Delft University of Technology
Master's Thesis in Embedded Systems

3D Gradient Printing of Energetic Multi-Materials

Bart Rijnders

3D Gradient Printing of Energetic Multi-Materials

Master's Thesis in Embedded Systems

Embedded Software Section
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Bart Rijnders
B.Rijnders@student.tudelft.nl

1st July 2019

Author
Bart Rijnders (B.Rijnders@student.tudelft.nl)
Title
3D Gradient Printing of Energetic Multi-Materials
MSc presentation
10th July 2019

Abstract

The performance of gun and rocket propellants, which consist of energetic materials, is largely determined by their geometry and composition. Conventional production methods limit the performance by putting constraints on both. With additive manufacturing, or 3D-printing, there are significantly fewer geometry constraints and together with the ability to combine multiple materials into a continuous gradient new performance optimization opportunities are created. In the current 3D-printing world it is possible to print single-material or discrete gradient multi-material objects by translating a CAD model to printer instructions. This translation is done with slicer software that slices a 3D-model and outputs printer instructions in a G-Code file. This thesis looks at how an object with a continuous gradient can be printed. A modified version of the popular Cura slicing software is presented that can apply an approximation of a continuous gradient to an input model. The printer paths are simulated with the slicer software and ultimately printed using TNO's multi-material 3D-printer. While the print results show that energetic materials behave in such a different way than normal plastics that 3D-printing them it is not an easy task, printing a 3D-model with a multi-material continuous gradient is certainly viable.

Preface

Before I started my internship at TNO my knowledge of 3D-printing was very limited. When I spotted the ad by TNO on the Embedded Software website I became intrigued by the exotic nature of the assignment. I was aware that 3D-printing was becoming increasingly popular, but I could never have imagined research would be done on 3D-printing explosives and rocket fuel. After some discussions with Michiel about the topic I accepted the assignment. When the internship started I soon became overwhelmed with all kinds of terms and thousands of lines of code that described how 3D-printing worked. But after countless of hours of coding and printing in a bunker, without any windows or sunlight, I became skilled on the subject, which resulted in the work presented in this thesis.

Without the help of my supervisors Koen and Michiel throughout this project I probably would have never completed it. I would like to thank them for their guidance, support, and especially for their patience during some difficult times that I endured during the start of my internship. Besides my supervisors I would like to thank my parents for their patience and support through all the years that it took to complete my studies at this university. Further I would like to thank my fellow students Jos and Nathan. Jos for keeping me company in the bunker during the countless experiments and Nathan who I collaborated with on multiple courses and especially my last course, which I could not have completed without his hard work. Finally I am grateful for all my friends from ”Bru” for their encouragement with Axel in particular, who supported me when I needed it the most during the start of my internship.

Bart Rijnders

Delft, The Netherlands
1st July 2019

Contents

Chapter 1

Introduction

In recent years 3D-printing has become an increasingly hot topic. Various companies have emerged that produce software and hardware with increasing precision and advanced options to print 3D-models for industrial and consumer purposes. Besides printing plastic, the material most consumer printers use, printers exist that print other conventional materials ranging from metals to glass and epoxy resins. More exotic options that print concrete structures [12], bone tissue [10], organs [13], and electronics [4] are also being researched. Other than the various materials that can be printed, different print techniques also exist. The most commonly used technique is extrusion-based printing, which usually involves melting plastic and pushing it through a nozzle and then building the 3D-object layer upon layer. While different variations exist of extrusion-based printing, in the 3D-printing world they are unified under the name Fused Deposition Modeling or FDM. Other techniques are SLA (stereolithograpy), which uses UV-light to cure material in a photopolymer vat, sheet lamination, powder bed printing and material jetting.

At TNO a new 3D-printer has been developed that is capable of combining two materials in virtually any ratio. This printer uses the FDM printing technique combined with UV-curing and was built to print Energetic Materials instead of conventional plastics. But instead of filament extrusion it uses RAM-based extrusion, which involves a piston pushing paste through a syringe that is connected to the nozzle. Energetic Materials or EM are substances in any state (liquid, solid, gaseous or a combination of these) that store a high amount of chemical energy. This energy can be released in a very short time when initiated by mechanical, thermal or shock stimuli. Typical applications of EM consist, but are not limited to, products like propellants, explosives and pyrotechnics. When producing EM such as propellants and explosives, the production methods are limited in terms of geometry and composition, which have a large impact on the performance of these products. With 3D-printing these limitations during production are significantly reduced by the freedom in geometry that 3D-printing offers. However, the ability to combine multiple materials and thus the ability to create functional gradients in the printed object is what allows the biggest performance optimization opportunity.

Printing with multiple materials and creating gradients introduces significant complexity for the software that generates instructions for the 3D-printer. The software that generates printer instructions from a 3D-model is called slicer software. It essentially slices a 3D-model into multiple layers and generates a printer path for each layer. But instead of just slicing a uniform volume, multiple materials now have to be taken into account. Other than multiple materials, new parameters such as the intensity of the UV-light and when to change the mixture ratio have to be considered when optimizing printer paths. This thesis investigates the 3D-printing of energetic materials with these new added challenges.

1.1 Problem Statement

Can multi-material objects with continuous gradients be modeled and 3D-printed while taking into account UV-curing?

The main goal of this thesis is to develop a process for modeling and printing of multi-material objects with continuous gradients. Conventional slicing software accepts simple 3D-CAD models that contain only information about the geometry of the object. With a gradient in the object, somewhere in the process the composition has to be added either numerically or analytically. This could be done in the CAD model, during slicing or after the final step when the printer instructions have been generated. The quality of the gradient in the printed object is of higher priority than print speed and the print paths should be optimized accordingly. Besides these challenges the interfaces between potential steps in the process have to be robust and the expertise required by the end user should be minimized. To our knowledge only software that can apply a color gradient to an input model's surface exists. A solution will have to be designed to apply a continuous functional material gradient to a simple input object.

1.2 Thesis Outline

The next chapter describes the related work for this thesis by discussing several papers and software currently on the market. Chapter 3 explains general information about 3D-printing required for the rest of the thesis. Followed by Chapter 4, which presents the design solution to print multi-material objects with a continuous gradient. Chapter 5 describes the implementation of the design and Chapter 6 presents the experimental results obtained. Finally Chapter 7 shows the conclusions and future work.

Chapter 2

Related Work

This chapter describes the related work for this thesis. Several relevant subjects for 3D-printing are looked into such as model representation in Section 2.1, path optimization in Section 2.2 and finally multi-material printing in Section 2.3.

2.1 Model Representation

Since this assignment requires a way to describe a multi-material object; literature was researched on 3D-model representation and CAD modeling. Several existing systems and file formats exist that can describe multi-material models. The most used format for 3D models in industry is STL, which contains simple geometry information about the object and is natively supported by almost all CAD software. The downside is however that it is not able to describe material properties of the object, which is why other file formats have been designed. One alternative is described later in this section.

Hsieh et al. developed a system that is based on the FDM method [1]. Their Multi-CAD system can design models and do multi-material slicing of these models. When models have been sliced the print paths can be simulated. This study focuses on the entire process, which includes a file format they created that contains information about geometry, build materials and fabrication process information. While their system is outdated, paper was published in 2002, the format to store the model geometry and build materials could be useful for this thesis.

A more modern format to represent 3D models used in 3D printing has been proposed by Hiller et al. in 2009 [5]. They describe a new file format that could handle the requirements that arose with new technological developments in 3D printing, such as multiple and graded materials. The Additive Manufacturing File format or AMF can describe material properties and geometry in a resolution-independent way. The geometry can be described in regions using triangle meshes, functions or voxel bitmaps. Each region can have material properties that can be described by single or multiple materials. When multiple materials are used a function can be defined enabling a smooth gradient. An example of a model described with a continuous gradient can be seen in Figure 2.1 This could be a very useful format in this thesis.

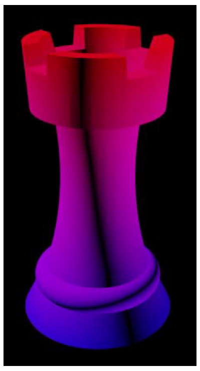


Figure 2.1: A 3D-model with continuous gradient described by AMF.

2.2 Path Optimization

Qui et al. studied a void-eliminating toolpath for extrusion-based multi-material layered manufacturing [2]. They describe three forms of void creating issues:

- SVV: Sub-perimeter vector-direction void, which is a void in the printing direction and is predictable. It happens when two print paths intersect and one of them is blocked, leaving a void in the extended direction of that path.
- SRV: Sub-perimeter road-turn void. Happens when the print head makes a turn and leaves a small void. Is also predictable
- IV: Irregular void. Is because of irregular behaviors of machine or material flow. It is unpredictable and cannot be accounted for with slicing software.

The voids created by the printer are shown in Figure 2.2. The authors use adaptive roadwidth and path extension as solutions for these voids, which are significantly reduced with the proposed solutions. They still, however, only make use of a single material per layer, only allowing gradients in the vertical of the printer. For gradients in the horizontal direction the optimization of the gradient also has to be taken into account and the described

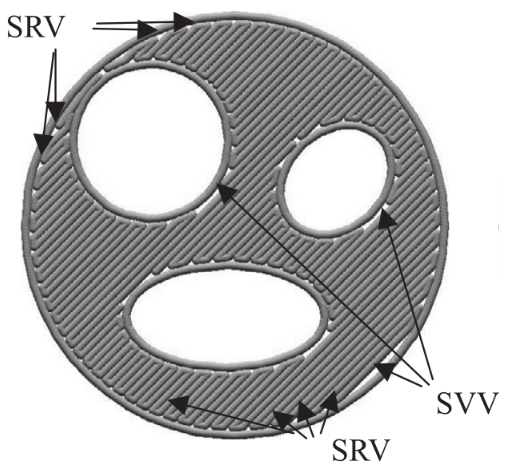


Figure 2.2: Vizualization of toolpath voids.

solutions may no longer be optimal. This paper only considers one standard infill pattern, which is not concentric, meaning the generated model will be asymmetric. A more symmetric concentric infill pattern is likely more useful to create radial gradients.

Jin et al. developed an algorithm to address common issues faced with additive manufacturing: low deposition quality and poor surface finish of printed parts [7]. They propose a path-planning technique that focuses on an optimization scheme for contour parallel-based path generation. It optimizes the spacing between adjacent paths and sharp corners, which is where most quality issues stem from. Figure 2.3 shows the parallel contour paths generated by the infill method to be optimized.

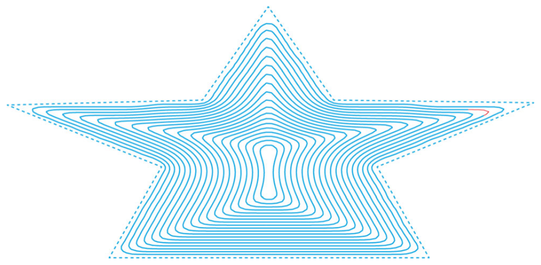


Figure 2.3: Contour parallel-based path generation.

Each contour is generated by applying an offset to points in the outer contour and is terminated when no new contours can be generated. It can be observed that some sharp corners appear along the offset contours that affect quality of the printed part. To counter the sharp corners they apply a smoothing function. However, after applying the smoothing function some paths suffer from non-uniform path spacing. To make the path spacing more uniform a local optimization algorithm is applied that takes circular samples along a center line of the object and shifts paths closer or further apart to create a more uniform path width distribution. Both these optimizations are used and applied to a layer of a 3D model to improve the quality of the deposition paths. Figure 2.4 shows the original paths and the optimization result that clearly has improved deposition path quality.

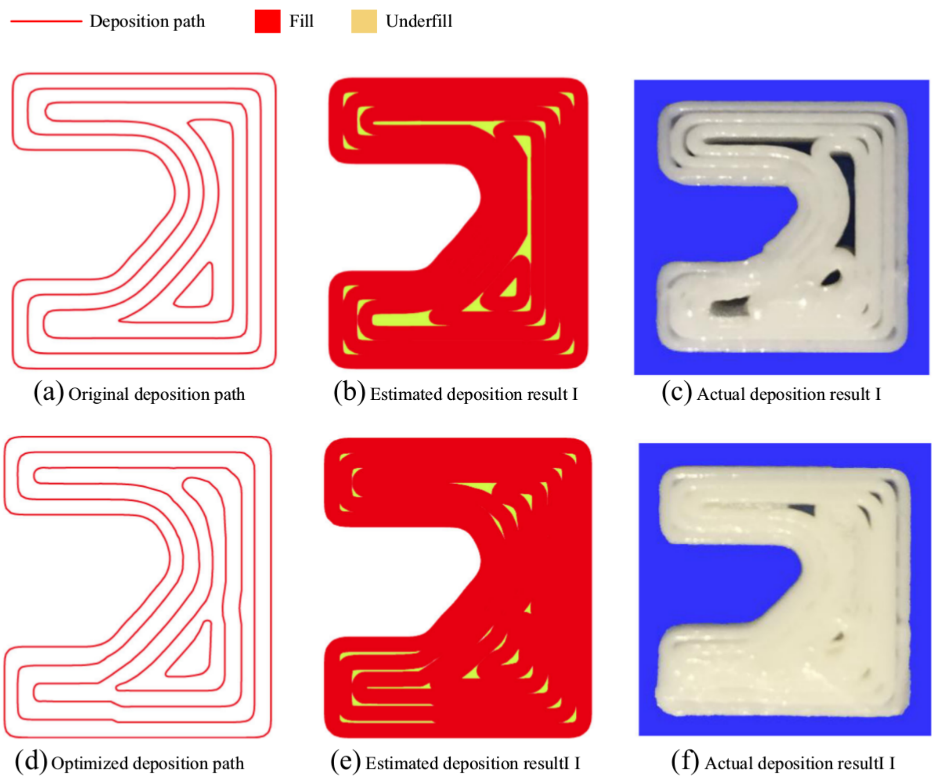

Deposition path
Fill
Underfill
(a) Original deposition path
(b) Estimated deposition result I
(c) Actual deposition result I
(d) Optimized deposition path
(e) Estimated deposition resultI I
(f) Actual deposition resultI I

Figure 2.4: Optimization of deposition paths.

2.3 Multi-Material Printing

Vidimče et al. developed OpenFab [11]. OpenFab is a programmable pipeline for multi-material fabrication. It is inspired by modern GPU pipelines and aims to easily design, slice and print multi-material objects. Their pipeline evaluates geometric models procedurally including material properties. It makes use of shader-like fablets. Shaders are used in computer graphics to procedurally define appearance of an object. Similarly a fablet is a description of an object in OpenFL, a C-like programming language, that describes both volume and surface together as methods of the fablet object. Fablets are streamed to the 3D printer, thus producing output with little delay. It seems an ideal solution for printing objects with functional gradients. However, because of the streaming implementation for specific hardware it is not very applicable to this thesis. It claims to be open source, but the source still has not been released since the paper came out 5 years ago.

Sitthi-Amorn et al. developed MultiFab, which is a multi-material 3D-printing platform that also includes a machine vision system [9]. This system enables self-calibration of printheads, 3D scanning and a closed-feedback loop to correct the printer in case of imperfections. Their software architecture is specific to their printer hardware, it requires a voxel (volume pixel) based 3D model as input where each voxel of the model contains a material ID as seen in Figure 2.5. Since each voxel contains a single material ID, no true gradients can be applied, but rather a discrete approximation by varying the density of voxels with a certain material ID can be used. Each voxel is printed by a droplet from the nozzle with a certain size dependent on the printer resolution, which is variable. Commands are generated for the positioning system, printhead modules, UV-curing module and a printhead cleaning sequence. The commands generated by the software are sent over Ethernet and USB to the microcontrollers that control the printer.

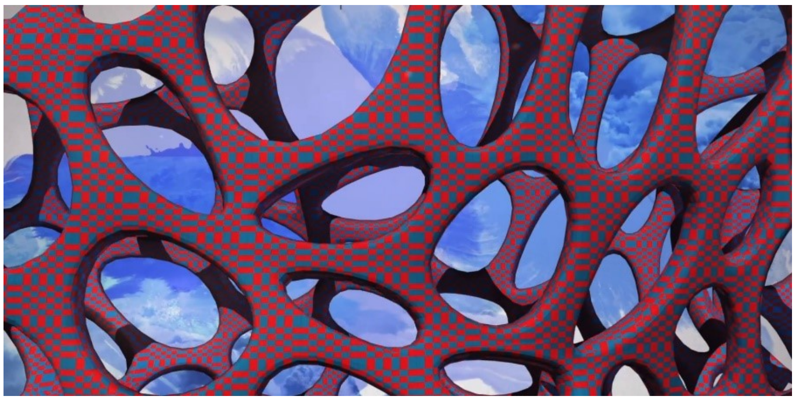


Figure 2.5: A model described by voxels

Lefebvre et al. discussed improving software for multi-filament 3D prints in a paper titled Clean Color [6]. When multiple extruders, a device that pushes material through an ozzle, are used like with TNOs printer; an issue called oozing reduces print quality. Oozing is the bleeding of material from one of the extruders when it should not be extruding. This causes small imperfections in the print result, which is especially noticable when printing with colors. While colors are not important when printing energetic materials the potentially small model size means these small imperfections matter significantly more than with conventional sizes. Three complementary techniques are introduced that should improve print quality: choosing a better azimuth angle for the printed part, a disposable rampart close to the print for nozzle wiping, and improved path planning which avoids already printed parts.

Wang et al. designed a method to fabricate functionally graded materials using inkjet color printing [8]. Inkjet printing mixes two starting components in a reservoir before printing; however, it is only capable of handling gradients with simple profiles. If the composition of the gradient becomes too complex or too steep it is difficult to change the composition of the reservoir dramatically to match the gradient change. Their system uses two reservoirs and the printer deposits drops of a mix of materials with a diameter varying from 5 to 50 µm. In Figure 2.6 the results of their prints can be seen. The gradients were designed by Adobe Photoshop and printed with a simple HP inkjet printer. Their study proves inkjet printing can be used to print a functionally graded material by stacking thin sheets of material. Major drawbacks are that inkjet printing is slow and requires your materials to be non-solid.

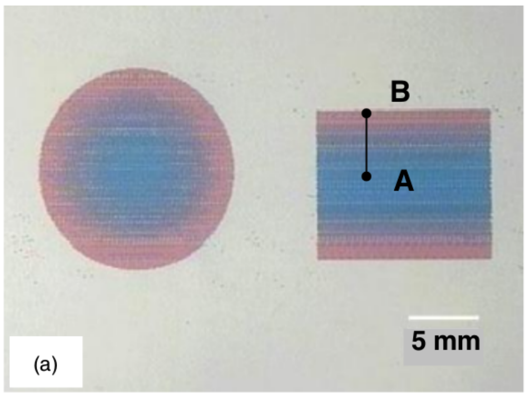


Figure 2.6: Gradients printed with inkjet printing.

Kokkinis et al. developed a system to 3D-print objects with Bioinspired Mechanical Gradients [3]. Their system prints resins and uses direct ink writing with a custom static mixer that homogenizes the resins before extrusion through a needle. They cure the prints with UV-lights after an entire layer has been printed. Their goal is to print an object where its elasticity is determined by material ratios defined over the objects geometry. A combination of both resins results in a higher elasticity. Figure 2.7 shows their final print result with the gradient indicated.

The inner and outer structure were combined as two seperate CAD files created with GrabCAD software. Slic3r, see Section 2.4.3, was used to convert the input models to printer instructions. The printer instructions were modified after they were generated to allow for the use of their custom static mixer and to implement the gradient changes. They placed a glass circular disc as stress concentrator in the middle of the gradient region and printed circular lines of a distinct material composition around the disc to

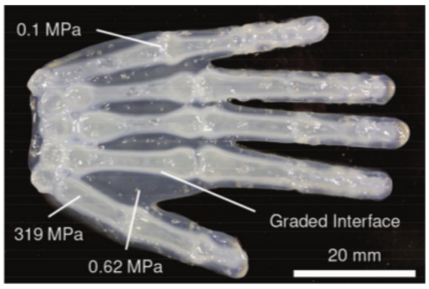


Figure 2.7: Object printed with inner and outer structure with a graded interface between them.

create the desired gradients. The material ratio was changed from line to line in the circular rings. In Figure 2.8 a layer of the print can be seen with the glass disc in the middle and the surrounding circular rings with different material ratios.

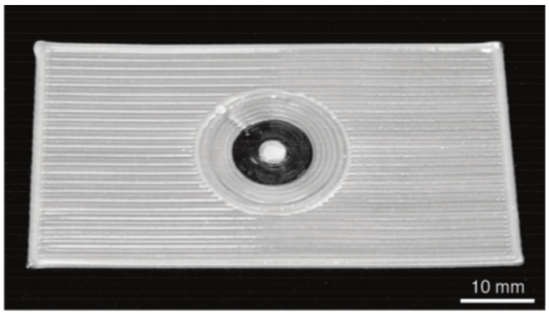


Figure 2.8: Print showing the circular rings with different material ratios.

2.4 Slicer Software

Currently various slicer software is available on the market that is both closed and open source. Below the most popular and most common slicer software is briefly discussed. Slicing software slices a geometric model into layers containing paths, these paths consist of printer instructions which can be various commands such as an extrusion move, travel move and retraction move. Slicing software is discussed in more detail in Section 5.1 in the implementation chapter. A visualization of the general printing process can be seen in Figure 2.9.

2.4.1 Ultimaker Cura

Cura is the slicing software developed by Ultimaker, which is a company that also builds 3D printers. It is open source and currently has been downloaded

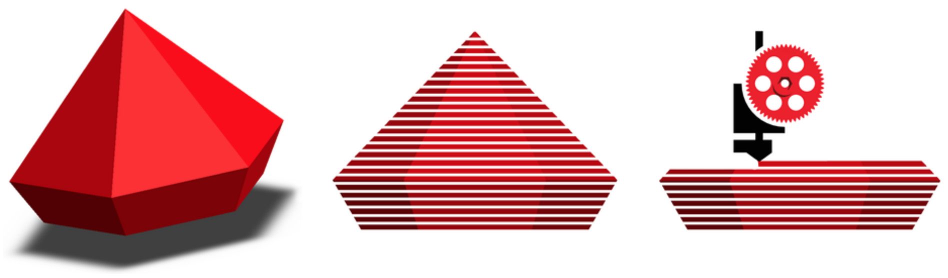


Figure 2.9: The general slicing and printing process.

over half a million times according to Ultimaker. It is continuously being developed by a large group and a new version is released every few months. The software has over 300 options to customize the printing process, making it also one of the most advanced slicers on the market. Documentation is also widely available. The slicing engine is written in C++ and the user interface is written in Python.

2.4.2 Simplify3D

Simplify3D is a commercial slicer available on the market, but is therefore closed source. It is the software used by TNO in the current implementation, which uses a simple post-processor to modify the output of Simplify3D. According to online reviews it generates high-quality prints [1], but these are not official reviews or published tests. Earlier tests with this software show viable results, however since the customization is constrained to only changing the settings or the post-processor the usage of this software is limited. A comprehensive manual is available online together with online support by the developers.

2.4.3 Slic3r

Slic3r is among the oldest slicer software available, it is however only developed part time by a team of volunteer developers. This can be seen in the features available, it is less customizable than both Cura and Simplify3D. The print quality seems comparable to Cura, but this is not verified by official research. Extensive documentation is available online, together with source description. The slice engine is written in C++ and the user interface is written in Perl.

[1]https://m.all3dp.com/simplify3d-review-best-slicer-3d-printing/

2.4.4 Repetier

Repetier is host software to directly connect to a printer that is compatible with other slicing engines. It has its own graphical user interface written in C++ and uses slicer engines of other available software like Cura or Slic3r. It offers less customization than the other slicers and is convenient when the printer is controlled from embedded devices such as an Arduino or a Raspberry Pi.

Chapter 3

Background

This chapter provides some background information regarding 3D-printing and specifics about the 3D-printer used at TNO. Section 3.1 describes the 3D-printer hardware and its different configurations. In Section 3.3 the programming language used with 3D-printers is briefly described. And finally Section 4.1 describes the requirements for the design of the software.

3.1 Printer Hardware

The 3D-printer used at TNO was designed in-house by the Additive Manufacturing department since no commercially off-the-shelf FDM (Fused Deposition Modeling) printers that use UV curing are available on the market. An overview of the printer can be seen in Figure 3.1.

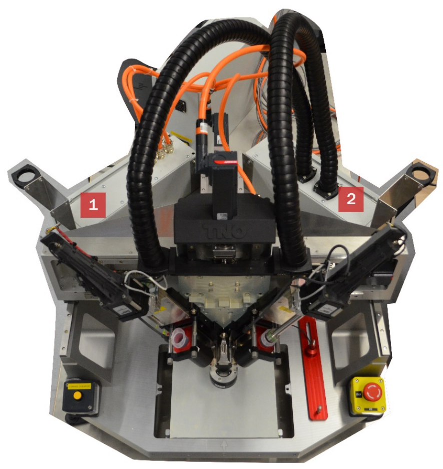


Figure 3.1: The FDM-UV printer

The printer can use different nozzles with variable diameters and commonly uses a path diameter of 1 mm. Most 3D printers currently on the market use heating to melt the plastic before it is pushed through the nozzle and use fans to cool the nozzle. The printer at TNO is also capable of heating the material but is currently not used to avoid having additional parameters to optimize. It has no fans to cool the nozzle. The printer uses PLCs (Programmable Logic Controller) from Beckhoff, which is a german company that implements automation systems. The Beckhoff PLC was chosen for its high reliability required in industrial standards and ease of configurability. The printer is extrusion-based and can use up to two extruders. In Figure 3.2 a cross section of an extruder can be seen.

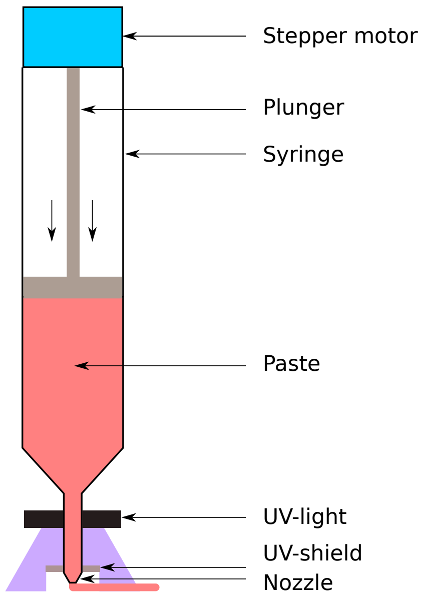


Figure 3.2: Extruder cross section.

The stepper motor drives the plunger into the syringe and pushes out the paste through the nozzle. The position of the plunger, or extruder axis, is relative since a stepper motor is used. A linear position sensor, not shown in the picture, is used to track the position of the plunger and defines the extruder axis. The UV-light shield prevents the material from curing inside the nozzle to prevent blocking.

3.1.1 Printer Configurations

Since various types of objects have to be printed the printer has three possible configurations. An overview of the configurations is shown in Figure 3.3.

The single nozzle configuration shown in Figure 3.3a is used when printing with just one material. The double-straight configuration shown in Fig-

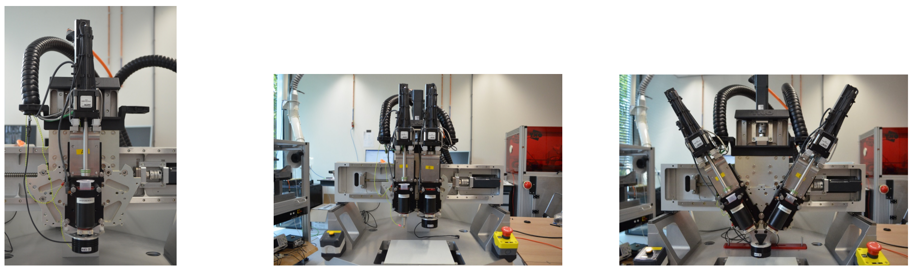

(a) Single nozzle. (b) Double straight. (c) Y-configuration

Figure 3.3: Different printer configurations.

ure 3.3b is used when printing with two materials. But only one material at a time, which creates a discrete gradient in the output print. The third configuration is the Y-configuration, which enables the use of the Y-nozzle and the co-extrusion nozzle. These two nozzles can mix materials together and as such can be used to print models with a continuous gradient.

3.2 Terminology

In the 3D-printing world quite a lot of jargon is used. To give some more insight into various terms used in this thesis a list was made to provide explanations.

Extrusion move Moving of the nozzle while extruding material; creating a print path.

Travel move Moving of the nozzle without extruding material.

Line or Print path Created by an extrusion move. A line or print path is defined by two points in where between material is deposited by an extrusion move.

Retraction The plunger is moved upwards to relieve some of the pressure in the syringe to prevent material dripping from the nozzle during travel moves.

Oozing Unwanted dripping of material from the nozzle; can be compensated for by retraction.

Prime Priming literally means "to make something ready for use or action". Priming an extruder means printing paths that are not part of a model to create pressure in the nozzle such that when the actual model is printed the extruder is ready for use.

Build plate The plateau used to print on. TNO's printer uses a steel plate that can be removed.

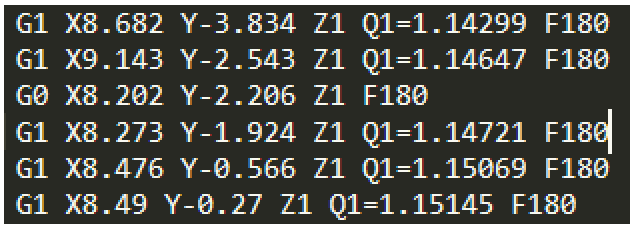
Build plate adhesion Several concentric rings can be printed attached to a model to increase the surface of the model that touches the build plate. This is to reduce the chance of a nozzle knocking over the print.

Skirt or Brim Concentric rings used to prime extruders and increase build plate adhesion. Skirt rings do not touch the actual model and only serve to prime extruders. Brim is attached to the model and increases print surface for better build plate adhesion.

Z-hop Lifting of the nozzle when retracting. When a travel move has to be made usually retraction is used to prevent oozing. Z-hop serves as an extra safety measure that lifts the nozzle to prevent accidental collision of the nozzle with print paths.

3.3 G-Code

As previously described slicer software slices a geometric model into layers containing printer instructions. These printer instructions are part of a numerical control programming language, which is named G-Code. It describes movement and actions of machine tools, such as a nozzle, a mill or a cutting tool. Several G-Code dialects exist with the most common one being Marlin. The 3D printer used at TNO uses a quite uncommon dialect called Beckhoff named after the company that produces the PLCs controlling the 3D printer. The PLCs support customized G-Code commands specific for this 3D printer meaning there is no universal standard for all commands in the Beckhoff dialect. A snippet of Beckhoff G-Code can be seen in Figure 3.4.



```
G1 X8.682 Y-3.834 Z1 Q1=1.14299 F180
G1 X9.143 Y-2.543 Z1 Q1=1.14647 F180
G0 X8.202 Y-2.206 Z1 F180
G1 X8.273 Y-1.924 Z1 Q1=1.14721 F180
G1 X8.476 Y-0.566 Z1 Q1=1.15069 F180
G1 X8.49 Y-0.27 Z1 Q1=1.15145 F180
```

Figure 3.4: G-Code snippet

The snippet contains extrusion moves (G1) and travel moves (G0) together with a speed indicated by F. Table 3.1 shows a detailed description of the symbols in the commands seen in Figure 3.4. In Beckhoff G-Code dialect, a movement command should always contain a speed and at least one coordinate or auxiliary axis (plunger position). An extrusion move containing a G1 moves the nozzle from the current position to the specified position while moving the plunger to the position specified by the Q-axis. This creates a straight line of printed material. A G0 command moves the nozzle to the specified position at maximum speed.

Command	Description
G0	Rapid Traverse
G1	Straight line interpolation
X, Y, Z	Position coordinates
Q	Position of plunger in mm
F	Speed in mm/s

Table 3.1: G-Code command overview

Chapter 4

Design

This chapter presents the design of the software to print a multi-material model with a continuous gradient. In Section 4.1 the requirements of the design are presented. Section 4.2 gives some general info about the purpose of the design. Section 4.3 proposes a solution to apply continuous gradients to an input model. Section 4.4 briefly explains the behaviour of the UV-lights followed by Section 4.5, which discusses support for multiple-material nozzles. Section 4.6 summarizes the design.

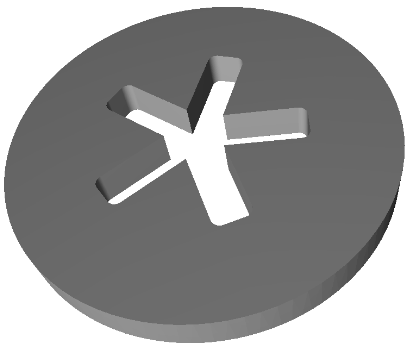
4.1 Requirements

TNO poses several requirements for the software, which can be derived from the problem statement. Summarized they are:

- Use TNO's printer hardware
- Design robust interfaces between various steps of the process
- Minimize the level of expertise required by the end user
- Use open-source software whenever possible

4.2 General

As defined by the requirements, TNO's printer hardware had to be used. The 3D printer uses a G-Code dialect called Beckhoff that is not natively supported by most slicer software and will have to be implemented. After supporting the new G-Code dialect more specific software adjustments have to be made such as support for different nozzles, UV-light support and ultimately the printing of continuous gradients. Combined these changes serve the purpose of printing a continuous gradient multi-material object where gradient quality is of the highest importance. The gradient defines the ratio of materials over the geometry of the object. An example of a 3D model, a star-grain rocket propellant, which could be used as input for the slicer, is shown in Figure 4.1a. The gradient applied to the models is always radial in the horizontal plane. In Figure 4.1b a cross section of the model with an ideal continuous gradient applied is shown.


(a) Example of a 3D input model.

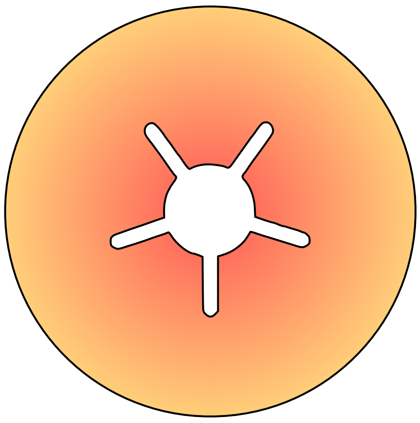

(b) Cross section with ideal continuous gradient.

Figure 4.1: Two star grain models.

The purpose of a gradient is to influence the burn profile of the propellant grains. By using two materials where one material burns faster than the other material; the gradient is able to define the burn speed as a function of the geometry of the propellant grain. The burn profile essentially burns in rings from the outside of the model to the inside. The outside of the model is defined as the outside shell touching the air. Some models have holes in the middle to accelerate the burn rate of the entire print or to only let the print burn from the inner ring to the outer ring. Since the 3D-printer has only limited precision the gradient will have to be discretized. In Figure 4.2 a cross section is shown with a discretized gradient and the direction of the burn profile is indicated with arrows. This gives a realistic indication of the desired print result for the star-grain rocket propellant with a continuous gradient.

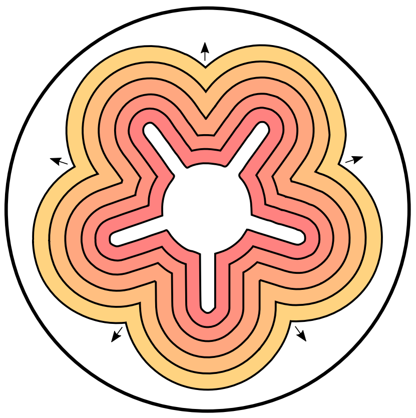


Figure 4.2: A star-grain rocket propellant with gradient.

4.3 Continuous Gradients

Printing a multi-material object with a continuous functional gradient requires an implementation of the material change at some point in the overall process from 3D-model to the actual printing. This gives three options as to when to implement the material changes:

1. In the initial 3D-model
2. During the slicing process
3. After the G-Code has been generated

Table 4.1 highlights the advantages and disadvantages of implementing the functional gradient in each step.

Process Step	Gradient Quality	Ease of Use	Software Complexity
Initial 3D-model	High/Medium	Requires CAD expertise Slicer easy to use	Process model material properties
Slicing Process	High	Requires input file of gradient	Passing gradient to engine
In G-Code	Low	Simple evaluation of gradient	Simple implementation

Table 4.1: Advatage and disadvantages of applying the functional gradient in each step of the process.

Defining the functional gradient in the initial 3D-model gives the designer of the 3D-model the best tools in defining the functional gradient together with the 3D-model using CAD software. The file STL format of 3D-models most commonly used does ,however, not support material properties. The proposed Additive Manufacturing Format, AMF, in Section 2.1 could be a solution [5]. A downside of defining the gradient in the initial 3D-model is that the printer has only limited precision and the model will be discretized during the slicing process. This makes a very precise definition of the functional gradient in the model obsolete and could result in small discrepancies of the gradient in the actual print. The user should also have knowledge about specialized CAD software supporting material properties instead of just geometric model design.

In the second approach the gradient is applied during the discretization step of the slicer. Therefore the actual precision of the functional gradient is defined by the path diameter of the print paths. Defining the functional gradient in this step means it will be defined with the precision the printer is capable of. The discrepancies described in the previous step will be minimized and there is a high probability of the best gradient quality. Implementing the functional gradient during this step also gives a high degree of control over the implementation. The user can configure the entire process in the slicer software with a simple input model. The downside of defining the functional gradient during slicing is that some format for the gradient together with a method to pass the definition of the gradient to the slicing engine will have to be designed. This means the user is now responsible for creating a functional gradient that matches the format required by the slicer instead of a generic gradient that could be defined in the initial 3D-model.

In the third approach the gradient is applied after the G-Code has been generated. Here a functional gradient could be easily evaluated by using coordinates of the commands for the printer as input in an analytic function that outputs the material ratio. This also provides a clear way for the user to define the gradient. The material ratio between two points cannot be changed. Meaning if points are spaced far apart and a change in material ratio is required the path should be split in multiple points. Evaluating the gradient here also requires building a post-processor besides modifiying the slicer software. The slicer software will have to be modified to include indicators for the post-processor where to insert gradients. Using a post-processor gives less control over implementing the gradient than during the slicing process and requires the user to also have knowledge about the post-processor instead of just slicing software.

4.3.1 Chosen Implementation Step

In the problem statement it became clear that gradient quality is the most important metric and that user expertise should also be considered. For a commercial product using a gradient definition in the initial 3D-model is likely the best choice; it offers a solution where the user can create a 3D-model with material properties and let the software take care of the rest. With careful optimization the gradient quality is also likely to be comparable to defining the gradient during the slicing process. However, since implementing the functional gradient during the slicing process has the highest chance of producing the best gradient quality it is the best option for this research. Even if it impacts user expertise somewhat negatively it allows for the highest degree of control over the gradient implementation. A format of the functional gradient definition will be designed together with a way to pass this definition to the slicing engine.

4.3.2 Gradient Specification

A format to specify the gradient and to pass it to the slicing engine is required. The functional gradient describing the material ratio in the 3D-model is rarely a linear gradient so simple linear interpolation between two values is not a viable option. The format should allow a material ratio to be defined or calculated for each ring in a print as shown by the star-grain in Figure 4.2. In Table 4.2 several options are listed together with their advantages and disadvantages.

Gradient format	Advantages	Disadvantages
Analytic	Easy evaluation of gradient	Difficult to generate complex curves Difficult to parse function(s) in software
Interpolation between points	Convient to specify complex functions Interpolation is simple to implement in software	Requires the user to build a table
Individual ring specification	Direct gradient specification Easy to implement in software	Requires pre-computation of number if rings

Table 4.2: Advantages and disadvantages of gradient specification options.

While an *analytic specification* of the functional gradient allows for unlimited precision; such precision is not needed as described in Section 4.3. The difficulty of specifiying complex functions and the implementation to parse these function in software makes analytic gradient specification not a very attractive option. Using *point interpolation* specified by a table is a convienient way to specify and evaluate complex gradient functions. Implementing interpolation is simply done in the slicing engine. The only downside to this method is that it requires the user to build a table with function coordinates that should be passed to the slicing engine. Specifying the material ratio for each *individual ring* gives the user an option to directly specify the gradient without any calculation steps in between. This greatly simplifies the evaluation of the gradient in the software as values can be directly read from user input. It does however require a pre-computation of the number of rings needing a different material ratio. This number is not known until late in the slicing process. The slicing process will have to be paused at some point and has to prompt the user for the requested values of each ring.

Both the analytic and individual ring gradient specification options introduce unnecesarry difficulties for the user and the software implementation. Leaving interpolation between points as the best option in specifying the functional gradient. The user will have to create a table in a file that will be passed to the slicing engine.

4.3.3 Path Planning

An important part of printing a multi-material object with a continuous functional gradient is path planning. In 3D-printed objects printed by consumers the infill pattern is usually not very relevant except for the strength of the print. Rarely are printed objects completely filled to create a solid and some space is left inside to save material. Various infill patterns exist in most slicer software. Figure 4.3 shows some of the most common patterns.

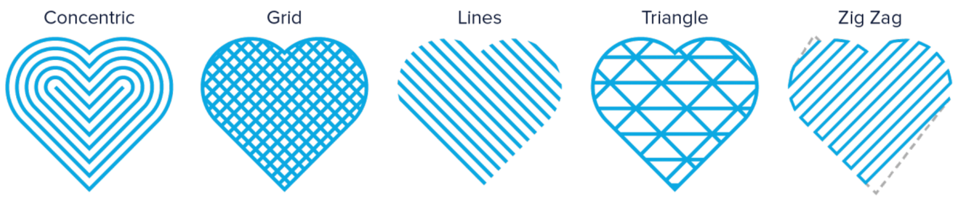


Figure 4.3: Common infill patterns used in slicer software.

When printing energetic materials, however, the infill pattern becomes quite important. Not because of the strength of the printed object, but because of the desired burn profile, which was described in Section 4.2. In Figure 4.4 an example of a star-grain rocket propellant with a single material is shown is shown where the rings burn from the inner ring to the outer ring.

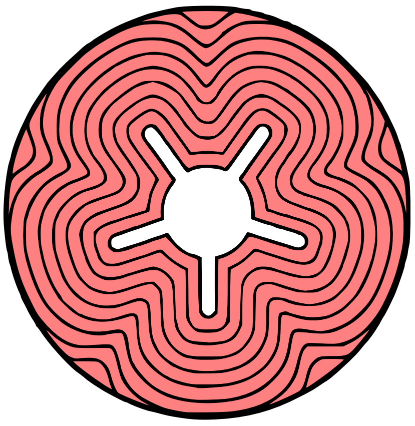


Figure 4.4: A star-grain propellant.

The concentric infill pattern shown in Figure 4.3 shows great similarities between the desired infill pattern shown in the star-grain in Figure 4.4. The difference, which is not visible here, is that the conventional concentric infill pattern adds concentric rings emerging from the entire shell. Meaning the infill pattern meets itself in the middle of the object. For grains that burn both from the inner and outer ring, such as in gun powder grains, this is desirable. In Figure 4.5 it is clearly visualized how the concentric infill pattern works. However, for grains such as the rocket propellant in Figure 4.4 a different inner structure is required.

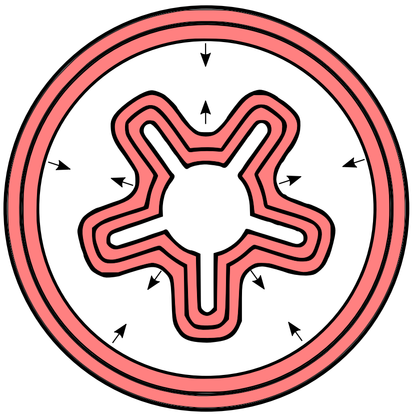


Figure 4.5: Concentric infill emerging from shell.

In Figure 4.6 the final result of the concentric infill is shown. Here the differences between the star-grain propellant shown in Figure 4.4 are clearly visible.

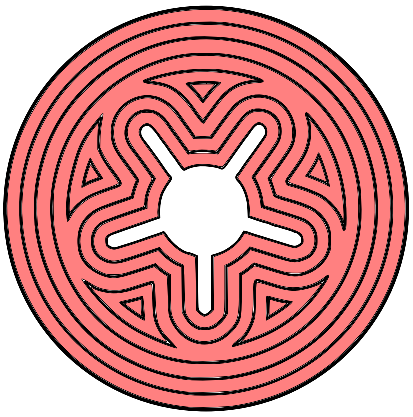


Figure 4.6: Final result of concentric infill pattern.

A new infill pattern will be designed to generate the required infill pattern in the star-grain propellant in Figure 4.4. The simplest solution to do this is to only add concentric lines emerging from the inner ring of the shell. This way the pattern of the cavity present in the middle of the model will emerge from the inner ring until it intersects with the outer ring of the model. This will result in the desired infill pattern shown in the star-grain propellant.

4.4 UV-lights

In order to cure energetic materials a UV-light is used that is mounted on the extruder near the print head. The UV-lights can be controlled with several G-Code commands defined in the PLCs that control the 3D-printer. It takes only a few seconds for the UV-lights to harden the energetic materials. During long travel moves the UV-light cures the material currently in the nozzle, even with the UV-shield, and causes it to become blocked. When this happens the print process will have to be restarted to clean the nozzle. A simple solution to prevent this from happening is turning off the UV-lights when an extruder is not extruding material. The user will be given the option to configure whether the UV-lights are turned off during travel moves. Certain print path types such as paths designed to increase build plate adhesion are not part of the model and the user should be able to specify if the UV-lights are to be used during these paths.

4.5 Multiple-Material Nozzles

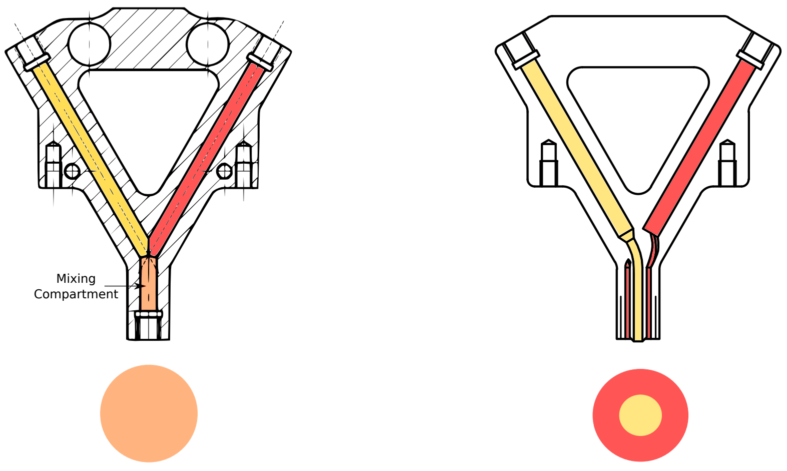
When mixing materials two different multiple-material nozzles are available: the Y-Nozzle, Figure 4.7a, and the Coaxial-Nozzle, Figure 4.7b. A cross section of the paths produced by the nozzles is shown at the bottom of the figures. The mixture or coaxial ratio can be controlled by the speed of the plungers when extruding. The Y-Nozzle is expected to suffer some complications by joining the channels, while the Coaxial-Nozzle channels are joined at the very end making complications less likely. The problems that could be introduced by the Y-Nozzle are:

- The shared mixing compartment introduces delay in command and execution
- When one plunger retracts it also affects pressure in the other channel
- Material from one channel could be pushed into the other channel

To some extent these effects could be compensated for in the software, but not completely. The first and most important issue is the shared mixing compartment delay. This could be compensated for by shifting the material switch command by a certain volume. In Figure 4.8 the delay is visualized.

A second problem that is introduced is a transition region of one material to an other material. The transition region is defined by the region where the material ratio changes from the previous ratio to the current desired ratio. This can only be solved by dumping the transition region somewhere on the side of the printbed. In Figure 4.9 the transition region is shown.

When using TNO's Y-Nozzle the most common path diameter of 1 mm is quite small compared to the volume of the mixing compartment: around 250



(a) Y-Nozzle. (b) Coaxial Nozzle.

Figure 4.7: Multiple material nozzles with cross section of its produced path.

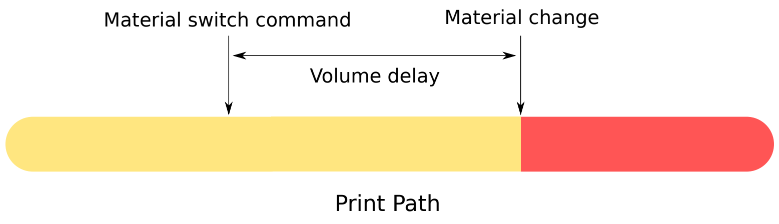


Figure 4.8: Volume delay caused by mixing compartment.

mm^3. A path length of at least 318 mm is needed to empty the mixing compartment assuming a constant path diameter of 1 mm. Paths are described in a layer, which consists of a list of points together with a material id for each path. Considering the small size of some of the printed objects a single layer could have less volume than the mixing compartment. Meaning the new location of the material switch, after compensation for the delay, is not in the same layer it started in and will require modifications of layers that are already in the process of being translated to G-Code. This introduces the need for a buffer that is used to shift material switches between layers. The calculation of the new material switch point has to be done after a layer plan has been generated as the volume is not known before a layer plan has been created. Layers have to be kept in the buffer long enough until it is certain it no longer has to be modified.

In the first layer, layer zero, the first extruder shift is shifted into the skirt or brim paths. These are concentric paths around the model to improve build plate adhesion and to prime the extrusion process. If these paths do not have a volume at least equal to the required shift volume the slicing process is terminated as it will result in incorrect prints. If a layer is empty nothing

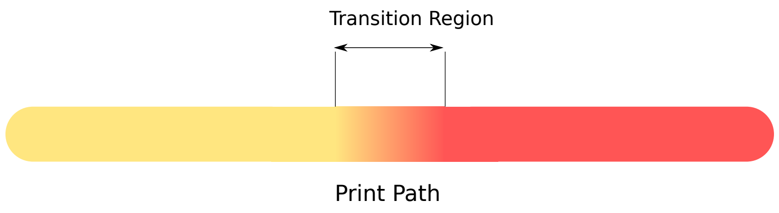


Figure 4.9: The transition region.

is done. Layers below the current layer are flushed since they will not receive new material switches anymore. This is guaranteed as the volume between two material switches will remain constant. Keeping layers in the buffer has a drawback of longer overall G-Code production time. Since print quality is more important than G-Code production speed this is not an issue.

4.6 Conclusion

Application of the functional gradient will be done during the slicing process. A table built by the user is used to specify the gradient and together with a new infill pattern the desired results can be produced: a 3D-printed multi-material model with a continuous functional gradient. In Figure 4.10 a star-grain rocket propellant with a gradient can be seen. Each ring emerging from the center has a distinct materal ratio. This is similar to the Bioinspired Mechanical Gradients used by Kokkinis et al. [3].

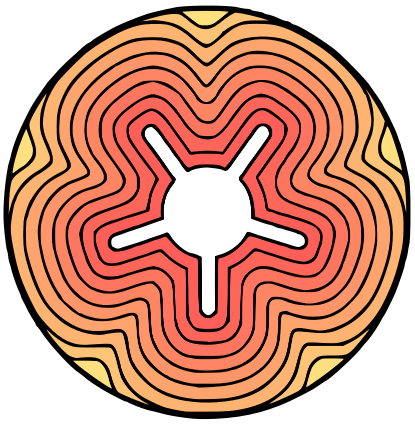


Figure 4.10: Star-grain rocket propellant with gradient.

Figure 4.11 shows a gun propellant with gradient. Both figures assume the Y-Nozzle is used. When printing with the Coaxial-Nozzle the gradient is less visible from the outside.

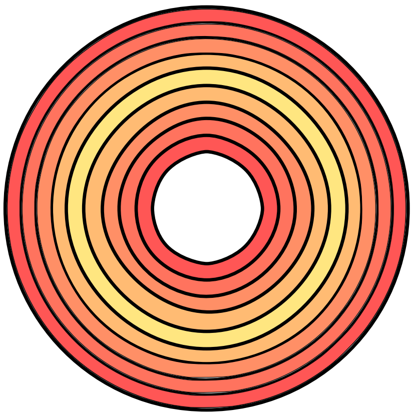


Figure 4.11: A gun propellant with gradient.

Chapter 5

Implementation

This chapter describes the implementation of the design presented in the previous chapter. Section 5.1 compares currently available slicers on various requirements, which is followed by Section 5.2 that describes the architecture of the selected slicer: Cura. Section 5.3 describes how Cura is modified to print continuous gradients. The implementation of support for different nozzles is described in Section 5.4. The required syntax changes for Beckhoff G-Code dialect and UV-light support are shown in Section 5.5 and Section 5.6 respectively.

5.1 Software Selection

To select the right software as a base for this thesis, a requirements table has been made to compare available choices.

Name	Open Source	Engine Language	GUI Language	Speed	Print Quality	Customization Degree
Cura	Yes	C++	Python	Fast	Good	Very High
Simplify3D	No	N/A	N/A	Very Fast	Good	High
Slic3r	Yes	C++	Perl	Slow	Good	High
Repetier	Yes	N/A	C++	N/A	N/A	Low

Table 5.1: Specs of slicer software candidates

The most popular available slicers are rated according to different criteria: whether it is open source, language of engine and GUI (Graphical User Interface), slicing speed, expected print quality and the current options available to customize the print.

Since Simplify3D is not open source, the use of this software for this thesis is very limited. The post processor built by TNO could be further extended but this would introduce unneccesary constraints.

Repetier is essentially host software that can use the slicing engine of other available slicers. It is fully written in C++, which is more convenient for programming than other GUI languages. However, since the native GUI of a slicing engine supports all of the available options of the engine instead of the generic options that Repetier offers; the native GUI with its engine is preferred. Repetier can be convenient when the printer is controlled from embedded devices such as an Arduino or a Raspberry Pi; with TNO's printer this is not the case.

The remaining two candidates are Cura and Slic3r, the biggest open source slicers currently available. While they are comparable in options for print customization; Cura is more actively developed by a larger community, which means more frequent software updates. This could, however, introduce more conflicts during development if the software must be kept up to date. The biggest difference is probably the language of the GUI, in which Cura has the advantage. Python is a more modern language than Perl, but this is also a personal preference. Both have had their source inspected, were ran from source and were tested for debugging to see if any problems arose regarding long term development. Since Cura has the ability to just build an executable, which was significantly faster than Slic3r build times, this gave a slight advantage. Debugging with Cura also was significantly easier by having a built-in option to connect a custom engine to the GUI with debugging enabled. Inspection of the source revealed nothing worth mentioning. When comparing slicing speed Cura is supposedly faster; while slicing speed is of less importance than print quality it is still favorable.

Overall Cura has preference as the base slicing software for this thesis: it is open source, actively developed, written in preferred languages, highly customizable, has good support for development, and faster slicing times.

5.2 Cura Architecture

This section describes the software architecture of Cura, the slicer software used as base for this thesis. For a detailed class description see Section 8.2.

5.2.1 Program Structure

Cura has a front end (the GUI), and a back end (the engine). When running Cura the GUI allows the user to configure options for the printer, the extruders and the slicing process. Models can be loaded into the workspace and can be moved around, scaled, and rotated. When the slice button is pressed all settings are converted to a string that also contains the file location of the model to be sliced. This string is sent to the engine where it is parsed. It is also possible to just use the slicing engine via a terminal, but building a settings string by hand is of course tiring work. All the settings are stored in the settings container system in the slicing engine, which will be discussed in Section 5.2.2. When the slicing process has finished, the engine sends the slicing data back to the front end where it is visualized. Figure 5.1 shows the general architecture.

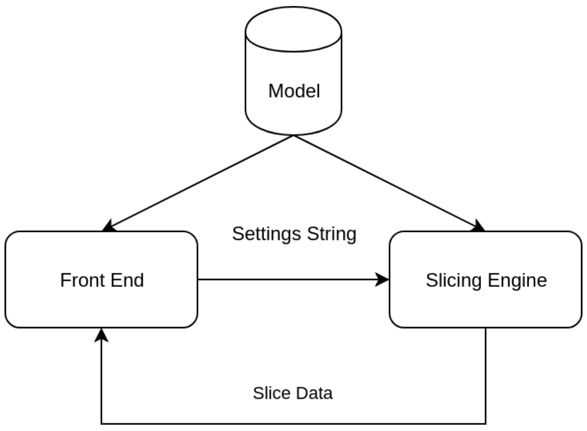


Figure 5.1: General overview of Cura

5.2.2 Settings

Before the actual slicing the settings string from the command line argument is parsed in the engine, also when using the GUI. Settings are stored in an unordered map with key value pairs and can be retrieved with various methods. All settings in the front end are defined in a JSON file, which is also read by the slicing engine to define variable types. When an unregistered setting is passed to the engine, meaning it is not defined in the JSON file that the engine reads, a warning is issued but the setting is still stored. A definition of the material-diameter variable is shown below.

```
"material_diameter":
{
    "label": "Diameter",
    "description": "Adjusts␣the␣diameter␣of␣the␣filament␣used.",
    "unit": "mm",
    "type": "float",
    "default_value": 2.85,
    "minimum_value": "0.0001",
    "minimum_value_warning": "0.4",
    "maximum_value_warning": "3.5",
    "enabled": "machine_gcode_flavor␣!=␣\"UltiGCode\"",
    "settable_per_mesh": false,
    "settable_per_extruder": true
}
```

The ”type” field is used to identify variable types and the ”unit” field is used for conversion when the variable is requested in a different unit than defined. The other fields are used by the front end. When new custom settings have to be added they can simply be appended to the JSON file.

5.3 Continuous Gradients

As discussed in Section 4.3 the gradients will be applied during the slicing process. To find a suitable moment in the slicing process a simplified version is briefly described in steps below. For a more detailed description see Section 8.1.

1. **Loading of 3D-models** The 3D input models are loaded and converted to a collection of points (vertices) in 3D space.
2. **Slicing** The 3D-model is intersected with horizontal planes to create layer outlines: the inner and outer walls, specified as polygon objects.
3. **Infill** The areas between the outline polygons are filled with a desired infill pattern.
4. **G-Code generation** All layers are converted to G-Code in the correct order (bottom to top).

Polygon objects are collections of points and can be modified using a clipper library. Since all paths in a layer will require a material ratio to define a gradient the best option is to apply the gradient during the infill step where all paths in a layer are created: step 3. Cura generates configurations for different path types (such as walls, infill, support etc.) before the start of step 3. These configurations are kept constant in a storage object and cannot be modified after they have been generated to avoid accidental changes during the remaining steps. Each path contains a reference to the storage object to define the path's configuration. Configurations contain important attributes of paths such as line width and print speed of the path. In Figure 5.2 the hierarchy of a layer is visualized.

Only two paths are shown in the figure. A layer can, however, contain an unlimited number of paths. Simply adding a material ratio to the existing path configurations when the configurations are normally generated is not an option. This would mean the configurations are kept constant while they do require changes later during the process when the ratios are known. Modifying the storage object with configurations to non-constant would require significant changes to the entire engine. Besides, each layer could require a different configuration if gradients in the vertical plane are used. The logical solution is adding a storage to each layer where configurations are defined that could be used instead of the standard storage object. In this way it can be modified when needed without having to make significant changes to the engine. It was chosen to create an entire configuration in a layer for a path with a specific ratio instead of just storing the ratio and using the conventional configuration storage object for the remaining attributes. This is done to allow future tweaks to other attributes besides material ratio as

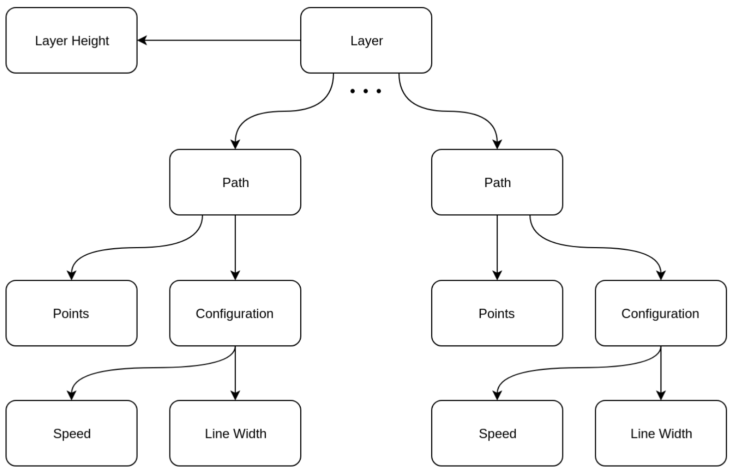


Figure 5.2: Hierarchy of a layer consisting of points. An arrow indicates "contains".

defined by the gradient. A negligible downside is more memory usage, but this is hardly a constraint.

Currently only the Concentric and Concentric Grain infill patterns can be used when printing gradients. While other infill patterns are not likely to be used a more generic implementation could be possible. By first creating all polygons including material ratios and sorting them in the desired order; then adding them to a layer plan would add support for any infill pattern.

5.3.1 Path Planning

A new infill pattern, Concentric Grain, was implemented to support a pattern that expands the shape of the cavity in the middle of the object until the walls are reached. Infill patterns are built with polygon objects. To manipulate polygons Cura uses a library, ClipperLib, that can perform various operations to create new polygons. It supports offsets and various boolean operations that together create the characteristic infill patterns shown in Figure 4.3. Figure 5.3 shows the four boolean operations used in ClipperLib and Figure 5.4 shows how a new polygon is generated with an offset.

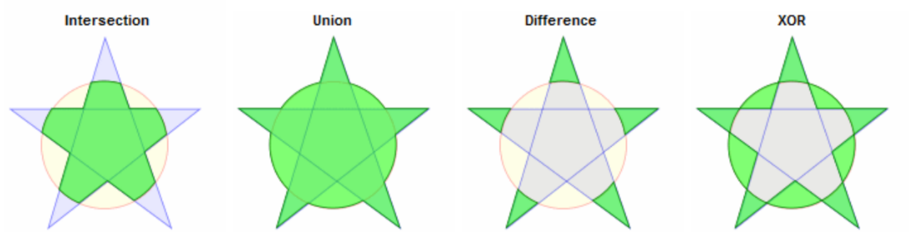


Figure 5.3: Boolean operations provided by ClipperLib library.

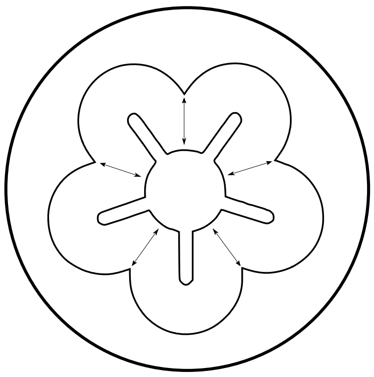


Figure 5.4: Creating a new polygon by applying an offset to the center cavity. Arrows indicate offset value.

To create the new desired infill pattern the offset value, which is dependent on the line width, is looped until it intersects with the outer wall. When a new polygon intersects with the outer wall the boolean intersection operator is used to cut the polygon at the edge and creates lines instead of polygons. In Figure 5.5 an example is shown where a polygon that was generated by an offset is cut to generate individual lines. The infill generator is terminated when a new offset polygon has zero points.

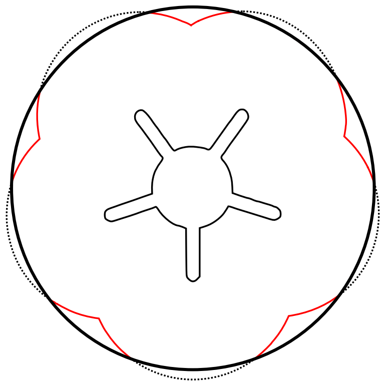
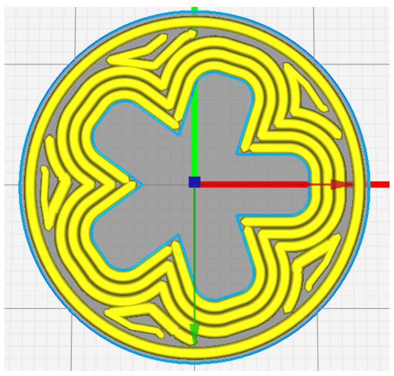


Figure 5.5: Lines generated by boolean intersection indicated in red.

Normally all polygons generated by infill are added to a layer through an optimizer in a single object. The optimizer tries to decrease the travel moves needed by using a traveling salesman computation to achieve the lowest print time. When printing a gradient, however, ideally all paths on a layer with the same material ratio are printed subsequently to reduce the amount of material ratio changes. A single infill object is still generated, but the object is split in different polygons based on material ratio. These polygons are then added separately to a layer to allow printing them in the desired order and to reduce the amount of material ratio changes while printing. While creating an infill object the infill generator counts the number of rings, which is used to create the gradient configuration in a later step.

Cura by default attempts to merge infill lines, not polygons, when points are close together to reduce unecessary point density. However, the algorithm often produces erratic results and for an unknown reason is not applied to polygons regardless of their point density. An option to disable infill line merge has been added that shows improved print quality when the merging has been turned off. Since polygons with equal point density apparently do not require merging of lines it is not expected that disabling the infill line merger will not negatively impact print quality. Figure 5.6 shows the implemented Concentric Grain infill pattern together with the impact of disabling the infill line merger.


(a) Infill line merge enabled.

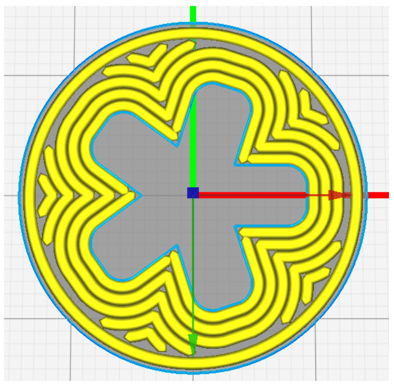

(b) Infill line merge disabled.

Figure 5.6: The concentric grain infill pattern

5.3.2 Gradient Specification

To pass the gradient to the slicing engine a function to read a text file with sample coordinates (consisting of x, y pairs) was implemented. It uses a straightforward algorithm that parses all coordinates until an end-of-file is reached and places all coordinates in an ordered map. It then applies piecewise linear interpolation to generate a gradient curve. All coordinate pairs have to be normalized; meaning the key and value pairs are doubles ranging from 0 to 1. With 0 being the ratio of the inner wall and 1 being the ratio of the outer wall. In the infill generator the normalized key of each ring is calculated based on the total ring count, which also includes the walls. This way the gradient can be mapped to the rings by iterating over the map until a key greater than the key of a specific ring is found. Linear interpolation is then used between the current and previous key value pairs to obtain the material ratio of the ring. Figure 5.7 shows the result of piecewise linear interpolation of a square root function with five samples used as gradient specification. Figure 5.8 shows the interface of Cura after a model has been sliced with the specified square root gradient applied.

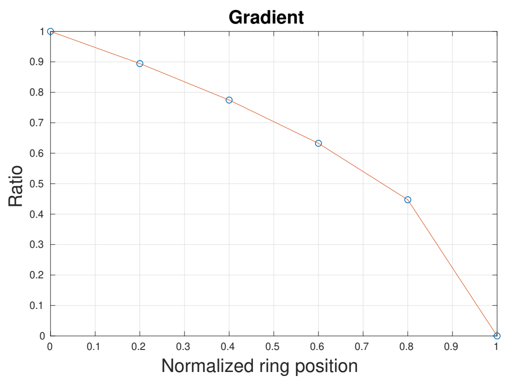
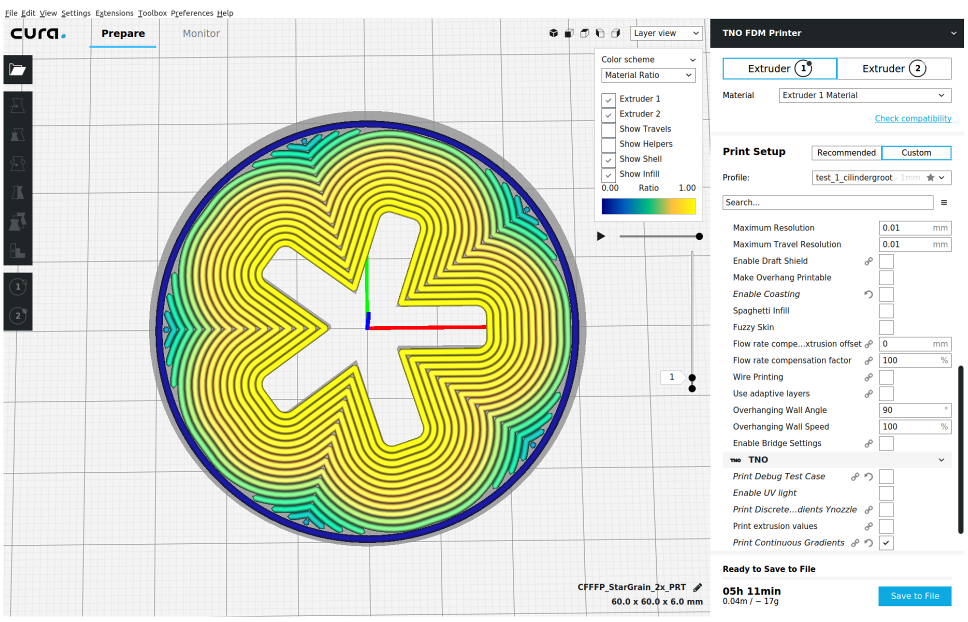


Figure 5.7: Piecewise linear interpolation between points in ordered map.



Color scheme: Material Ratio
✓ Extruder 1
✓ Extruder 2
Show Travels
Show Helpers
✓ Show Shell
✓ Show Infill
0.00 Ratio 1.00

Material: Extruder 1 Material

Print Setup
Profile: test_1_cilindergroot - 1mm

Maximum Resolution	0.01 mm
Maximum Travel Resolution	0.01 mm
Enable Draft Shield	
Make Overhang Printable	
Enable Coasting	
Spaghetti Infill	
Fuzzy Skin	
Flow rate compe...xtrusion offset	0 mm
Flow rate compensation factor	100 %
Wire Printing	
Use adaptive layers	
Overhanging Wall Angle	90 °
Overhanging Wall Speed	100 %
Enable Bridge Settings	
TNO	
Print Debug Test Case	
Enable UV light	
Print Discrete...dients Ynozzle	
Print extrusion values	
Print Continuous Gradients	✓

Ready to Save to File
05h 11min
0.04m / ~ 17g

CFFFP_StarGrain_2x_PRT
60.0 x 60.0 x 6.0 mm

Figure 5.8: Interface of Cura with the piecewise linear gradient shown in Figure 5.7 applied.

5.4 Multiple Material Nozzles

To implement the shifting of material switches discussed in Section 4.5 a buffer is needed to buffer layers. Cura has a built-in buffer that is used to allow temperature command insertions in earlier layers to heat up extruders in advance for use in later layers. It can also be used for the material switch shift without interfering with temperature insertions even though temperature commands are are not implemented on the TNO printer. Figure 5.9 shows a flow diagram of the behaviour of the buffer.

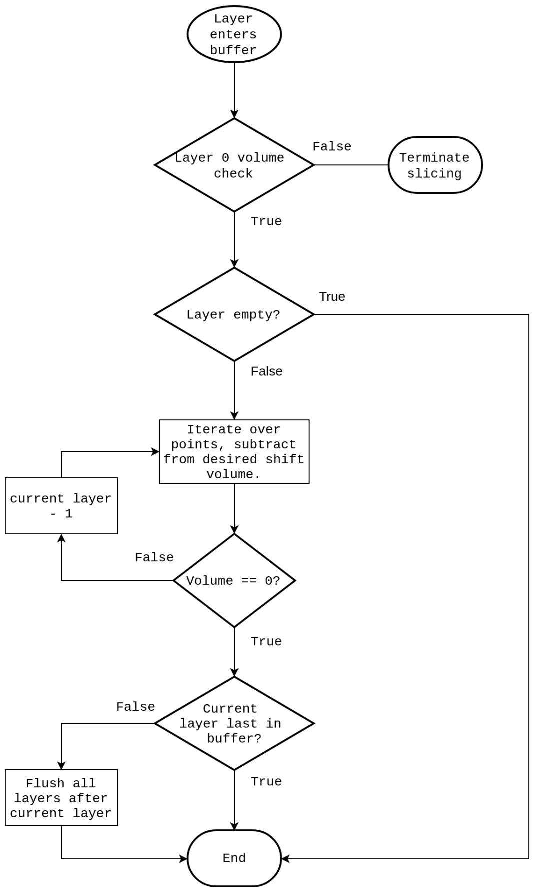


Figure 5.9: Flowchart of layer buffer.

If the skirt or brim paths in layer zero do not have enough volume the slicing process is terminated and a log error is given. The material switch shift calculation starts in layer zero and iterates over all layers from bottom to top. When starting the calculation in a layer it iterates over all points in a layer and subtracts the volume between points from the desired shift volume until it reaches zero. If the volume has not reached zero in a layer the subtraction continues in the layers below.

5.5 Beckhoff Implementation

To support the Beckhoff G-Code dialect used by TNOs printer the output syntax of the produced G-Code will have to be modified. In Listings 5.2 and 5.1 a comparison of Beckhoff dialect with the most common dialect Marlin is shown.

Listing 5.1: Marlin dialect.

```
;LAYER_COUNT:4
;LAYER:0
G1 F300 Z2
G92 E0
T1
M107
G0 F600 X5.071 Y-18.803 Z2
;TYPE:SKIRT
G1 F300 Z1
G1 F360 X6.905 Y-18.21 E0.00489
G1 X7.611 Y-17.927 E0.00682
G1 X9.361 Y-17.079 E0.01176
```

Listing 5.2: Beckhoff dialect.

```
(LAYER_COUNT:4)
(LAYER:0)
G1 F300 Z2
M31 @714
D02
G1 X5.071 Y-18.803 Z2 F600
(TYPE:SKIRT)
G1 F300 Z1
G1 X6.905 Y-18.21 Z1 Q2=0.00489 F360
G1 X7.611 Y-17.927 Z1 Q2=0.00682 F360
G1 X9.361 Y-17.079 Z1 Q2=0.01176 F360
```

An overview of the most important differences is shown in Table 5.2. Some commands are custom defined for this 3D printer and not part of the standard Beckhoff dialect.

Command	Marlin	Beckhoff
Comment style	;	()
Absolute coordinates	M82	G90
Relative coordinates	M83	G91
Extruder reset	G92 E*X*	M3*X* @714
Tool select	T*XX*	D*XX*
Fan speed	M107	N/A
Temperature	M104 T*X* S*XXX*	N/A
Extruder symbol	E	Q=
Travel move	G0	G1/G0
Move speed	F	F

Table 5.2: Syntax comparison.

Multiple differences can be spotted in the two dialects: comment style, different symbols and syntax of move commands. Besides different symbols an extrusion or travel move in Beckhoff always requires a speed value at the end of the command. While in Marlin it can be provided once following the initial command after which it will be used until specified otherwise.

5.6 UV-lights Implementation

In Table 5.3 an overview of the commands can be seen. When the printer is used in the double straight configuration, meaning parallel extruders, one UV-light is used for each extruder. The lock intensity commands will set the intensity of the corresponding UV-light specified in the R value. The intensity can be changed at anytime during the print process.

Command	Extruder A	Extruder B
Turn on UV-light	M3	M8
Turn off UV-light	M5	M10
Intensity value	R2=XXX	R3=XXX
Lock intensity	M4	M9

Table 5.3: UV-light G-Code commands.

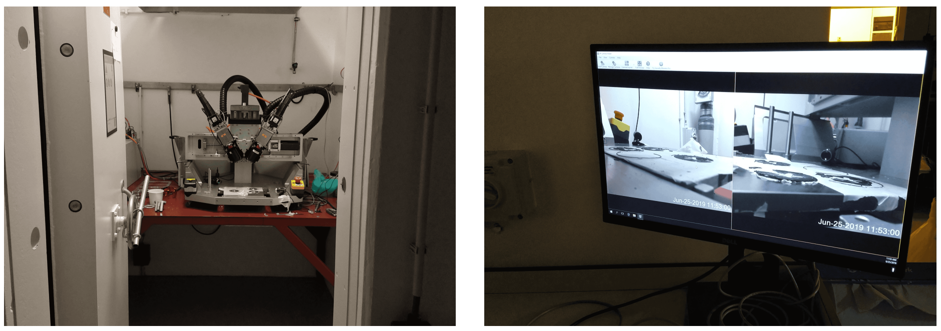
Chapter 6

Experimental Results

This chapter describes the experiments performed with the modified version of Cura. First a brief description is given of the experimental setup in Section 6.1. Followed by two sections that discuss several experiments using play-doh (Section 6.2 and Section 6.3). And finally Section 6.4 discusses the printing of energetic materials. It is recommended to read the terminology in Section 3.2 before reading this chapter.

6.1 Experimental Setup

The printer is located in a bunker for safety reasons, which has to be closed when printing energetic materials. This complicates closely observing the behaviour when printing, which is why two cameras are setup next to the printer that also record the experiments. Figure 6.1 shows the printer and the screens to observe the printer when the bunker door is closed.



(a) The printer located in the bunker. (b) The screen with the camera views that observe the printer room.

Figure 6.1: Experimental setup when printing.

6.2 Discrete Gradient

The first experiments were done to test the printer behaviour with the new software. Instead of energetic materials play-doh was used to be able to observe the printer without having to close the bunker door. The dual-straight printer configuration was used for these experiments. Two concentric cylinders with different materials were printed. Figure 6.2 shows the cylinders in the Cura interface. In the final result the red cylinder should completely encase the yellow cylinder, however, in this model the top lid was removed to show both cylinders. These cylinders should eventually be printed with energetic materials where the red cylinder consists of slow burning material and the yellow cylinder consists of fast burning material.

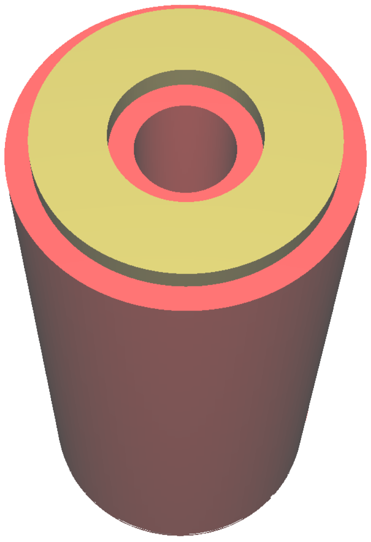


Figure 6.2: 3D-model in Cura of two concentric cylinders consisting of different materials.

These cylinders were printed with the settings shown in Table 6.1.

Line Width	Layer Height	Print Speed	Infill Pattern
1 mm	1 mm	6 mm/s	Concentric

Table 6.1: Print settings of first experiment.

Figure 6.3 below shows the print result of the two cylinders. It can be seen that it is quite messy because of over-extrustion, which is when an extruder extrudes too much material at a certain region. These regions are indicated in the figure; the number indicates the origin, which are explained in a numbered list. Cura alternates the start extruder in each layer to minimize the number of material switches, which is why in some layers the infill seeps through the walls.

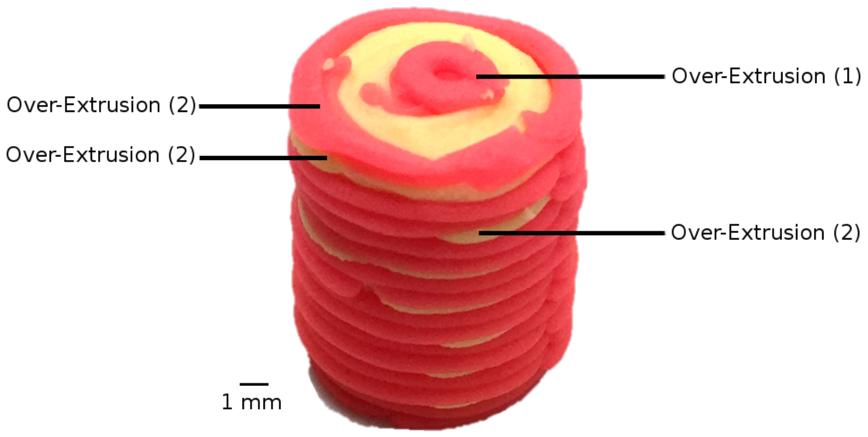


Figure 6.3: Print result of the two concentric cylinders.

Over-extrusion happens with both extruders and both materials. It was observed that over-extrusion happens at two points during printing:

1. At the end of a print path
2. Stacked paths in infill and walls

The first problem, which is called oozing in 3D-printing jargon, happens because the built-up pressure inside the extruder cannot be changed instantaneously and decreases gradually at the end of a print path. This causes some of the material to be pushed out; creating an unwanted blob of material. Cura, and all other slicer software, has a solution for this problem called *Coasting*. It replaces the end of a print path with a small travel move to use oozing to print the last part of a print path. Figure 6.4 shows a print path with coasting. To find the optimal coasting value (specified in mm) it should be gradually increased until the blobs no longer occur at the end of print paths, but not too far to cause under-extrusion.

The second problem is specific to Cura. Cura has problems with an uneven number of lines in both the walls and the standard *Concentric* infill pattern (see Figure 4.5). The walls are always generated in pairs, which results in a minimum of 2 lines for the inner ring and 2 lines for the outer ring. The red cylinder in Figure 6.2 that forms the walls has a wall thickness of 1 mm. Cura will, however, try to fit two lines in the 1 mm of space available causing over-extrusion. Cura has an option *Compensate Wall Overlaps* that tries to reduce the amount of extrusion when printing a wall if a wall is already in place, but does not completely remove them. However, combined with an experimental feature *Minimum Wall Flow*, which removes all wall lines

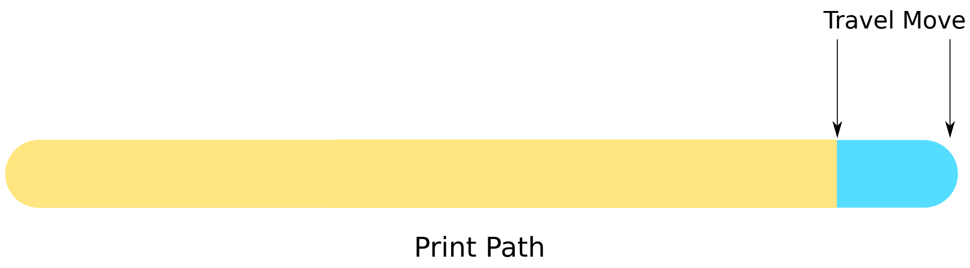


Figure 6.4: Coasting replaces end of a path with a travel move (in blue).

below a certain flow (percentage that is multiplied with extrusion values thus creating thinner lines) amount, the stacked wall problem is solved. To solve the stacking of lines during infill, the new *Concentric Grain* pattern is used. The pattern does not generate infill lines in pairs and thus is able to create an infill pattern with an uneven number of lines. Because the wall overlap compensation options in Cura already solve the wall stacking problem it was chosen to keep the current wall generation algorithm. A new print was made using the revised settings listed in Table 6.2. A minimum wall flow of 100% removes all walls that are not extruded at full volume. That means when a wall is already in place and a second wall is placed at the same location; the second wall has its flow reduced significantly and is removed as the flow is not at least the minimum 100%.

Line Width	Layer Height	Print Speed	Infill Pattern	Minimum Wall Flow	Coasting
1 mm	1 mm	6 mm/s	Concentric Grain	100%	1.5 mm^3

Table 6.2: Print settings of second experiment.

Figure 6.5 shows the result of the second experiment with the new settings.

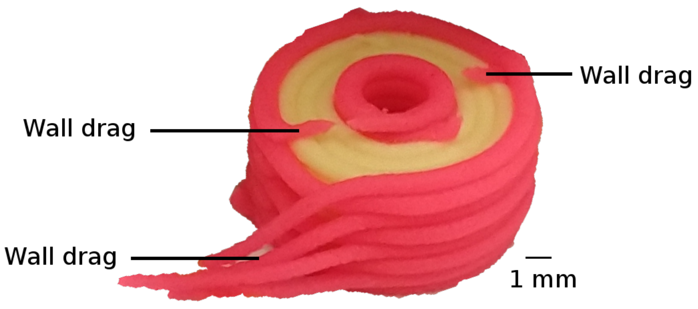


Figure 6.5: Second print with revised settings in Table 6.2.

The new settings have greatly reduced the over-extrusion problem. However, a new issue arises where the walls are dragged next to the model as can be seen in Figure 6.5. When an extruder switch occurs, which is sometimes at the end of a wall line, the extruder drags a part of the wall with it. To compensate for this effect Cura has an option called *Outer Wall Wipe*, which adds a small travel move along the wall after the end of the wall. Figure 6.6 shows how outer wall wipe works. The arrows indicate the added travel move after a wall has been printed. Too long wall wipes introduce the risk of over-extrusion, however, if the coasting volume is correctly configured this is not an issue.

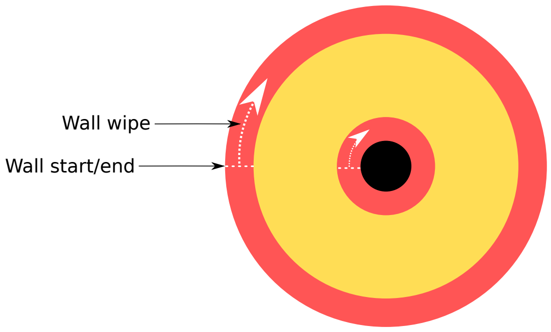
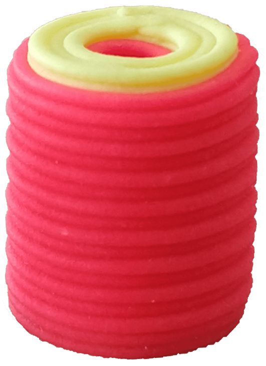
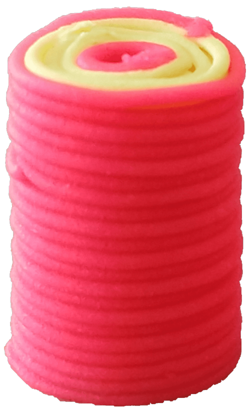


Figure 6.6: Adding a travel move with outer wall wipe.

A new print was made using the settings in Table 6.2 including the outer wall wipe option set to 2 mm, which resulted in the print shown in Figure 6.7a.



(a) Coasting compensation enabled.



(b) Coasting compensation disabled.

Figure 6.7: Two prints with wall wipe.

The outer wall wipe option solved the wall dragging problem completely and produced an acceptable print result. However, by closer inspection it was observed that the line thickness increased quite a bit at various points in the print. After code inspection it became clear Cura tries to extrude slightly more material after a coasting move to compensate for the material oozed during coasting. This causes over-extrusion after each coasting move in the print; creating much thicker lines than the expected 1 mm thickness. Figure 6.7b shows the print result with the coasting compensation option disabled. The print in Figure 6.7a was stopped earlier than the print in Figure 6.7b, which is why the yellow lines stick out of the print in the left figure.

6.3 Continuous Gradient

In this section the experiments to print a continuous gradient are discussed. The printer is used in the Y-configuration with the coaxial nozzle (see Figure 4.7b) and play-doh as print material. A star-grain is used as input model (See Figure 4.1) with a continuous gradient applied. Figure 6.8 shows the input model after it has been sliced in the Cura interface.

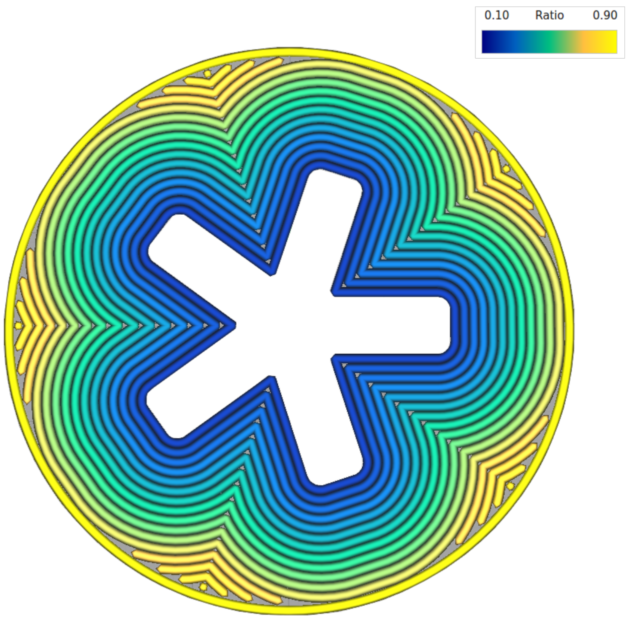


Figure 6.8: Sliced star-grain model with an applied gradient.

The *Concentric Grain* infill pattern was used to produce the desired star-grain infill. The inner wall material ratio is 0.10 and is expanded linearly with each ring until the outer wall, which has a ratio of 0.90. The settings used in the print are shown in Table 6.3. Apparently the coaxial nozzle suffers more from oozing, which is why the coasting value is increased compared to previous prints. Besides increasing coasting the nozzle tip was lifted by 0.2 mm to prevent the sticking of play-doh to the nozzle, which the coaxial nozzle tip suffers from to a greater extent than the previously used nozzle tips. This slight height increase does introduce a brief delay when starting a print path, which results in bigger voids. The end of a print path is expected to have slightly bigger blobs because of the increased nozzle tip height. The print result is shown in Figure 6.9.

Line Width	Layer Height	Print Speed	Retraction Distance	Coasting
1 mm	1 mm	6 mm/s	1.5 mm	3 mm^3

Table 6.3: Settings used in first continuous gradient test.

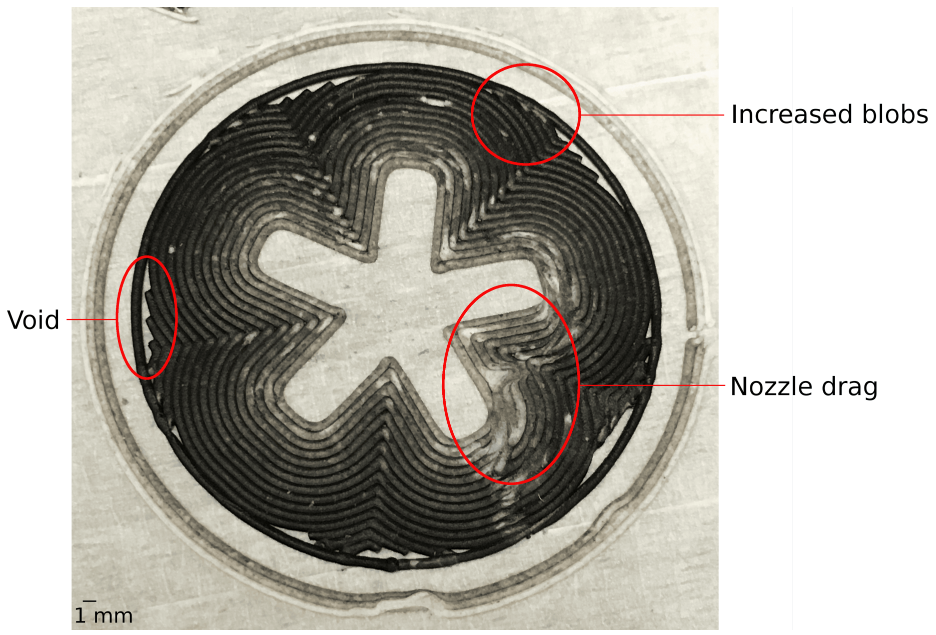


Figure 6.9: Star-grain with applied gradient; printed with the coaxial nozzle.

The voids caused by the delay at the beginning of a print path and the increased blob size at the end of a print path are indicated. Unfortunately some material stuck to the nozzle during printing and dragged some paths with it, which is also indicated in the figure. The nozzle tip size should be decreased to prevent the dragging of paths caused by material sticking to the tip. Reduced nozzle tip size would also mean the height of the tip can be lowered to prevent increased voids and blobs. Since the print was made with the coaxial nozzle the gradient is less visible from the outside; a cross section was made to inspect the material ratio. Figure 6.10 shows the cross section of the print with a clearly visible gradient.

In the cross section a gradient can be clearly seen where the ratio of the core (white) and the mantle (black) of the coaxial paths gradually change from the inner ring to the outer ring. The print was cut after the material

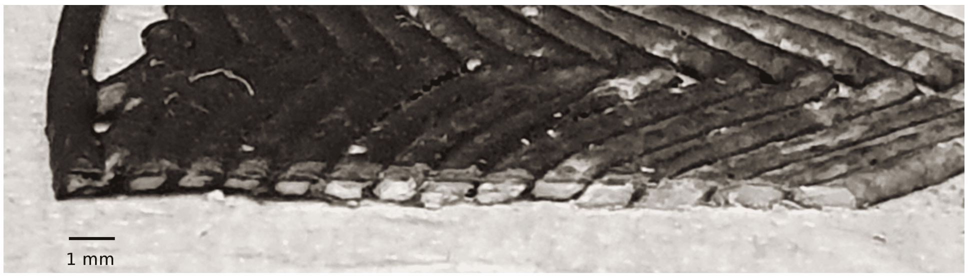


Figure 6.10: Visible gradient in cross section of print.

had dried for 24 hours, which decreased path width by water evaporation and flattening of the lines caused by sagging of the material. If this experiment can be performed with energetic materials the produced star-grain print can be stacked to create solid rocket fuel where the gradient produces a burn profile with either increasing of decreasing thrust, which depends on the material properties. The next section discusses the printing of energetic materials.

6.4 Printing Energetic Materials

After the play-doh experiments to characterize the printer behaviour new experiments were performed with a single energetic material. The first energetic material consisted of: 70% RDX (explosive powder used in C4), 10% energetic plasticizer (decreases viscosity) and 20% acrylate (the UV-curable material). The mixture is a shear thinning material, which means it behaves in a non-Newtonian way and its viscosity decreases under stress. This could result in unpredictable behaviour when printing. No UV-lights were used during this experiment. The model seen in Figure 6.2 was used as input. The first print result can be seen in Figure 6.11.

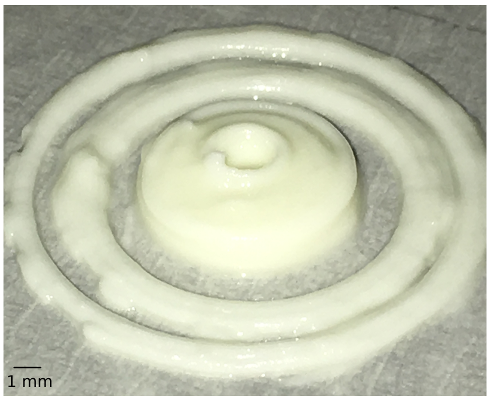


Figure 6.11: Concentric cylinders with skirt surrounding the print.

The rings around the center cylinder form the skirt that is used to prime the extruders. It can be seen that it is quite liquid compared to the play-doh material used in the previous experiments. The printer settings will have to be tuned accordingly to print with these different material properties.

To tune the settings a test setup was implemented that can be seen in Figure 6.12. The test setup overrides the slicing process and prints lines with increasing travel moves between them (indicated in blue), which can be used to determine the retraction or coasting settings. The travel move size can be freely configured from the interface; during these experiments the travel move distance starts at 1 mm and increases by 2mm with each line up to 19 mm. By disabling retraction the travel moves will act as coasting distance and the coasting settings can be determined. The four lines at the bottom are used to prime the extruder.

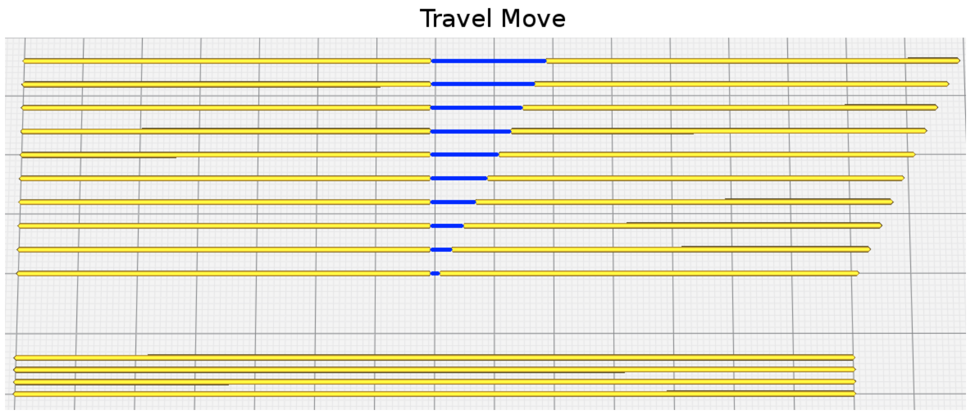


Figure 6.12: Test setup used to determine settings for new materials.

During the following experiments several parameters were investigated:

Retraction distance The retraction distance determines the total pressure relief in the nozzle during retraction. With values too low oozing is expected. Retraction distance probably has little impact on the blob size at the end of a print path, which is expected to be determined by retraction speed and coasting distance.

Retraction speed A higher retraction speed results in faster relief of pressure in the nozzle when the plunger retracts. A faster retraction speed probably results in smaller blobs at the end of a print path.

Coasting volume A higher coasting volume is expected to reduce the blob size at the end of a print path. It has no impact on the blobs at the beginning of print paths.

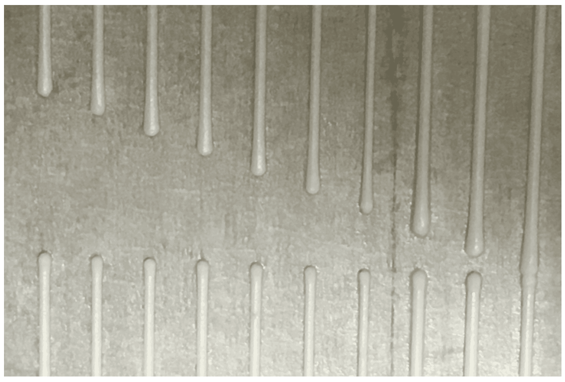
UV-curing Curing could have impact on the line width. If unwanted blobs are cured this could cause collisions with the nozzle.

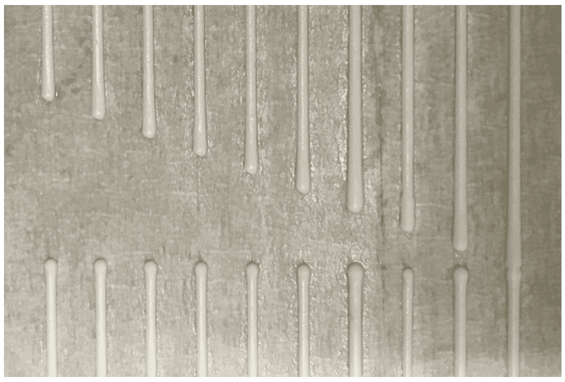
6.4.1 Retraction Distance

In the first few tests the effect of retraction distance was tested. The print settings used can be seen in Table 6.4. Figure 6.13 shows three prints with increasing retraction distance to determine its impact on blobs.

Line Width	Layer Height	Print Speed	Retraction Speed	Coasting
1 mm	1 mm	6 mm/s	1 mm/s	1 mm^3

Table 6.4: Settings used in retraction distance test.


(c) 2 mm retraction


(b) 1 mm retraction

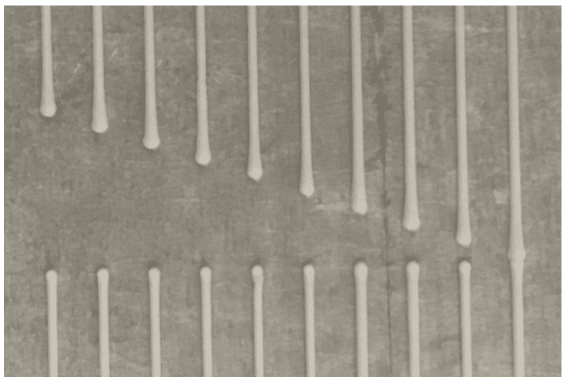

(a) 0.5 mm retraction

Figure 6.13: Prints with increasing retraction distance.

The impact of the retraction distance is minimal in the prints. While blobs at the start of a print path are slightly reduced by increasing the retraction distance to 1 mm; increasing it further yields no visible results except for longer retraction times. The inconsistent line thickness seen at random points in the prints could be the result of the plunger exerting force on the material and causing a temporary decrease in viscosity.

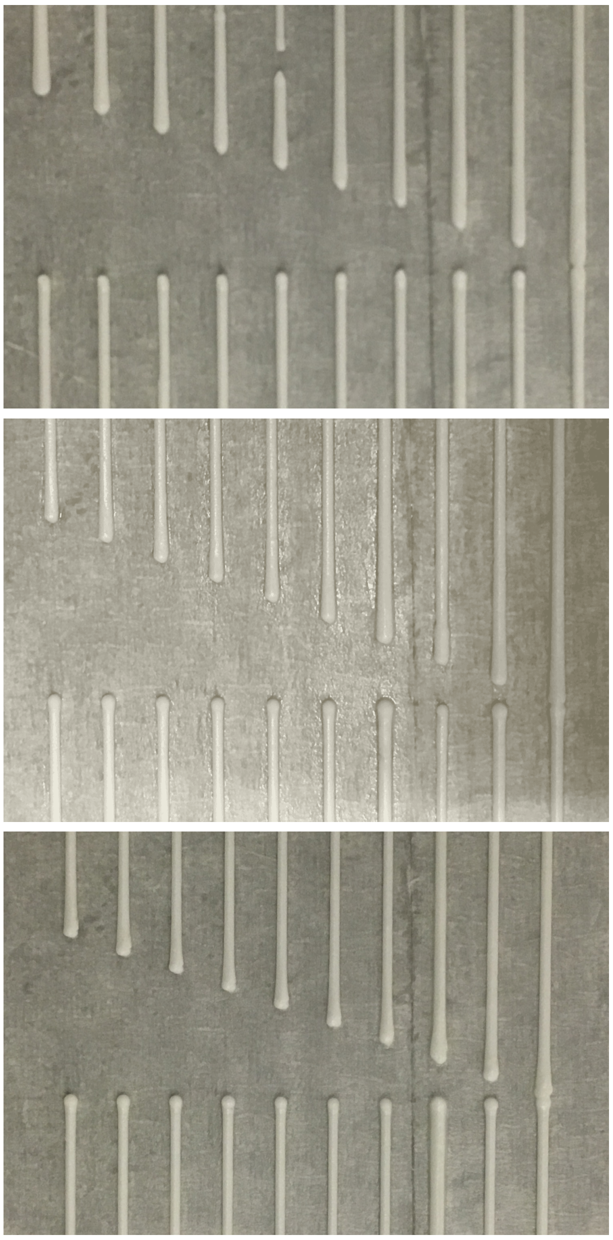
6.4.2 Retraction Speed

In a second experiment the effect of the retraction speed was researched to see wheter this has any influence on blobs and varying line thickness. Table 6.5 shows the settings used, and Figure 6.14 shows the results of the experiment.

Line Width	Layer Height	Print Speed	Retraction Distance	Coasting
1 mm	1 mm	6 mm/s	1 mm	1 mm^3

Table 6.5: Settings used in retraction speed test.

While little difference can be spotted in the experiments, increasing the retraction speed to 2 mm/s does seem to decrease the blob size at the end of a print path. Inconsistent line thickness is still present in all three prints and even a consistent difference in line thickness can be seen before and after travel moves in Figure 6.14c. An increased retraction speed will result in faster pressure drop when retracting at the end of a print path. This could, however, cause a higher decrease in viscosity and produce thicker lines when printing the next line.


(a) 0.5 mm/s retraction speed.

(b) 1 mm/s retraction speed.

(c) 2 mm/s retraction speed.

Figure 6.14: Prints with increasing retraction speed.

6.4.3 Coasting

To determine the coasting settings, retraction was disabled and the test print was executed. This way the travel moves indicated in blue in Figure 6.12 act as coasting distance. In theory a single line followed by a long travel move should be enough to determine the coasting volume, however, the previous experiments showed random variation in line thickness, which is why it was chosen to print multiple lines. The extra line at the top of the figure is followed by a long travel move. Figure 6.15 shows the result of the coasting test print.

Line Width	Layer Height	Print Speed
1 mm	1 mm	6 mm/s

Table 6.6: Settings used in coasting test.

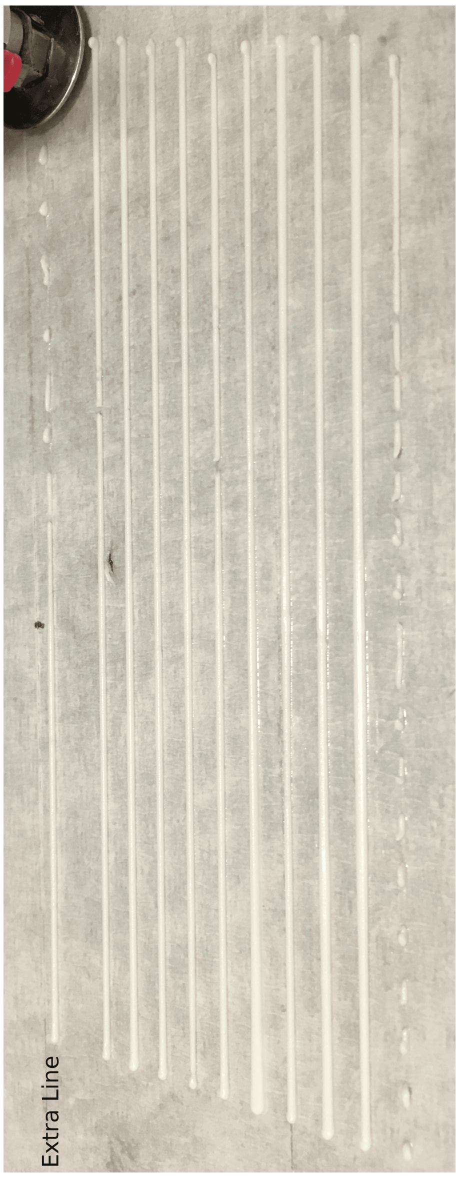


Figure 6.15: Test print to determine coasting settings.

Apparently the material has such low viscosity that, in every case except the upper line, each travel move is filled by oozing. Line thickness does usually decrease during each travel move and increases with a delay after extrusion has started again. Random variation in line thickness is also still present in several lines. It is difficult to determine a suitable coasting distance after this experiment as no clear boundary is seen on any of the lines. For an unknown reason the extra line at the top starts thinning and producing interrupted blobs much sooner than any of the previous lines.

While the goal of the experiment was to determine the coasting settings, it resulted in little relevant results. Using a different test setup in the future where multiple lines with the same coasting value are printed would probably be a more robust way to determine coasting settings. Figure 6.16 shows a different setup. Repeating this setup while gradually increasing coasting distance is a more reliable way of determining the correct settings.

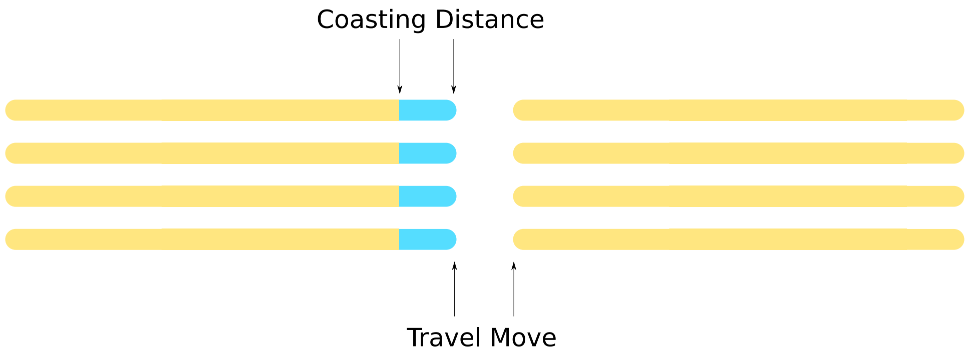


Figure 6.16: Multiple lines with same coasting distance, in case of random line thickness variation.

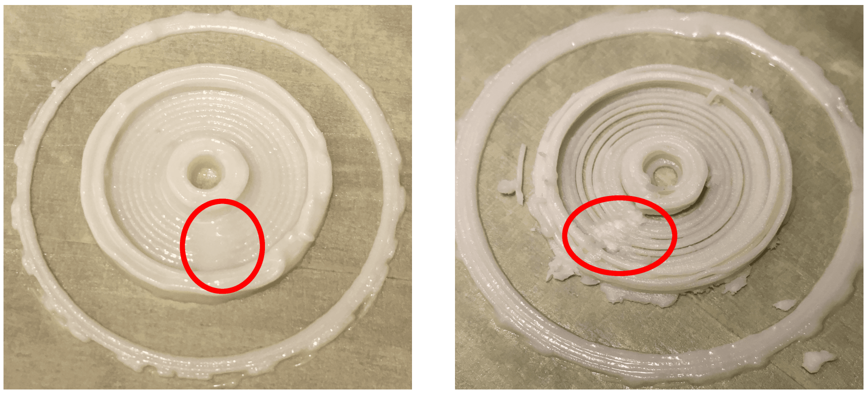
6.4.4 UV-Curing

The behaviour of the material with UV-curing was researched in an experiment where the cylinders in Figure 6.2 were scaled to three times their original size. The scaling reduces the relative size of the blobs compared to the total print to see wheter an acceptable result could be printed of this size. Table 6.7 shows the settings used and Figure 6.17 shows the print results.

Line Width	Layer Height	Print Speed	Retraction Distance	Retraction Speed	Coasting	UV Intensity
1 mm	1 mm	6 mm/s	1 mm	1 mm/s	2 mm^3	30%

Table 6.7: Settings used in UV experiment.

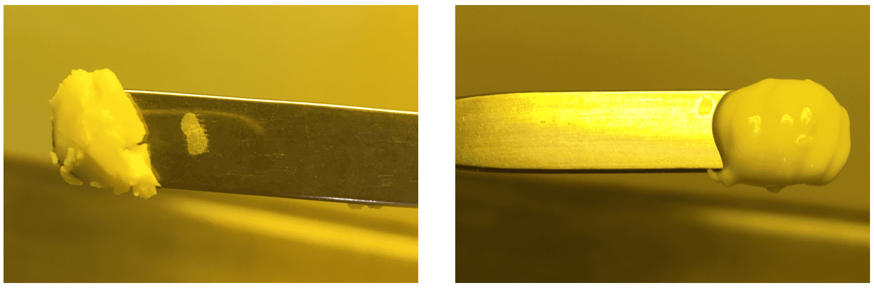
The location of the blobs is indicated in both subfigures with a red circle. The blobs are the same height or higher than the nozzle tip because the


(a) Print without using UV-lights.

(b) Print with UV-lights at 30% intensity.

Figure 6.17: Prints with and without UV-curing.

printhead is lifted by a Z-hop at the end of a line. If the nozzle tip collides with cured material it can cause small pieces to break and scatter across the print as seen in Figure 6.17b. Curing also significantly reduces line width, which decreases the density of the print and reveals the build plate below. Such significant shrinking has not been observed in earlier tests. Demixing of the material in the syringe is likely causing a higher ratio of acrylates when it exits the nozzle, which could cause more shrinkage when curing. Figure 6.18 shows a comparison of the material before it is inserted in the syringe and after it has been used in a print.


(a) Material before insertion into syringe.

(b) Material deposited by the printer.

Figure 6.18: Comparison of print material before and after printing.

A clear difference can be seen, which is caused by the demixing of the material in the syringe. This results in blobs and significant shrinkage making it very difficult to produce a proper print with this material. A new mixture of materials will have to be designed that suffers less from demixing in the syringes.

6.5 Conclusion

The performed experiments show that printer settings can compensate for undesired behaviour of the material to some extent, however, a proper material is crucial to produce a sufficient quality print result. Printing with energetic materials resulted in unexpected behaviour, which is probably due to the sheer thinning material properties. Prints showed random variation in line thickness, numerous blobs, and demixing in the syringe. When the energetic material was cured with UV-light it showed significant shrinkage of line thickness, which is a result of the demixing in the syringe. The experiments performed with play-doh revealed that, after tuning the printer settings to the material, it is possible to print a multi-material 3D-model with a continuous gradient.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

The goal of this thesis was to research the 3D-printing of energetic materials using TNO's printer hardware that is able to print a mix of two materials. The ability to combine multiple materials into a continuous gradient allows for new performance optimizations of gun and rocket propellants.

A design that can apply a gradient to an input CAD model was presented, which is a modified version of the Cura slicer software. It can slice a CAD model to create printer instructions and ultimately produce a multi-material object with a continuous gradient. The gradient is applied during the slicing process and requires a gradient specification in a text file with function coordinates. These function coordinates are mapped to the 3D-model using piecewise linear interpolation. To support the printing of rocket propellants that burn from the inner to the outer ring the *Concentric Grain* infill pattern was designed, which enables printing of the characteristic star-grain propellant.

The design was first tested by printing with play-doh, which allowed printing with close observation without having to close the bunker door for safety. These experiments showed that the design is able to produce discrete and continuous prints with proper tuning of the slicing settings.

Printing with energetic materials proved to be more challenging. The mixture used as print material behaves in such a different way than play-doh that the settings could not be properly tuned to create a proper print result. Random variation in line thickness was observed in several experiments as well as blobs at the beginning and ends of print paths. Moreover UV-curing showed that the material shrinks significantly after it has been cured, which is the result of demixing of the mixture in the syringe. The demixing causes a higher ratio of acrylates, which is the UV-curable material, to be deposited by the printer. These acrylates shrink after curing and result in thinner print paths.

In conclusion, printing a multi-material object with a continuous gradient is certainly possible with the proper materials.

7.2 Future Work

A more user friendly approach could be built in future work. Instead of specifying a gradient in a text file the gradient could be added in a 3D-CAD model with CAD design software. This gives the user better design tools to produce a desired print. The slicing software will have to be modified to accept the new 3D-CAD model format with material properties. The CAD design software could be further extended with with an option to specify a desired burn profile such that it automatically calculates and applies a gradient on a 3D-model.

Little work has been done in this thesis on path optimization besides creating a new infill pattern. Various path optimization techniques exist as shown in Section 2.2, which could be applied on the current design to improve print quality. Void reduction optimizations could be especially helpful as voids were still present in the star-grain print in Figure 6.9.

Currently no quality metric exists to measure the gradient quality. Some sort of way to determine wheter a print result is sufficient could be designed. The quality of a print is mostly determined by voids between layers and print paths. Figure 7.1 shows the voids that are created during printing that influence the gradient quality.

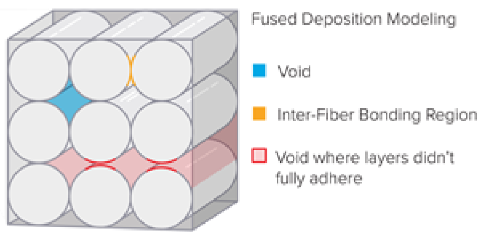

Fused Deposition Modeling
🟦 Void
🟧 Inter-Fiber Bonding Region
🟥 Void where layers didn't fully adhere

Figure 7.1: Voids created by the nature of Fused Deposition Modeling printers.

Bibliography

[1] Noshir A. Langrana Cheng-Tiao Hsieh. A system approach in extrusion-based multi-material cad. *SFF Symposium*, 2001.

[2] Noshir A. Langrana Dan Qiu. Void eliminating toolpath for extrusion-based multi-material layered manufacturing. *Rapid Prototyping Journal*, 8(1):38–45, 2002.

[3] Florian Bouville Dimitri Kokkinis and Andr R. Studart. 3D printing of materials with tunable failure via bioinspired mechanical gradients. *Advanced Materials*, 30(19), May 2018.

[4] David Espalin Mireya Perez Efrain Aguilera Dan Muse Ryan B. Wicker Eric Macdonald, Rudy Salas. 3D printing for the rapid prototyping of structural electronics. *IEEE Access*, 2:234–242, March 2013.

[5] J.D. Hiller and H Lipson. Stl 2.0: A proposal for a universal multi-material additive manufacturing file format. *20th Annual International Solid Freeform Fabrication Symposium*, pages 266–278, January 2009.

[6] Sylvain Lefebvre Jean Hergel. Clean color: Improving multi-filament 3D prints. *Computer Graphics Forum*, 33(2), May 2014.

[7] Yuan Jin, Jianke Du, Zhiyong Ma, Anbang Liu, and Yong He. An optimization approach for path planning of high-quality and uniform additive manufacturing. *The International Journal of Advanced Manufacturing Technology*, 92(1):651–662, Sep 2017.

[8] Leon L. Shaw Jiwen Wang. Fabrication of functionally graded materials via inkjet color printing. *Journal of the American Ceramic Society*, 89(10):3285–3289, September 2006.

[9] Yuwang Wangy Joyce Kwan Justin Lan Wenshou Wang Wojciech Matusik Pitchaya Sitthi-Amorn, Javier E. Ramos. Multifab: a machine vision assisted platform for multi-material 3D printing. *ACM Trans. Graph*, 34(4):129, 2015.

[10] Amit Bandyopadhyay Susmita Bose, Sahar Vahabzadeh. Bone tissue engineering using 3D printing. *Materials Today*, 16(12):496–503, December 2013.

[11] Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik. Openfab: A programmable pipeline for multi-material fabrication. *ACM Transactions on Graphics*, 32(4):11, July 2013.

[12] R.J.M. Wolfs. 3D printing of concrete structures. MSc thesis, Eindhoven University of Technology, February 2015.

[13] Kaiming Ye Zengmin Xia, Sha Jin. Tissue and organ 3D bioprinting. *SLAS Technology*, 23(4):301314, 2018.

Chapter 8

Appendix A

8.1 Slicing Process

The slicing process can be generally described in multiple steps:

1. Loading all geometric models into a `MeshGroup` object
2. Generate outline of `MeshGroup` object as polygons
3. Process outline information and generate inset perimeter polygons
4. Process slice data and produce `LayerPlan` object for each layer in parallel
5. Convert `LayerPlan` objects in ordered non-parallel way to G-Code
6. Finalize the G-Code
7. Report statistics and G-Code to front end

Each step in the slicing process is discussed in this section, highlighting the most important parts.

1. Loading models Loading models into a MeshGroup object is fairly straightforward, for each model in the workspace a file stream is opened which parses the file. When parsing an STL file, which is the input model extension, it can be either in binary or ASCII format. Both formats define faces with vertices, for each face the vertices are parsed and stored into `Point3` objects after which these points are stored in a `MeshFace` object. All faces together in an array form a `Mesh` object which together form the final MeshGroup object. A MeshGroup object is then used as input into the general slicing process.

2. **Generating Outline** The first slicing step is intersecting a mesh group object with layer planes, creating 2D outlines of the mesh group for each layer height. This creates lists of polygons without any order or structure. When slicing the layer plane can intersect in several ways with a triangle face, projection and linear interpolation is used to create a straight line through a triangle face that represents the intersection. The next step is grouping all polygons into parts, where each part represents an isolated area in the 2D plane. This ensures each part is printed entirely before moving to an other part in the same 2D plane.

3. **Process outline and inset** After the outline has been generated and ordered the inset polygons are generated. When the model has been sliced the layers are further processed, generating wall inset areas and areas to be filled with support and infill patterns. Wall insets are extra concentric wall lines if a wall thickness of one line is not enough. The actual infill and support patterns are generated in a later stage.

4. **LayerPlan production** The `LayerPlan` production is done with multi-threading. Before a layer plan object is created several settings are retrieved from the settings storage and some variables are determined for the layer plan constructor. After the layer plan has been constructed it is further processed, if it is not a pre model layer (raft, skin) all remaining features are added: the infill areas are filled with the desired infill patterns, the skin areas are filled and if enabled gaps between features can be filled. To fill an area, polygons are generated that fit info the infill area. The generated polygons can be added to a layer plan with various methods, converting these polygons to extrusion and travel moves and adding them to the extruder plans.

5. **LayerPlan consumption** Before layer plans are converted to G-Code, they are added to the layer plan buffer that connects them. If the buffer limit is exceeded the front layer plan is consumed and added to the G-Code output stream. Consuming a layer plan is simply iterating over all extruder plans and using `GcodeExport` methods to add all print moves to the output stream.

6. **Finalizing G-Code** When all layer plans have been added to the G-Code, the end code is written. The end code is a string that can be defined in the interface for custom G-Code commands that are printer specific. Besides the end code, various printer settings such as absolute extrusion mode, print acceleration and maximum feed rate are reset to their default values.

7. **Report to frond end** After finalizing the G-Code the output stream is flushed and sent to the front end together with statistics such as slice

time, estimated print time and estimated amount of filament required for the print. The paths can then be viewed in the print preview window of Cura, showing all planned paths.

8.2 Class descriptions

This section describes the most important classes in the Cura slicer.

`MeshGroup` class which contains all geometric models from the GUI, can have multiple `Mesh` objects which contain faces and vertices. Also stores an array of `ExtruderTrain` objects which contain extruder specific settings for each used extruder. This class is used as the input in the `FffProcessor` class.

`Slicer` class which does the slicing, it outputs coordinates of plane intersections with the geometric model that are used in the next step.

`FffProcessor` short for Fused Filament Fabrication Processor, class hold all general objects used in the slicing process. It is responsible for the finalizing of the gcode, reporting statistics to the front end, time estimation of the slicing progress, parsing the settings string and contains the important subclasses `FffPolygonGenerator` and `FffGcodeWriter`.

`FffPolygonGenerator` class that is responsible for the generation of the polygons. Polygons are represented as objects with a list of points. The `FffPolygonGenerator` class takes the unordered input of the slicer object and creates polygons that represent outlines: the boundaries between the inside and the outside of the object.

`FffGcodeWriter` class that is responsible for the actual G-Code generation, also generates support and infill patterns in designated areas. All polygons that were generated by the `FffPolygonGenerator` class are converted into G-Code by using the `GcodeExport` class. It also contains the consumer producer system of `LayerPlan` objects and the `LayerPlanBuffer` which handles all produced layer plans.

`LayerPlan` class that contains all extrusion and travel moves for each layer. Polygons and lines can be added to a layer plan with various methods, adding them to the path planning. It contains several important variables used in generating the G-Code:

- `extruder_plans`: vector of `ExtruderPlan` objects, an extruder plan object contains an extruder number that indicates which extruder will be used for this path. Besides the extruder number it also contains a vector of `GCodePath` objects, these G-Code paths contain the actual points together with a configuration for this point collection.

- `z`: integer which contains the height of this layer in the z plane, unit is in micron (micrometer).
- `layer_thickness`: integer that contains the thickness of the current layer in micron.
- `layer_nr`: integer that acts as identifier, zero is first layer and negative layer numbers are pre model layers (raft, skin etc).

`GcodepathConfig` class that holds several properties of a G-Code path, such as the type (wall, infill, skin etc.), line width, flow and the extrusion volume amount per mm line traversed. When new paths have to be added to a layer plan, the path configuration is compared to the last path. If it is equal the points of the new path are simply added to the last G-Code path. If the configuration is different, a new G-Code path object is created with the new configuration and added to the extruder plan.

`GcodeExport` class that handles the extrusion moves, travel moves, retractions and other printer actions from a layer plan and outputs them in the correct G-Code format, which is G-Code flavour dependent. It writes all G-Code to a `std::ostream` object as a buffer, which can be flushed and sent to the front end.

`LayerPlanBuffer` class that buffers all produced layer plans. Since layer plans are produced in parallel they are not ordered during production, hence the need for a buffer. The producer consumer system guarantees that layer plans are consumed in an ordered non parallel way, when a layer plan is consumed it is pushed into the end of the buffer. If at least two layer plans are present in the buffer, a travel move is added between the last and second last layer plan in the buffer to connect them. A variable buffer limit can be set, if it is exceeded the front layer plan is popped and it's extruder plans are converted to G-Code with `GcodeExport`.