers connect at
each connection point. Retraction lets the upward lines harden in the right position, but
may cause filament grinding. A knot can be made at the end of an upward line to
heighten the chance of connecting to it and to let the line cool; however it may require
slow printing speeds. Another strategy is to compensate for the sagging of the top of an
upward line; however, the lines won't always fall down as predicted.",
          "type": "enum",
          "options": [
            "Compensate",
            "Knot",
            "Retract"
          ],
          "default": "Compensate",
          "visible": false,
          "active_if": {
            "setting": "wireframe",
            "value": true
          }
        },
        "wireframe_straight_before_down": {
          "label": "Straighten downward lines",
          "description": "Percentage of a diagonally downward line which is covered
by a horizontal line piece. This can prevent sagging of the top most point of upward
lines.",
          "type": "float",
          "unit": "%",
          "default": 20.0,
          "visible": false,
          "active_if": {
            "setting": "wireframe",
            "value": true
          }
        },
        "wireframe_roof_fall_down": {
          "label": "Roof fall down",
          "description": "The distance which horizontal roof lines printed 'in thin air'
fall down when being printed. This distance is compensated for.",
          "type": "float",
          "unit": "mm",
          "default": 2.0,
          "visible": false,
          "active_if": {
```

```
                        "setting": "wireframe",
                        "value": true
                    }
                },
                "wireframe_roof_drag_along": {
                    "label": "Roof drag along",
                    "description": "The distance of the end piece of an inward line which gets
dragged along when going back to the outer outline of the roof. This distance is
compensated for.",
                    "type": "float",
                    "unit": "mm",
                    "default": 0.8,
                    "visible": false,
                    "active_if": {
                        "setting": "wireframe",
                        "value": true
                    }
                },
                "wireframe_roof_outer_delay": {
                    "label": "Roof outer delay",
                    "description": "Time spent at the outer perimeters of hole which is to become
a roof. Larger times can ensure a better connection.",
                    "type": "boolean",
                    "unit": "sec",
                    "type": "float",
                    "default": 0.2,
                    "visible": false,
                    "active_if": {
                        "setting": "wireframe",
                        "value": true
                    }
                },
                "wireframe_height": {
                    "label": "Connection height.",
                    "description": "The height of the upward and diagonally downward lines
between two horizontal parts.",
                    "type": "float",
                    "unit": "mm",
                    "default": 3.0,
                    "visible": false,
                    "active_if": {
                        "setting": "wireframe",
                        "value": true
                    }
                },
                "wireframe_roof_inset": {
```

```
            "label": "Roof inset distance",
            "description": "The distance covered when making a connection from a roof
outline inward.",
            "type": "float",
            "unit": "mm",
            "default": 3.0,
            "visible": false,
            "active_if": {
              "setting": "wireframe",
              "value": true
            },
            "inherit_function": "wireframe_height"
          },
          "wireframe_nozzle_clearance": {
            "label": "Nozzle clearance",
            "description": "Distance between the nozzle and horizontally downward
lines. Larger clearance results in diagonally downward lines with a less steep angle,
which in turn results in less upward connections with the next layer.",
            "type": "float",
            "unit": "mm",
            "default": 1.0,
            "visible": false,
            "active_if": {
              "setting": "wireframe",
              "value": true
            }
          }
        }
      }
    }
}
```

```
//fffProcessor.h
#ifndef FFF_PROCESSOR_H
#define FFF_PROCESSOR_H

//#define M_PI 3.14159265358979323846  /* pi */

#include <algorithm>
#include <sstream>
#include <fstream>
#include "utils/gettime.h"
#include "utils/logoutput.h"
#include "sliceDataStorage.h"
#include "modelFile/modelFile.h"
#include "slicer.h"
#include "support.h"
#include "multiVolumes.h"
#include "layerPart.h"
#include "inset.h"
#include "skirt.h"
#include "raft.h"
#include "skin.h"
#include "infill.h"
#include "bridge.h"
#include "pathOrderOptimizer.h"
#include "gcodePlanner.h"
#include "gcodeExport.h"
#include "commandSocket.h"
#include "Weaver.h"
#include "Wireframe2gcode.h"
#include "utils/polygonUtils.h"
//@ std::setprecision
#include <iomanip>

namespace cura {

//FusedFilamentFabrication processor.
class fffProcessor : public SettingsBase
{
private:
    int maxObjectHeight;
    int fileNr; //!< used for sequential printing of objects
    GCodeExport gcode;
    TimeKeeper timeKeeper;
    CommandSocket* commandSocket;
    std::ofstream output_file;
```

```
public:
   fffProcessor()
   {
      fileNr = 1;
      maxObjectHeight = 0;
      commandSocket = NULL;
   }

   void resetFileNumber()
   {
      fileNr = 1;
   }

   void setCommandSocket(CommandSocket* socket)
   {
      commandSocket = socket;
   }

   void sendPolygons(PolygonType type, int layer_nr, Polygons& polygons, int
line_width)
   {
      if (commandSocket)
         commandSocket->sendPolygons(type, layer_nr, polygons, line_width);
   }

   bool setTargetFile(const char* filename)
   {
      output_file.open(filename);
      if (output_file.is_open())
      {
         gcode.setOutputStream(&output_file);
         return true;
      }
      return false;
   }

   void setTargetStream(std::ostream* stream)
   {
      gcode.setOutputStream(stream);
   }

   bool processFiles(const std::vector<std::string> &files)
   {
      timeKeeper.restart();
      PrintObject* model = nullptr;
```

```
    model = new PrintObject(this);
    for(std::string filename : files)
    {
        log("Loading %s from disk...\n", filename.c_str());

        FMatrix3x3 matrix;
        if (!loadMeshFromFile(model, filename.c_str(), matrix))
        {
            logError("Failed to load model: %s\n", filename.c_str());
            return false;
        }
    }
    model->finalize();

    log("Loaded from disk in %5.3fs\n", timeKeeper.restart());
    return processModel(model);
}

bool processModel(PrintObject* model)
{
    timeKeeper.restart();
    if (!model)
        return false;

    TimeKeeper timeKeeperTotal;

    if (model->getSettingBoolean("wireframe_enabled"))
    {
        log("starting Neith Weaver...\n");

        Weaver w(this);
        w.weave(model, commandSocket);

        log("starting Neith Gcode generation...\n");
        preSetup();
        Wireframe2gcode gcoder(w, gcode, this);
        gcoder.writeGCode(commandSocket, maxObjectHeight);
        log("finished Neith Gcode generation...\n");

    } else
    {
        SliceDataStorage storage;
        preSetup();

        if (!prepareModel(storage, model))
            return false;
            processSliceData(storage);
            writeGCode(storage);

    std::cerr << "machine_gcode_flavor = " << model-
>getSettingString("machine_gcode_flavor") << std::endl;
    std::cerr << "machine_gcode_flavor = " << model-
>getSettingAsGCodeFlavor("machine_gcode_flavor") << std::endl;
        }

        logProgress("process", 1, 1);//Report the GUI that a file has been fully processed.
        log("Total time elapsed %5.2fs.\n", timeKeeperTotal.restart());

        return true;
    }

    void finalize()
    {
        gcode.finalize(maxObjectHeight,
getSettingInMillimetersPerSecond("speed_travel"),
getSettingString("machine_end_gcode").c_str());
        for(int e=0; e<MAX_EXTRUDERS; e++)
            gcode.writeTemperatureCommand(e, 0, false);
    }

    double getTotalFilamentUsed(int e)
    {
        return gcode.getTotalFilamentUsed(e);
    }

    double getTotalPrintTime()
    {
        return gcode.getTotalPrintTime();
    }

private:
    void preSetup()
    {
        for(unsigned int n=1; n<MAX_EXTRUDERS;n++)
        {
            std::ostringstream stream;
            stream << "machine_extruder_offset" << n;
            if (hasSetting(stream.str() + "_x") || hasSetting(stream.str() + "_y"))
                gcode.setExtruderOffset(n, Point(getSettingInMicrons(stream.str() + "_x"),
getSettingInMicrons(stream.str() + "_y")));
```

```
        }
        for(unsigned int n=0; n<MAX_EXTRUDERS;n++)
        {
            std::ostringstream stream;
            stream << n;
            if (hasSetting("machine_pre_extruder_switch_code" + stream.str()) ||
hasSetting("machine_post_extruder_switch_code" + stream.str()))
                gcode.setSwitchExtruderCode(n,
getSettingString("machine_pre_extruder_switch_code" + stream.str()),
getSettingString("machine_post_extruder_switch_code" + stream.str()));

            gcode.setFilamentDiameter(n, getSettingInMicrons("material_diameter")); //
TODO: separate for each nozzle!
        }

        gcode.setFlavor(getSettingAsGCodeFlavor("machine_gcode_flavor"));

gcode.setRetractionSettings(getSettingInMicrons("machine_switch_extruder_retraction_
amount"),
getSettingInMillimetersPerSecond("material_switch_extruder_retraction_speed"),
getSettingInMillimetersPerSecond("material_switch_extruder_prime_speed"),
getSettingInMicrons("retraction_extrusion_window"),
getSettingAsCount("retraction_count_max"));
    }

    bool prepareModel(SliceDataStorage& storage, PrintObject* object) /// slices the
model
    {
        storage.model_min = object->min();
        storage.model_max = object->max();
        storage.model_size = storage.model_max - storage.model_min;

        log("Slicing model...\n");
        int initial_layer_thickness = object->getSettingInMicrons("layer_height_0");
        int layer_thickness = object->getSettingInMicrons("layer_height");
        if (object->getSettingAsPlatformAdhesion("adhesion_type") == Adhesion_Raft)
        {
            initial_layer_thickness = layer_thickness;
        }
        int initial_slice_z = (initial_layer_thickness - layer_thickness / 2);
        int layer_count = (storage.model_max.z - initial_slice_z) / layer_thickness + 1;
        std::vector<Slicer*> slicerList;
        for(Mesh& mesh : object->meshes)
        {
            Slicer* slicer = new Slicer(&mesh, initial_slice_z, layer_thickness, layer_count,
mesh.getSettingBoolean("meshfix_keep_open_polygons"),
mesh.getSettingBoolean("meshfix_extensive_stitching"));
            slicerList.push_back(slicer);
            /*
            for(SlicerLayer& layer : slicer->layers)
            {
                //Reporting the outline here slows down the engine quite a bit, so only do so
when debugging.
                //sendPolygons("outline", layer_nr, layer.z, layer.polygonList);
                //sendPolygons("openoutline", layer_nr, layer.openPolygonList);
            }
            */
        }

        if (false) { // remove empty first layers
            int n_empty_first_layers = 0;
            for (int layer_idx = 0; layer_idx < layer_count; layer_idx++)
            {
                bool layer_is_empty = true;
                for (Slicer* slicer : slicerList)
                {
                    if (slicer->layers[layer_idx].polygonList.size() > 0)
                    {
                        layer_is_empty = false;
                        break;
                    }
                }

                if (layer_is_empty)
                {
                    n_empty_first_layers++;
                } else
                {
                    break;
                }
            }

            if (n_empty_first_layers > 0)
            {
                for (Slicer* slicer : slicerList)
                {
                    std::vector<SlicerLayer>& layers = slicer->layers;
                    layers.erase(layers.begin(), layers.begin() + n_empty_first_layers);
                    for (SlicerLayer& layer : layers)
                    {
```

```
                    layer.z -= n_empty_first_layers * layer_thickness;
                }
            }
            layer_count -= n_empty_first_layers;
        }
    }

    log("Layer count: %i\n", layer_count);
    log("Sliced model in %5.3fs\n", timeKeeper.restart());

    object->clear();///Clear the mesh data, it is no longer needed after this point, and it
saves a lot of memory.

    log("Generating layer parts...\n");
    storage.meshes.reserve(slicerList.size());
    for(unsigned int meshIdx=0; meshIdx < slicerList.size(); meshIdx++)
    {
        storage.meshes.emplace_back(&object->meshes[meshIdx]);
        SliceMeshStorage& meshStorage = storage.meshes[meshIdx];
        createLayerParts(meshStorage, slicerList[meshIdx], meshStorage.settings-
>getSettingBoolean("meshfix_union_all"), meshStorage.settings-
>getSettingBoolean("meshfix_union_all_remove_holes"));
        //@createLayerParts(meshStorage, slicerList[meshIdx], true,
meshStorage.settings->getSettingBoolean("meshfix_union_all_remove_holes"));
        delete slicerList[meshIdx];

        bool has_raft = meshStorage.settings-
>getSettingAsPlatformAdhesion("adhesion_type") == Adhesion_Raft;
        for(unsigned int layer_nr=0; layer_nr<meshStorage.layers.size(); layer_nr++)
        {
            //Add the raft offset to each layer.
            if (has_raft)
            {
                meshStorage.layers[layer_nr].printZ +=
                    meshStorage.settings->getSettingInMicrons("raft_base_thickness")
                    + meshStorage.settings->getSettingInMicrons("raft_interface_thickness")
                    + meshStorage.settings->getSettingAsCount("raft_surface_layers") *
getSettingInMicrons("layer_height") //raft_surface_thickness")
                    + meshStorage.settings->getSettingInMicrons("raft_airgap")
                    - initial_slice_z;
            }
            else
            {
                meshStorage.layers[layer_nr].printZ +=
                    meshStorage.settings->getSettingInMicrons("layer_height_0")
                    - initial_slice_z;
            }
        }
    }
    log("Generated layer parts in %5.3fs\n", timeKeeper.restart());

    log("Finished prepareModel.\n");
    return true;
}

void processSliceData(SliceDataStorage& storage)
{
    if (commandSocket)
      commandSocket->beginSendSlicedObject();

    // const
    unsigned int totalLayers = storage.meshes[0].layers.size();

    //carveMultipleVolumes(storage.meshes);
    generateMultipleVolumesOverlap(storage.meshes,
getSettingInMicrons("multiple_mesh_overlap"));
    //dumpLayerparts(storage, "c:/models/output.html");
    if (getSettingBoolean("magic_polygon_mode"))
    {
        for(unsigned int layer_nr=0; layer_nr<totalLayers; layer_nr++)
        {
            for(SliceMeshStorage& mesh : storage.meshes)
            {
                SliceLayer* layer = &mesh.layers[layer_nr];
                for(SliceLayerPart& part : layer->parts)
                {
                    sendPolygons(Inset0Type, layer_nr, part.outline, mesh.settings-
>getSettingInMicrons("wall_line_width_x"));
                }
            }
        }
        return;
    }

    for(unsigned int layer_nr=0; layer_nr<totalLayers; layer_nr++)
    {
        for(SliceMeshStorage& mesh : storage.meshes)
        {
            if(commandSocket)
            {
                int initial_layer_thickness = mesh.settings-
>getSettingInMicrons("layer_height_0");
```

```
                int layer_thickness = mesh.settings->getSettingInMicrons("layer_height");
                if (mesh.settings->getSettingAsPlatformAdhesion("adhesion_type") ==
Adhesion_Raft)
                {
                    initial_layer_thickness = layer_thickness;
                }
                commandSocket->sendLayerInfo(layer_nr, mesh.layers[layer_nr].printZ,
layer_nr == 0 ? initial_layer_thickness : layer_thickness);
            }

            int insetCount = mesh.settings->getSettingAsCount("wall_line_count");
            if (mesh.settings->getSettingBoolean("magic_spiralize") &&
static_cast<int>(layer_nr) < mesh.settings->getSettingAsCount("bottom_layers") &&
layer_nr % 2 == 1)//Add extra insets every 2 layers when spiralizing, this makes bottoms
of cups watertight.
                insetCount += 5;
            SliceLayer* layer = &mesh.layers[layer_nr];
            int wall_line_width_0 = mesh.settings-
>getSettingInMicrons("wall_line_width_0");
            int wall_line_width_x = mesh.settings-
>getSettingInMicrons("wall_line_width_x");
            int inset_count = insetCount;
            if (mesh.settings->getSettingBoolean("alternate_extra_perimeter"))
                inset_count += layer_nr % 2;
            generateInsets(layer, wall_line_width_0, wall_line_width_x, inset_count,
mesh.settings->getSettingBoolean("wall_overlap_avoid_enabled"));

            for(unsigned int partNr=0; partNr<layer->parts.size(); partNr++)
            {
                if (layer->parts[partNr].insets.size() > 0)
                {
                    sendPolygons(Inset0Type, layer_nr, layer->parts[partNr].insets[0],
wall_line_width_x);
                    for(unsigned int inset=1; inset<layer->parts[partNr].insets.size(); inset++)
                        sendPolygons(InsetXType, layer_nr, layer->parts[partNr].insets[inset],
wall_line_width_x);
                }
            }
        }
        logProgress("inset",layer_nr+1,totalLayers);
        if (commandSocket) commandSocket->sendProgress(1.0/3.0 * float(layer_nr) /
float(totalLayers));
    }

    { // remove empty first layers
        int n_empty_first_layers = 0;
```

```
            for (unsigned int layer_idx = 0; layer_idx < totalLayers; layer_idx++)
            {
                bool layer_is_empty = true;
                for (SliceMeshStorage& mesh : storage.meshes)
                {
                    if (mesh.layers[layer_idx].parts.size() > 0)
                    {
                        layer_is_empty = false;
                        break;
                    }
                }

                if (layer_is_empty)
                {
                    n_empty_first_layers++;
                } else
                {
                    break;
                }
            }

            if (n_empty_first_layers > 0)
            {
                log("Removing %d layers because they are empty\n", n_empty_first_layers);
                for (SliceMeshStorage& mesh : storage.meshes)
                {
                    std::vector<SliceLayer>& layers = mesh.layers;
                    layers.erase(layers.begin(), layers.begin() + n_empty_first_layers);
                    for (SliceLayer& layer : layers)
                    {
                        layer.printZ -= n_empty_first_layers *
getSettingInMicrons("layer_height");
                    }
                }
                totalLayers -= n_empty_first_layers;
            }
        }
        if (totalLayers < 1)
        {
            log("Stopping process because there are no layers.\n");
            return;
        }

        if (getSettingBoolean("ooze_shield_enabled"))
        {
            for(unsigned int layer_nr=0; layer_nr<totalLayers; layer_nr++)
            {
               Polygons oozeShield;
               for(SliceMeshStorage& mesh : storage.meshes)
               {
                  for(SliceLayerPart& part : mesh.layers[layer_nr].parts)
                  {
                     oozeShield = 
oozeShield.unionPolygons(part.outline.offset(MM2INT(2.0))); // TODO: put hard coded 
value in a variable with an explanatory name (and make var a parameter, and perhaps 
even a setting?)
                  }
               }
               storage.oozeShield.push_back(oozeShield);
            }

            for(unsigned int layer_nr=0; layer_nr<totalLayers; layer_nr++)
               storage.oozeShield[layer_nr] = storage.oozeShield[layer_nr].offset(-
MM2INT(1.0)).offset(MM2INT(1.0)); // TODO: put hard coded value in a variable with 
an explanatory name (and make var a parameter, and perhaps even a setting?)
            int offsetAngle = tan(getSettingInAngleRadians("ooze_shield_angle")) * 
getSettingInMicrons("layer_height");//Allow for a 60deg angle in the oozeShield.
            for(unsigned int layer_nr=1; layer_nr<totalLayers; layer_nr++)
               storage.oozeShield[layer_nr] = 
storage.oozeShield[layer_nr].unionPolygons(storage.oozeShield[layer_nr-1].offset(-
offsetAngle));
            for(unsigned int layer_nr=totalLayers-1; layer_nr>0; layer_nr--)
               storage.oozeShield[layer_nr-1] = storage.oozeShield[layer_nr-
1].unionPolygons(storage.oozeShield[layer_nr].offset(-offsetAngle));
         }
         log("Generated inset in %5.3fs\n", timeKeeper.restart());

         log("Generating support areas...\n");
         for(SliceMeshStorage& mesh : storage.meshes)
         {
            generateSupportAreas(storage, &mesh, totalLayers);
         }
         log("Generated support areas in %5.3fs\n", timeKeeper.restart());

         for(unsigned int layer_nr=0; layer_nr<totalLayers; layer_nr++)
         {
            if (!getSettingBoolean("magic_spiralize") || static_cast<int>(layer_nr) < 
getSettingAsCount("bottom_layers"))    //Only generate up/downskin and infill for the 
first X layers when spiralize is choosen.
            {
               for(SliceMeshStorage& mesh : storage.meshes)
               {
                  int extrusionWidth = mesh.settings-
>getSettingInMicrons("wall_line_width_x");
                  generateSkins(layer_nr, mesh, extrusionWidth, mesh.settings-
>getSettingAsCount("bottom_layers"), mesh.settings->getSettingAsCount("top_layers"),
mesh.settings->getSettingAsCount("skin_outline_count"), mesh.settings-
>getSettingBoolean("wall_overlap_avoid_enabled"));
                  if (mesh.settings->getSettingInMicrons("infill_line_distance") > 0)
                  {
                     int infill_skin_overlap = 0;
                     if (mesh.settings->getSettingInMicrons("infill_line_distance") >
mesh.settings->getSettingInMicrons("infill_line_width") + 10)
                     {
                        infill_skin_overlap = extrusionWidth / 2;
                     }
                     generateSparse(layer_nr, mesh, extrusionWidth, infill_skin_overlap);
                     if (mesh.settings->getSettingString("fill_perimeter_gaps") == "Skin")
                     {
                        generatePerimeterGaps(layer_nr, mesh, extrusionWidth, mesh.settings-
>getSettingAsCount("bottom_layers"), mesh.settings-
>getSettingAsCount("top_layers"));
                     }
                     else if (mesh.settings->getSettingString("fill_perimeter_gaps") ==
"Everywhere")
                     {
                        generatePerimeterGaps(layer_nr, mesh, extrusionWidth, 0, 0);
                     }
                  }

                  SliceLayer& layer = mesh.layers[layer_nr];
                  for(SliceLayerPart& part : layer.parts)
                  {
                     for (SkinPart& skin_part : part.skin_parts)
                     {
                        sendPolygons(SkinType, layer_nr, skin_part.outline, extrusionWidth);
                     }
                  }
               }
            }
            logProgress("skin", layer_nr+1, totalLayers);
            if (commandSocket) commandSocket->sendProgress(1.0/3.0 + 1.0/3.0 *
float(layer_nr) / float(totalLayers));
         }
         for(unsigned int layer_nr=totalLayers-1; layer_nr>0; layer_nr--)
         {
            for(SliceMeshStorage& mesh : storage.meshes)
```

```
            combineSparseLayers(layer_nr, mesh, mesh.settings-
>getSettingAsCount("fill_sparse_combine"));
        }
        log("Generated up/down skin in %5.3fs\n", timeKeeper.restart());

        if (getSettingInMicrons("wipe_tower_distance") > 0 &&
getSettingInMicrons("wipe_tower_size") > 0)
        {
            PolygonRef p = storage.wipeTower.newPoly();
            int tower_size = getSettingInMicrons("wipe_tower_size");
            int tower_distance = getSettingInMicrons("wipe_tower_distance");
            p.add(Point(storage.model_min.x - tower_distance, storage.model_max.y +
tower_distance));
            p.add(Point(storage.model_min.x - tower_distance, storage.model_max.y +
tower_distance + tower_size));
            p.add(Point(storage.model_min.x - tower_distance - tower_size,
storage.model_max.y + tower_distance + tower_size));
            p.add(Point(storage.model_min.x - tower_distance - tower_size,
storage.model_max.y + tower_distance));

            storage.wipePoint = Point(storage.model_min.x - tower_distance - tower_size / 2,
storage.model_max.y + tower_distance + tower_size / 2);
        }

        int adhesion_line_width = 0;
        switch(getSettingAsPlatformAdhesion("adhesion_type"))
        {
        case Adhesion_None:
            adhesion_line_width = getSettingInMicrons("skirt_line_width");
            generateSkirt(storage, getSettingInMicrons("skirt_gap"), adhesion_line_width,
getSettingAsCount("skirt_line_count"), getSettingInMicrons("skirt_minimal_length"));
            break;
        case Adhesion_Brim:
            adhesion_line_width = getSettingInMicrons("skirt_line_width");
            generateSkirt(storage, 0, adhesion_line_width,
getSettingAsCount("brim_line_count"), getSettingInMicrons("skirt_minimal_length"));
            break;
        case Adhesion_Raft:
            generateRaft(storage, getSettingInMicrons("raft_margin"));
            break;
        }

        sendPolygons(SkirtType, 0, storage.skirt, adhesion_line_width);
    }

    void writeGCode(SliceDataStorage& storage)
    {
        gcode.resetTotalPrintTimeAndFilament();

        if (commandSocket)
            commandSocket->beginGCode();

        //Setup the retraction parameters.
        storage.retraction_config.amount =
INT2MM(getSettingInMicrons("retraction_amount"));
        storage.retraction_config.primeAmount =
INT2MM(getSettingInMicrons("retraction_extra_prime_amount"));
        storage.retraction_config.speed =
getSettingInMillimetersPerSecond("retraction_retract_speed");
        storage.retraction_config.primeSpeed =
getSettingInMillimetersPerSecond("retraction_prime_speed");
        storage.retraction_config.zHop = getSettingInMicrons("retraction_hop");
        for(SliceMeshStorage& mesh : storage.meshes)
        {
            mesh.retraction_config.amount = INT2MM(mesh.settings-
>getSettingInMicrons("retraction_amount"));
            mesh.retraction_config.primeAmount = INT2MM(mesh.settings-
>getSettingInMicrons("retraction_extra_prime_amount"));
            mesh.retraction_config.speed = mesh.settings-
>getSettingInMillimetersPerSecond("retraction_retract_speed");
            mesh.retraction_config.primeSpeed = mesh.settings-
>getSettingInMillimetersPerSecond("retraction_prime_speed");
            mesh.retraction_config.zHop = mesh.settings-
>getSettingInMicrons("retraction_hop");
        }

        if (fileNr == 1)
        {
            if (gcode.getFlavor() != GCODE_FLAVOR_ULTIGCODE)
            {//@ RepRap
                if (hasSetting("material_bed_temperature") &&
getSettingInDegreeCelsius("material_bed_temperature") > 0)

gcode.writeBedTemperatureCommand(getSettingInDegreeCelsius("material_bed_temper
ature"), true);

                for(SliceMeshStorage& mesh : storage.meshes)
                    if (mesh.settings->hasSetting("material_print_temperature") &&
mesh.settings->getSettingInDegreeCelsius("material_print_temperature") > 0)
                        gcode.writeTemperatureCommand(mesh.settings-
>getSettingAsIndex("extruder_nr"), mesh.settings-
>getSettingInDegreeCelsius("material_print_temperature"));
```

```
            for(SliceMeshStorage& mesh : storage.meshes)
               if (mesh.settings->hasSetting("material_print_temperature") &&
mesh.settings->getSettingInDegreeCelsius("material_print_temperature") > 0)
                  gcode.writeTemperatureCommand(mesh.settings-
>getSettingAsIndex("extruder_nr"), mesh.settings-
>getSettingInDegreeCelsius("material_print_temperature"), true);
            gcode.writeCode(getSettingString("machine_start_gcode").c_str());
            //@ set welder_on and welder_off gcode string if metal printing
            if (getSettingBoolean("machine_metal_printing")){
               gcode.setIsMetalPrinting(getSettingBoolean("machine_metal_printing"));
               gcode.setWelderOn(getSettingString("machine_welder_on_gcode"));
               gcode.setWelderOff(getSettingString("machine_welder_off_gcode"));

gcode.setMinDistWelderOff(getSettingInMillimetersPerSecond("machine_min_dist_wel
der_off"));
               gcode.setIsWelding(false); //@ initial that welding is on or not
            }
         }
         gcode.writeComment("Generated with Cura_SteamEngine " VERSION);
         //@ add line_width and layer_height to the gcode output
         gcode.writeComment("////////////////////////////////////////");
         std::string tempString;
         tempString = getSettingString("skin_line_width");
         gcode.writeComment("Line width: " + tempString + " mm.");
         tempString = getSettingString("layer_height");
         gcode.writeComment("Layer height: " + tempString + " mm.");
         tempString = getSettingString("speed_print");
         gcode.writeComment("Printing speed: " + tempString + " mm/s");
         tempString = getSettingString("material_diameter");
         gcode.writeComment("Material diameter: " + tempString + " mm.");

         double lineWidth = INT2MM(getSettingInMicrons("infill_line_width"));
         double layerHeight = INT2MM(getSettingInMicrons("layer_height"));
         double speedPrint = INT2MM(getSettingInMicrons("speed_print"));
         double expectedMM3PerSec = lineWidth * layerHeight * speedPrint;
         double materialDiameter = INT2MM(getSettingInMicrons("material_diameter"));
         double crossSectionalArea =
M_PI*(materialDiameter/2.0)*(materialDiameter/2.0);
         double wireSpeed;
         double wireMMPerSec;
         for(int i=1;i<=100;i++){
            //@ this equation only for Millermatic 190
            //@ wireMMPerSec = 0.0254 * (0.45*i/100.0+20.5) * i/100.0;
            wireMMPerSec = (2.216*i)-19;
            if ((crossSectionalArea*wireMMPerSec) < expectedMM3PerSec){
               wireSpeed = i;
```

```
                }
                else {
                    break;
                }
            }
            //@ expected material in volume
            std::ostringstream expVolume;
            expVolume << expectedMM3PerSec;
            gcode.writeComment("Expected Material: " + expVolume.str() + " mm3/s.");
            gcode.writeComment("///////////////////////////////////////");
            gcode.writeComment("Recommended welder (Millermatic 190) settings");
            gcode.writeComment("Voltage setting: 5");
            if (wireSpeed >=10)
            {
                std::ostringstream tempStr;
                tempStr << (wireSpeed - 2.0) << " - " << (wireSpeed + 2.0);
                gcode.writeComment("Wire speed: " + tempStr.str());
            }
            else
            {
                gcode.writeComment("Wire speed: <10 (Cannot be set!)");
            }
            gcode.writeComment("///////////////////////////////////////\n");

            if (gcode.getFlavor() == GCODE_FLAVOR_BFB)
            {
                gcode.writeComment("enable auto-retraction");
                std::ostringstream tmp;
                tmp << "M227 S" << (getSettingInMicrons("retraction_amount") * 2560 /
1000) << " P" << (getSettingInMicrons("retraction_amount") * 2560 / 1000);
                gcode.writeLine(tmp.str().c_str());
            }
        }
        else
        {
            gcode.writeFanCommand(0);
            gcode.resetExtrusionValue();
            gcode.setZ(maxObjectHeight + 5000);
            gcode.writeMove(gcode.getPositionXY(),
getSettingInMillimetersPerSecond("speed_travel"), 0);
            gcode.writeMove(Point(storage.model_min.x, storage.model_min.y),
getSettingInMillimetersPerSecond("speed_travel"), 0);
        }
        fileNr++;

        unsigned int totalLayers = storage.meshes[0].layers.size();
        //gcode.writeComment("Layer count: %d", totalLayers);

        bool has_raft = getSettingAsPlatformAdhesion("adhesion_type") ==
Adhesion_Raft;
        if (has_raft)
        {
            //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter has_raft"); //@ for
test.
            GCodePathConfig raft_base_config(&storage.retraction_config, "SUPPORT");

raft_base_config.setSpeed(getSettingInMillimetersPerSecond("raft_base_speed"));
            raft_base_config.setLineWidth(getSettingInMicrons("raft_base_linewidth"));
            raft_base_config.setLayerHeight(getSettingInMicrons("raft_base_thickness"));
            raft_base_config.setFlow(getSettingInPercentage("material_flow"));
            GCodePathConfig raft_interface_config(&storage.retraction_config,
"SUPPORT");

raft_interface_config.setSpeed(getSettingInMillimetersPerSecond("raft_base_speed"));
            raft_interface_config.setLineWidth(getSettingInMicrons("raft_base_linewidth"));

raft_interface_config.setLayerHeight(getSettingInMicrons("raft_base_thickness"));
            raft_interface_config.setFlow(getSettingInPercentage("material_flow"));
            GCodePathConfig raft_surface_config(&storage.retraction_config, "SUPPORT");

raft_surface_config.setSpeed(getSettingInMillimetersPerSecond("raft_base_speed"));
            raft_surface_config.setLineWidth(getSettingInMicrons("raft_base_linewidth"));
            raft_surface_config.setLayerHeight(getSettingInMicrons("raft_base_thickness"));
            raft_surface_config.setFlow(getSettingInPercentage("material_flow"));

            {
                gcode.writeLayerComment(-3);
                gcode.writeComment("RAFT");
                GCodePlanner gcodeLayer(gcode, &storage.retraction_config,
getSettingInMillimetersPerSecond("speed_travel"),
getSettingInMicrons("retraction_min_travel"));
                if (getSettingAsIndex("support_extruder_nr") > 0)
                    gcodeLayer.setExtruder(getSettingAsIndex("support_extruder_nr"));
                gcode.setZ(getSettingInMicrons("raft_base_thickness"));
                gcodeLayer.addPolygonsByOptimizer(storage.raftOutline, &raft_base_config);

                Polygons raftLines;
                int offset_from_poly_outline = 0;
                generateLineInfill(storage.raftOutline, offset_from_poly_outline, raftLines,
getSettingInMicrons("raft_base_linewidth"), getSettingInMicrons("raft_line_spacing"),
getSettingInPercentage("fill_overlap"), 0);
                gcodeLayer.addLinesByOptimizer(raftLines, &raft_base_config);
                gcode.writeFanCommand(getSettingInPercentage("cool_fan_speed_max"));
                gcodeLayer.writeGCode(false, getSettingInMicrons("raft_base_thickness"));
            }

            {
                gcode.writeLayerComment(-2);
                gcode.writeComment("RAFT");
                GCodePlanner gcodeLayer(gcode, &storage.retraction_config,
getSettingInMillimetersPerSecond("speed_travel"),
getSettingInMicrons("retraction_min_travel"));
                gcode.setZ(getSettingInMicrons("raft_base_thickness") +
getSettingInMicrons("raft_interface_thickness"));

                Polygons raftLines;
                int offset_from_poly_outline = 0;
                int raft_interface_line_width = getSettingInMicrons("wall_line_width_x"); //
getSettingInMicrons("raft_interface_line_width")
                int raft_interface_line_spacing = getSettingInMicrons("raft_line_spacing"); //
getSettingInMicrons("raft_interface_line_spacing")
                generateLineInfill(storage.raftOutline, offset_from_poly_outline, raftLines,
raft_interface_line_width, raft_interface_line_spacing,
getSettingInPercentage("fill_overlap"), getSettingAsCount("raft_surface_layers") > 0 ?
45 : 90);
                gcodeLayer.addLinesByOptimizer(raftLines, &raft_interface_config);

                gcodeLayer.writeGCode(false,
getSettingInMicrons("raft_interface_thickness"));
            }

            for (int raftSurfaceLayer=1;
raftSurfaceLayer<=getSettingAsCount("raft_surface_layers"); raftSurfaceLayer++)
            {
                gcode.writeLayerComment(-1);
                gcode.writeComment("RAFT");
                int raft_surface_thickness = getSettingInMicrons("layer_height"); //
getSettingInMicrons("raft_surface_thickness")
                GCodePlanner gcodeLayer(gcode, &storage.retraction_config,
getSettingInMillimetersPerSecond("speed_travel"),
getSettingInMicrons("retraction_min_travel"));
                gcode.setZ(getSettingInMicrons("raft_base_thickness") +
getSettingInMicrons("raft_interface_thickness") +
raft_surface_thickness*raftSurfaceLayer);

                Polygons raftLines;
                int offset_from_poly_outline = 0;
                int raft_surface_line_width = getSettingInMicrons("wall_line_width_0"); //
getSettingInMicrons("raft_surface_line_width")
                int raft_surface_line_spacing = raft_surface_line_width; //
getSettingInMicrons("raft_surface_line_spacing")
                generateLineInfill(storage.raftOutline, offset_from_poly_outline, raftLines,
raft_surface_line_width, raft_surface_line_spacing,
getSettingInPercentage("fill_overlap"), (raftSurfaceLayer % 2 == 0)? 0 : 90);
                gcodeLayer.addLinesByOptimizer(raftLines, &raft_surface_config);

                gcodeLayer.writeGCode(false,
getSettingInMicrons("raft_interface_thickness"));
         }
      }
      //@ add vairables for pause time between layers
      double pauseTime = INT2MM(getSettingInMicrons("machine_layer_pause_time"));
      double pauseIncrease =
INT2MM(getSettingInMicrons("machine_layer_pause_increase"));
      std::string pauseGcode = getSettingString("machine_layer_pause_gcode");
      //@ add variable for move the printer head up at the end of each layer
      double upLayerEnd = INT2MM(getSettingInMicrons("machine_up_layer_end"));
      //@ welder off gcode
      std::string welderOffGCode = getSettingString("machine_welder_off_gcode");
      //@ boolean layer pause
      bool layerPause = getSettingBoolean("machine_layer_pause");

      for(unsigned int layer_nr=0; layer_nr<totalLayers; layer_nr++)
      {
         logProgress("export", layer_nr+1, totalLayers);
         if (commandSocket) commandSocket->sendProgress(2.0/3.0 + 1.0/3.0 *
float(layer_nr) / float(totalLayers));

         int layer_thickness = getSettingInMicrons("layer_height");
         if (layer_nr == 0 && !has_raft)
         {
                layer_thickness = getSettingInMicrons("layer_height_0");
         }

         storage.skirt_config.setSpeed(getSettingInMillimetersPerSecond("skirt_speed"));
         storage.skirt_config.setLineWidth(getSettingInMicrons("skirt_line_width"));
         storage.skirt_config.setFlow(getSettingInPercentage("material_flow"));
         storage.skirt_config.setLayerHeight(layer_thickness);


storage.support_config.setLineWidth(getSettingInMicrons("support_line_width"));

storage.support_config.setSpeed(getSettingInMillimetersPerSecond("speed_support"));
```

```
        storage.support_config.setFlow(getSettingInPercentage("material_flow"));
        storage.support_config.setLayerHeight(layer_thickness);
        for(SliceMeshStorage& mesh : storage.meshes)
        {
            mesh.inset0_config.setLineWidth(mesh.settings-
>getSettingInMicrons("wall_line_width_0"));
            mesh.inset0_config.setSpeed(mesh.settings-
>getSettingInMillimetersPerSecond("speed_wall_0"));
            mesh.inset0_config.setFlow(mesh.settings-
>getSettingInPercentage("material_flow"));
            mesh.inset0_config.setLayerHeight(layer_thickness);

            mesh.insetX_config.setLineWidth(mesh.settings-
>getSettingInMicrons("wall_line_width_x"));
            mesh.insetX_config.setSpeed(mesh.settings-
>getSettingInMillimetersPerSecond("speed_wall_x"));
            mesh.insetX_config.setFlow(mesh.settings-
>getSettingInPercentage("material_flow"));
            mesh.insetX_config.setLayerHeight(layer_thickness);

            mesh.skin_config.setLineWidth(mesh.settings-
>getSettingInMicrons("skin_line_width"));
            mesh.skin_config.setSpeed(mesh.settings-
>getSettingInMillimetersPerSecond("speed_topbottom"));
            mesh.skin_config.setFlow(mesh.settings-
>getSettingInPercentage("material_flow"));
            mesh.skin_config.setLayerHeight(layer_thickness);

            for(unsigned int idx=0; idx<MAX_SPARSE_COMBINE; idx++)
            {
                mesh.infill_config[idx].setLineWidth(mesh.settings-
>getSettingInMicrons("infill_line_width") * (idx + 1));
                mesh.infill_config[idx].setSpeed(mesh.settings-
>getSettingInMillimetersPerSecond("speed_infill"));
                mesh.infill_config[idx].setFlow(mesh.settings-
>getSettingInPercentage("material_flow"));
                mesh.infill_config[idx].setLayerHeight(layer_thickness);
            }
        }

        int initial_speedup_layers = getSettingAsCount("speed_slowdown_layers");
        if (static_cast<int>(layer_nr) < initial_speedup_layers)
        {
            int initial_layer_speed = getSettingInMillimetersPerSecond("speed_layer_0");
            storage.support_config.smoothSpeed(initial_layer_speed, layer_nr,
initial_speedup_layers);
```

```
            for(SliceMeshStorage& mesh : storage.meshes)
            {
                mesh.inset0_config.smoothSpeed(initial_layer_speed, layer_nr,
initial_speedup_layers);
                mesh.insetX_config.smoothSpeed(initial_layer_speed, layer_nr,
initial_speedup_layers);
                mesh.skin_config.smoothSpeed(initial_layer_speed, layer_nr,
initial_speedup_layers);
                for(unsigned int idx=0; idx<MAX_SPARSE_COMBINE; idx++)
                {
                    mesh.infill_config[idx].smoothSpeed(initial_layer_speed, layer_nr,
initial_speedup_layers);
                }
            }
        }
        //@ start layer
        gcode.writeLayerComment(layer_nr);

        GCodePlanner gcodeLayer(gcode, &storage.retraction_config,
getSettingInMillimetersPerSecond("speed_travel"),
getSettingInMicrons("retraction_min_travel"));

        int z = storage.meshes[0].layers[layer_nr].printZ;

        gcode.setZ(z);
        gcode.resetStartPosition();

        if (layer_nr == 0)
        {
            if (storage.skirt.size() > 0)
                gcodeLayer.addTravel(storage.skirt[storage.skirt.size()-
1].closestPointTo(gcode.getPositionXY()));
            gcodeLayer.addPolygonsByOptimizer(storage.skirt, &storage.skirt_config);
        }

        bool printSupportFirst = (storage.support.generated &&
getSettingAsIndex("support_extruder_nr") > 0 &&
getSettingAsIndex("support_extruder_nr") == gcodeLayer.getExtruder());
        if (printSupportFirst)
            addSupportToGCode(storage, gcodeLayer, layer_nr);

        if (storage.oozeShield.size() > 0)
        {
            //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter storage oozeShield
size > 0"); //@ for test.
            gcodeLayer.setAlwaysRetract(true);
            gcodeLayer.addPolygonsByOptimizer(storage.oozeShield[layer_nr],
&storage.skirt_config);
            gcodeLayer.setAlwaysRetract(!getSettingBoolean("retraction_combing"));
        }

        //Figure out in which order to print the meshes, do this by looking at the current
extruder and preferer the meshes that use that extruder.
        std::vector<SliceMeshStorage*> mesh_order = calculateMeshOrder(storage,
gcodeLayer.getExtruder());
        for(SliceMeshStorage* mesh : mesh_order)
        {
            addMeshLayerToGCode(storage, mesh, gcodeLayer, layer_nr);
        }
        if (!printSupportFirst)
            addSupportToGCode(storage, gcodeLayer, layer_nr);

        { //Finish the layer by applying speed corrections for minimal layer times and
determine the fanSpeed
            double travelTime;
            double extrudeTime;
            gcodeLayer.getTimes(travelTime, extrudeTime);

gcodeLayer.forceMinimalLayerTime(getSettingInSeconds("cool_min_layer_time"),
getSettingInMillimetersPerSecond("cool_min_speed"), travelTime, extrudeTime);

            // interpolate fan speed (for cool_fan_full_layer and for
cool_min_layer_time_fan_speed_max)
            int fanSpeed = getSettingInPercentage("cool_fan_speed_min");
            double totalLayerTime = travelTime + extrudeTime;
            if (totalLayerTime < getSettingInSeconds("cool_min_layer_time"))
            {
                fanSpeed = getSettingInPercentage("cool_fan_speed_max");
            }
            else if (totalLayerTime <
getSettingInSeconds("cool_min_layer_time_fan_speed_max"))
            {
                // when forceMinimalLayerTime didn't change the extrusionSpeedFactor, we
adjust the fan speed
                double minTime = (getSettingInSeconds("cool_min_layer_time"));
                double maxTime =
(getSettingInSeconds("cool_min_layer_time_fan_speed_max"));
                int fanSpeedMin = getSettingInPercentage("cool_fan_speed_min");
                int fanSpeedMax = getSettingInPercentage("cool_fan_speed_max");
                fanSpeed = fanSpeedMax - (fanSpeedMax-fanSpeedMin) * (totalLayerTime
- minTime) / (maxTime - minTime);
            }
            if (static_cast<int>(layer_nr) < getSettingAsCount("cool_fan_full_layer"))
            {
               //Slow down the fan on the layers below the [cool_fan_full_layer], where
layer 0 is speed 0.
               fanSpeed = fanSpeed * layer_nr / getSettingAsCount("cool_fan_full_layer");
            }
            gcode.writeFanCommand(fanSpeed);
         }
         //@ start write GCode for each layer
         gcodeLayer.writeGCode(getSettingBoolean("cool_lift_head"), layer_nr > 0 ||
getSettingAsPlatformAdhesion("adhesion_type") == Adhesion_Raft?
getSettingInMicrons("layer_height") : getSettingInMicrons("layer_height_0"));
         if (commandSocket)
            commandSocket->sendGCodeLayer();
         //@ add pause to each layer
         if (layerPause){
            //@ turn off the welder
            //gcode.writeCode(getSettingString("machine_welder_off_gcode").c_str());
            gcode.writeCode(welderOffGCode.c_str());
            //@ move printer head up in mm unit
            std::string tempUpLayerEnd;
            std::ostringstream tempUp;
            double upZ = INT2MM(gcode.getPositionZ()) + upLayerEnd;
            //tempUp.precision(3);
            tempUp << std::fixed << std::setprecision(3) << ";Move print head up\nG0 Z"
<< upZ << "\n";
            tempUpLayerEnd = tempUp.str();
            gcode.writeCode(tempUpLayerEnd.c_str());
            //@ pause the pringting
            std::string tempGcode;
            double tempPauseTime;
            std::ostringstream temp;
            tempPauseTime = pauseTime + (pauseTime*(pauseIncrease/100)*layer_nr);
            temp << (int)tempPauseTime << "\n";
            tempGcode = pauseGcode + temp.str();

            gcode.writeCode(tempGcode.c_str());
            //@ set that the welder is off
            gcode.setIsWelding(false);
         }
      }//@ end for each layer
      gcode.writeRetraction(&storage.retraction_config, true);

      log("Wrote layers in %5.2fs.\n", timeKeeper.restart());
      gcode.writeFanCommand(0);
```

```
        //Store the object height for when we are printing multiple objects, as we need to
clear every one of them when moving to the next position.
        maxObjectHeight = std::max(maxObjectHeight, storage.model_max.z);

        if (commandSocket)
        {
            finalize();
            commandSocket->sendGCodeLayer();
            commandSocket->endSendSlicedObject();
            if (gcode.getFlavor() == GCODE_FLAVOR_ULTIGCODE)
            {
                std::ostringstream prefix;
                prefix << ";FLAVOR:UltiGCode\n";
                prefix << ";TIME:" << int(gcode.getTotalPrintTime()) << "\n";
                prefix << ";MATERIAL:" << int(gcode.getTotalFilamentUsed(0)) << "\n";
                prefix << ";MATERIAL2:" << int(gcode.getTotalFilamentUsed(1)) << "\n";
                commandSocket->sendGCodePrefix(prefix.str());
            }
        }
    }

    std::vector<SliceMeshStorage*> calculateMeshOrder(SliceDataStorage& storage, int
current_extruder)
    {
        std::vector<SliceMeshStorage*> ret;
        std::vector<SliceMeshStorage*> add_list;
        for(SliceMeshStorage& mesh : storage.meshes)
            add_list.push_back(&mesh);

        int add_extruder_nr = current_extruder;
        while(add_list.size() > 0)
        {
            for(unsigned int idx=0; idx<add_list.size(); idx++)
            {
                if (add_list[idx]->settings->getSettingAsIndex("extruder_nr") ==
add_extruder_nr)
                {
                    ret.push_back(add_list[idx]);
                    add_list.erase(add_list.begin() + idx);
                    idx--;
                }
            }
            if (add_list.size() > 0)
                add_extruder_nr = add_list[0]->settings->getSettingAsIndex("extruder_nr");
        }
        return ret;
    }

    //Add a single layer from a single mesh-volume to the GCode
    void addMeshLayerToGCode(SliceDataStorage& storage, SliceMeshStorage* mesh,
GCodePlanner& gcodeLayer, int layer_nr)
    {
        //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter
addMeshLayerToGCode function"); //@ for test.
        int prevExtruder = gcodeLayer.getExtruder();
        bool extruder_changed = gcodeLayer.setExtruder(mesh->settings-
>getSettingAsIndex("extruder_nr"));

        if (extruder_changed)
            addWipeTower(storage, gcodeLayer, layer_nr, prevExtruder);

        SliceLayer* layer = &mesh->layers[layer_nr];

        if (getSettingBoolean("magic_polygon_mode"))
        {
            //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter
magic_polygon_mode"); //@ for test.
            Polygons polygons;
            for(unsigned int partNr=0; partNr<layer->parts.size(); partNr++)
            {
                for(unsigned int n=0; n<layer->parts[partNr].outline.size(); n++)
                {
                    for(unsigned int m=1; m<layer->parts[partNr].outline[n].size(); m++)
                    {
                        Polygon p;
                        p.add(layer->parts[partNr].outline[n][m-1]);
                        p.add(layer->parts[partNr].outline[n][m]);
                        polygons.add(p);
                    }
                    if (layer->parts[partNr].outline[n].size() > 0)
                    {
                        Polygon p;
                        p.add(layer->parts[partNr].outline[n][layer-
>parts[partNr].outline[n].size()-1]);
                        p.add(layer->parts[partNr].outline[n][0]);
                        polygons.add(p);
                    }
                }
            }
            for(unsigned int n=0; n<layer->openLines.size(); n++)
            {
                for(unsigned int m=1; m<layer->openLines[n].size(); m++)
                {
                    Polygon p;
                    p.add(layer->openLines[n][m-1]);
                    p.add(layer->openLines[n][m]);
                    polygons.add(p);
                }
            }
            if (mesh->settings->getSettingBoolean("magic_spiralize"))
                mesh->inset0_config.spiralize = true;

            gcodeLayer.addPolygonsByOptimizer(polygons, &mesh->inset0_config);
            return;
        }

        PathOrderOptimizer partOrderOptimizer(gcode.getStartPositionXY());
        for(unsigned int partNr=0; partNr<layer->parts.size(); partNr++)
        {
            partOrderOptimizer.addPolygon(layer->parts[partNr].insets[0][0]);
        }
        partOrderOptimizer.optimize();

        for(unsigned int partCounter=0; partCounter<partOrderOptimizer.polyOrder.size();
partCounter++)
        {
            SliceLayerPart* part = &layer-
>parts[partOrderOptimizer.polyOrder[partCounter]];

            if (getSettingBoolean("retraction_combing"))
                gcodeLayer.setCombBoundary(&part->combBoundery);
            else
                gcodeLayer.setAlwaysRetract(true);

            int fillAngle = 45;
            if (layer_nr & 1)
                fillAngle += 90;
            int extrusionWidth = getSettingInMicrons("infill_line_width");

            //Add thicker (multiple layers) sparse infill.
            int sparse_infill_line_distance = getSettingInMicrons("infill_line_distance");
            double infill_overlap = getSettingInPercentage("fill_overlap");
            if (sparse_infill_line_distance > 0)
            {
                //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter
spare_infill_line_distance greater than zero"); //@ for test.
                //Print the thicker sparse lines first. (double or more layer thickness, infill
combined with previous layers)
```

```
            for(unsigned int n=1; n<part->sparse_outline.size(); n++)
            {
               Polygons fillPolygons;
               switch(getSettingAsFillMethod("fill_pattern"))
               {
               case Fill_Grid:
                  generateGridInfill(part->sparse_outline[n], 0, fillPolygons,
extrusionWidth, sparse_infill_line_distance * 2, infill_overlap, fillAngle);
                  gcodeLayer.addLinesByOptimizer(fillPolygons, &mesh-
>infill_config[n]);
                  break;
               case Fill_Lines:
                  generateLineInfill(part->sparse_outline[n], 0, fillPolygons,
extrusionWidth, sparse_infill_line_distance, infill_overlap, fillAngle);
                  gcodeLayer.addLinesByOptimizer(fillPolygons, &mesh-
>infill_config[n]);
                  break;
               case Fill_Triangles:
                  generateTriangleInfill(part->sparse_outline[n], 0, fillPolygons,
extrusionWidth, sparse_infill_line_distance * 3, infill_overlap, 0);
                  gcodeLayer.addLinesByOptimizer(fillPolygons, &mesh-
>infill_config[n]);
                  break;
               case Fill_Concentric:
                  generateConcentricInfill(part->sparse_outline[n], fillPolygons,
sparse_infill_line_distance);
                  gcodeLayer.addPolygonsByOptimizer(fillPolygons, &mesh-
>infill_config[n]);
                  break;
               case Fill_ZigZag:
                  generateZigZagInfill(part->sparse_outline[n], fillPolygons,
extrusionWidth, sparse_infill_line_distance, infill_overlap, fillAngle, false, false);
                  gcodeLayer.addPolygonsByOptimizer(fillPolygons, &mesh-
>infill_config[n]);
                  break;
               default:
                  logError("fill_pattern has unknown value.\n");
                  break;
               }
               sendPolygons(InfillType, layer_nr, fillPolygons, extrusionWidth);
            }
         }

         //Combine the 1 layer thick infill with the top/bottom skin and print that as one
thing.
         Polygons infillPolygons;
```

```
        Polygons infillLines;
        if (sparse_infill_line_distance > 0 && part->sparse_outline.size() > 0)
        {
            //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter infillPolygons");
//@ for test.
            switch(getSettingAsFillMethod("fill_pattern"))
            {
            case Fill_Grid:
                generateGridInfill(part->sparse_outline[0], 0, infillLines, extrusionWidth,
sparse_infill_line_distance * 2, infill_overlap, fillAngle);
                break;
            case Fill_Lines:
                generateLineInfill(part->sparse_outline[0], 0, infillLines, extrusionWidth,
sparse_infill_line_distance, infill_overlap, fillAngle);
                break;
            case Fill_Triangles:
                generateTriangleInfill(part->sparse_outline[0], 0, infillLines,
extrusionWidth, sparse_infill_line_distance * 3, infill_overlap, 0);
                break;
            case Fill_Concentric:
                generateConcentricInfill(part->sparse_outline[0], infillPolygons,
sparse_infill_line_distance);
                break;
            case Fill_ZigZag:
                generateZigZagInfill(part->sparse_outline[0], infillLines, extrusionWidth,
sparse_infill_line_distance, infill_overlap, fillAngle, false, false);
                break;
            default:
                logError("fill_pattern has unknown value.\n");
                break;
            }
        }
        gcodeLayer.addPolygonsByOptimizer(infillPolygons, &mesh->infill_config[0]);
        gcodeLayer.addLinesByOptimizer(infillLines, &mesh->infill_config[0]);

        sendPolygons(InfillType, layer_nr, infillLines, extrusionWidth);

        if (getSettingAsCount("wall_line_count") > 0)
        {
            if (getSettingBoolean("magic_spiralize"))
            {
                //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter
magic_spiralize"); //@ for test.
                if (static_cast<int>(layer_nr) >= getSettingAsCount("bottom_layers"))
                    mesh->inset0_config.spiralize = true;
                if (static_cast<int>(layer_nr) == getSettingAsCount("bottom_layers") &&
part->insets.size() > 0)
                    gcodeLayer.addPolygonsByOptimizer(part->insets[0], &mesh-
>insetX_config);
            }
            for(int insetNr=part->insets.size()-1; insetNr>-1; insetNr--)
            {
                if (insetNr == 0)
                    gcodeLayer.addPolygonsByOptimizer(part->insets[insetNr], &mesh-
>inset0_config);
                else
                    gcodeLayer.addPolygonsByOptimizer(part->insets[insetNr], &mesh-
>insetX_config);
            }
        }

        Polygons skinPolygons;
        Polygons skinLines;
        for(SkinPart& skin_part : part->skin_parts)
        {
            int bridge = -1;
            if (layer_nr > 0)
                bridge = bridgeAngle(skin_part.outline, &mesh->layers[layer_nr-1]);
            if (bridge > -1)
            {
                generateLineInfill(skin_part.outline, 0, skinLines, extrusionWidth,
extrusionWidth, infill_overlap, bridge);
            }else{
                switch(getSettingAsFillMethod("top_bottom_pattern"))
                {
                case Fill_Lines:
                    for (Polygons& skin_perimeter : skin_part.insets)
                    {
                        gcodeLayer.addPolygonsByOptimizer(skin_perimeter, &mesh-
>skin_config); // add polygons to gcode in inward order
                    }
                    if (skin_part.insets.size() > 0)
                    {
                        generateLineInfill(skin_part.insets.back(), -extrusionWidth/2,
skinLines, extrusionWidth, extrusionWidth, infill_overlap, fillAngle);
                        if (getSettingString("fill_perimeter_gaps") != "Nowhere")
                        {
                            generateLineInfill(skin_part.perimeterGaps, 0, skinLines,
extrusionWidth, extrusionWidth, 0, fillAngle);
                        }
                    }
```

```
                        else
                        {
                            generateLineInfill(skin_part.outline, 0, skinLines, extrusionWidth,
extrusionWidth, infill_overlap, fillAngle);
                        }
                        break;
                    case Fill_Concentric:
                        {
                            //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter
skinPolygons Fill_Concentric"); //@ for test.
                            Polygons in_outline;
                            offsetSafe(skin_part.outline, -extrusionWidth/2, extrusionWidth,
in_outline, getSettingBoolean("wall_overlap_avoid_enabled"));
                            if (getSettingString("fill_perimeter_gaps") != "Nowhere")
                            {
                                generateConcentricInfillDense(in_outline, skinPolygons, &part-
>perimeterGaps, extrusionWidth, getSettingBoolean("wall_overlap_avoid_enabled"));
                            }
                        }
                        break;
                    default:
                        logError("Unknown fill method for skin\n");
                        break;
                    }
                }
            }

            // handle gaps between perimeters etc.
            if (getSettingString("fill_perimeter_gaps") != "Nowhere")
            {
                generateLineInfill(part->perimeterGaps, 0, skinLines, extrusionWidth,
extrusionWidth, 0, fillAngle);
            }


            gcodeLayer.addPolygonsByOptimizer(skinPolygons, &mesh->skin_config);
            gcodeLayer.addLinesByOptimizer(skinLines, &mesh->skin_config);

            sendPolygons(SkinType, layer_nr, skinLines, extrusionWidth);

            //After a layer part, make sure the nozzle is inside the comb boundary, so we do
not retract on the perimeter.
            if (!getSettingBoolean("magic_spiralize") || static_cast<int>(layer_nr) <
getSettingAsCount("bottom_layers"))
                gcodeLayer.moveInsideCombBoundary(extrusionWidth * 2);
        }
        gcodeLayer.setCombBoundary(nullptr);
    }

    void addSupportToGCode(SliceDataStorage& storage, GCodePlanner& gcodeLayer,
int layer_nr)
    {
        if (!storage.support.generated)
            return;

        if (getSettingAsIndex("support_extruder_nr") > -1)
        {
            int prevExtruder = gcodeLayer.getExtruder();
            if (gcodeLayer.setExtruder(getSettingAsIndex("support_extruder_nr")))
                addWipeTower(storage, gcodeLayer, layer_nr, prevExtruder);
        }
        Polygons support;
        if (storage.support.generated)
            support = storage.support.supportAreasPerLayer[layer_nr];

        sendPolygons(SupportType, layer_nr, support,
getSettingInMicrons("wall_line_width_x"));

        std::vector<Polygons> supportIslands = support.splitIntoParts();

        PathOrderOptimizer islandOrderOptimizer(gcode.getPositionXY());
        for(unsigned int n=0; n<supportIslands.size(); n++)
        {
            islandOrderOptimizer.addPolygon(supportIslands[n][0]);
        }
        islandOrderOptimizer.optimize();

        for(unsigned int n=0; n<supportIslands.size(); n++)
        {
            Polygons& island = supportIslands[islandOrderOptimizer.polyOrder[n]];

            Polygons supportLines;
            int support_line_distance = getSettingInMicrons("support_line_distance");
            double infill_overlap = getSettingInPercentage("fill_overlap");
            if (support_line_distance > 0)
            {
                int extrusionWidth = getSettingInMicrons("wall_line_width_x");
                switch(getSettingAsFillMethod("support_pattern"))
                {
                case Fill_Grid:
                    {
                        int offset_from_outline = 0;
                        if (support_line_distance > extrusionWidth * 4)
                        {
                            generateGridInfill(island, offset_from_outline, supportLines,
extrusionWidth, support_line_distance*2, infill_overlap, 0);
                        }else{
                            generateLineInfill(island, offset_from_outline, supportLines,
extrusionWidth, support_line_distance, infill_overlap, (layer_nr & 1) ? 0 : 90);
                        }
                    }
                    break;
                case Fill_Lines:
                    {
                        int offset_from_outline = 0;
                        if (layer_nr == 0)
                        {
                            generateGridInfill(island, offset_from_outline, supportLines,
extrusionWidth, support_line_distance, infill_overlap + 150, 0);
                        }else{
                            generateLineInfill(island, offset_from_outline, supportLines,
extrusionWidth, support_line_distance, infill_overlap, 0);
                        }
                    }
                    break;
                case Fill_ZigZag:
                    {
                        int offset_from_outline = 0;
                        if (layer_nr == 0)
                        {
                            generateGridInfill(island, offset_from_outline, supportLines,
extrusionWidth, support_line_distance, infill_overlap + 150, 0);
                        }else{
                            generateZigZagInfill(island, supportLines, extrusionWidth,
support_line_distance, infill_overlap, 0, getSettingBoolean("support_connect_zigzags"),
true);
                        }
                    }
                    break;
                default:
                    logError("Unknown fill method for support\n");
                    break;
                }
            }

            gcodeLayer.forceRetract();
            if (getSettingBoolean("retraction_combing"))
                gcodeLayer.setCombBoundary(&island);
```

```
         if (getSettingAsFillMethod("support_pattern") == Fill_Grid || (
getSettingAsFillMethod("support_pattern") == Fill_ZigZag && layer_nr == 0 ) )
            gcodeLayer.addPolygonsByOptimizer(island, &storage.support_config);
         gcodeLayer.addLinesByOptimizer(supportLines, &storage.support_config);
         gcodeLayer.setCombBoundary(nullptr);

         sendPolygons(SupportInfillType, layer_nr, supportLines,
getSettingInMicrons("wall_line_width_x"));
      }
   }

   void addWipeTower(SliceDataStorage& storage, GCodePlanner& gcodeLayer, int
layer_nr, int prevExtruder)
   {
      if (getSettingInMicrons("wipe_tower_size") < 1)
         return;

      int64_t offset = -getSettingInMicrons("wall_line_width_x");
      if (layer_nr > 0)
         offset *= 2;

      //If we changed extruder, print the wipe/prime tower for this nozzle;
      std::vector<Polygons> insets;
      if ((layer_nr % 2) == 1)
         insets.push_back(storage.wipeTower.offset(offset / 2));
      else
         insets.push_back(storage.wipeTower);
      while(true)
      {
         Polygons new_inset = insets[insets.size() - 1].offset(offset);
         if (new_inset.size() < 1)
            break;
         insets.push_back(new_inset);
      }
      for(unsigned int n=0; n<insets.size(); n++)
      {
         gcodeLayer.addPolygonsByOptimizer(insets[insets.size() - 1 - n],
&storage.meshes[0].insetX_config);
      }

      //Make sure we wipe the old extruder on the wipe tower.
      gcodeLayer.addTravel(storage.wipePoint - gcode.getExtruderOffset(prevExtruder) +
gcode.getExtruderOffset(gcodeLayer.getExtruder()));
   }
};
```

```
}//namespace cura

#endif//FFF_PROCESSOR_H
```

```
//gcodeExport.cpp
/**Yuenyong Nilsiam based on */
/** Copyright (C) 2013 David Braam - Released under terms of the AGPLv3 License */
#include <stdarg.h>
#include <iomanip>

#include "gcodeExport.h"
#include "utils/logoutput.h"

namespace cura {

GCodeExport::GCodeExport()
: output_stream(&std::cout), currentPosition(0,0,0),
startPosition(INT32_MIN,INT32_MIN,0)
{
   extrusion_amount = 0;
   retraction_extrusion_window = 0.0;
   extruderSwitchRetraction = 14.5;
   current_extruder = 0;
   currentFanSpeed = -1;

   totalPrintTime = 0.0;
   for(unsigned int e=0; e<MAX_EXTRUDERS; e++)
   {
      totalFilament[e] = 0.0;
      currentTemperature[e] = 0;
      filament_diameter[e] = 0;
   }

   currentSpeed = 1;
   retractionPrimeSpeed = 1;
   isRetracted = false;
   isZHopped = false;
   setFlavor(GCODE_FLAVOR_REPRAP);
   memset(extruderOffset, 0, sizeof(extruderOffset));
}

GCodeExport::~GCodeExport()
{
}

void GCodeExport::setOutputStream(std::ostream* stream)
{
   output_stream = stream;
   *output_stream << std::fixed;
}
```

```
void GCodeExport::setExtruderOffset(int id, Point p)
{
   extruderOffset[id] = p;
}

Point GCodeExport::getExtruderOffset(int id)
{
   return extruderOffset[id];
}

void GCodeExport::setSwitchExtruderCode(int id, std::string preSwitchExtruderCode,
std::string postSwitchExtruderCode)
{
   this->preSwitchExtruderCode[id] = preSwitchExtruderCode;
   this->postSwitchExtruderCode[id] = postSwitchExtruderCode;
}

void GCodeExport::setFlavor(EGCodeFlavor flavor)
{
   this->flavor = flavor;
   if (flavor == GCODE_FLAVOR_MACH3)
      for(int n=0; n<MAX_EXTRUDERS; n++)
         extruderCharacter[n] = 'A' + n;
   else
      for(int n=0; n<MAX_EXTRUDERS; n++)
         extruderCharacter[n] = 'E';
   if (flavor == GCODE_FLAVOR_ULTIGCODE || flavor ==
GCODE_FLAVOR_REPRAP_VOLUMATRIC)
   {
      is_volumatric = true;
   }
   else
   {
      is_volumatric = false;
   }
}

EGCodeFlavor GCodeExport::getFlavor()
{
   return this->flavor;
}

void GCodeExport::setRetractionSettings(int extruderSwitchRetraction, int
extruderSwitchRetractionSpeed, int extruderSwitchPrimeSpeed, int
retraction_extrusion_window, int retraction_count_max)
```

```
{
   this->extruderSwitchRetraction = INT2MM(extruderSwitchRetraction);
   this->extruderSwitchRetractionSpeed = extruderSwitchRetractionSpeed;
   this->extruderSwitchPrimeSpeed = extruderSwitchPrimeSpeed;
   this->retraction_extrusion_window = INT2MM(retraction_extrusion_window);
   this->retraction_count_max = retraction_count_max;
}

void GCodeExport::setZ(int z)
{
   this->zPos = z;
}

Point3 GCodeExport::getPosition()
{
   return currentPosition;
}
Point GCodeExport::getPositionXY()
{
   return Point(currentPosition.x, currentPosition.y);
}

int GCodeExport::getPositionZ()
{
   return currentPosition.z;
}

void GCodeExport::resetStartPosition()
{
   startPosition.x = INT32_MIN;
   startPosition.y = INT32_MIN;
}

Point GCodeExport::getStartPositionXY()
{
   return Point(startPosition.x, startPosition.y);
}

int GCodeExport::getExtruderNr()
{
   return current_extruder;
}

double GCodeExport::getFilamentArea(unsigned int extruder)
{
   double r = INT2MM(filament_diameter[extruder]) / 2.0;
```

```
    double filament_area = M_PI * r * r;
    return filament_area;
}

void GCodeExport::setFilamentDiameter(unsigned int n, int diameter)
{
    filament_diameter[n] = diameter;
}

double GCodeExport::getExtrusionAmountMM3(unsigned int extruder)
{
    if (is_volumatric)
    {
        return extrusion_amount;
    }
    else
    {
        return extrusion_amount * getFilamentArea(extruder);
    }
}

double GCodeExport::getTotalFilamentUsed(int e)
{
    if (e == current_extruder)
        return totalFilament[e] + getExtrusionAmountMM3(e);
    return totalFilament[e];
}

double GCodeExport::getTotalPrintTime()
{
    return totalPrintTime;
}

void GCodeExport::resetTotalPrintTimeAndFilament()
{
    totalPrintTime = 0;
    for(unsigned int e=0; e<MAX_EXTRUDERS; e++)
    {
        totalFilament[e] = 0.0;
        currentTemperature[e] = 0;
    }
    extrusion_amount = 0.0;
    estimateCalculator.reset();
}

void GCodeExport::updateTotalPrintTime()
```

```
{
   totalPrintTime += estimateCalculator.calculate();
   estimateCalculator.reset();
}

void GCodeExport::writeComment(std::string comment)
{
   *output_stream << ";" << comment << "\n";
}

void GCodeExport::writeTypeComment(const char* type)
{
   *output_stream << ";TYPE:" << type << "\n";
}
void GCodeExport::writeLayerComment(int layer_nr)
{
   *output_stream << ";LAYER:" << layer_nr << "\n";
}

void GCodeExport::writeLine(const char* line)
{
   *output_stream << line << "\n";
}

void GCodeExport::resetExtrusionValue()
{
   if (extrusion_amount != 0.0 && flavor != GCODE_FLAVOR_MAKERBOT &&
flavor != GCODE_FLAVOR_BFB)
   {
      *output_stream << "G92 " << extruderCharacter[current_extruder] << "0\n";
      totalFilament[current_extruder] += getExtrusionAmountMM3(current_extruder);
      for (unsigned int i = 0; i < extrusion_amount_at_previous_n_retractions.size(); i++)
         extrusion_amount_at_previous_n_retractions[i] -= extrusion_amount;
      extrusion_amount = 0.0;
   }
}

void GCodeExport::writeDelay(double timeAmount)
{
   *output_stream << "G4 P" << int(timeAmount * 1000) << "\n";
   totalPrintTime += timeAmount;
}

void GCodeExport::writeMove(Point p, int speed, double extrusion_mm3_per_mm)
{
   writeMove(p.X, p.Y, zPos, speed, extrusion_mm3_per_mm);
```

```
}

void GCodeExport::writeMove(Point3 p, int speed, double extrusion_mm3_per_mm)
{
    writeMove(p.x, p.y, p.z, speed, extrusion_mm3_per_mm);
}
void GCodeExport::writeMove(int x, int y, int z, int speed, double
extrusion_mm3_per_mm)
{
    if (currentPosition.x == x && currentPosition.y == y && currentPosition.z == z)
        return;

    double extrusion_per_mm = extrusion_mm3_per_mm;
    if (!is_volumatric)
    {
        extrusion_per_mm = extrusion_mm3_per_mm / getFilamentArea(current_extruder);
    }

    if (flavor == GCODE_FLAVOR_BFB)
    {
        //For Bits From Bytes machines, we need to handle this completely differently. As
they do not use E values but RPM values.
        float fspeed = speed * 60;
        float rpm = extrusion_per_mm * speed * 60;
        const float mm_per_rpm = 4.0; //All BFB machines have 4mm per RPM extrusion.
        rpm /= mm_per_rpm;
        if (rpm > 0)
        {
            if (isRetracted)
            {
                if (currentSpeed != int(rpm * 10))
                {
                    //fprintf(f, "; %f e-per-mm %d mm-width %d mm/s\n", extrusion_per_mm,
lineWidth, speed);
                    //fprintf(f, "M108 S%0.1f\r\n", rpm); //M108 set extruder speed
                    *output_stream << "M108 S" << std::setprecision(1) << rpm << "\r\n";
                    currentSpeed = int(rpm * 10);
                }
                //Add M101 or M201 to enable the proper extruder.
                *output_stream << "M" << int((current_extruder + 1) * 100 + 1) << "\r\n";
                isRetracted = false;
            }
            //Fix the speed by the actual RPM we are asking, because of rounding errors we
cannot get all RPM values, but we have a lot more resolution in the feedrate value.
            // (Trick copied from KISSlicer, thanks Jonathan)
            fspeed *= (rpm / (roundf(rpm * 100) / 100));
        //Increase the extrusion amount to calculate the amount of filament used.
        Point3 diff = Point3(x,y,z) - getPosition();

        extrusion_amount += extrusion_per_mm * diff.vSizeMM();
      }else{
        //If we are not extruding, check if we still need to disable the extruder. This
causes a retraction due to auto-retraction.
        if (!isRetracted)
        {
          *output_stream << "M103\r\n";
          isRetracted = true;
        }
      }
      *output_stream << std::setprecision(3) << "G1 X" << INT2MM(x -
extruderOffset[current_extruder].X) << " Y" << INT2MM(y -
extruderOffset[current_extruder].Y) << " Z" << INT2MM(z) << std::setprecision(1) << "
F" << fspeed << "\r\n";
    }else{
      //Normal E handling. //@ RepRap
      //@ move this diff outside so able to use it in else for G0
      Point3 diff = Point3(x,y,z) - getPosition();
      //@ t_tmp for temporary calculation time
      //@ double t_tmp;
      if (extrusion_mm3_per_mm > 0.000001)
      {
        //@ Point3 diff = Point3(x,y,z) - getPosition();
        if (isZHopped > 0)
        {
          *output_stream << std::setprecision(3) << "G1 Z" <<
INT2MM(currentPosition.z) << "\n";
          isZHopped = false;
        }
        if (isRetracted)
        {
          if (flavor == GCODE_FLAVOR_ULTIGCODE || flavor ==
GCODE_FLAVOR_REPRAP_VOLUMATRIC)
          {
            *output_stream << "G11\n";
            //Assume default UM2 retraction settings.

estimateCalculator.plan(TimeEstimateCalculator::Position(INT2MM(currentPosition.x),
INT2MM(currentPosition.y), INT2MM(currentPosition.z), extrusion_amount), 25.0);
          }else{
```

```
                    *output_stream << "G1 F" << (retractionPrimeSpeed * 60) << " " <<
extruderCharacter[current_extruder] << std::setprecision(5) << extrusion_amount <<
"\n";
                    currentSpeed = retractionPrimeSpeed;

estimateCalculator.plan(TimeEstimateCalculator::Position(INT2MM(currentPosition.x),
INT2MM(currentPosition.y), INT2MM(currentPosition.z), extrusion_amount),
currentSpeed);
                }
                if (getExtrusionAmountMM3(current_extruder) > 10000.0) //According to
https://github.com/Ultimaker/CuraEngine/issues/14 having more then 21m of extrusion
causes inaccuracies. So reset it every 10m, just to be sure.
                    resetExtrusionValue(); //
                isRetracted = false;
            }
            //@ if isMetalPrinting then cal time and new speed
            if (isMetalPrinting)
            {
                //@t_tmp = (extrusion_per_mm * diff.vSizeMM())/(double)speed;
                //@speed = (int)(diff.vSizeMM()/t_tmp); //@ cal new speed
                if (!isWelding)
                {
                    isWelding = true;
                    *output_stream << welder_on;
                }
            }
            extrusion_amount += extrusion_per_mm * diff.vSizeMM();
            *output_stream << "G1"; //@ to print

        }else{//@ only moving
            //@ if it is metal printing
            if (isMetalPrinting)
            {
                //@ check with min_dist_welder_off
                if (isWelding && diff.vSizeMM() > min_dist_welder_off){
                    isWelding = false;
                    *output_stream << welder_off;
                }
            }

            *output_stream << "G0"; //@ to move
        }

        if (currentSpeed != speed)
        {
            *output_stream << " F" << (speed * 60); //@feedrate per minute
            currentSpeed = speed;
        }

        *output_stream << std::setprecision(3) << " X" << INT2MM(x -
extruderOffset[current_extruder].X) << " Y" << INT2MM(y -
extruderOffset[current_extruder].Y);
        if (z != currentPosition.z)
            *output_stream << " Z" << INT2MM(z);
        if (!isMetalPrinting)
        {
            if (extrusion_mm3_per_mm > 0.000001)
                *output_stream << " " << extruderCharacter[current_extruder] <<
std::setprecision(5) << extrusion_amount;
        }

        *output_stream << "\n";
    }

    currentPosition = Point3(x, y, z);
    startPosition = currentPosition;

estimateCalculator.plan(TimeEstimateCalculator::Position(INT2MM(currentPosition.x),
INT2MM(currentPosition.y), INT2MM(currentPosition.z), extrusion_amount), speed);
}

void GCodeExport::writeRetraction(RetractionConfig* config, bool force)
{
    if (flavor == GCODE_FLAVOR_BFB)//BitsFromBytes does automatic retraction.
        return;
    if (isRetracted)
        return;
    if (config->amount <= 0)
        return;

    if (!force && retraction_count_max > 0 &&
int(extrusion_amount_at_previous_n_retractions.size()) == retraction_count_max - 1
        && extrusion_amount < extrusion_amount_at_previous_n_retractions.back() +
retraction_extrusion_window)
        return;

    if (config->primeAmount > 0)
        extrusion_amount += config->primeAmount;
    retractionPrimeSpeed = config->primeSpeed;

    if (flavor == GCODE_FLAVOR_ULTIGCODE || flavor ==
GCODE_FLAVOR_REPRAP_VOLUMATRIC)
```

```
    {
        *output_stream << "G10\n";
        //Assume default UM2 retraction settings.
        double retraction_distance = 4.5;

estimateCalculator.plan(TimeEstimateCalculator::Position(INT2MM(currentPosition.x),
INT2MM(currentPosition.y), INT2MM(currentPosition.z), extrusion_amount -
retraction_distance), 25); // TODO: hardcoded values!
    }else{
        *output_stream << "G1 F" << (config->speed * 60) << " " <<
extruderCharacter[current_extruder] << std::setprecision(5) << extrusion_amount -
config->amount << "\n";
        currentSpeed = config->speed;

estimateCalculator.plan(TimeEstimateCalculator::Position(INT2MM(currentPosition.x),
INT2MM(currentPosition.y), INT2MM(currentPosition.z), extrusion_amount - config-
>amount), currentSpeed);
    }
    if (config->zHop > 0)
    {
        *output_stream << std::setprecision(3) << "G1 Z" << INT2MM(currentPosition.z +
config->zHop) << "\n";
        isZHopped = true;
    }
    extrusion_amount_at_previous_n_retractions.push_front(extrusion_amount);
    if (int(extrusion_amount_at_previous_n_retractions.size()) == retraction_count_max)
    {
        extrusion_amount_at_previous_n_retractions.pop_back();
    }
    isRetracted = true;
}

void GCodeExport::switchExtruder(int newExtruder)
{
    if (current_extruder == newExtruder)
        return;

    if (flavor == GCODE_FLAVOR_BFB)
    {
        if (!isRetracted)
            *output_stream << "M103\r\n";

        isRetracted = true;
        return;
    }
    resetExtrusionValue();
    if (flavor == GCODE_FLAVOR_ULTIGCODE || flavor ==
GCODE_FLAVOR_REPRAP_VOLUMATRIC)
    {
        *output_stream << "G10 S1\n";
    }else{
        *output_stream << "G1 F" << (extruderSwitchRetractionSpeed * 60) << " " <<
extruderCharacter[current_extruder] << std::setprecision(5) << (extrusion_amount -
extruderSwitchRetraction) << "\n";
        currentSpeed = extruderSwitchRetractionSpeed;
    }

    current_extruder = newExtruder;
    if (flavor == GCODE_FLAVOR_MACH3)
        resetExtrusionValue();
    isRetracted = true;
    writeCode(preSwitchExtruderCode[current_extruder].c_str());
    if (flavor == GCODE_FLAVOR_MAKERBOT)
        *output_stream << "M135 T" << current_extruder << "\n";
    else
        *output_stream << "T" << current_extruder << "\n";
    writeCode(postSwitchExtruderCode[current_extruder].c_str());

    //Change the Z position so it gets re-writting again. We do not know if the switch code
modified the Z position.
    currentPosition.z += 1;
}

void GCodeExport::writeCode(const char* str)
{
    *output_stream << str;
    if (flavor == GCODE_FLAVOR_BFB)
        *output_stream << "\r\n";
    else
        *output_stream << "\n";
}

void GCodeExport::writeFanCommand(int speed)
{
    if (currentFanSpeed == speed)
        return;
    if (speed > 0)
    {
        if (flavor == GCODE_FLAVOR_MAKERBOT)
            *output_stream << "M126 T0\n"; //value = speed * 255 / 100 // Makerbot cannot
set fan speed...;
```

```
        else
            *output_stream << "M106 S" << (speed * 255 / 100) << "\n";
    }
    else
    {
        if (flavor == GCODE_FLAVOR_MAKERBOT)
            *output_stream << "M127 T0\n";
        else
            *output_stream << "M107\n";
    }
    currentFanSpeed = speed;
}

void GCodeExport::writeTemperatureCommand(int extruder, int temperature, bool wait)
{
    if (!isMetalPrinting){
        if (!wait && currentTemperature[extruder] == temperature)
            return;

        if (wait){
            *output_stream << "M109";//@ remove set extruder temperature and wait
        }
        else{
            *output_stream << "M104";//@ remove set extruder temperature
        }
        if (extruder != current_extruder)
            *output_stream << " T" << extruder;//@
        *output_stream << " S" << temperature << "\n";//@
    }

    currentTemperature[extruder] = temperature;
}

void GCodeExport::writeBedTemperatureCommand(int temperature, bool wait)
{
    if (wait)
        *output_stream << "M190 S";
    else
        *output_stream << "M140 S";
    *output_stream << temperature << "\n";
}

void GCodeExport::finalize(int maxObjectHeight, int moveSpeed, const char* endCode)
{
    std::cerr << "maxObjectHeight : " << maxObjectHeight << std::endl;
    writeFanCommand(0);
    setZ(maxObjectHeight + 5000);
    writeMove(Point3(0,0,maxObjectHeight + 5000) + getPositionXY(), moveSpeed, 0);
    writeCode(endCode);
    log("Print time: %d\n", int(getTotalPrintTime()));
    log("Filament: %d\n", int(getTotalFilamentUsed(0)));
    for(int n=1; n<MAX_EXTRUDERS; n++)
        if (getTotalFilamentUsed(n) > 0)
            log("Filament%d: %d\n", n + 1, int(getTotalFilamentUsed(n)));
    output_stream->flush();
}
//@ set welder_on GCode
void GCodeExport::setWelderOn(std::string welder_on_gcode)
{
    welder_on = welder_on_gcode;
}

//@ set welder_off GCode
void GCodeExport::setWelderOff(std::string welder_off_gcode)
{
    welder_off = welder_off_gcode;
}

//@ set Minimum Distance to move with welder off
void GCodeExport::setMinDistWelderOff(double machine_min_dist_welder_off)
{
    min_dist_welder_off = machine_min_dist_welder_off;
}

//@ set metal printing boolean
void GCodeExport::setIsMetalPrinting(bool machine_metal_printing){
    isMetalPrinting = machine_metal_printing;
}

//@ set is welding
void GCodeExport::setIsWelding(bool is_welding){
    isWelding = is_welding;
}

}//namespace cura
```

```
//gcodeExport.h
/**Yuenyong Nilsiam based on */
/** Copyright (C) 2013 David Braam - Released under terms of the AGPLv3 License */
#ifndef GCODEEXPORT_H
#define GCODEEXPORT_H

#include <stdio.h>
#include <deque> // for extrusionAmountAtPreviousRetractions

#include "settings.h"
#include "utils/intpoint.h"
#include "timeEstimate.h"

namespace cura {

class RetractionConfig
{
public:
   double amount; //!< The amount
   int speed;
   int primeSpeed;
   double primeAmount;
   int zHop;
};

//The GCodePathConfig is the configuration for moves/extrusion actions. This defines at
which width the line is printed and at which speed.
class GCodePathConfig
{
private:
   int speed;
   int line_width;
   int flow;
   int layer_thickness;
   double extrusion_mm3_per_mm;
public:
   const char* name;
   bool spiralize;
   RetractionConfig* retraction_config;

   GCodePathConfig() : speed(0), line_width(0), extrusion_mm3_per_mm(0.0),
name(nullptr), spiralize(false), retraction_config(nullptr) {}
   GCodePathConfig(RetractionConfig* retraction_config, const char* name) : speed(0),
line_width(0), extrusion_mm3_per_mm(0.0), name(name), spiralize(false),
retraction_config(retraction_config) {}
```

```
    void setSpeed(int speed)
    {
        this->speed = speed;
    }

    void setLineWidth(int line_width)
    {
        this->line_width = line_width;
        calculateExtrusion();
    }

    void setLayerHeight(int layer_height)
    {
        this->layer_thickness = layer_height;
        calculateExtrusion();
    }

    void setFlow(int flow)
    {
        this->flow = flow;
        calculateExtrusion();
    }

    void smoothSpeed(int min_speed, int layer_nr, int max_speed_layer)
    {
        speed = (speed*layer_nr)/max_speed_layer + (min_speed*(max_speed_layer-
layer_nr)/max_speed_layer);
    }

    double getExtrusionMM3perMM()
    {
        return extrusion_mm3_per_mm;
    }

    int getSpeed()
    {
        return speed;
    }

    int getLineWidth()
    {
        return line_width;
    }

private:
    void calculateExtrusion()
    {
        extrusion_mm3_per_mm = INT2MM(line_width) * INT2MM(layer_thickness) *
double(flow) / 100.0;
    }
};

//The GCodeExport class writes the actual GCode. This is the only class that knows how
GCode looks and feels.
//  Any customizations on GCodes flavors are done in this class.
class GCodeExport
{
private:
    std::ostream* output_stream;
    double extrusion_amount; // in mm or mm^3
    double extruderSwitchRetraction;
    int extruderSwitchRetractionSpeed;
    int extruderSwitchPrimeSpeed;
    double retraction_extrusion_window;
    int retraction_count_max;
    std::deque<double> extrusion_amount_at_previous_n_retractions; // in mm or mm^3
    Point3 currentPosition;
    Point3 startPosition;
    Point extruderOffset[MAX_EXTRUDERS];
    char extruderCharacter[MAX_EXTRUDERS];
    int currentTemperature[MAX_EXTRUDERS];
    int currentSpeed;
    int zPos;
    bool isRetracted;
    bool isZHopped;
    int retractionPrimeSpeed;
    int current_extruder;
    int currentFanSpeed;
    EGCodeFlavor flavor;
    std::string preSwitchExtruderCode[MAX_EXTRUDERS];
    std::string postSwitchExtruderCode[MAX_EXTRUDERS];

    double totalFilament[MAX_EXTRUDERS]; // in mm^3
    double filament_diameter[MAX_EXTRUDERS]; // in mm^3
    double totalPrintTime;
    TimeEstimateCalculator estimateCalculator;

    bool is_volumatric;
    //std::string welder_on = "G4 P0\nM42 P0 S1\n";//@ GCode to turn welder on
    //std::string welder_off = "G4 P0\nM42 P0 S0\n";//@ Gcode to turn welder off
    std::string welder_on;//@ GCode to turn welder on
    std::string welder_off;//@ GCode to turn welder off
    double min_dist_welder_off; //@ minimum distance to move with welder off, unit mm
    bool isWelding; //@ true = welder is on, false = welder is off
    bool isMetalPrinting; //@ true = metal printing, false = not metal printing
public:

    GCodeExport();
    ~GCodeExport();

    void setOutputStream(std::ostream* stream);

    void setExtruderOffset(int id, Point p);
    Point getExtruderOffset(int id);
    void setSwitchExtruderCode(int id, std::string preSwitchExtruderCode, std::string
postSwitchExtruderCode);

    void setFlavor(EGCodeFlavor flavor);
    EGCodeFlavor getFlavor();

    void setRetractionSettings(int extruderSwitchRetraction, int
extruderSwitchRetractionSpeed, int extruderSwitchPrimeSpeed, int
minimalExtrusionBeforeRetraction, int retraction_count_max);

    void setZ(int z);

    Point3 getPosition();

    Point getPositionXY();

    void resetStartPosition();

    Point getStartPositionXY();

    int getPositionZ();

    int getExtruderNr();

    void setFilamentDiameter(unsigned int n, int diameter);
    double getFilamentArea(unsigned int extruder);

    double getExtrusionAmountMM3(unsigned int extruder);

    double getTotalFilamentUsed(int e);

    double getTotalPrintTime();
    void updateTotalPrintTime();
    void resetTotalPrintTimeAndFilament();
```

```
   void writeComment(std::string comment);
   void writeTypeComment(const char* type);
   void writeLayerComment(int layer_nr);

   void writeLine(const char* line);

   void resetExtrusionValue();

   void writeDelay(double timeAmount);

   void writeMove(Point p, int speed, double extrusion_per_mm);

   void writeMove(Point3 p, int speed, double extrusion_per_mm);
private:
   void writeMove(int x, int y, int z, int speed, double extrusion_per_mm);
public:
   void writeRetraction(RetractionConfig* config, bool force=false);

   void switchExtruder(int newExtruder);

   void writeCode(const char* str);

   void writeFanCommand(int speed);

   void writeTemperatureCommand(int extruder, int temperature, bool wait = false);
   void writeBedTemperatureCommand(int temperature, bool wait = false);

   void finalize(int maxObjectHeight, int moveSpeed, const char* endCode);
   void setWelderOn(std::string welder_on_gcode);
   void setWelderOff(std::string welder_off_gcode);
   void setMinDistWelderOff(double machine_min_dist_welder_off);
   void setIsMetalPrinting(bool machine_metal_printing);
   void setIsWelding(bool is_welding);
};

}

#endif//GCODEEXPORT_H
```

```
//gcodePlanner.cpp
#include "gcodePlanner.h"
#include "pathOrderOptimizer.h"

namespace cura {

GCodePath* GCodePlanner::getLatestPathWithConfig(GCodePathConfig* config)
{
    if (paths.size() > 0 && paths[paths.size()-1].config == config && !paths[paths.size()-
1].done)
        return &paths[paths.size()-1];
    paths.push_back(GCodePath());
    GCodePath* ret = &paths[paths.size()-1];
    ret->retract = false;
    ret->config = config;
    ret->extruder = currentExtruder;
    ret->done = false;
    return ret;
}

void GCodePlanner::forceNewPathStart()
{
    if (paths.size() > 0)
        paths[paths.size()-1].done = true;
}

GCodePlanner::GCodePlanner(GCodeExport& gcode, RetractionConfig*
retraction_config, int travelSpeed, int retractionMinimalDistance)
: gcode(gcode), travelConfig(retraction_config, "MOVE")
{
    lastPosition = gcode.getPositionXY();
    travelConfig.setSpeed(travelSpeed);
    comb = nullptr;
    extrudeSpeedFactor = 100;
    travelSpeedFactor = 100;
    extraTime = 0.0;
    totalPrintTime = 0.0;
    forceRetraction = false;
    alwaysRetract = false;
    currentExtruder = gcode.getExtruderNr();
    this->retractionMinimalDistance = retractionMinimalDistance;
}
GCodePlanner::~GCodePlanner()
{
    if (comb)
        delete comb;
```

```
}

void GCodePlanner::addTravel(Point p)
{
   GCodePath* path = getLatestPathWithConfig(&travelConfig);
   if (forceRetraction)
   {
      if (!shorterThen(lastPosition - p, retractionMinimalDistance))
      {
         path->retract = true;
      }
      forceRetraction = false;
   }else if (comb != nullptr)
   {
      std::vector<Point> pointList;
      if (comb->calc(lastPosition, p, pointList))
      {
         for(unsigned int n=0; n<pointList.size(); n++)
         {
            path->points.push_back(pointList[n]);
         }
      }else{
         if (!shorterThen(lastPosition - p, retractionMinimalDistance))
            path->retract = true;
      }
   }else if (alwaysRetract)
   {
      if (!shorterThen(lastPosition - p, retractionMinimalDistance))
         path->retract = true;
   }
   path->points.push_back(p);
   lastPosition = p;
}

void GCodePlanner::addExtrusionMove(Point p, GCodePathConfig* config)
{
   getLatestPathWithConfig(config)->points.push_back(p);
   lastPosition = p;
}

void GCodePlanner::moveInsideCombBoundary(int distance)
{
   //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter
moveInsideCombBoundary"); //@ for test.
   if (!comb || comb->inside(lastPosition)) return;
   Point p = lastPosition;
```

```
    if (comb->moveInside(&p, distance))
    {
        //Move inside again, so we move out of tight 90deg corners
        comb->moveInside(&p, distance);
        if (comb->inside(p))
        {
            addTravel(p);
            //Make sure the that any retraction happens after this move, not before it by
starting a new move path.
            forceNewPathStart();
        }
    }
}

void GCodePlanner::addPolygon(PolygonRef polygon, int startIdx, GCodePathConfig*
config)
{
    Point p0 = polygon[startIdx];
    addTravel(p0);
    for(unsigned int i=1; i<polygon.size(); i++)
    {
        Point p1 = polygon[(startIdx + i) % polygon.size()];
        addExtrusionMove(p1, config);
        p0 = p1;
    }
    if (polygon.size() > 2)
        addExtrusionMove(polygon[startIdx], config);
}

void GCodePlanner::addPolygonsByOptimizer(Polygons& polygons, GCodePathConfig*
config)
{
    //log("addPolygonsByOptimizer");
    PathOrderOptimizer orderOptimizer(lastPosition);
    for(unsigned int i=0;i<polygons.size();i++)
        orderOptimizer.addPolygon(polygons[i]);
    orderOptimizer.optimize();
    for(unsigned int i=0;i<orderOptimizer.polyOrder.size();i++)
    {
        int nr = orderOptimizer.polyOrder[i];
        addPolygon(polygons[nr], orderOptimizer.polyStart[nr], config);
    }
}
void GCodePlanner::addLinesByOptimizer(Polygons& polygons, GCodePathConfig*
config)
{
    LineOrderOptimizer orderOptimizer(lastPosition);
    for(unsigned int i=0;i<polygons.size();i++)
        orderOptimizer.addPolygon(polygons[i]);
    orderOptimizer.optimize();
    for(unsigned int i=0;i<orderOptimizer.polyOrder.size();i++)
    {
        int nr = orderOptimizer.polyOrder[i];
        addPolygon(polygons[nr], orderOptimizer.polyStart[nr], config);
    }
}

void GCodePlanner::forceMinimalLayerTime(double minTime, int minimalSpeed,
double travelTime, double extrudeTime)
{
    double totalTime = travelTime + extrudeTime;
    if (totalTime < minTime && extrudeTime > 0.0)
    {
        double minExtrudeTime = minTime - travelTime;
        if (minExtrudeTime < 1)
            minExtrudeTime = 1;
        double factor = extrudeTime / minExtrudeTime;
        for(unsigned int n=0; n<paths.size(); n++)
        {
            GCodePath* path = &paths[n];
            if (path->config->getExtrusionMM3perMM() == 0)
                continue;
            int speed = path->config->getSpeed() * factor;
            if (speed < minimalSpeed)
                factor = double(minimalSpeed) / double(path->config->getSpeed());
        }

        //Only slow down with the minimal time if that will be slower then a factor already
set. First layer slowdown also sets the speed factor.
        if (factor * 100 < getExtrudeSpeedFactor())
            setExtrudeSpeedFactor(factor * 100);
        else
            factor = getExtrudeSpeedFactor() / 100.0;

        if (minTime - (extrudeTime / factor) - travelTime > 0.1)
        {
            this->extraTime = minTime - (extrudeTime / factor) - travelTime;
        }
        this->totalPrintTime = (extrudeTime / factor) + travelTime;
    }else{
        this->totalPrintTime = totalTime;
    }
```

```
}

void GCodePlanner::getTimes(double& travelTime, double& extrudeTime)
{
   travelTime = 0.0;
   extrudeTime = 0.0;
   Point p0 = gcode.getPositionXY();
   for(unsigned int n=0; n<paths.size(); n++)
   {
      GCodePath* path = &paths[n];
      for(unsigned int i=0; i<path->points.size(); i++)
      {
         double thisTime = vSizeMM(p0 - path->points[i]) / double(path->config-
>getSpeed());
         if (path->config->getExtrusionMM3perMM() != 0)
            extrudeTime += thisTime;
         else
            travelTime += thisTime;
         p0 = path->points[i];
      }
   }
}

void GCodePlanner::writeGCode(bool liftHeadIfNeeded, int layerThickness)
{
   GCodePathConfig* lastConfig = nullptr;
   int extruder = gcode.getExtruderNr();

   for(unsigned int n=0; n<paths.size(); n++)
   {
      GCodePath* path = &paths[n];
      if (extruder != path->extruder)
      {
         extruder = path->extruder;
         gcode.switchExtruder(extruder);
      }else if (path->retract)
      {
         gcode.writeRetraction(path->config->retraction_config);
      }
      if (path->config != &travelConfig && lastConfig != path->config)
      {
         //printf("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!enter writeTypecomment
in WriteGCode\n"); //@ for test.
         gcode.writeTypeComment(path->config->name);
         lastConfig = path->config;
      }
```

```
        int speed = path->config->getSpeed();

        if (path->config->getExtrusionMM3perMM() != 0)// Only apply the
extrudeSpeedFactor to extrusion moves
            speed = speed * extrudeSpeedFactor / 100;
        else
            speed = speed * travelSpeedFactor / 100;

        if (path->points.size() == 1 && path->config != &travelConfig &&
shorterThen(gcode.getPositionXY() - path->points[0], path->config->getLineWidth() *
2))
        {
            //Check for lots of small moves and combine them into one large line
            Point p0 = path->points[0];
            unsigned int i = n + 1;
            while(i < paths.size() && paths[i].points.size() == 1 && shorterThen(p0 -
paths[i].points[0], path->config->getLineWidth() * 2))
            {
                p0 = paths[i].points[0];
                i ++;
            }
            if (paths[i-1].config == &travelConfig)
                i --;
            if (i > n + 2)
            {
                p0 = gcode.getPositionXY();
                for(unsigned int x=n; x<i-1; x+=2)
                {
                    int64_t new_width = vSize(p0 - paths[x].points[0]); // = old_length
                    Point newPoint = (paths[x].points[0] + paths[x+1].points[0]) / 2;
                    int64_t old_width = path->config->getLineWidth();
                    if (old_width > 0)
                    {
                        if (new_width > 0)
                            gcode.writeMove(newPoint, speed * old_width / new_width, path-
>config->getExtrusionMM3perMM() * new_width / old_width);
                        else
                            gcode.writeMove(newPoint, speed, path->config-
>getExtrusionMM3perMM());
                    }
                    p0 = paths[x+1].points[0];
                }
                gcode.writeMove(paths[i-1].points[0], speed, path->config-
>getExtrusionMM3perMM());
                n = i - 1;
                continue;
            }
        }

        bool spiralize = path->config->spiralize;
        if (spiralize)
        {
            //Check if we are the last spiralize path in the list, if not, do not spiralize.
            for(unsigned int m=n+1; m<paths.size(); m++)
            {
                if (paths[m].config->spiralize)
                    spiralize = false;
            }
        }
        if (spiralize)
        {
            //If we need to spiralize then raise the head slowly by 1 layer as this path
progresses.
            float totalLength = 0.0;
            int z = gcode.getPositionZ();
            Point p0 = gcode.getPositionXY();
            for(unsigned int i=0; i<path->points.size(); i++)
            {
                Point p1 = path->points[i];
                totalLength += vSizeMM(p0 - p1);
                p0 = p1;
            }

            float length = 0.0;
            p0 = gcode.getPositionXY();
            for(unsigned int i=0; i<path->points.size(); i++)
            {
                Point p1 = path->points[i];
                length += vSizeMM(p0 - p1);
                p0 = p1;
                gcode.setZ(z + layerThickness * length / totalLength);
                gcode.writeMove(path->points[i], speed, path->config-
>getExtrusionMM3perMM());
            }
        }else{
            for(unsigned int i=0; i<path->points.size(); i++)
            {
                gcode.writeMove(path->points[i], speed, path->config-
>getExtrusionMM3perMM());
            }
        }
    }
    gcode.updateTotalPrintTime();
    if (liftHeadIfNeeded && extraTime > 0.0)
    {
        gcode.writeComment("Small layer, adding delay");
        if (lastConfig)
            gcode.writeRetraction(lastConfig->retraction_config, true);
        gcode.setZ(gcode.getPositionZ() + MM2INT(3.0));
        gcode.writeMove(gcode.getPositionXY(), travelConfig.getSpeed(), 0);
        gcode.writeMove(gcode.getPositionXY() - Point(-MM2INT(20.0), 0),
travelConfig.getSpeed(), 0);
        gcode.writeDelay(extraTime);
    }
}

}//namespace cura
```

```
//gcodePlanner.h
#ifndef GCODE_PLANNER_H
#define GCODE_PLANNER_H

#include <vector>

#include "gcodeExport.h"
#include "comb.h"
#include "utils/polygon.h"
#include "utils/logoutput.h"

namespace cura
{

class GCodePath
{
public:
    GCodePathConfig* config;
    bool retract;
    int extruder;
    std::vector<Point> points;
    bool done;//Path is finished, no more moves should be added, and a new path should
be started instead of any appending done to this one.
};

//The GCodePlanner class stores multiple moves that are planned.
// It facilitates the combing to keep the head inside the print.
// It also keeps track of the print time estimate for this planning so speed adjustments can
be made for the minimal-layer-time.
class GCodePlanner
{
private:
    GCodeExport& gcode;

    Point lastPosition;
    std::vector<GCodePath> paths;
    Comb* comb;

    GCodePathConfig travelConfig;
    int extrudeSpeedFactor;
    int travelSpeedFactor;
    int currentExtruder;
    int retractionMinimalDistance;
    bool forceRetraction;
    bool alwaysRetract;
    double extraTime;
    double totalPrintTime;

private:
    GCodePath* getLatestPathWithConfig(GCodePathConfig* config);
    void forceNewPathStart();
public:
    GCodePlanner(GCodeExport& gcode, RetractionConfig* retraction_config, int
travelSpeed, int retractionMinimalDistance);
    ~GCodePlanner();

    bool setExtruder(int extruder)
    {
        if (extruder == currentExtruder)
            return false;
        currentExtruder = extruder;
        return true;
    }

    int getExtruder()
    {
        return currentExtruder;
    }

    void setCombBoundary(Polygons* polygons)
    {
        if (comb)
            delete comb;
        if (polygons)
            comb = new Comb(*polygons);
        else
            comb = nullptr;
    }

    void setAlwaysRetract(bool alwaysRetract)
    {
        this->alwaysRetract = alwaysRetract;
    }

    void forceRetract()
    {
        forceRetraction = true;
    }

    void setExtrudeSpeedFactor(int speedFactor)
    {
        if (speedFactor < 1) speedFactor = 1;
        this->extrudeSpeedFactor = speedFactor;
    }
    int getExtrudeSpeedFactor()
    {
        return this->extrudeSpeedFactor;
    }
    void setTravelSpeedFactor(int speedFactor)
    {
        if (speedFactor < 1) speedFactor = 1;
        this->travelSpeedFactor = speedFactor;
    }
    int getTravelSpeedFactor()
    {
        return this->travelSpeedFactor;
    }

    void addTravel(Point p);

    void addExtrusionMove(Point p, GCodePathConfig* config);

    void moveInsideCombBoundary(int distance);

    void addPolygon(PolygonRef polygon, int startIdx, GCodePathConfig* config);

    void addPolygonsByOptimizer(Polygons& polygons, GCodePathConfig* config);

    void addLinesByOptimizer(Polygons& polygons, GCodePathConfig* config);

    void forceMinimalLayerTime(double minTime, int minimalSpeed, double travelTime,
double extrusionTime);

    void getTimes(double& travelTime, double& extrudeTime);

    void writeGCode(bool liftHeadIfNeeded, int layerThickness);
};

}//namespace cura

#endif//GCODE_PLANNER_H
```

```
//mesh.cpp
#include "mesh.h"
#include "utils/logoutput.h"


const int vertex_meld_distance = MM2INT(0.03);
static inline uint32_t pointHash(Point3& p)
{
   return ((p.x + vertex_meld_distance/2) / vertex_meld_distance) ^ (((p.y +
vertex_meld_distance/2) / vertex_meld_distance) << 10) ^ (((p.z +
vertex_meld_distance/2) / vertex_meld_distance) << 20);
}

Mesh::Mesh(SettingsBase* parent)
: SettingsBase(parent)
{
}

void Mesh::addFace(Point3& v0, Point3& v1, Point3& v2)
{
   int vi0 = findIndexOfVertex(v0);
   int vi1 = findIndexOfVertex(v1);
   int vi2 = findIndexOfVertex(v2);
   if (vi0 == vi1 || vi1 == vi2 || vi0 == vi2) return; // the face has two vertices which get
assigned the same location. Don't add the face.

   int idx = faces.size(); // index of face to be added
   faces.emplace_back();
   MeshFace& face = faces[idx];
   face.vertex_index[0] = vi0;
   face.vertex_index[1] = vi1;
   face.vertex_index[2] = vi2;
   vertices[face.vertex_index[0]].connected_faces.push_back(idx);
   vertices[face.vertex_index[1]].connected_faces.push_back(idx);
   vertices[face.vertex_index[2]].connected_faces.push_back(idx);
}

void Mesh::clear()
{
   faces.clear();
   vertices.clear();
   vertex_hash_map.clear();
}

void Mesh::finish()
{
   // Finish up the mesh, clear the vertex_hash_map, as it's no longer needed from this
point on and uses quite a bit of memory.
   vertex_hash_map.clear();

   // For each face, store which other face is connected with it.
   for(unsigned int i=0; i<faces.size(); i++)
   {
       MeshFace& face = faces[i];
       face.connected_face_index[0] = getFaceIdxWithPoints(face.vertex_index[0],
face.vertex_index[1], i); // faces are connected via the outside
       face.connected_face_index[1] = getFaceIdxWithPoints(face.vertex_index[1],
face.vertex_index[2], i);
       face.connected_face_index[2] = getFaceIdxWithPoints(face.vertex_index[2],
face.vertex_index[0], i);
   }
}

Point3 Mesh::min()
{
   if (vertices.size() < 1)
       return Point3(0, 0, 0);
   Point3 ret = vertices[0].p;
   for(unsigned int i=0; i<vertices.size(); i++)
   {
       ret.x = std::min(ret.x, vertices[i].p.x);
       ret.y = std::min(ret.y, vertices[i].p.y);
       ret.z = std::min(ret.z, vertices[i].p.z);
   }
   return ret;
}
Point3 Mesh::max()
{
   if (vertices.size() < 1)
       return Point3(0, 0, 0);
   Point3 ret = vertices[0].p;
   for(unsigned int i=0; i<vertices.size(); i++)
   {
       ret.x = std::max(ret.x, vertices[i].p.x);
       ret.y = std::max(ret.y, vertices[i].p.y);
       ret.z = std::max(ret.z, vertices[i].p.z);
   }
   return ret;
}

int Mesh::findIndexOfVertex(Point3& v)
{
   uint32_t hash = pointHash(v);

   for(unsigned int idx = 0; idx < vertex_hash_map[hash].size(); idx++)
   {
       if ((vertices[vertex_hash_map[hash][idx]].p - v).testLength(vertex_meld_distance))
       {
           return vertex_hash_map[hash][idx];
       }
   }
   vertex_hash_map[hash].push_back(vertices.size());
   vertices.emplace_back(v);
   return vertices.size() - 1;
}

/*!
Returns the index of the 'other' face connected to the edge between vertices with indices
idx0 and idx1.
In case more than two faces are connected via the same edge, the next face in a counter-
clockwise ordering (looking from idx1 to idx0) is returned.

\cond DOXYGEN_EXCLUDE
   [NON-RENDERED COMENTS]
   For two faces abc and abd with normals n and m, we have that:
   \f{eqnarray*}{
   n &=& \frac{ab \times ac}{\|ab \times ac\|}     \\
   m &=& \frac{ab \times ad}{\|ab \times ad\|}     \\
   n \times m &=& \|n\| \cdot \|m\| \mathbf{p} \sin \alpha  \\
   && (\mathbf{p} \perp n \wedge \mathbf{p} \perp m) \\
   \sin \alpha &=& \|n \times m \|
   &=& \left\| \frac{(ab \times ac) \times (ab \times ad)}{\|ab \times ac\| \cdot \|ab \times
ad\|}  \right\|    \\
   &=& \left\| \frac{ (ab \cdot (ac \times ad)) ab  }{\|ab \times ac\| \cdot \|ab \times ad\|}
\right\|    \\
   &=&  \frac{ (ab \cdot (ac \times ad)) \left\| ab   \right\| }{\|ab\| \|ac\| \sin bac \cdot \|ab\|
\|ad\| \sin bad}    \\
   &=&  \frac{  ab \cdot (ac \times ad)  }{\|ab\| \|ac\| \|ad\|  \sin bac \sin bad}    \\
   \f}}
\endcond

See <a href="http://stackoverflow.com/questions/14066933/direct-way-of-computing-
clockwise-angle-between-2-vectors">Direct way of computing clockwise angle between
2 vectors</a>



*/
int Mesh::getFaceIdxWithPoints(int idx0, int idx1, int notFaceIdx)
```

```
{
   std::vector<int> candidateFaces; // in case more than two faces meet at an edge,
multiple candidates are generated
   int notFaceVertexIdx = -1; // index of the third vertex of the face corresponding to
notFaceIdx
   for(int f : vertices[idx0].connected_faces) // search through all faces connected to the
first vertex and find those that are also connected to the second
   {
      if (f == notFaceIdx)
      {
         for (int i = 0; i<3; i++) // find the vertex which is not idx0 or idx1
            if (faces[f].vertex_index[i] != idx0 && faces[f].vertex_index[i] != idx1)
               notFaceVertexIdx = faces[f].vertex_index[i];
         continue;
      }
      if ( faces[f].vertex_index[0] == idx1 // && faces[f].vertex_index[1] == idx0 // next
face should have the right direction!
        || faces[f].vertex_index[1] == idx1 // && faces[f].vertex_index[2] == idx0
        || faces[f].vertex_index[2] == idx1 // && faces[f].vertex_index[0] == idx0
         )  candidateFaces.push_back(f);

   }

   if (candidateFaces.size() == 0) { cura::logError("Couldn't find face connected to face
%i.\n", notFaceIdx); return -1; }
   if (candidateFaces.size() == 1) { return candidateFaces[0]; }


   if (notFaceVertexIdx < 0) { cura::logError("Couldn't find third point on face %i.\n",
notFaceIdx); return -1; }

   if (candidateFaces.size() % 2 == 0) cura::log("Warning! Edge with uneven number of
faces connecting it!(%i)\n", candidateFaces.size()+1);

   FPoint3 vn = vertices[idx1].p - vertices[idx0].p;
   FPoint3 n = vn / vn.vSize(); // the normal of the plane in which all normals of faces
connected to the edge lie => the normalized normal
   FPoint3 v0 = vertices[idx1].p - vertices[idx0].p;

// the normals below are abnormally directed! : these normals all point counterclockwise
(viewed from idx1 to idx0) from the face, irrespective of the direction of the face.
   FPoint3 n0 = FPoint3(vertices[notFaceVertexIdx].p - vertices[idx0].p).cross(v0);

   if (n0.vSize() <= 0) cura::log("Warning! Face %i has zero area!", notFaceIdx);

   double smallestAngle = 1000; // more then 2 PI (impossible angle)
```

```
    int bestIdx = -1;

    for (int candidateFace : candidateFaces)
    {
        int candidateVertex;
        {// find third vertex belonging to the face (besides idx0 and idx1)
            for (candidateVertex = 0; candidateVertex<3; candidateVertex++)
                if (faces[candidateFace].vertex_index[candidateVertex] != idx0 &&
faces[candidateFace].vertex_index[candidateVertex] != idx1)
                    break;
        }

        FPoint3 v1 = vertices[candidateVertex].p -vertices[idx0].p;
        FPoint3 n1 = v1.cross(v0);

        double dot = n0 * n1;
        double det = n * n0.cross(n1);
        double angle = std::atan2(det, dot);
        if (angle < 0) angle += 2*M_PI; // 0 <= angle < 2* M_PI

        if (angle == 0)
        {
            cura::log("Warning! Overlapping faces: face %i and face %i.\n", notFaceIdx,
candidateFace);
            std::cerr<< n.vSize() <<"; "<<n1.vSize()<<";"<<n0.vSize() <<std::endl;
        }
        if (angle < smallestAngle)
        {
            smallestAngle = angle;
            bestIdx = candidateFace;
        }
    }
    if (bestIdx < 0) cura::logError("Couldn't find face connected to face %i.\n",
notFaceIdx);
    return bestIdx;
}
```

```
//mesh.h
#ifndef MESH_H
#define MESH_H

#include "settings.h"

/*!
Vertex type to be used in a Mesh.

Keeps track of which faces connect to it.
*/
class MeshVertex
{
public:
   Point3 p; //!< location of the vertex
   std::vector<uint32_t> connected_faces; //!< list of the indices of connected faces

   MeshVertex(Point3 p) : p(p) {} //!< doesn't set connected_faces
};

/*! A MeshFace is a 3 dimensional model triangle with 3 points. These points are already
converted to integers

A face has 3 connected faces, corresponding to its 3 edges.

Note that a correct model may have more than 2 faces connected via a single edge!
In such a case the face_index stored in connected_face_index is the one connected via the
outside; see ASCII art below:

: horizontal slice through vertical edge connected to four faces :

\verbatim
[inside] x|
       x| <--+--- faces which contain each other in their connected_face_index fiels
  xxxxxxx|   \|/
  -------+-------
     ^   |xxxxxxx
     +-->|x
     |   |x [inside]
     |
   faces which contain each other in their connected_face_index fiels
\endverbatim
*/
class MeshFace
{
public:
   int vertex_index[3] = {-1}; //!< counter-clockwise ordering
   int connected_face_index[3]; //!< same ordering as vertex_index (connected_face 0 is
connected via vertex 0 and 1, etc.)
};


/*!
A Mesh is the most basic representation of a 3D model. It contains all the faces as
MeshFaces.

See MeshFace for the specifics of how/when faces are connected.
*/
class Mesh : public SettingsBase // inherits settings
{
   //! The vertex_hash_map stores a index reference of each vertex for the hash of that
location. Allows for quick retrieval of points with the same location.
   std::map<uint32_t, std::vector<uint32_t> > vertex_hash_map;
public:
   std::vector<MeshVertex> vertices;//!< list of all vertices in the mesh
   std::vector<MeshFace> faces; //!< list of all faces in the mesh

   Mesh(SettingsBase* parent); //!< initializes the settings

   void addFace(Point3& v0, Point3& v1, Point3& v2); //!< add a face to the mesh
without settings it's connected_faces.
   void clear(); //!< clears all data
   void finish(); //!< complete the model : set the connected_face_index fields of the
faces.

   Point3 min(); //!< min (in x,y and z) vertex of the bounding box
   Point3 max(); //!< max (in x,y and z) vertex of the bounding box

private:
   int findIndexOfVertex(Point3& v); //!< find index of vertex close to the given point, or
create a new vertex and return its index.
   /*!
   Get the index of the face connected to the face with index \p notFaceIdx, via vertices \p
idx0 and \p idx1.
   In case multiple faces connect with the same edge, return the next counter-clockwise
face when viewing from \p idx1 to \p idx0.
   */
   int getFaceIdxWithPoints(int idx0, int idx1, int notFaceIdx);
};


#endif//MESH_H
```

```
//settings.cpp
#include <cctype>
#include <fstream>
#include <stdio.h>
#include <sstream> // ostringstream
#include "utils/logoutput.h"

#include "settings.h"
#include "settingRegistry.h"

//c++11 no longer defines M_PI, so add our own constant.
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

SettingsBase::SettingsBase()
: parent(NULL)
{
}

SettingsBase::SettingsBase(SettingsBase* parent)
: parent(parent)
{
}

void SettingsBase::setSetting(std::string key, std::string value)
{
    if (SettingRegistry::getInstance()->settingExists(key))
    {
        setting_values[key] = value;
    }
    else
    {
        cura::logError("Warning: setting an unregistered setting %s\n", key.c_str() );
        setting_values[key] = value; // Handy when programmers are in the process of
introducing a new setting
    }
}

std::string SettingsBase::getSettingString(std::string key)
{
    if (setting_values.find(key) != setting_values.end())
    {
        return setting_values[key];
    }
    if (parent)
    {
        return parent->getSettingString(key);
    }

    if (SettingRegistry::getInstance()->settingExists(key))
    {
        setting_values[key] = SettingRegistry::getInstance()->getSettingConfig(key)-
>getDefaultValue();
        cura::logError("Using default for: %s = %s\n", key.c_str(),
setting_values[key].c_str());
    }
    else
    {
        setting_values[key] = "";
        cura::logError("Unregistered setting %s\n", key.c_str());
    }
    return setting_values[key];
}

bool SettingsBase::hasSetting(std::string key)
{
    if (setting_values.find(key) != setting_values.end())
    {
        return true;
    }
    if (parent)
    {
        return parent->hasSetting(key);
    }

    return false;
}

int SettingsBase::getSettingAsIndex(std::string key)
{
    std::string value = getSettingString(key);
    return atoi(value.c_str());
}

int SettingsBase::getSettingAsCount(std::string key)
{
    std::string value = getSettingString(key);
    return atoi(value.c_str());
}

int SettingsBase::getSettingInMicrons(std::string key)
```

```
{
   std::string value = getSettingString(key);
   return atof(value.c_str()) * 1000.0;
}

double SettingsBase::getSettingInAngleRadians(std::string key)
{
   std::string value = getSettingString(key);
   return atof(value.c_str()) / 180.0 * M_PI;
}

bool SettingsBase::getSettingBoolean(std::string key)
{
   std::string value = getSettingString(key);
   if (value == "on")
      return true;
   if (value == "yes")
      return true;
   if (value == "true" or value == "True") //Python uses "True"
      return true;
   return atoi(value.c_str()) != 0;
}

double SettingsBase::getSettingInDegreeCelsius(std::string key)
{
   std::string value = getSettingString(key);
   return atof(value.c_str());
}

double SettingsBase::getSettingInMillimetersPerSecond(std::string key)
{
   std::string value = getSettingString(key);
   return std::max(1.0, atof(value.c_str()));
}

double SettingsBase::getSettingInPercentage(std::string key)
{
   std::string value = getSettingString(key);
   return std::max(0.0, atof(value.c_str()));
}

double SettingsBase::getSettingInSeconds(std::string key)
{
   std::string value = getSettingString(key);
   return std::max(0.0, atof(value.c_str()));
}
```

```
EGCodeFlavor SettingsBase::getSettingAsGCodeFlavor(std::string key)
{
   std::string value = getSettingString(key);
   if (value == "RepRap")
      return GCODE_FLAVOR_REPRAP;
   else if (value == "UltiGCode")
      return GCODE_FLAVOR_ULTIGCODE;
   else if (value == "Makerbot")
      return GCODE_FLAVOR_MAKERBOT;
   else if (value == "BFB")
      return GCODE_FLAVOR_BFB;
   else if (value == "MACH3")
      return GCODE_FLAVOR_MACH3;
   else if (value == "RepRap (Volumatric)")
      return GCODE_FLAVOR_REPRAP_VOLUMATRIC;
   return GCODE_FLAVOR_REPRAP;
}

EFillMethod SettingsBase::getSettingAsFillMethod(std::string key)
{
   std::string value = getSettingString(key);
   if (value == "Lines")
      return Fill_Lines;
   if (value == "Grid")
      return Fill_Grid;
   if (value == "Triangles")
      return Fill_Triangles;
   if (value == "Concentric")
      return Fill_Concentric;
   if (value == "ZigZag")
      return Fill_ZigZag;
   return Fill_None;
}

EPlatformAdhesion SettingsBase::getSettingAsPlatformAdhesion(std::string key)
{
   std::string value = getSettingString(key);
   if (value == "Brim")
      return Adhesion_Brim;
   if (value == "Raft")
      return Adhesion_Raft;
   return Adhesion_None;
}

ESupportType SettingsBase::getSettingAsSupportType(std::string key)
```

```
{
    std::string value = getSettingString(key);
    if (value == "Everywhere")
        return Support_Everywhere;
    if (value == "Touching Buildplate")
        return Support_PlatformOnly;
    return Support_None;
}
```

```
//settings.h
#ifndef SETTINGS_H
#define SETTINGS_H

#include <vector>
#include <map>

#include "utils/floatpoint.h"

#ifndef VERSION
#define VERSION "MOSTMetalCura"
#endif

/*!
 * Different flavors of GCode. Some machines require different types of GCode.
 * The GCode flavor definition handles this as a big setting to make major or minor
modifications to the GCode.
 */
enum EGCodeFlavor
{
/**
 * RepRap flavored GCode is Marlin/Sprinter/Repetier based GCode.
 *  This is the most commonly used GCode set.
 *  G0 for moves, G1 for extrusion.
 *  E values give mm of filament extrusion.
 *  Retraction is done on E values with G1. Start/end code is added.
 *  M106 Sxxx and M107 are used to turn the fan on/off.
 **/
   GCODE_FLAVOR_REPRAP = 0,
/**
 * UltiGCode flavored is Marlin based GCode.
 *  UltiGCode uses less settings on the slicer and puts more settings in the firmware. This
makes for more hardware/material independed GCode.
 *  G0 for moves, G1 for extrusion.
 *  E values give mm^3 of filament extrusion. Ignores the filament diameter setting.
 *  Retraction is done with G10 and G11. Retraction settings are ignored. G10 S1 is used
for multi-extruder switch retraction.
 *  Start/end code is not added.
 *  M106 Sxxx and M107 are used to turn the fan on/off.
 **/
   GCODE_FLAVOR_ULTIGCODE = 1,
/**
 * Makerbot flavored GCode.
 *  Looks a lot like RepRap GCode with a few changes. Requires MakerWare to convert
to X3G files.
 *   Heating needs to be done with M104 Sxxx T0
 *   No G21 or G90
 *   Fan ON is M126 T0 (No fan strength control?)
 *   Fan OFF is M127 T0
 *   Homing is done with G162 X Y F2000
 **/
   GCODE_FLAVOR_MAKERBOT = 2,

/**
 * Bits From Bytes GCode.
 *  BFB machines use RPM instead of E. Which is coupled to the F instead of
independed. (M108 S[deciRPM])
 *  Need X,Y,Z,F on every line.
 *  Needs extruder ON/OFF (M101, M103), has auto-retrection (M227 S[2560*mm]
P[2560*mm])
 **/
   GCODE_FLAVOR_BFB = 3,

/**
 * MACH3 GCode
 *  MACH3 is CNC control software, which expects A/B/C/D for extruders, instead of E.
 **/
   GCODE_FLAVOR_MACH3 = 4,
/**
 * RepRap volumatric flavored GCode is Marlin based GCode.
 *  Volumatric uses less settings on the slicer and puts more settings in the firmware. This
makes for more hardware/material independed GCode.
 *  G0 for moves, G1 for extrusion.
 *  E values give mm^3 of filament extrusion. Ignores the filament diameter setting.
 *  Retraction is done with G10 and G11. Retraction settings are ignored. G10 S1 is used
for multi-extruder switch retraction.
 *  M106 Sxxx and M107 are used to turn the fan on/off.
 **/
   GCODE_FLAVOR_REPRAP_VOLUMATRIC = 5,
};

/*!
 * In Cura different infill methods are available.
 * This enum defines which fill patterns are available to get a uniform naming troughout
the engine.
 * The different methods are used for top/bottom, support and sparse infill.
 */
enum EFillMethod
{
   Fill_Lines,
   Fill_Grid,
   Fill_Triangles,
    Fill_Concentric,
    Fill_ZigZag,
    Fill_None
};

/*!
 * Type of platform adheasion
 */
enum EPlatformAdhesion
{
    Adhesion_None,
    Adhesion_Brim,
    Adhesion_Raft
};

/*!
 * Type of support material to generate
 */
enum ESupportType
{
    Support_None,
    Support_PlatformOnly,
    Support_Everywhere
};

#define MAX_EXTRUDERS 16

//Maximum number of sparse layers that can be combined into a single sparse extrusion.
#define MAX_SPARSE_COMBINE 8

/*!
 * Base class for every object that can hold settings.
 * The SettingBase object can hold multiple key-value pairs that define settings.
 * The settings that are set on a SettingBase are checked against the SettingRegistry to
ensure keys are valid.
 * Different conversion functions are available for settings to increase code clarity and in
the future make
 * unit conversions possible.
 */
class SettingsBase
{
private:
    std::map<std::string, std::string> setting_values;
    SettingsBase* parent;
public:
    SettingsBase();
    SettingsBase(SettingsBase* parent);

    bool hasSetting(std::string key);

    void setSetting(std::string key, std::string value);

    std::string getSettingString(std::string key);
    int getSettingAsIndex(std::string key);
    int getSettingAsCount(std::string key);

    double getSettingInAngleRadians(std::string key);
    int getSettingInMicrons(std::string key);
    bool getSettingBoolean(std::string key);
    double getSettingInDegreeCelsius(std::string key);
    double getSettingInMillimetersPerSecond(std::string key);
    double getSettingInPercentage(std::string key);
    double getSettingInSeconds(std::string key);

    EGCodeFlavor getSettingAsGCodeFlavor(std::string key);
    EFillMethod getSettingAsFillMethod(std::string key);
    EPlatformAdhesion getSettingAsPlatformAdhesion(std::string key);
    ESupportType getSettingAsSupportType(std::string key);
};


#endif//SETTINGS_
```