Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

3D Printing of Nonplanar Layers for Smooth Surface Generation

vorgelegt von

Daniel Ahlers

MIN-Fakultät

Fachbereich Informatik

Technical Aspects of Multimodal Systems, TAMS

Studiengang: Informatik

Matrikelnummer: 6424701

Erstgutachter: Prof.Dr. Jianwei Zhang

Zweitgutachter: Dr. Norman Hendrich

Abstract

Stair stepping artifacts reduce the surface quality of 3D printed objects, especially when the slope is close to horizontal. Previous researchers showed that nonplanar surfaces could reduce these artifacts but only presented prototypes for the toolpath generation. The combination of nonplanar and planar layers results in smoother surfaces and the object is still printable on a three-axis 3D printer. Adding the ability to print nonplanar surfaces to an open-source slicer increases the usability and provides a general-purpose approach. This work presents a slicer that is capable of generating nonplanar toolpaths from any object. While printing, the printhead does not collide with the object since this is checked beforehand. The surface quality of objects with nonplanar layers is significantly better than with planar layers only.

Zusammenfassung

Der Treppeneffekt auf 3D gedruckten Objekten, der vor allem auf flach verlaufenden Steigungen auftritt, verringert die Oberflächenqualität deutlich. Frühere Arbeiten haben gezeigt, das nicht-planare Oberflächen dieses Problem beheben können. Die Pfadgenerierung war dabei aber nur in Prototypen implementiert. Die Kombination von planaren und nicht-planaren Schichten führt zu glatteren Oberflächen auf Objekten dabei kann das Objekt mit einem Drei-Achs-Drucker gedruckt werden. Durch die Integration der nicht-planaren Schichten in einen Open-Source Slicer wird dieser Ansatz universell nutzbar. Außerdem erhöht sich die Benutzbarkeit deutlich. Diese Arbeit zeigt einen Slicer der in der Lage ist, nicht-planare Schichten in beliebigen Objekten zu generieren. Beim Druck der Objekte treten keine Kollisionen auf, da dies von vornherein überprüft wird. Die Oberflächenqualität der gedruckten Objekte ist dabei besser als die der planar gedruckten Objekte.

Contents

1 Introduction **1**

1.1 Problem Statement 2
1.2 Goal of this Thesis 3
1.3 Outline 3

2 Principles **5**

2.1 3D Printing 5
2.2 STL 7
2.3 G-Code 9
2.4 FDM Slicing Software 10
2.5 FDM Printers 12

3 Related Work **15**

3.1 Multi-Direction Slicing 15
3.2 Inclined Layer Printing 16
3.3 Active-Z Printing 17
3.4 Multi-Axis Material Extrusion 18
3.5 Fully Three-Dimensional Toolpath Generation 18
3.6 Curved Layer Fused Deposition Modeling 20
3.7 Combining Flat and Curved Layers 20
3.8 Path Planning for CLFDM 21

4 Implementation **23**

4.1 Planar Toolpath Generation 23
4.1.1 Layer Generation 24
4.1.2 Perimeter Generation 25
4.1.3 Prepare Filling 26
4.1.4 Surface Filling 27
4.1.5 Support Generation 29
4.1.6 Skirt and Brim Generation 30
4.1.7 G-code Generation 31

4.2 Hardware Limitations 33
4.2.1 Nozzle Geometry 34
4.3 Nonplanar Toolpath Generation 36
4.3.1 Identifying Printable Nonplanar Surfaces 37
4.3.2 Collision Avoidance 39
4.3.3 Surface Generation for Nonplanar Layers 41
4.3.4 Toolpath Generation for Nonplanar Layers 44
4.3.5 G-code Generation for Nonplanar Layers 50
4.3.6 Toolpath Visualization 52
4.4 Limitations 53
4.4.1 Unusable Slic3r Features 53
4.4.2 Over- and Under-filling 54

5 Evaluation **57**
5.1 Stair-Stepping 57
5.2 Complex Surfaces 58
5.3 Printability of Different Angles 59
5.4 Print and Slicing Speed 60
5.5 Approximation Error 61

6 Conclusion **63**
6.1 Outlook 64

Bibliography **67**

List of Figures

1.1 Quarter sphere with stair-stepping 2
1.2 Growth of stair-stepping 3

2.1 3D printing process 6
2.2 The ASCII STL file format 8
2.3 The binary STL file format 9
2.4 Cura and Slic3r 11
2.5 Two different FDM printers 12
2.6 A bowden and a direct drive extruder 14

3.1 Inclined layer printing 16
3.2 Bread slicer 17
3.3 Nozzle sweeping out a surface 19
3.4 Curved on top of planar slices 21
3.5 Printing height compensation 22

4.1 Layer generation 24
4.2 Perimeter generation 25
4.3 Layer classification 27
4.4 Surface filling 28
4.5 Finding overhangs 29
4.6 Collision model 34
4.7 Different nozzle geometries 35
4.8 Penetration depth 36
4.9 Printable surface 37
4.10 Simplified printhead collider 40
4.11 Object colliders 42
4.12 Nonplanar surface generation 43
4.13 Point in facet 46
4.14 Toolpath projection 47
4.15 Travel moves to acoid collisions 50

4.16 Extrusion correction factor 51
4.17 Toolpath visualization 52
4.18 Support collisions 54
4.19 Underfilling 55

5.1 Stair-stepping on a flat surface 57
5.2 Stair-stepping on a curved surface 58
5.3 Complex surface 59
5.4 Printability test 60
5.5 Approximation error test 62

1 Introduction

Three Dimensional (3D) printing is a process where 3D models can be printed as physical objects. It is possible to print objects with a complex geometry that is impossible to produce with classic fabrication methods [Gibson et al., 2015]. Expiring patents on some 3D printing techniques helped to spread low-cost printers on the market. They are affordable for home and small business use cases. The hype in the consumer market also pushed the Additive Manufacturing (AM) industry further where prototypes and even products with small quantities are printed. 3D printing mostly uses different types of plastics. In the industry, metal and ceramics are also common. Today different materials can be printed at once, and even electronics are integrated directly into the object [Wasserfall, 2015].

Fused Deposition Modeling (FDM) is an important technique for small-scale plastic 3D printers. Like in other techniques, the objects are built by stacking planar layers onto each other. Each layer is built by pushing melted plastic through a nozzle. Due to the discretized layer structure, printed objects have approximation errors from the original model like stair-stepping. These artifacts occur especially on surfaces with a slope close to horizontal. The intensity of the stair-stepping artifacts can be lowered by reducing the layer thickness. This can either be done on all layers evenly or adaptively on layers where it is necessary [Tyberg, 1998]. But since FDM printers have relatively thick layers, the stair-stepping will still be visible. They can be removed by post-processing the object with either chemicals or a Computerized Numerical Control (CNC) mill. Both of these methods require additional, often manually triggered work steps.

To get rid of stair-stepping, [Chakraborty et al., 2008] proposed to use curved layers on the surface of objects. These curved layers are nonplanar and follow the surface of the object instead of spreading it over different layers and creating a stair-stepping effect. [Huang and Singamneni, 2014] and [Llewellyn-Jones et al., 2016] used these curved layers to build prototypes with planar layers below and curved layers on top.

Figure 1.1: A quarter sphere with generated layers (green). The visible stair-stepping artifacts lead to approximation errors from the model (red).

1.1 Problem Statement

Due to the discretized layer structure of printed objects, stair-stepping artifacts occur on the surfaces where the slope is close to horizontal. Theses stair-steps are an approximation error from the model. Figure 1.1 shows stair-stepping artifacts on a quarter sphere and the approximation error from the model. The stair-stepping is way worse on surfaces with a low ramp angle than on those with a high ramp angle. Figure 1.2 shows the growth of the stair-step length when the ramp angle decreases on different layer heights. The stair-step length is calculated with a right triangle which is placed between two layers. The adjacent side of the angle forms the stair-step length while the opposite side represents the layer height. Stair-stepping is also unaesthetic and influences the mechanical properties of the printed object. Friction, fluid-dynamics, and aerodynamics are different from the designed object. For example, on a printed wing, these stair-stepping artifacts can produce air turbulence and reduce the uplift. Nonplanar layers can reduce the stair-stepping effect on surfaces. These objects look more aesthetic and have mechanical properties which are closer to the modeled object. Lastly, the objects are stronger since a nonplanar surface bonds multiple layers together.

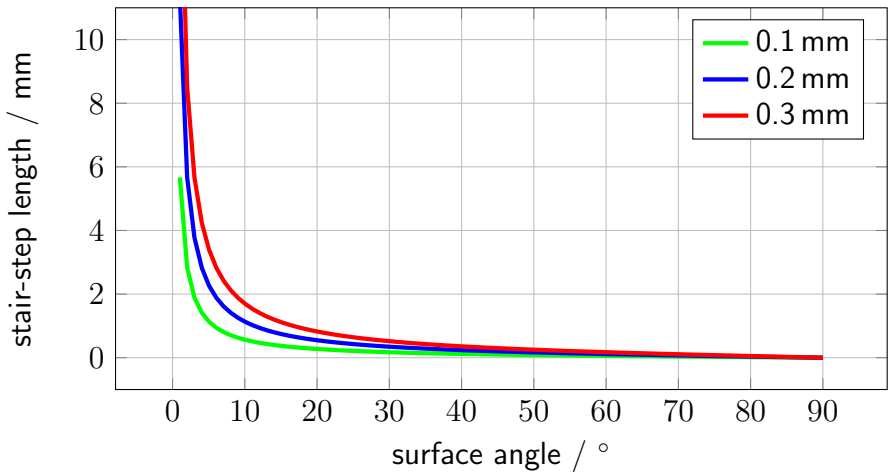


Figure 1.2: The stair-stepping length in relation to the angle of the surface. The stair-stepping length rises drastically below 20°.

1.2 Goal of this Thesis

The goal of this work is to develop a method to print nonplanar layers on top of planar layers in any object. This will lead to smoother surfaces with less stair-stepping artifacts. The toolpaths that are generated will be printable on a three-axis 3D printer without further modifications. To be able to generate the nonplanar toolpath, the surfaces are identified by their angle. They are checked for collisions during the printing process. The actual printing layers are created by moving the surfaces upwards from their original position below the nonplanar surface. The toolpaths are generated for a planar layer and are then projected downwards along the original surface mesh. To achieve a general purpose method which can be used by anyone, an existing open-source slicing software is modified to generate toolpaths for nonplanar surfaces. This increases the usability that previous implementations did not have. The nonplanar surfaces are only generated on surfaces facing upwards.

1.3 Outline

Chapter 1 introduces this work and sets its goals. The principles which are necessary to understand this work are presented in chapter 2. The related work, done by other researchers, is analyzed in chapter 3. Chapter 4 shows how the nonplanar surfaces can be found and how a toolpath is generated for them. The evaluation in chapter 5 presents some test prints with nonplanar surfaces and compares them with regular prints. Lastly, chapter 6 concludes this work.

2 Principles

This chapter describes the basic principles necessary for this work. First, the general 3D printing process is described in section 2.1. Then, the file format STL, which is the most used format in 3D printing, is presented in section 2.2. Then, the relevant open-source slicing tools are described in section 2.4. Lastly, an FDM printer is described in section 2.5.

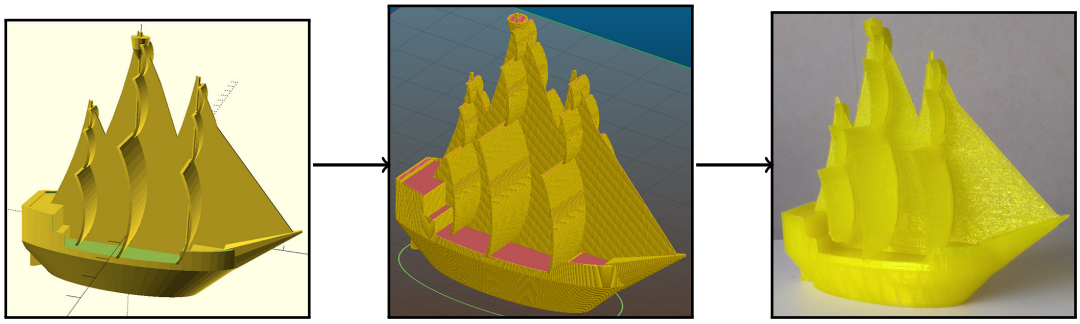
2.1 3D Printing

3D printing is similar to regular Two Dimensional (2D) printing on paper. It just adds the third dimension to produce objects that can be taken into hand. Like in paper printing it transforms the digital data from the computer into a real physical object. A term that is often mentioned as 3D printing is additive manufacturing. Which means 3D printing, but often in a more industrial environment. 3D printing can also be used for mass production of objects without the process planning and tool making of traditional fabrication processes. 3D printing saves complex planning and makes the process more flexible, but that does not mean that it is simple.

The general 3D printing process can be separated into three steps: Model generation (figure 2.1(a)), slicing (figure 2.1(b)), and printing (figure 2.1(c)). For the model generation, the model has to be designed, downloaded or scanned. Slicing means that the printing instructions have to be generated in such a way that the printer can print the model. The last step is the actual printing of the model on the 3D printer. These steps are described further in the following paragraphs.

Model Generation

The first step in the model generation process is to define what should be modeled. Maybe the model has to be generated by scanning an existing 3D object. The model could also be downloaded from a 3D model sharing platform or is sent by an external supplier like a customer. The new generation of a model within



(a) Model generation (b) Slicing (c) Printing

Figure 2.1: The 3D printing process. First, the model is generated. Then it is sliced with a slicing software and lastly, the object is printed on a 3D printer. [Ahlers, 2015]

a computer is called Computer Aided Design (CAD). There is a wide variety of CAD tools for different fields. Not every CAD software is capable of producing 3D printable models but since 3D printing became such a big topic in the last years, most vendors added the capability to their software. Common tools are OpenSCAD, Tinkercad, Autodesk Inventor or Blender. OpenSCAD is an open-source tool where code-like instructions generate the model. Tinkercad has a drag and drop mouse centered design approach. Autodesk Inventor is a more engineering centered tool with the capability of creating printable parts and also assembling different parts or performing stress tests. Blender is a visual or sculptural program, often used by artists and game designers.

Formats which most 3D slicing programs can process are STereoLithograhy (STL), OBJ and Additive manufacturing file format (AMF). The de facto standard is STL which is further described in section 2.2. The OBJ and AMF are also facet based formats like STL with a similar structure. They add further information about color and could handle models with different parts.

Slicing

Since printers cannot print the model directly, it has to be transferred into a form that the printer can understand and execute. This is quite similar to paper printing where the text or image that should be printed is translated into instructions for the printhead and its movements. In 3D printing, this transfer from the model to a form that the printer can print it is called slicing. The instructions generated while slicing are called G-code and are further described in section 2.3.

Common slicing tools in FDM printing are Cura, Slic3r, and Simplify3D. The different FDM slicing tools are further described in section 2.4. The slicing process as it is implemented in Slic3r is described in detail in section 4.1.

Printing

When the G-code, is generated by the slicer, it has to be executed on the printer to get the desired object. To execute the G-code, it has to be sent to the printer. It can be transferred through USB by a print server like OctoPrint or with a G-code sender integrated into the slicing software. The G-code could also be sent via network to the printer or stored on an SD-card which is then put into the printer. The printer executes the G-code commands one after another in the same order as they appear in the G-code file. The whole printing process runs automated and does not need assistance. Since mistakes may always occur, monitoring the machine while printing is quite useful. When starting the print, the printer prepares itself for the print job. All axes are driven to their home position to ensure their positions. When the printer supports print bed leveling, the leveling process is executed. The printer then heats up the print bed and the nozzle. Some printers prime their printheads before starting the print. Then the printer starts with the execution of the G-code commands which print the actual object. Usually, a print has several hundred thousand instruction lines. The amount of print instructions depends on the size and complexity of the object. The printer usually gives feedback to the user while printing. When the print is finished, the printer shuts down its nozzle and print bed and moves the printhead aside. The finished object can now be removed from the print bed by the user. The hardware of a 3D printer is further described in section 2.5.

2.2 STL

STL is a file format which was created by 3D Systems [3D Systems, 1988]. It stands for STereoLithograhy and is the de facto standard for 3D printing. It describes the geometry of the model without additional information like color or texture. The surface of STL models is composed out of multiple facets. Each facet is defined by three vertices and a facet normal. The facets lay beside each other to form the model. Each facet is separate and has no direct relation to other facets. The only relation neighboring facets have is that they both contain the same edge. Since

facets cannot be round, the facet mesh is only an approximation of the actual model as long as it contains any round surface or edge. This approximation could also lead to visible facets on the outside of the model. Each facet has a surface normal vector to determine the outside of the facet. The units in the STL format are arbitrary and they have no scaling information. Common units are mm, inch or meter. When the surface normal vector is zero, it can be calculated with the right-hand rule. To be printable, a STL model has to be watertight and manifold. Watertight means that the model has no holes in its surface. This is reached when each facet has exactly three neighbors. Manifold means that the facets of the model are arranged correctly and facing the right direction. Watertight and manifold models are generally allowed in the STL format and have to be checked before printing. Many CAD programs can output STL models. STL files can be stored in American Standard Code for Information Interchange (ASCII) (figure 2.2) or binary (figure 2.3) format. In the ASCII format, all coordinates are stored as written float values with a describing text. STL files in ASCII format take up a lot of disk space and are rather inefficient. To reduce the necessary storage, STL

solid *name*
$\left\{\begin{array}{l} \textbf{facet normal } n_i\ n_j\ n_k \\ \quad \textbf{outer loop} \\ \quad\quad \textbf{vertex } v1_x\ v1_y\ v1_z \\ \quad\quad \textbf{vertex } v2_x\ v2_y\ v2_z \\ \quad\quad \textbf{vertex } v3_x\ v3_y\ v3_z \\ \quad \textbf{endloop} \\ \textbf{endfacet} \end{array}\right\}^{+}$
endsolid *name*

Figure 2.2: The ASCII STL file format. Each triangle is stored with its three vertices and a facet normal. [Burns, 1993]

files are often stored in a binary format. It is about 80 % smaller than the same file stored in ASCII format. The binary format has an 80 byte header which is not further defined, followed by the number of facets in the whole object. The facets are then listed, beginning with the surface normal, followed by the three vertices. Lastly, each facet has an attribute count value.

$Byte[80]$	ASCII Header
$unsigned\ long\ int$	Number of facets
$\left\{ \right.$	
$float$	n_i Normal vector
$float$	n_j
$float$	n_k
$float$	$v1_x$ Vetex 1
$float$	$v1_y$
$float$	$v1_z$
$float$	$v2_x$ Vetex 2
$float$	$v2_y$
$float$	$v2_z$
$float$	$v3_x$ Vetex 3
$float$	$v3_y$
$float$	$v3_z$
$unsigned\ int$	Attribute byte count
$\left. \right\}^+$	

Figure 2.3: The binary STL file format. To save file size, only the values are stored as float. [Burns, 1993]

2.3 G-Code

G-code is a numerical control language which is widely used in Computer Aided Manufacturing (CAM). It is used to automate machine tools such as CNC mills, laser cutters, lathes or 3D printers. G-code tells the machine how to move its axis to produce the object. For example, the single G-code instructions tell the machine how fast and where to move. The machine then moves its motors to reach the given position. G-code can also control other machine behaviors like the temperature or control a servo. There are many different standards for G-code. In the past, controllers had to be aware of the differences in G-code. Today, most G-code generators can output different styles for different machine types. G-code is very machine specific, a code that works flawlessly on a machine can damage another one. The different G-code commands start with either a **G** for preparatory commands or an **M** for miscellaneous functions. Some example G-code commands are:

G28 X Y Z Homes the x-, y- and z-axis by moving each one of them to its endstop and setting the coordinate zero there.

G1 X100 Y100 Z100 E10 F2400 Moves linear interpolated to the position x 100, y 100, z 100 and moves the extruder 10 mm with the maximum speed of 2400 mm/min. In linear interpolated movement, each axis dives with different speed so all axis reaches their target at the same time with the tool moving at the desired speed. The **G1** command is by far the most used command in 3D printing because the whole object is built with this command.

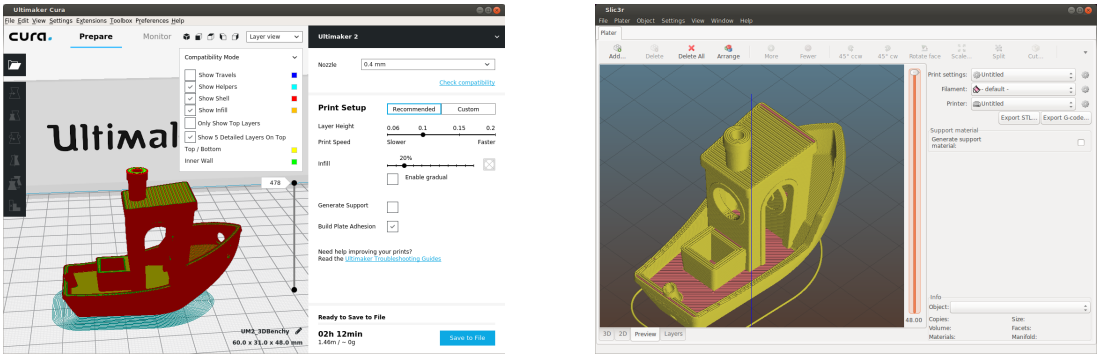
M109 S210 T0 Heats up the extruder 0 to 210 C° and waits for the extruder to reach its temperature before continuing with the next command.

2.4 FDM Slicing Software

There is a wide variety of different slicing software for FDM printers. Some are made for a specific printer manufacturer others are general-purpose. The general-purpose slicers can be divided into three subgroups: paid-, free- and open-source-slicers. Since paid- and free- slicers have inaccessible source code, they cannot be modified to add nonplanar layers. Therefore, these slicers will not be considered any further here. What's left are the open-source slicers. [Cura] and [Slic3r] are currently the only two open-source slicers which are actively developed and can be called state of the art. Skeinforge is outdated and no longer developed and other slicers mostly use Slic3r or Cura Engine for slicing and just provide a different user interface. Cura and Slic3r can be called the only two relevant open-source slicers and are described further in the following sections.

Cura

Cura is an open-source slicing software developed by Ultimaker B.V.. It is licensed under the GNU Lesser General Public License (LGPL) license. Ultimaker does the main development and the slicer is therefore optimized for the Ultimaker printers. Other printers are also supported and some have already predefined profiles which are shipped with Cura. When a printer is not predefined, a custom printer can be configured. Cura has also included different material profiles, both for branded and generic filament. The development is rather active with a new release every 1-3 month. The software is divided into two parts, Cura and the Cura Engine. Where Cura is the frontend written in Python and Cura Engine is the actual slicing software written in C++. Cura has its own plugin infrastructure where different features can be added. Features and usability can be called state of the art. Cura



(a) (b)

Figure 2.4: The interface of (a) Cura 3.4.1. and (b) Slic3r 1.3.1-dev.

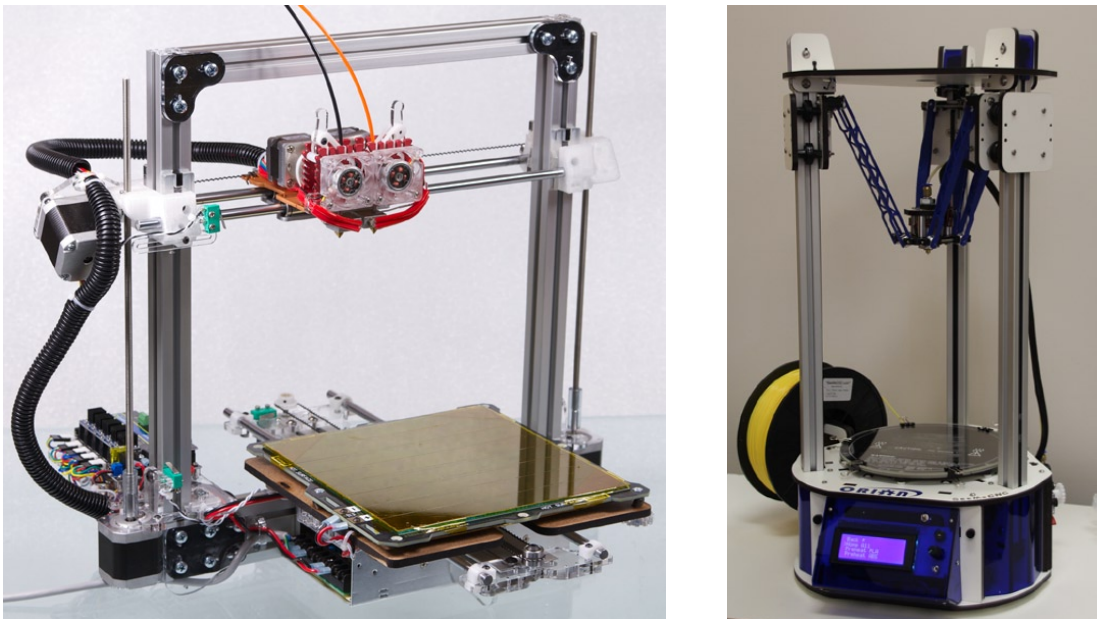
has a recommended mode where the user can only adjust five different settings and a custom mode with over 200 different print settings. The interface of Cura can be seen in figure 2.4(a). It is possible to load STL, 3D Manufacturing Format (3MF) and OBJ files into Cura. A detailed user documentation is available online.

Slic3r

Slic3r is an open-source slicer started by Alessandro Ranellucci. It is not sponsored or developed by any company and is licensed under the GNU Affero General Public License (AGPL). Slic3r war originally written in Perl. Today most of the core features are ported to C++, mainly because of performance improvements. The GUI and some leftovers are still in Perl. This combination of two languages makes it sometimes hard to understand the source code. Slic3r supports models in STL, OBJ, AMF and 3MF file format. Printer profiles are usually provided from the printer manufacturers but can be also created for an own printer with the configuration wizard. Slic3r has a simple and an expert mode in the current stable version, the simple mode was removed in the current development version. The interface of Slic3r can be seen in figure 2.4(b). Due to the complexity, Slic3r is not very beginner friendly but rather feature rich. The user documentation is slightly outdated. The development is active and new features are constantly added, but the last stable release is over two years old. Due to this lack of active releases, there is a fork, the Slic3r Prusa edition by Prusa Research s.r.o.. This fork natively supports Pursa 3D printers and adds new features like a new support structure written in C++. It also has a fast release cycle with up to multiple new releases per month.

2.5 FDM Printers

FDM Printers are very common printers, especially in the low-cost segment. The hardware of these printers can be divided into the motion system and the extrusion system. The motion system moves the printhead to the desired position in the 3D printer while the extrusion system lays down the material.



(a) (b)

Figure 2.5: (a) A Cartesian motion system with a Cartesian-xz-head configuration. (b) A delta printer with its three-axis all moving along the z-axis. [Horvath, 2014]

Motion System

Other CNC applications like milling, laser cutting, pick and place machines and robotics have inspired motion systems of 3D printers. The most common motion system of 3D printers is the Cartesian. These printers move each axis of the printhead with a distinct motor. The x-axis is usually left and right, the y-axis is front and back and the z-axis moves up and down. Stepper motors drive either belts or leadscrews to transfer the rotary motion into linear motion alongside the axis. Since belt systems are a bit faster, these are often used for the x- and y-axis while leadscrews are mostly used for the z-axis. There are two common Cartesian configurations. The first being the Cartesian-xz-head, where the printhead is moved

along x- and z-axis while the print bed is moved back and forth along the y-axis. The other common is the Cartesian-xy-head configuration where the printhead is moved along the x- and y-axis while the print bed moves up and down along the z-axis. A Cartesian printer can be seen in figure 2.5(a). Delta printers are another motion system which was originally used in pick and place applications. The printhead is mounted on a carriage with three attached arms which move individually along the z-axis. To reach the desired coordinate, the position of the individual arms is calculated with inverse kinematics. The printhead is capable of reaching every-position above a round print bed. Belts or leadscrews with stepper motors are usually used to position the individual arms. The main advantage of this configuration is that the actual end effector carries little weight so it can move faster. Calibrating a delta printer is slightly harder than a Cartesian printer thus all axis have to be calibrated at once and not one after another. A delta printer configuration can be seen in figure 2.5(b).

Extrusion System

The extrusion system of an FDM printer is usually a stepper motor which pushes filament through a heated nozzle. The part that moves the filament forward is called the feeder while the heated nozzle block is called the hotend. The pushing into the feeder is done by a drive gear which presses against the filament string. The drive gear is driven directly from the stepper motor or with a transmission gear to increase the pushing force. The filament is a very long plastic string with a fixed diameter. Common sizes of filament are 1.75 and 2.85 mm. The filament can be fed directly into the hotend or with a Bowden tube in between. The advantage of the Bowden system is that the stepper motor does not have to be directly on the print head. That reduces the weight of the carriage but it adds extra flexibility in the filament path. This flexibility causes issues in the extrusion especially, with flexible filaments. With a directly driven extruder without a Bowden tube, the printhead has to carry the heavy stepper motor of the feeder. The purpose of the hotend is to heat the filament up and bring it to a defined width to get a continuous filament path while printing. The heating is done by a heating element which is monitored by a thermistor. The heater is controlled by a Proportional Integral Derivative (PID) controller forming a closed loop system with the thermistor. To get the filament into a defined width, it is pushed through a heated brass nozzle. A nozzle is conical and has a hole with a very precise diameter on its pointy end. Common diameters are between 0.25 mm to 1.0 mm. The diameter of the tip is a

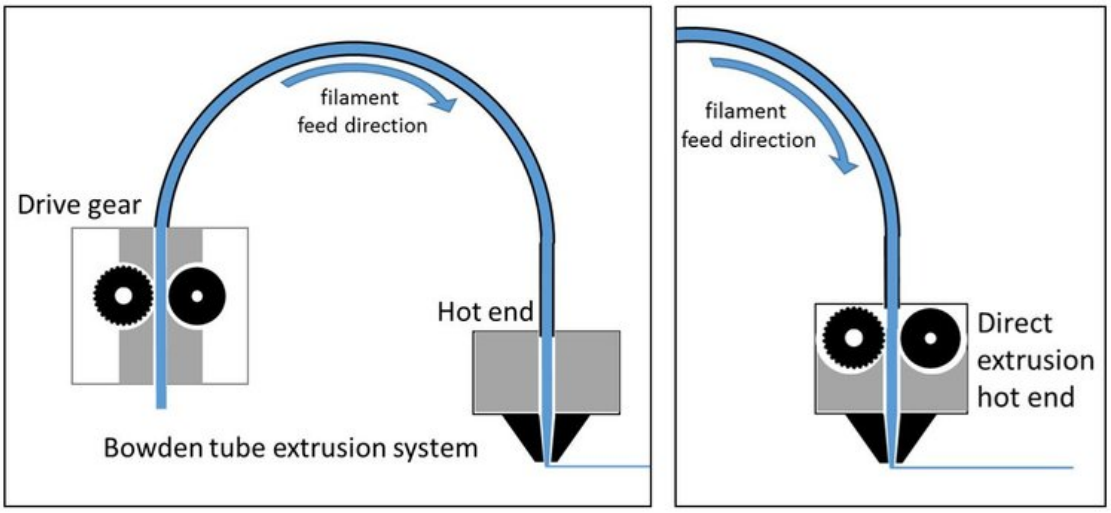


Figure 2.6: Left: A Bowden extruder with the feeder, the Bowden tube and the hotend. Right: A direct drive extruder with the feeder directly above the hotend. [Forefront Filament, 2016]

trade-off between speed and printable details because bigger diameters extruding wider traces which leads to faster prints. The width of the print trace can be variated a little bit by pushing more or less filament through a nozzle.
The first layer of the filament extrusion is laid out on the print bed. A print bed is a flat surface where the print object is built on. To increase adhesion of the print and to reduce warping of prints, most print beds are heatable. The adhesion can also be increased with special print surfaces, a layer of glue stick or painters tape.
The printer electronics control all motors and heaters. Stepper motor drivers drive stepper motors, heaters are controlled by Metal Oxide Semiconductor Field Effect Transistors (MOSFETs), the temperatures are read by thermistors and the homing position is found with attached switches. An 8-bit Arduino style microcontroller drive most FDM printers. Some newer models use 32-bit architecture. The microcontroller reads the G-code commands and executes them with the attached hardware. The G-code commands are sent by a control software via a serial interface or read from an attached SD-card.

3 Related Work

Several researchers already used nonplanar layers or other forms of 3D toolpaths to get better objects. Better could either mean printing without support material, improved part strength or printing esthetically better-looking objects with a smoother surface. The different researchers either use special printers with at least five-axis or standard three-axis printers that can usually be bought off the shelf. In the following paragraphs, each approach is presented with a rough overview of how each problem was solved.

3.1 Multi-Direction Slicing

To print objects with heavy overhangs without using support structures, [Ding et al., 2015] used multi-direction toolpath planning. The toolpaths where generated for a five-axis printer. The multi-direction toolpath is generated by slicing the model in different directions so that some layers are not horizontal. Two separate modules are used to generate the desired toolpath. The first module is for volume decomposition and regrouping. It takes the object and decomposes it by searching for closed concave loops in the volume mesh. Concave loops contain only edges which are also concave. On concave edges, the adjacent facets form an angle that is smaller than 180°. Holes in the mesh that emerge when decomposing the object are filled so that these meshes can be sliced later. For the later performed regrouping of the subvolumes, the topology information and the position of each subvolume is stored. The second module is responsible for the actual slicing process of the different subvolumes. For each subvolume, the best possible printing direction is calculated and the subvolume is sliced with a regular planar slicing algorithm. The different toolpaths are then translated and rotated back to their original position and arranged according to their topology information. In the whole process, collision detection is not taken into account. The toolpaths where only shown in theory, no actual printed objects where shown.

3.2 Inclined Layer Printing

Like in the previous approach, [Zhao et al., 2018] printed objects with a multi-direction toolpath. But instead of a five-axis printer, they use a standard three-axis printer to print the overhang regions with different printing angles. Overhanging regions are especially hard to print when the offset of the previous layer is bigger than the extrusion width of the line printed in the current layer. These lines are printed into thin air and often hang down instead of forming a layer as intended. Support structures are usually generated to print such regions. The support has to be removed from the finished print and the contact areas of the object stay rather rough. So the idea is to print some regions with a different non-horizontal direction to get rid of the support structures. The object is partitioned into different regions regarding their overhang to generate different printing directions. The whole process is visualized in Figure 3.1. Each partition is rotated by the desired printing angle and translated onto the print bed. All models are then sliced with a standard slicing tool for horizontal layers. The generated paths of the partitioned object are then translated and rotated back into their original position. There can be multiple printing angles in one object. The generated G-codes are executed one after another to get the correct printing order. Zhao et al. also observed that with rising printing angle of the non-horizontal extrusions the bonding between the layers decreases. They assumed that this happens because the force that the extruder pushes the filament on the previous layer is heading in a different direction than on horizontal layers. This only occurs on three-axis printers, with five or more axis the printer can direct this pushing force onto the previous layer by tilting the nozzle towards it. They also observed that gravity is an issue when the surface angle is very high. Single extrusion lines can hang down due to sticking issues to the previous layer looking similar to overhanging extrusions.

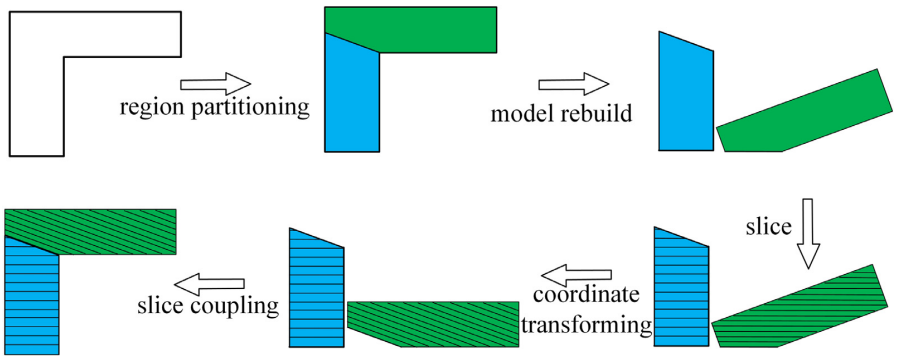


Figure 3.1: The inclined layer printing method. The model runs through the steps to generate toolpath in another direction than horizontal. [Zhao et al., 2018]

3.3 Active-Z Printing

3D printed objects with planar layers are weak when load is applied perpendicularly to the layers interfaces. [Khurana et al., 2017] had the idea is to use nonplanar 3D shaped layers to improve the mechanical strength of the printed objects with a standard three-axis 3D printer. This nonplanar layer printing was called active-z printing because it involves motions in the x-, y- and z-axis simultaneously to print the desired layers. They use the open-source [Bread slicer]. The bread slicer needs two STL files for the actual slicing process (Figure 3.2). The first STL file is the object that should be printed later. The second STL file holds the shape that the layers of the object should later have. Process parameters like the desired layer thickness are added in a configuration file. The slicer shapes the object regarding the layer shape modifier through the whole object. The 3D toolpaths are later generated for these layers. Because of the early development status of the bread slicer, it produces many unwanted travel moves which sometimes cause collisions and increase the print time unnecessarily. To get a fast toolpath without collisions while traveling, the G-code is post-processed with a MATLAB script. The authors did several test prints with a sinus formed layer structure with different amplitudes and performed several mechanical tests with them. It shows that the parts with nonplanar layers are both stronger and stiffer than regular printed objects with

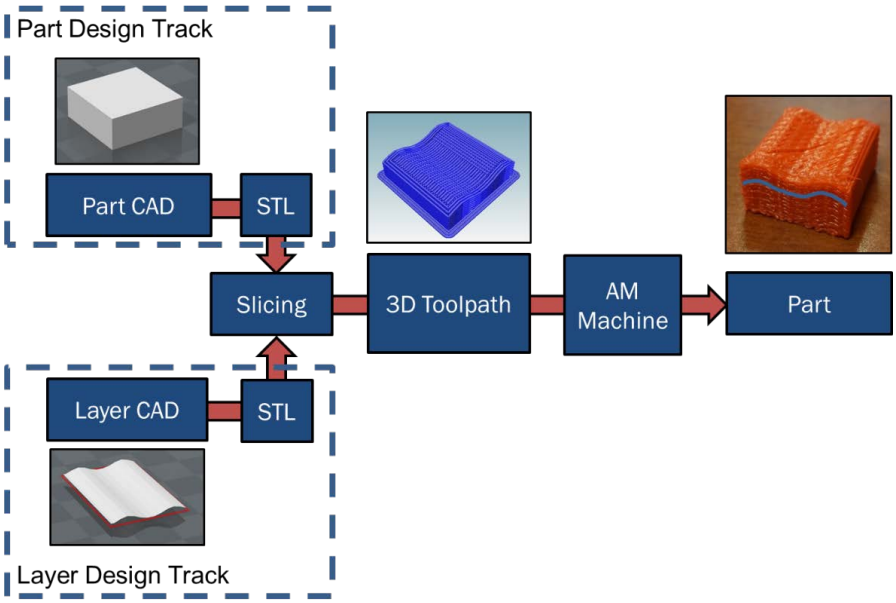


Figure 3.2: The bread slicer which combines the model STL with the layers STL to generate a 3D toolpath. [Khurana et al., 2017]

planar layers. Parts with a small rotation of the filling structure are stronger while parts which were printed with a higher amplitude get stiffer. Overall, active-z printing with nonplanar layers increases the mechanical strength of the printed parts where the force is applied perpendicularly to the interface layers.

3.4 Multi-Axis Material Extrusion

To improve the mechanical properties along the printing axis, [Kubalak et al., 2018] printed a reinforced surface onto a regularly printed core with a six-axis robotic arm printing system. The idea is to shift the stress from the layer bonding to the shell of the object which is printed in a different direction as the core. Starting points are generated at the bottom contour of the object to generate this skin. The distance of the individual starting points depends on the desired line width for the shell extrusions. The path is generated for each starting point with a given angle around the object until the path reached the top of the object. The G-code commands for the paths are generated using the coordinates for the position and the inverse of the facet normals for the print heads rotational component. The algorithms can only process objects that have only one single closed loop perimeter. A clear maximum z-height with no local maxima where path generation would get stuck is necessary. Since the printhead would collide with the print bed, the skin cannot be printed directly from the bottom of the object. The object cannot be excessively concave to prevent collisions between the object and the printhead. Overall, the skinning process increases the mechanical strength of the object. The best performance is reached when the skin lines are printed along with the expected load.

3.5 Fully Three-Dimensional Toolpath Generation

To get rid of the stair-stepping artifacts caused by the layer based structure, [Micali and Dornfeld, 2016] generated a 3D toolpath which can follow a free-form surface. This approach is designed for nozzle based printing processes in standard three-axis 3D printers. CNC toolpath planning approaches inspire the 3D toolpath generation. The machine geometries are taken into account to check for collisions and guarantee a collision-free toolpath. Nozzles are most often conical, so it is possible for them to access narrow regions while printing. The ability to reach narrow regions decreases with shorter, more bulky nozzles with a lower ramp angle.

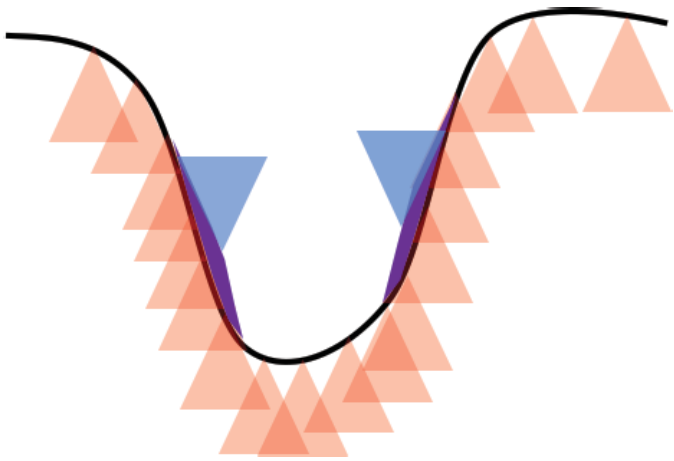


Figure 3.3: An upside down nozzle is sweeping out an envelope surface with its tip. This surface is used to check for collisions and build a toolpath that fills the surface. [Micali and Dornfeld, 2016]

The inverse toolpath offset is generated to calculate the printability of a path. The nozzle is therefore taken upside down and its tip follows the printing surface sweeping out an envelope with its body (Figure 3.3). The formed envelope surface is a printable, collision-free surface. When the envelope surface lies above the printing surface at any point, this point is not reachable with the nozzle. The surface in its current shape is not printable. When the difference between these two surfaces does not exceed a given tolerance, the generated envelope surface can be used for the toolpath generation instead of the object surface. The toolpath is generated by starting from one side of the surface and filling it along one edge until it reached the other side. This process is repeated with a new starting point near the previous starting point in an unfilled area until the surface is completely filled. The extrusion amount that is necessary to fill the surface is calculated by multiplying the Euclidean distance between two points of the path with the desired flow. The full 3D toolpath is now generated for this single shell and is guaranteed to be collision-free. This approach is only used to generate a toolpath for a given surface and does not provide a printable toolpath which produces a 3D printed object.

3.6 Curved Layer Fused Deposition Modeling

To improve the surface quality, [Chakraborty et al., 2008] printed nonplanar curved layers over different z-heights. This process was named Curved Layer Fused Deposition Modeling (CLFDM) by Chakraborty et al. They defined three key factors for the printability proper toolpath generation, proper filament orientation, and proper filament bonding. A three-axis printer can print the toolpaths, but a five-axis printer would be better. They are generated from the top to the bottom of the object. The curved surface is defined as a parametric surface. A MATLAB script was written to get the toolpaths for the layer along the parametric surface. The thickness of the object is generated by offsetting the top surface until the desired number of layers is reached. Chakraborty et al. assumed that the bonding of adjacent filament paths is worse than within planar layers. Mostly, paths do not necessarily lie on the same plane as they would with planar layers. They also assumed that the inter-layer bonding would be better than that of planar layers since curved layers have more surface area to create this bond. The generated toolpaths where just visualized and not printed on a real printer.

3.7 Combining Flat and Curved Layers

CLFDM was used in combination with planar layers to get rid of the stair-stepping artifacts in real printed objects. Both [Huang and Singamneni, 2014] and [Llewellyn-Jones et al., 2016] used three-axis printers to print the objects. They use STL files to provide the model or at least parts of it. To get a printable CLFDM surface, they use different approaches.

Huang and Singamneni

To identify the surface area which should be printed as a curved surface, all facets of the STL mesh are classified by their angle of the normal vector and the z-axis. Each facet is either part of the top, side or bottom surface. The user can define the angle to classify the facets between top and side surfaces. Next, all connected surfaces are defined as one single continuous top surface. The top surface is offset downwards along their facet normals to get the number of desired shell surfaces. The toolpath is generated by rasterizing the surface and generating the toolpath along those raster points. To get the planar layers, the offset surfaces are subtracted from the original STL and a new one is created. The planar layers are then generated from the new STL. The final layer structure can be seen in figure 3.4.

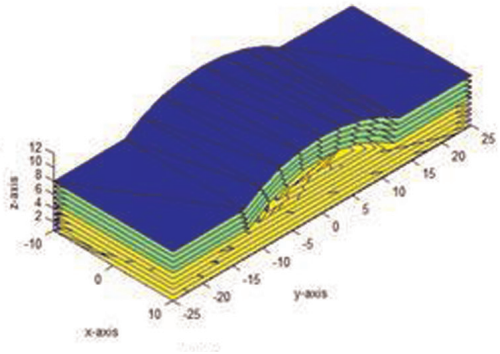


Figure 3.4: Curved slices are printed on top of the planar slices. This increases the surface smoothness while being still printable on a three-axis printer. [Huang and Singamneni, 2014]

Llewellyn-Jones et al.

The input STL file for this approach needs to be a thin skin of the object. It has to be thinner than the actual layer height. The skin is repeated over a number of layers by offsetting it in z-direction to get the desired model thickness. This approach can only produce an extrusion of the skin in z-direction. The toolpath is created by a MATLAB script which also generates a scaffold structure on which the skin layers can be printed. To create a toolpath of the skin, first, a perimeter is printed along the edge of the skin. The filling structure is generated again by rasterizing the surface as Huang et al. did. The scaffold structure is sliced as planar layers and is printed before the skin.

3.8 Path Planning for CLFDM

To get better surfaces with curved layers, [Jin et al., 2017] analyzed and optimized the toolpath generation. The surfaces of the different layers are generated by offsetting parametrized shells downwards. The focus lies on how to lay down toolpaths in a curved layer surface properly. For an optimal surface quality, the printhead should follow the surface with a five-axis printer. The printhead should be perpendicular to the surface at all times. Since this is impossible with three-axis printers, the printhead can cause collisions with the currently printed layer. The distance between the nozzle and the layer should be changed in regard to the print direction to prevent digging the nozzle in already printed areas or laying material down with a nozzle which is too high. The different print heights H for a curved surface with the angle θ and a nozzle tip diameter D is shown in figure 3.5. The

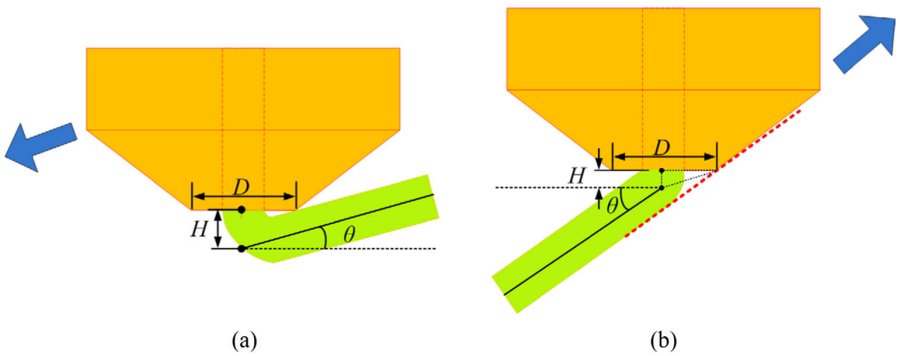


Figure 3.5: Different printing heights H regarding the print angle θ, the nozzle tip diameter D and the print direction. This correction factor should ensure a good surface extrusion without the nozzle penetrating the surface. [Jin et al., 2017]

distance to the top of the actual surface has to be larger when printing upwards than when printing downwards. The surfaces in the STL file are transferred into a B-spline which forms the reference surface to generate the toolpath. The toolpath is then generated along this surface. The void depth between two adjacent lines should be kept constant to get a constant extrusion. This results in a smooth surface on concave and convex surfaces. No real printed objects were presented in this work.

4 Implementation

In this chapter, the implementation of the nonplanar layer generation is described in detail. Nonplanar layers are added to an existing slicing software by modifying its source code. Slic3r is modified in this work because it is not owned by a single company and it is more likely that experimental features like nonplanar layers find a way into an actual software release. Furthermore, the visualization of Slic3r it better than the one from Cura. Lastly, the author of this work has already some knowledge about the Slic3r source code and its structure. The idea is to reuse as much as possible of the current implementation for planar layers and change the structure of the source code only where it is necessary. If possible, the implementation should use no additional external libraries for the nonplanar layers.

Since it is essential to understand the way planar toolpaths are generated, a thorough description follows in section 4.1. Section 4.2 describes the hardware limitations when printing nonplanar layers with a three-axis printer. It also shows parameters that could be used to build a proper collision detection. The implementation of nonplanar layers with all steps to print smooth surfaces is described in section 4.3. Lastly, the limitations of the current implementation are described in section 4.4.

4.1 Planar Toolpath Generation

In 3D printing, the toolpath generation is often called slicing. It is an essential step in the whole 3D printing process. The appearance of the printed model is set in this step, for example by using different layer heights. The overall print quality is also heavily influenced by the toolpath generation. For example, by applying retractions to the filament while moving the print head, the print does not suffer from oozing. Mechanical properties like the strength of a part or its weight are influenced by the thickness of the layers and the density of the internally printed structures. When the printer works correctly, the toolpath generation makes the difference between a good, a bad or even a failed print. The toolpath generation

is a software process step that every 3D printer needs. Some printers may take object models directly for printing. The toolpath generation of these devices is then done internally.

This chapter shows how toolpaths are generated in Slic3r. Each toolpath is only usable for one specific FDM printer since each one needs its own configuration. The process starts with a loaded object model, usually in STL format. The toolpath generation can be started automatically when a model is loaded, or manually by the user. Slic3r then generates a toolpath for each object on the print surface. The slicing can be separated into layer generation, perimeter generation, fill preparation, surface filling, support generation, skit and brim generation, and the G-code generation. In the following paragraphs, each step is described in detail.

4.1.1 Layer Generation

First, the layers are generated for the object height. These layers are distributed evenly along the z-axis of the printed object. The first layer is usually slightly thicker than the rest of the layers. Variable layer thickness is also possible and can be generated adaptively according to the geometry of the object. The object mesh is transformed, scaled and rotated when the user has used these features. Then the slicing of the object mesh into layers begins. The model is cut horizontally on the height of the layer and the contour of this cut is taken as the layer. To get the contour, each facet is checked against all layers that lay between their minimum and maximum z-coordinates. Then, each facet edge is checked for intersections with the layer. For each intersection that is found, the intersecting line is added to the layer. When a layer lies exactly on the edge of the facet, this edge is added to the layer. No line will be added when the layer crosses one of the facets vertices.

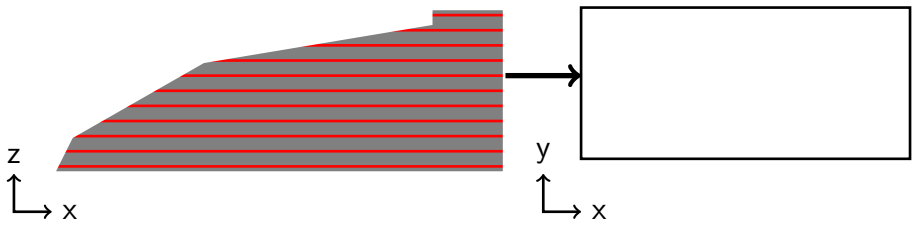


Figure 4.1: The generation of the layers from the contours of the object. Each red line represents a different layer. An example layer is shown in top-down view on the right.

These lines are chained together to generate contour loops from these independent lines. First, all tangent lines are removed because they hold no information about the shape of the object. The first line that was found is chosen to start a new loop. Then, the line that starts where the previous line ends is appended to the new loop. The process is repeated until the starting point of the first line is reached again. When there are unchained lines left, the process is repeated until no lines are left. Now, all loops of the current layer are found. Since these lines have the same start and endpoint, they are actually polygons. Next, expolygons are created from the found polygons where they represent either the contour or a hole. An expolygon is a contour polygon with multiple hole polygons representing one connected region on one layer. All the found expolygons of the layer represent the contour of the model on a specific height. An example cut can be seen in figure 4.1.

4.1.2 Perimeter Generation

In the next step, the perimeters are generated on each layer. A perimeter is the outer printed wall along the edges of the object. Perimeters are generated by offsetting the outline polygons, which were found in the previous step, by half the extrusion width to the inside of the object. Usually, more than one perimeter line is generated, therefore the outline is offset even more. When the path intersects with itself or another path, the loops are cut into different paths. The number of perimeters is defined by the user; common are two to three perimeter lines. Passages which are too narrow for two perimeters to pass each other are filled with a single extrusion line called gap fill. Extrusion objects are formed from the loops to print the generated perimeters later. Extrusion objects contain a path that will be printed and additional printing information. The perimeter of a layer is shown in figure 4.2.

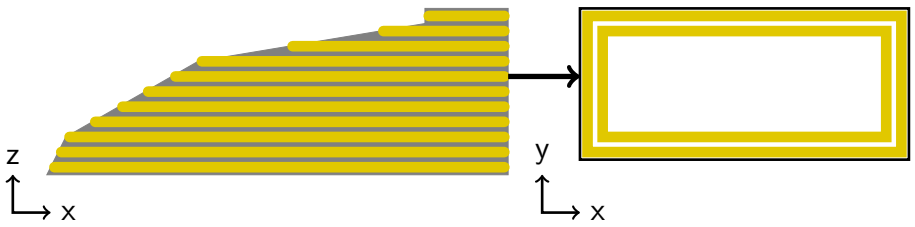


Figure 4.2: Two perimeters generated on each layer from the outline of the object by offsetting it to the inside. Each layer is filled like the example layer on the right.

4.1.3 Prepare Filling

Since there are different types of surfaces in an object with different print characteristics, the surfaces have to be classified into different types. Surfaces that are on the first layer are bottom surfaces as long as no raft is printed below the object. The outline of the layer above the current layer is subtracted from the current layer to detect top surfaces. The remaining area is classified as a top surface since it has nothing above itself. For the last layer, the full layer is classified as top surface because there is no layer above it. The same difference is calculated with the layer below the current layer to identify the bottom layers. They are either classified as bottom when dissoluble support is used or as bottom-bridge when normal or no support is used. This differentiation is important since both types need different print characteristics for a good print. All surface areas that remained unclassified yet are classified as internal surfaces. When the user does not want top or bottom surfaces, these surfaces are set back to internal. Usually, surfaces with a small area are set to internal solid to prevent internal surfaces with no infill structure.

Since the user can configure more than one top and bottom surface, these areas have to be grown to the layers inside the object. The multiple surfaces are called shells. The areas top, bottom, and bottom-bridge are grown into the upper or lower layers until the desired thickness is achieved. All found shells are merged by type and set as internal solid surface. To get a better solid layer directly over the infill, the flow of these areas is set to the same value as used for bridges. This is done as long as the infill density is below 25% and the additional material would fit into the void area of the previous layers. This is checked by calculating the additional volume that will be extruded, and by comparing it with the volume of the void areas from the layer below. In figure 4.3 classified object layers are shown with an example layer.

To print continuous bridges, internal areas which are surrounded by bridges are set to bottom-bridge by offsetting them. Then, adjacent bottom-bridge surfaces are merged into one surface when the bridges have the same direction. All bottom-bridges, which are not supported by two opposite edges on the layer below, are marked as unsupported bridges.

For faster printing and stiffer objects, the infill of multiple layers can be combined into one higher layer with thicker extrusion lines. Perimeters or surface

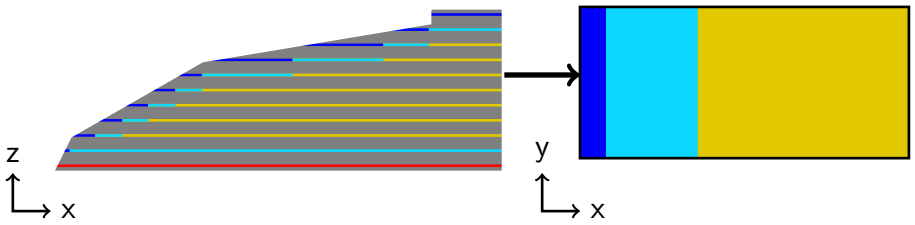


Figure 4.3: Classification of the layers with an example layer on the right. Top surfaces are blue, internal solid surfaces are light blue, bottom surfaces are red, and internal surfaces are yellow.

areas which are not marked as infill are not combined and stay untouched. When the combine infill option is set, intersecting areas which are marked as internal are searched. The user sets the layer range in which the layers are combined. To combine the layers, the lower layers of the found group are set to void and the material flow in the topmost layer is increased so it would fill its infill over multiple layers. All areas which are not an intersection with the combined layers remain unchanged and are filled regularly.

4.1.4 Surface Filling

After classifying every surface on the layers of the object, the paths can be filled with a toolpath. Each layer is filled separately although, some filling might reach down into other layers. For a more even and better-connected filling structure, all surfaces with the same proprieties are grouped. These properties are surface type, surface thickness, and bridge angle. Then a fill pattern is assigned to each group. Rectilinear is usually used for solid layers like the top, bottom, and bridges. It is also used for infilling an object with a gap between the lines. Especially for the infill, there are much more patterns like honeycomb, cubic or triangles. For a smoother surface finish on top layers, these layers get a specific flow. Then the groups with similar properties are merged into one surface regardless of their type. This is necessary to print connected print paths even when the type of the surface changes. For each surface, the specific flow is calculated taking the layer height and the extrusion width into account, and whether the surface is a bridge or whether a surface lies on the first layer. Next, the specific fill pattern of every surface is generated.

Since the rectilinear fill pattern is used for solid fills and also often for infill, it

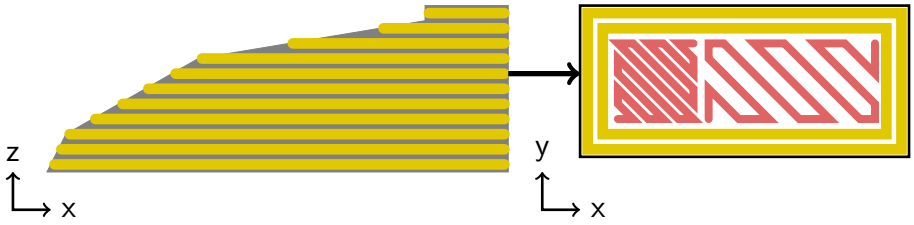


Figure 4.4: The filled layers where the top and internal solid surfaces are filled with 100% and the infill surface is filled with 40%. Each layer is filled according to the classification of the previous step. One example layer is shown in top-down view on the right.

is described here. The fill direction alternates by 90° on every layer; usually, 45° and 135° are used. On bridges, the bridge angle is used as fill direction. For the actual fill generation, the polygon is rotated by the fill direction angle to get lines along the y-axis while generating the pattern. This makes the actual pattern generation simpler than with a tilted pattern. To get a pattern with the desired density, the distance between the individual lines which leads to this density has to be calculated. Then the bounding box of the polygon is calculated and rectilinear lines with the calculated distance are generated inside the bounding box along the y-axis. All intersections between the polygon and the lines are searched to get only the lines inside the polygon. The intersection points of each line are sorted descending by their y-position and grouped into pairs. Each pair forms a line with an upper and a lower endpoint and lies inside the polygon. To generate a continuous path from the independent lines, they are chained together. The upper endpoint of a line is connected to the nearest upper endpoint of another line. Then, the lower endpoint is connected to the nearest lower endpoint of another line. This process is repeated until each line is part of the path. The surface is now filled with lines which are connected with a zigzag pattern. Since it is not possible to connect all lines with a continuous path, travel moves are added when two points of a chained path do not lie directly next to each other. The generated toolpath is rotated back into its original position to get the toolpath back in the orientation of the layer. Then, extrusion entities are generated from the path and appended to the fill extrusion objects of the actual layer. Figure 4.4 shows a filled layer with a tilted rectilinear pattern. The top surface is filled with 100%, while the internal surface is filled with 40% material.

4.1.5 Support Generation

Since some objects might need support material to be printable, it must be generated during the slicing process. The support is generated for each object separately. To generate useful support, the areas which need it have to be found.

Finding Overhangs

Whether support is needed on an overhang surface or not is defined by a threshold angle. A right triangle with the threshold angle is formed between the previous and the current layer to find the area where the angle is greater (figure 4.5). Every part of the surface which is greater than the triangle also has a greater angle. The length of the upper side of this triangle d can be calculated with the following equation where h is the layer height and θ is the threshold angle.

$$d = h \cdot \tan\theta \tag{4.1}$$

To get the actual areas where the support is later generated, the previous layer is offset by the calculated d and then subtracted from the current layer. All areas that remain have a higher angle than the threshold and need to be supported.

Since supported bridges usually do not need additional support, these areas are removed from the found surfaces. When the support is set to build-plate-only, the top surfaces of all layers below the current layer are merged and removed from

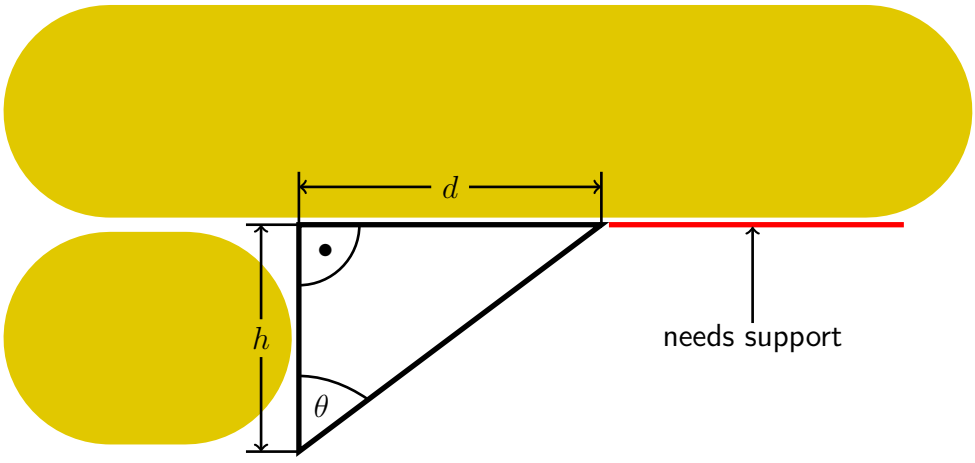


Figure 4.5: The right triangle which shows the maximum support angle θ, the layer height h, the maximum overhang distance d, and the area that needs support.

the potential supported areas. The remaining area is offset by half an extrusion width to support perimeters properly. This offset area is set as contact surface and stored for further support generation.

Finding Contact Areas

As long as the option build-plate-only is not selected, the top surfaces where the support is attached are searched. The algorithm iterates through all layers from top to bottom to find these contact areas. Each time an overhang surface is found it is added to the supported areas. These areas are intersected with the top surfaces of each layer. When they intersect, the intersection is stored as a top contact area and is removed from the support area. When the supported area is empty, or the first layer is reached, the top contact layers are all found.

Support Generation

When the overhangs and the top surface areas are found, support structures can be generated between these two. First, support layers are generated by taking the height of the overhang surfaces and the top contact surfaces and adding more layers in between. To generate the interfaces below the overhang surfaces, they are propagated down to the layers below. The number of propagated layers depends on the desired interface thickness. Since the down-propagated layers could intersect with the object, its silhouette is removed from the interface layers. The interface layers are merged with already generated support surfaces and contact areas and are projected downwards to generate the support surfaces below the interface. Since these support surfaces should not intersect with the already found top contact surface, interface surfaces, and overhang surfaces, they are removed before adding them to the support layers. To get interface surfaces above top contact surfaces, all areas above these are marked as bottom interface areas. The generation of the support layers and its surfaces is now finished and they are added to the actual object. The toolpaths are then generated for all surface types. The actual path generation of the support surfaces is pretty similar to the toolpath generation of regular layers but with slightly different patterns.

4.1.6 Skirt and Brim Generation

Since a good print relies heavily on a primed extruder with a constant flow, a skirt is printed around the object to have a constant extrusion when the actual model is printed. The skirt's job is to prime the extruder before printing and it is thrown

away after printing. It is possible to generate a skirt that has the same height as the object, but usually, a skirt is only generated for the first layer. All points of all polygons from every layer where a skirt is generated are collected from all objects on the build plate to generate the skirt without colliding with any printed object. Then a convex hull is calculated from all points. This convex hull represents the outline of all objects on the print bed. To print the skirt with distance to the objects, the convex hull it is offset. When more than one skirt line should be generated it is offset again. To later print the skirt, extrusion objects are generated for the offset path. This process is repeated for every layer until the maximum height is reached. When a minimum skirt length is defined, it is generated until the desired length of the skirt is reached. Extrusion loops are generated from the paths to print them later.

A brim is a specialized skirt which is attached to the printed object. Its primary purpose is to hold the part down to the print bed. This minimizes warping and increases the overall adhesion. A brim is only created on the first layer of the print. Since all objects get a brim when activated, their contours of the first layer including support are collected. Then multiple offsets are generated from these contours until the brim has the desired width. The loops are chained together when they intersect to get rid of overlapping brim. The brim is now successfully generated and extrusion loops are generated from the paths.

4.1.7 G-code Generation

The toolpaths for all objects are now generated. To be able to print the toolpath on a 3D printer, the extrusion objects have to be translated into G-code. The generation is started when the user decides to save the G-code, and not directly after the toolpath generation. First, the commands for setting the extruder and bed temperature are written to the file. The defined start G-code is also written to the file. If a skirt or brim exists, these are processed first. Then the toolpaths of the actual objects are processed. To minimize traveling moves between the objects, they are sorted by their nearest neighbor. Then all layers of all objects are sorted by their print height. Next, the layers are processed from bottom to top and in the sorted object order. For each object, the support G-code is generated first to prevent oozing on already printed areas. Then the G-code for the individual object layers is generated. On every layer change, the layer change G-code is inserted into the file. All layers are processed by the following steps:

All extrusions inside a layer are grouped by the extruder to minimize tool changes. The lastly used extruder is used first. Then the toolpaths are grouped by independent islands for every extruder. For each island, the perimeter is printed first and then the infill if not configured differently. The perimeter is made out of closed loops. They are cut open either on the same position for every layer or close to the previous extruder position. The fill paths are ordered by a greedy algorithm to minimize travels. The algorithm always takes the nearest path to its current position. When a path is marked as reversible, it is also possible that the algorithm takes the end of a path instead of the start and reverses the path. The paths are chained until no path is left. All extrusion operations are now paths with different speeds and material flow.

Since some paths can be unnecessarily complicated, each path is first simplified with the Douglas-Peucker algorithm [Douglas and Peucker, 1973]. The algorithm removes points in the path which do not change the overall geometry of the object. The simplification is done by recursively analyzing the furthest point between the start and the endpoint of a path. If the furthest point is closer than a given threshold, only the start and endpoint are kept. Otherwise, the path is split into two sub-paths and the algorithm is called recursively. When the algorithm terminates, the now simplified path is put back together.

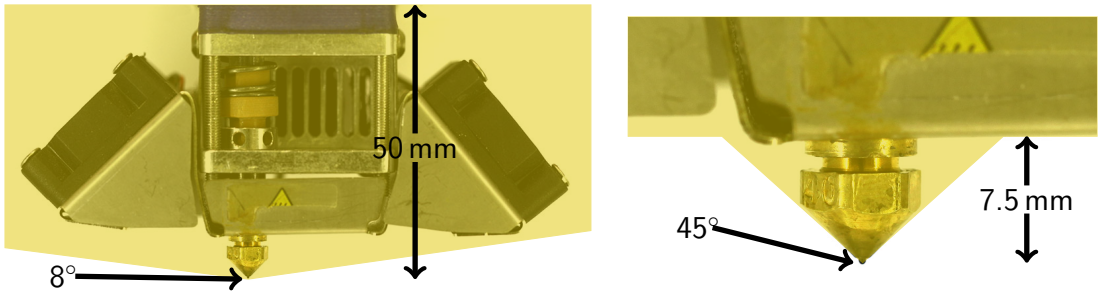
Next, the extrusion paths are converted into G-code. First, the G-code to move to the first point of the path is written to the file. Then the retraction is compensated if the extruder is retracted. Next, the acceleration speed for the current path is set in the G-code. When cooling is considered necessary, they are turned on. Then the different points of the path are added one after another. The extrusion volume is calculated by the length between the current position and the next planned position multiplied by the desired material flow. When the last point of a path is reached, the extruder position is updated and the G-code is written to the file. The speed of the whole layer is lowered when the minimum layer time is higher than the actual print time to give each layer enough time to cool properly. When all layers are generated, the end G-code is written to the file and the file handler is closed. The time and material usage estimation are shown in the interface to give the user some feedback. The desired path can now be printed with a 3D printer.

4.2 Hardware Limitations

When printing only planar layers, there should not be any collisions between the printhead and the print. This is because the print process strictly prints the layers from bottom to top without ever traveling back to a layer that was printed before. The only exception from this is when parts are printed one after another. The bounding box of the printhead, as well as the maximum height, is defined in the printer configuration to prevent collisions. The printhead sometimes collides with the previous layer which unintentionally warped up while cooling down. The slicer cannot prevent this collision because they cannot be calculated. Warping can be prevented for example by increasing the adhesion to the print bed with a brim.

With nonplanar layers involved, the printhead must travel down into regions with already printed structures. This can cause collisions with parts of the printhead. This could be the nozzle, fans, the extruder body or bed level sensors. The algorithm needs further information about the printheads geometry to prevent the collisions between the printhead and the previously printed structures. The more detailed this information about the printhead is, the better the slicer can decide if a nonplanar layer can be printed without collisions. One approach would be to include a full model of the printhead to check for collisions. These checks would be rather complex because of the many different shapes of the extruder model. Furthermore, each printer has a different printhead so a complex model would look different on every printer.

A simple parametrized print head model would be much easier because it could be measured and stored in the configuration of the slicer. The individual values can be measured on every printer model. The simple extruder model contains only two variables, first the maximum printing angle and second the maximum height that can be printed with a nonplanar layer. The maximum printing angle is the angle which the printer can print without collision in any direction of the printhead. The whole printhead has to be collision-free in this angle until the maximum height is reached. Figure 4.6(a) shows the maximum printing angle and the maximum height on the printhead of an Ultimaker 2. This maximum height is necessary to prevent collisions with parts of the motion system which do not move along with the printhead like the rails of the x-axis. These values can either be used to print small angles with a great height difference (Figure 4.6(a)) or to print high angles



(a) (b)

Figure 4.6: The collision model of the Ultimaker 2 either (a) taking the whole printhead into account with an 8° angle and 50 mm maximum height or (b) taking only the nozzle into account with an 45° angle and 7.5 mm maximum height. With these configurations, either large surfaces with a small angle or small surfaces with a large angle can be printed.

on a small height difference by only using the nozzle as a collision model (Figure 4.6(b)). The collision model forms a large cone with the angle that is in any case smaller than the ramp angle of the nozzle.

On most printers, the collisions are probably not caused by the nozzle or the heater block but by the attached fans or bed level sensors. When designing a printhead specifically for nonplanar printing, these parts can be designed so they would not cause any collisions. With dual nozzle printers, nonplanar printing is not possible on most printers because the second nozzle sits on the same height as the first one. This will always cause collisions on the slightest nonplanar surface angles.

4.2.1 Nozzle Geometry

Nozzles come in different forms and sizes. Some have an internal thread and some have an external thread. Nearly all nozzles are shaped like a pointy cone with a hole at the tip. Common nozzle types are the E3D nozzle (figure 4.7(a)) and the Olsson Block nozzle (figure 4.7(b)). There is a wide variety of nozzles, but mostly they all look similar. As [Jin et al., 2017] mentioned, nozzles are not entirely pointy. In fact, they have a flat surface at their ends. In planar printing, this surface has two purposes. First, it makes the nozzle more robust since the walls around its hole are more solid. Second, this flat surface squishes down the printed path to make it flatter. The surface is essentially ironed with the nozzle.

(a) E3D (b) Olsson Block (c) Fictional

Figure 4.7: Different nozzle geometries with differently formed tips. The pointier a tip is the better the result of the printed surface.

On planar layers, this results in a slightly better surface quality but can sometimes be an issue since more heat is applied to the previously printed surface. This heat can melt small printed structures. Most nozzle manufacturers do not publish any data about the size of the flat area at the tip of the nozzle. Only E3D has defined that the plateau size is 2.5 times the nozzle diameter. The Olsson Block nozzle of the Ultimaker 2 was measured with a flat area diameter of 0.8 mm on a 0.4 mm nozzle.

When printing nonplanar layers, the flat surface will melt previously printed areas while driving through them with the hot nozzle. This will result in a rough surface finish especially on surfaces with a high angle. As [Jin et al., 2017] already mentioned, there are also problems while printing upwards or downwards a slope. The nozzle is either a bit too high while printing upwards or too low while printing downwards. With a thin layer and a high printing angle, the nozzle will push the complete trace away while printing or sometimes even damage previous layers. Jin et al. also presented a method to compensate this penetration by adding additional height while printing, but this will also result in a rough surface because the extrusion line is not laid down constantly. The depth of the nozzle penetration p can be calculated with the following equation where D is the diameter of the flat surface and θ is the angle of the printed slope.

$$p = \frac{D}{2} \cdot \sin\theta \tag{4.2}$$

Using the Olssen Block nozzle with the 0.8 mm flat surface, the penetration depth evolves as shown in figure 4.8. This graph shows that it is better to use thicker layers with higher printing angles because the extrusion would otherwise collapse when the nozzle penetrates too deeply. However, the stair-stepping is much worse on surfaces with an angle below 10° where the penetration is relatively low. It might be a good idea to restrict the maximum ramping angle for a print even if the printer can print higher ramping angles to get the smoothest surface possible. The optimal nozzle for printing nonplanar layers is rather long and pointy and has a flat surface diameter very close to the actual nozzle diameter. This nozzle would be very fragile due to its thin tip. A sketch of this fictional nozzle can be seen in figure 4.7(c).

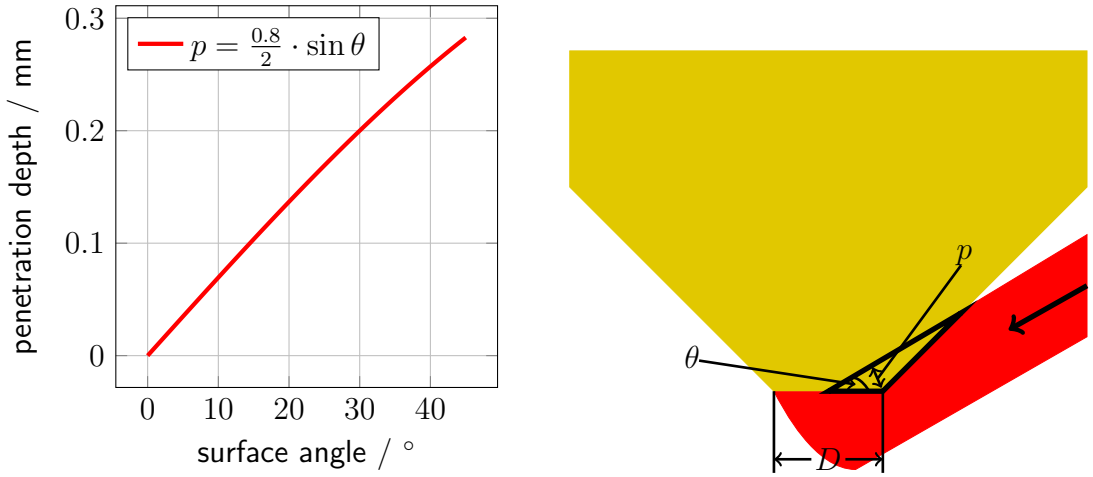


Figure 4.8: The penetration depth of the Olssen Block nozzle with 0.8 mm flat surface on its tip. The penetration problem is visualized on the right where the nozzle penetrates the path while printing.

4.3 Nonplanar Toolpath Generation

In this section, the toolpath generation of nonplanar layers is described. The individual steps are included in the planar slicing process described in section 4.1. The printable surfaces are searched in the STL model to generate the toolpaths for the nonplanar layers. Then the surfaces are checked for potential collisions. Next, the nonplanar surfaces are generated within the layer structure. Then the toolpaths are generated for the surfaces and projected according to the nonplanar geometry. Lastly, the G-code is generated from the toolpaths. The toolpath visualization of Slic3r is modified to be able to display 3D toolpaths.

Figure 4.9: The nonplanar printable surfaces (red) that are found and grouped from the model's mesh.

4.3.1 Identifying Printable Nonplanar Surfaces

To generate nonplanar toolpaths, nonplanar surfaces are searched in the facet mesh. This step is performed after the layer generation of the planar slicing process. The previously defined hardware limitations are the maximum printing angle and the maximum height. All facets from the provided STL that should belong to the nonplanar surface have to be within this limitation. Since every facet has a normal which represents the orientation of the facet, the z-component of it can be used to calculate the angle of the facet relative to the z-axis. First, all facets that meet the criteria $normal.z \geq \cos\theta$, where θ is the threshold angle, are stored as potentially printable facets. The x- and y-components of the normal can be entirely ignored because the threshold angle has to be valid in all directions. The potentially printable facets are stored as `NonplanarFacets`. This is a facet with an additional bounding box and the calculated facet surface area. A `NonplanarFacet` also provides scale and rotate functionality which is needed when the user scales or rotates the model.

Connected components are searched to get surfaces from the individual facets. The algorithm takes the first facet, marks it and all its neighbors, then their neighbors are marked as well. This is repeated until no more unmarked neighbors can be found. The marked facets form a connected component and are removed from the facet vector. The process is repeated with the next facet in the vector until every facet belongs to one connected component. The pseudo code can be seen in algorithm 1. All facets of a connected component are then added to a `NonplanarSurface`. A `NonplanarSurface` also has a bounding box and a calculated surface area. It provides rotation and scale operations like the `NonplanarFacet` as well as a horizontal projection of the surface. Each `NonplanarSurface` now represents a connected component and is further called nonplanar surface. The nonplanar printable surface can be seen in figure 4.9.

Algorithm 1 Find all connected components and group them into NonplanarSurfaces.

```
1:  function GROUP_SURFACES(facets)
2:      if facets empty then
3:          return
4:      end if
5:      MARK_NEIGHBOR_FACETS(facets.first)
6:      for all facets do
7:          if marked then
8:              add to NonplanarSurface
9:              erase from facets
10:         end if
11:     end for
12:     return NonplanarSurface and GROUP_SURFACES(facets)
13: end function
```

```
1:  function MARK_NEIGHBOR_FACETS(facet)
2:      if marked then
3:          return
4:      end if
5:      for all facet.neighbors do
6:          MARK_NEIGHBOR_FACETS(neighbor)
7:      end for
8:  end function
```

All nonplanar surfaces are now below the maximum printing angle because none of their facets are exceeding this limit. To meet the maximum printing height, all nonplanar surfaces are removed where the distance between the highest and the lowest point exceeds this limit. All remaining nonplanar surfaces are printable. Printing tiny areas does not improve the surface quality since the nonplanar surface would only contain a single outline with no infill. Therefore all surfaces where the area is below $20\,\mathrm{mm}^2$ are removed from the nonplanar surface vector.

The nonplanar surfaces are by definition collision-free for the printhead. Each surface is checked for collisions to ensure that the printhead will not collide with the surrounding layers that are not part of the nonplanar surface. When a surface causes a collision, it is removed from the nonplanar surface vector. The collision detection is further described in section 4.3.2. All remaining surfaces that meet all the criteria are printable and collision-free. A toolpath is generated for each of them in the following steps.

4.3.2 Collision Avoidance

While printing only planar layers, there are no collisions because the printhead never travels below the current printing layer. When printing nonplanar layers the printhead drives down into already printed layers. This can cause collisions with the printhead. To prevent these, the surface which causes this collision is not printed nonplanarly. Collisions can be found by checking on the complete printing path for when the printhead would collide with some already printed structures. The printhead is abstracted as a big pointy cone with a maximum printing angle and a maximum height. Since the nonplanar surface is by definition collision-free, the only parts which have to be checked are the contour path of the nonplanar surface. The printed object should only be checked up to the maximum height of the nonplanar surface since the area above is built after the nonplanar surface is printed.

Collision checks on two 3D shapes are not trivial. The following paragraphs present three different ways to check for collisions. First, the octree collision check, second the intersection of facets collision check and lastly the layer based collision check. The layer based collision check is the one that is used in the implementation.

Octree Collision Check

To check collisions with an octree a collider and the object are needed. The collider has to represent every possible position of the printhead while printing the object. This is, as mentioned before, the outline of the nonplanar surface. The collider is generated by building a Minkowski sum of the surface and the simplified pointy printhead cone. A Minkowski sum is built by sweeping one object along the borders of another and generating a hull from all positions. Figure 4.10 shows the simplified printhead, the nonplanar surface, and the generated Minkowski sum. An octree is built around the object between the minimum and the maximum nonplanar surface height to check for collisions. An octree represents a volume that is divided into eight equally sized cubes and each cube can be further divided into eight cubes. This forms a tree where every node has eight or zero children. Then the collider is added to the octree. Every cube of the octree is then examined, whether both objects are part of the cube. If not, the next cube is checked. Otherwise, the cube is split up into eight new cubes and the check is repeated for each one of them. There are no collisions when both the object and the collider occupy no cubes. There are collisions when both occupy cubes and the minimal cube size is reached.

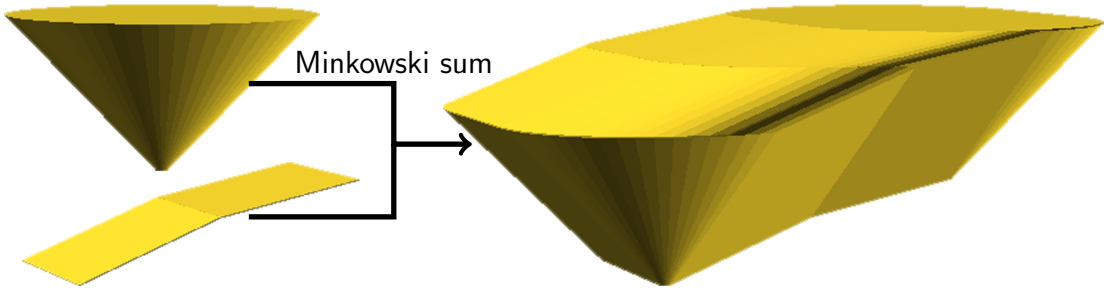


Figure 4.10: The simplified printhead defined by the maximum angle, the nonplanar surface that needs to be checked for possible collisions, and the Minkowski sum which is generated out of both.

The calculation can be aborted when the first collision is found. Since the collider will always collide with the nonplanar surface, this area has to be excluded from the calculations. The precision and the necessary calculations rely on the minimal cube size. To implement this method into Slic3r, the octree, the Minkowski sum, and the check if a cube is occupied have to be implemented or added by an external library.

Intersection of Facets Collision Check

Another way to check for collisions is to find out if the facets of both the collider and the object intersect at any point. The Minkowski sum collider is again used in this method. Both the collider and the object have to be represented as an STL. The object is cut above and below the minimal and maximal surface height to prevent checking for collisions above the actual nonplanar surface. Then every facet of the collider is checked against every facet of the object. If any two facets intersect, there is a collision; if not, there is none. This method is very precise but has a relatively high complexity of $O(n \cdot m)$ where n is the number of facets in the model and m is the number of facets in the collider. The computation time can be reduced by simplifying a complex model before checking for collisions. To implement this method into Slic3r the Minkowski sum and a facet intersection check have to be implemented or added by an external library. Cutting a model on a given height is already implemented and can be reused.

Layer Based Collision Check

A 3D printed object is not really printed three-dimensionally. In fact, the areas where the material is laid down are stacked 2D layers. Each of this layers can be checked for collisions to get the collisions on the whole toolpath. This results in a

bunch of simple 2D collision checks instead of one complex 3D collision check. To check for collisions, first, a horizontal projection of the nonplanar surface and an empty collider polygon is generated. Then the algorithm iterates through all layers between the minimum and the maximum nonplanar surface height. On each layer, it is checked if the collision polygon collides with anything from the surface polygon by calculating an intersection between these two. The projected nonplanar surface is excluded from this check because it would generate a false positive. When the intersection is not empty, there is a collision; otherwise, continue. On each layer, the intersection will be checked against the collider from the previous iteration. The first layer is checked against an empty collider since there can be no collisions on the first layer because the nozzle always stays above this layer. The potential top surfaces are calculated by the difference between the current and the next layer. Then the intersection between the nonplanar surface projection and the potential top surfaces is calculated with a polygon intersection. This intersection is added to the collider. Then the collider is offset by the maximum radius that the printhead collider would gain within one layer. This radius r is calculated with the following equation where h is the layer height and θ is the maximum printing angle.

$$r = \frac{h}{\tan\theta} \tag{4.3}$$

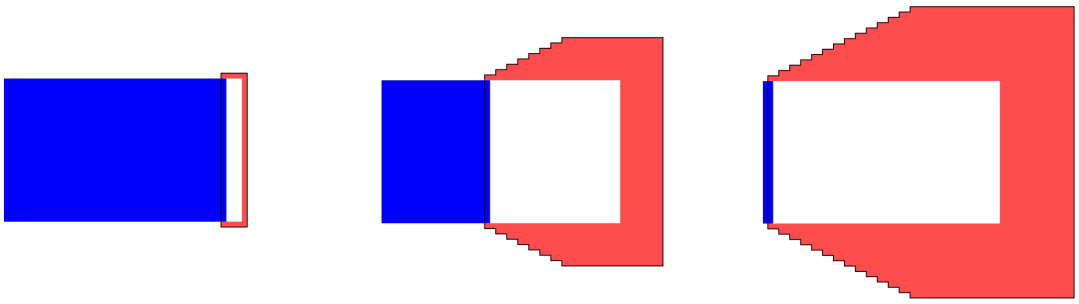
The collision check of the layer is now completed and is repeated with the next layer. The collider grows on every layer since it is offset in every iteration. The pseudo code of the algorithm is shown in algorithm 2. Three layers with their collision polygons can be seen in figure 4.11. Since all calculations are simple polygon operations like offset, intersection, and difference, it is relatively inexpensive to compute the possible collisions. The polygon operations are already present in Slic3r, so no additional libraries have to be added. This method is implemented because of its simplicity and because there is no need for new features.

4.3.3 Surface Generation for Nonplanar Layers

The found nonplanar surfaces are still only facet meshes. Now they have to be converted into areas on which a printable toolpath can be generated. The idea is to keep the planar layer structure of the object and print nonplanarly in the previously identified regions. The full object is sliced with the regular layer generation step as described in section 4.1.1 to generate the planar layers. The layer slices now also include the areas that should be printed nonplanarly. The top and

Algorithm 2 The layer based collision check.

```
1:  function CHECK_NONPLANAR_COLLISIONS(surface)
2:      Polygon collider
3:      Polygon nonplanar_surface = NONPLANAR_PROJECTION(surface)
4:      offset = layerheight / tan θ
5:      for all layers between surface.min and surface.max do
6:          layer_collider = DIFFERENCE(collider, nonplanar_surface)
7:          if INTERSECTION(layer.surface, layer_collider) ≠ empty then
8:              return collision
9:          else
10:             potential_top = DIFFERENCE(layer.surface, upper_layer.surface)
11:             new_collider = INTERSECTION(nonplanar_surface, potential_top)
12:             collider = OFFSET(UNION(collider, new_collider), offset)
13:         end if
14:     end for
15:     return no collision
16: end function
```



(a) (b) (c)

Figure 4.11: The generated collision polygon (red) of (a) the first, (b) the sixth and (c) the last layer of the nonplanar extrusion. The object outline that is checked for collisions is blue, the white area in between is from the nonplanar surface and is excluded from the collision check. The collider grows on each layer because it is constantly offset. The object can be seen in 5.1(b).

shell areas are removed and replaced by nonplanar layers to generate the needed space for the nonplanar extrusions. The replacement is done by first finding the layer where the nonplanar surface should be generated. This is achieved by iterating through all layers starting from the top and stopping when the height of the layers is smaller or equal to the height of the nonplanar surface. This layer is then marked as the home layer and the nonplanar surface is attached to the layer's `NonplanarSurface` list. The next step is to remove the top surfaces from the

lower layers and adding them to the home layer. This is done by iterating through all layers that lie between the minimum nonplanar surface height subtracted by the thickness of one layer and the home layer. In each layer, the potential top surfaces are taken by calculating the difference between the current layer's surface and the surface of the layer above. Then, the intersection between the potential top surfaces and the nonplanar projection is calculated (figure 4.12(a)). This intersection is removed from the current layer and added to the home layer (figure 4.12(b)). This will generate enough room for the nonplanar surfaces and will also

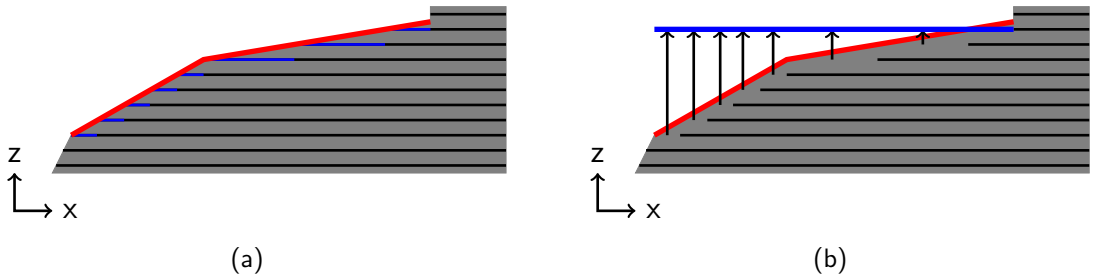


Figure 4.12: (a) The top surfaces (blue) which lie below the nonplanar surface(red) are (b) moved to the home layer of the nonplanar surface. The home layer is the first layer from top that lies below the nonplanar surface.

generate their surface in the layer structure. The moved surface is marked as `stTopNonplanar` to enable printing the toolpath generated from this surfaces with different parameters than other parts of the object. All remaining surfaces of the current layer are again marked as `stInternal` as they were before. The process is repeated until all layers in the given range are processed. Since usually top surfaces have more than one shell, the nonplanar surface should also have more than one shell. The previous projection process is repeated with the home layer shifted one layer downwards to create these additional surfaces. The moved surfaces are marked as `stInternalSolidNonplanar` since they are usually printed differently from the topmost layer. For later calculations, each layer where the surfaces are moved up has to store the distance to the top layer. For the topmost layer, this is 0, for the next shell layers, the thickness of the previous layers is added up. The home layers now contain the surface area of the nonplanar surface. The layers are floating above the other layers only with contact to the planar layers where the nonplanar surface is the highest. The surface contains multiple small surfaces. They are merged to generate toolpaths in the next step.

The planar surface preparation continues as usual. While identifying the top sur-

faces, it is essential to remove everything below the nonplanar surface from the potential top surfaces. Otherwise, the slicer will generate top surfaces below the nonplanar surfaces. To not exclude all top surfaces in this region, this only applies if the potential top surface lies between the minimum and maximum height of the nonplanar surface. It is important that the nonplanar surfaces which are already classified as `stTopNonplanar` and `stInternalSolidNonplanar` remain with this classification and are not classified as anything else.

4.3.4 Toolpath Generation for Nonplanar Layers

The surfaces of the planar and nonplanar layers are now entirely generated. The filling of the planar surfaces is done as usual. Until now, the nonplanar surfaces are also planar surfaces that float above their intended position in mid-air. The idea is to generate a regular planar toolpath for the nonplanar surface and project it downwards regarding the actual surface geometry. So, a 2D toolpath is created and changed to a 3D toolpath by adding the z-coordinate to each point. The perimeters and the fillings are generated as shown in section 4.1. Figure 4.14(a) shows the generated but unprojected toolpath. It is possible to generate the toolpaths for the nonplanar surfaces with a different algorithm that follows the surface geometry while generating the toolpath. But then it would be harder to print mixed layers where parts of the toolpath are planar and parts are nonplanar. A major drawback with the projected toolpath is that the filling cannot react to the geometry of the surface. This can be a problem on concave or convex surfaces and leads to over- and under-filling of those regions. The problem is further described in section 4.4.2.

Toolpath projection

The generated toolpath is now a regular 2D toolpath. To get a 3D toolpath from it, it is projected downwards regarding the facet geometry. Perimeters are extrusion loops which are a special case of an extrusion path where the start and the endpoint are the same. The filling is generated out of extrusion paths. All extrusion paths are polylines with some print specific additions like extrusion width, height and some more. A polyline is a multipoint which is a vector of points. Each of these points has an x- and y-coordinate which represents a point in the print volume. All coordinates in the print volume are represented as integer values. To ensure the necessary sub-mm precision, all float values are multiplied by 100.000 before

they are converted to integers. This new coordinate system is called the scaled coordinate system. All operations while slicing are done in this coordinate system. When generating G-code from the toolpaths, the values are calculated back into unscaled float coordinates. An extrusion path is basically an array of 2D points. To get the third dimension to the points, they get an additional z-component. To further use the current implementation of the 2D point, the z-component is set to a default value of -1. Every time the z-component is not needed, it belongs to -1.

The toolpath is projected downwards to get the path to follow the geometry of the nonplanar surface. On each layer where a nonplanar surface is attached, every point in every toolpath is processed. Since the z-coordinates of the extrusions represent the position which the nozzle would reach, the z-component of the points has to be set to the same height that the nonplanar mesh has at this position. This is done by first searching the facet of the nonplanar surface in which each point lies. So each point is checked against each facet which leads to a complexity of $O(n \cdot m)$ where n is the number of facets and m is the number of points in the extrusion path. To only check facets which lie close together, it is first checked if the point lies inside the bounding box of the facet. When the point lies inside the bounding box, there is still a 50% chance that the point does not lie inside the facet. To test that, the point in facet test is performed.

Point In Facet Test

The same side test is performed to check if a point lies inside a facet. The facet edges can be defined as three vectors: AB, BC, and CA. When a point is on the same side of all of all three vectors, it must be inside the facet. When the point lies on the other side of only one vector, it is not in the facet. To check on which side of the vector AB a point P lies, the cross product $AB \times AP$ is calculated. When the cross product is greater than 0, the point P lies on the right side of the vector AB; otherwise, it lies on the left. A point P is inside the facet if all cross products are greater than 0 or all smaller than 0. Since this algorithm just contains multiplications and subtractions, it is very efficient.

With the correct facet, the point needs to be projected onto the facet to get the correct z-height. When the point is interpreted as a vertical line, this can be done with a line plane intersection. The line plane intersection works with an infinite plane so it is important just to project points that are already checked to be inside the facet because otherwise, it would create false hits.

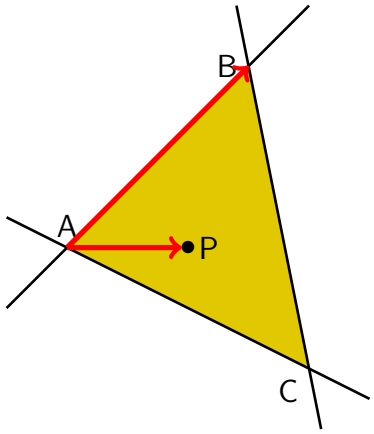


Figure 4.13: The facet with its three edges, each described as a vector. The cross product AB and AP is used to calculate if the point is either on the left or on the right side of the triangle. A point is inside the triangle when it lies on the same side of every edge vector.

Line Plane Intersection

For the line plane intersection, the plane has to be converted into the following from:

$$ax + by + cz + d = 0 \tag{4.4}$$

where a,b and c are the x-, y- and z-components of the surface normal and d is defined as follows:

$$d = -(N \cdot V) \tag{4.5}$$

where N is the normal vector and V is any vertex from the facet that is currently checked.

The intersection point of the line will now be calculated. Since the x- and y-component does not change with a vertical intersection line, only the z-component of the point $(P.z)$ has to be calculated with the following formula:

$$P.z = P.z + u \cdot (L2.z - L1.z) \tag{4.6}$$

where $L1$ is the position of the point that should be projected and $L2$ is the same position shifted upwards by one mm. This shift by one mm is used so that $L2.z - L1.z$ evaluates to one and can be ignored. The previously not mentioned component u is defined as follows:

$$u = -(N \cdot P + d)/N.z \tag{4.7}$$

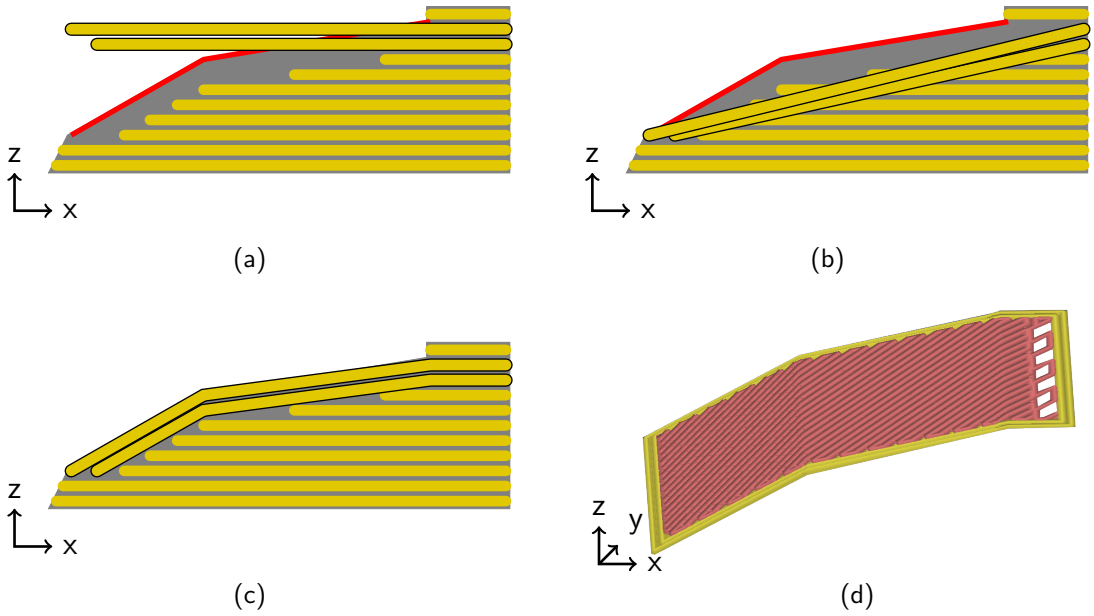


Figure 4.14: (a) The unprojected planar toolpath, (b) the points are projected downwards, (c) the line intersections are added, and (d) the projected example toolpath of the upper nonplanar layer.

where N is again the normal vector and P is the point that should be projected. The generated point stays unchanged in x- and y-axis. Only the z-component is changed to the position where the point is projected onto the plane which represents the facet.

To ensure that the interface layers are placed lower than the top shell layers, the found z-component is lowered by the distance to top, which was saved earlier. All points that did not get projected and still have a z-component at -1 are set to the current layer z-height. The x- and y-components of all points stay unchanged during the whole process. The toolpath with the projected points can be seen in figure 4.14(b).

The projected path does not follow the geometry yet, because an extrusion line takes the shortest path between the start and endpoint which usually goes from one side of the object to the other. This can be seen in figure 4.14(b). On nonplanar surfaces with more than just a flat surface, this does not represent the surface geometry and will also collide with the internal planar layers. The start and end points are already on their correct position. Each line has to add additional points every time the surface changes its direction to follow the actual

geometry of the nonplanar surface. These direction changes only occur along the edges of the facets. Within a facet, the surface is flat and the direction is constant. To get the path to follow the contour it has to be split up on every intersection with a facet edge from the nonplanar surface. Every line of the print path has to be checked against every edge of every facet of the nonplanar surface to get these intersections. This check has the complexity of $O(n \cdot m)$ where n is the number of facets and m is the number of extrusion lines. Since the path is projected along the z-axis, the intersection test can be simplified by only checking in 2D space and ignoring the z-component. The z-component is then added when the intersecting lines are found. The two-line intersection test is performed on every possible pair of lines.

Two-Line Intersection

The two-line intersection algorithm needs two lines which should be checked. They are described by four points $P1$, $P2$, $P3$, and $P4$ where the first two represent the line from the facet and the last two represent the extrusion line. The intersection point P of the two lines can be calculated as follows:

$$P.x = P1.x + u_a(P2.x - P1.x) \tag{4.8}$$

$$P.y = P1.y + u_a(P2.y - P1.y) \tag{4.9}$$

where the following formula defines u_a:

$$u_a = \frac{((P4.x - P3.x) \cdot (P1.y - P3.y)) - ((P4.y - P3.y) \cdot (P1.x - P3.x))}{((P4.y - P3.y) \cdot (P2.x - P1.x)) - ((P4.x - P3.x) \cdot (P2.y - P1.y))} \tag{4.10}$$

$$u_b = \frac{((P2.x - P1.x) \cdot (P1.y - P3.y)) - ((P2.y - P1.y) \cdot (P1.x - P3.x))}{((P4.y - P3.y) \cdot (P2.x - P1.x)) - ((P4.x - P3.x) \cdot (P2.y - P1.y))} \tag{4.11}$$

u_b is not needed for the calculation of the intersection point but for checking if the intersection happens on those two lines later. When the two lines are parallel and have therefore no intersection the denominator of the equation is zero. This has to be checked before calculating u_a and u_b because otherwise, it would be a division by zero. Since two endless lines always intersect unless they are parallel, it has to be checked if the found intersection lies between the start and end points of both lines before calling it an intersection. This is the case if both u_a and u_b are in the interval $[0, 1]$. When this is not the

case for just one of them, the two lines intersect, but not in between the start and end points of both lines. This is not an intersection and is not further handled. The equations 4.8 and 4.9 just calculated the x- and y-position of the point because this is a test in 2D space. Since the projection happens on a vertical axis, this is not a problem. The z-component $P.z$ of the point can be calculated by the following equation:

$$P.z = P1.z - \left(\frac{Dist(P1, P)}{Dist(P1, P2)} \right) \cdot (P1.z - P2.z) \tag{4.12}$$

where $Dist()$ is the Euclidean distance between these two points in 2D space. The intersection is now completely calculated.

When the intersection test is performed with all facets of the mesh, all intersections are stored in the potential intersection vector. Like with the projected points, the distance to top is subtracted from the z-component if the toolpath belongs to an inner shell. The intersection points are usually not in the correct order because it relies on the order of the facets. The points must be stored in the correct order to get a constant extrusion. They are sorted in the same direction where the previous line was heading. Each inner facing edge of a facet is twice in the mesh because each facet is described by all three edges and is independent of the other facets. These duplicate edges create duplicate intersection points which have to be eliminated. Because the points where sorted beforehand, the same points always lie next to each other. Each point is compared to the next point and is deleted if they match to eliminate the duplicates. This leaves valid and unique points behind. They are added between the start and end point of the initially checked extrusion line. This forms a bunch of new lines which all follow the nonplanar surface mesh. When all lines are processed, the complete toolpath follows its contour. The toolpath no longer floats in mid-air but lies directly above the planar layers. It seems easier to replicate the toolpath of the top layer to the inner shell layers. This is not possible since the lower layers do not have the same geometry. Furthermore, usually the angle of the filling changes in every layer. It is necessary to project each toolpath separately. Figure 4.14(c) shows the fully projected toolpath. Both planar and nonplanar layers are now entirely filled with printable toolpaths.

4.3.5 G-code Generation for Nonplanar Layers

The G-code generation now chains the independent toolpaths together to a continuous 3D printing path, calculates the extrusion amounts and generates the actual G-code which can be executed by the 3D printer. The whole process is pretty similar to the planar one. Most parts of it can be reused as they are. The nonplanar layers are located on the layer where the highest point of the nonplanar surface is. So they are handled as they were on this layer. To be able to generate nonplanar extrusion G-codes, the z-component has been added where it is present. When the z-component is not set, the point is part of a planar extrusion and the layer print height is used as the z-component. While extruding, no collisions can occur since this was already checked in the collision avoidance (section 4.3.2). Travel moves, on the other hand, can cause a collision even if the target is lower than the starting point of the travel move. This is because travel moves can go across the whole object with obstacles in between. Figure 4.15(a) shows a travel move with a potential collision. To avoid collisions, every travel move that starts or ends on a height lower than the current layer height needs special treatment. Instead of moving directly from the current point to the target point, the printhead is first moved straight up. Then it moves to the x- and y-position of the target point and when it reaches that, the printhead moves down to the desired z-position. The collision-free travel path is shown in figure 4.15(b). With the new travel path, the printhead avoids crashing into any obstacles on the travel path. To prevent this up-and-down movement on extremely short travels, like when switching from the perimeter to the infill, this is only performed when the travel is longer than 1 mm.

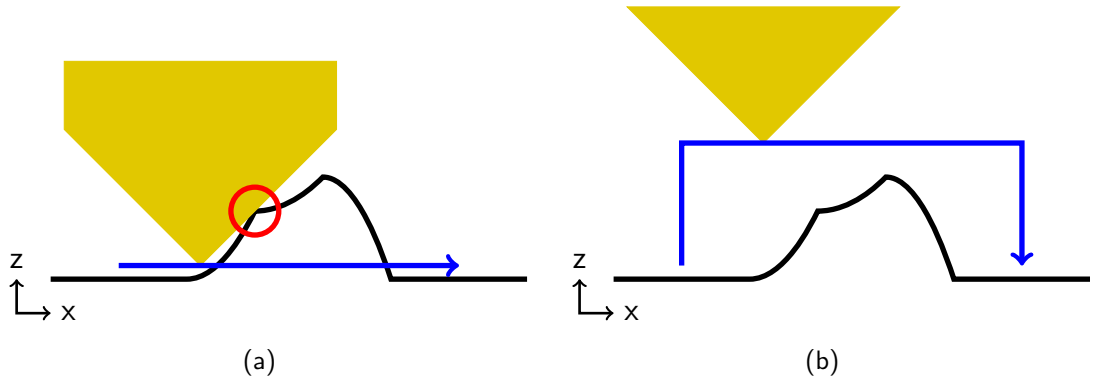


Figure 4.15: (a) Traveling directly to the target point can cause collisions with nonplanar structures. (b) When moving up to the current layer height before traveling avoids collisions with nonplanar structures.

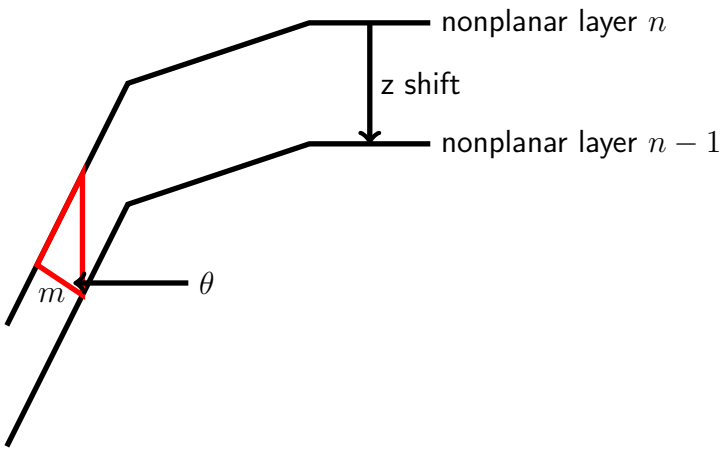


Figure 4.16: Shifting layers along its z-axis lowers the distance between the surfaces when they are not horizontal. The correction factor to compensate this difference can be calculated with a right angle perpendicular to the surface.

This method produces some unnecessary travels and the travel paths are rather long. It may be more effective to check if this collision-preventing travel move is necessary for this particular travel. It might also be a good idea to follow the surface while moving to prevent the extruder from oozing.

The necessary extrusion amount is calculated by multiplying the desired flow with the length of the path which is calculated with the Euclidean distance. The z-component is simply added to the calculation of the Euclidean distance.

The different shell layers are only shifted along the z-axis and not along their normals as [Huang and Singamneni, 2014] suggested. The z-shift leads to less volume between the layers with rising slope angle. The problem is shown in figure 4.16. When this is not compensated, layers that are not horizontal are overfilled. The compensation factor can be calculated by creating a right triangle that is perpendicular to the surface and calculating its height. Since the surface angle is not known in the G-code, the angle is calculated by the ratio between the z difference of both points and the overall extrusion length. This leads to the following correction factor m :

$$m = \cos\left(\arctan\left(\frac{(P2.z - P1.z)}{length(P1, P2)}\right)\right) \tag{4.13}$$

where $P1$ and $P2$ are both points of the extrusion line and $length()$ is the Euclidean distance of both points. This compensation factor is multiplied with the extrusion amount to prevent overfilling.

The G-code now generated and can be printed on the desired printer. The printing of the object with nonplanar layers does not differ from the planar one. The z-axis moves while extruding may be a little bit slower than expected because the firmware of the printer cannot exceed the maximum z-axis speed that is configured.

4.3.6 Toolpath Visualization

The toolpath that will be later printed can be visualized in Slic3r. The user can check if the toolpath is as intended and can scroll through the layers of the toolpath. This feature is essential for many users since they check every toolpath before printing. The feature comes in handy while developing since the toolpath can be checked while making changes in the source code without having to print each time. The slicer iterates through all extrusion objects and displays them to visualize the whole toolpath. Each path will be disassembled into single lines. For each line, a rhombic prism is generated out of facets with the length of the line. The prism has the width of the actual extrusion and the height of the layer. The ends and corners are covered with caps to prevent open polygons. To be able to display 3D toolpaths that go across different heights, the z-component of the build facets can no longer be taken from the layer height but rather from the z-component of the point. When the z-component of a point is not defined, the layer z-height is used as before. The new visualization can display toolpaths with nonplanar layers and can be seen in figure 4.17.

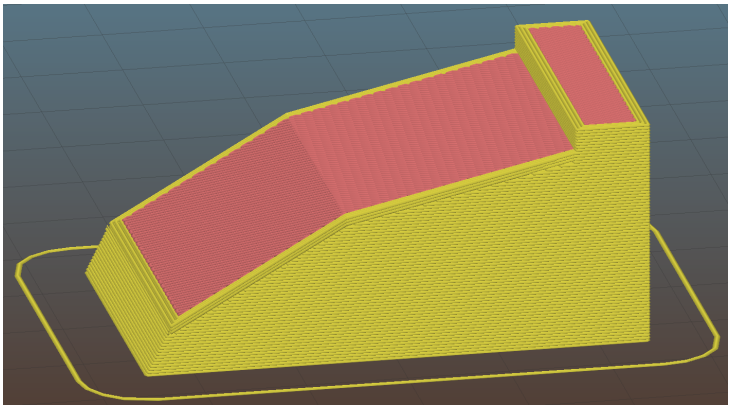


Figure 4.17: The nonplanar toolpath visualized in the toolpath preview in Slic3r. The preview now supports 3D toolpaths.

4.4 Limitations

The implementation of nonplanar layers in Slic3r works as intended. After adding the angle and the height of the hardware limitations to the configuration for the used printer, collision-free nonplanar surfaces can be printed. There are some limitations regarding the usable features of the slicer and with over- and underfilling of concave and convex structures. The next paragraphs show these limitations in detail.

4.4.1 Unusable Slic3r Features

Not all features that Slic3r provides are usable anymore. Some need further inspection to work again and some cannot be activated while printing nonplanar layers. Users can use these features but need to check the toolpath for collisions in the preview.

At this point of the implementation, it is not possible to generate support structures because a collision-free toolpath with nonplanar layers could not be guaranteed. This is mainly because the support is generated after the toolpath generation of the regular object. So the toolpaths of the nonplanar layers cannot check for those collisions and the support implementation does not expect nonplanar layers. It is possible to generate support structures, especially when the support is only on layers below the lowest printing point of a nonplanar surface. But the generation cannot be guaranteed to be collision-free. Collisions can occur when the support is printed alongside a not yet printed nonplanar surface. Figure 4.18(a) shows a nonplanar surface with support structure that collides with the printhead. Another more serious problem are support interfaces which are generated on top of an existing nonplanar surface. A support interface will be printed on every layer on top of the not yet existing nonplanar surface. When the nonplanar surface is printed, the support surface is already there and causes a collision. Figure 4.18(b) shows a collision with the support interface that is printed where the nonplanar layer belongs. To get collision-free support structures, the support that would cause a collision needs to be printed after printing the nonplanar surface. However, this could also cause collisions with other parts of the object. It is possible to generate collision-free support structures, but they are not implemented yet. Overall, support is not generally forbidden, but the user needs to be careful and check the actual printing preview for collisions before starting the print.

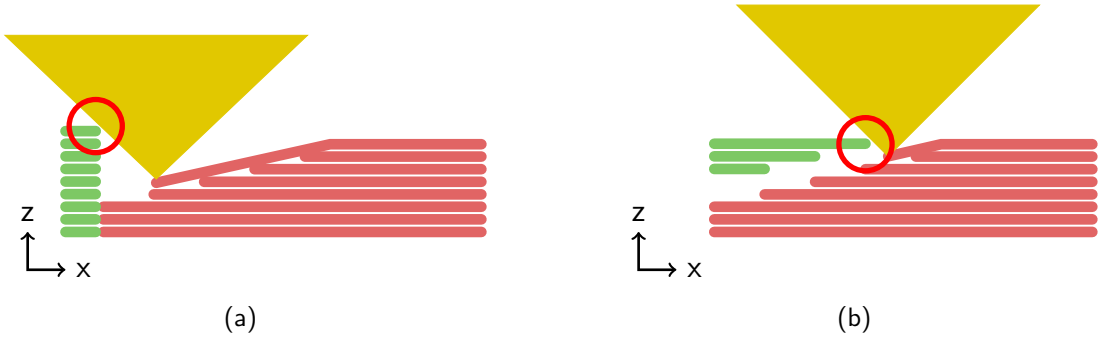


Figure 4.18: (a) The printhead collides with the previously printed support structure while printing the nonplanar layer. (b) The printhead collides with the support interface which is already printed above the nonplanr surface.

A brim cannot cause any problems with nonplanar layers because it is only printed on the first layer. A skirt, on the other hand, can cause collisions when the skirt is printed higher than the first layer. Since skirts that are higher than one layer are uncommon, this is not really a problem. However, Slic3r should forbid such configurations.

Nonplanar surfaces can be printed with a constant layer height. Even a different layer height on the first layer is not a problem since the first layer always stays untouched. Adaptive layer heights will cause a problem with the current implementation. While projecting the surfaces up to the home layer of the nonplanar surface, it is assumed that all surfaces have the same height. This is necessary to ensure that there will be enough space for the later down projected extrusion path. With adaptive layers, nonplanar layers are generally possible, but further research is needed to generate a planar base structure with enough space for the later above printed nonplanar layers.

4.4.2 Over- and Under-filling

When filling a nonplanar surface with a rectilinear pattern, gaps will emerge between the individual lines since the overall distance perpendicular to the filling direction increases. On concave surfaces, the lines are placed closer together and on convex surfaces, they are further apart. Figure 4.19(a) illustrates the problem on a convex surface. With a low maximum angle, these gaps or over-filling can be compensated by increasing or decreasing the extrusion amount. On a half sphere

(a) (b)

Figure 4.19: (a) A convex surface which suffers from underfilling. The gaps are especially visible on the side of the convex surface. (b) A different fill pattern to compensate these gaps. [Lim et al., 2016]

with a high maximum angle, this is impossible since the gaps get too big to compensate this by simply increasing the extrusion amount. Different fill patterns may be suitable to fill those extreme concave and convex surfaces. One pattern would be an endless volute track which starts on the outside polygon of the nonplanar surface. On each round, the track goes one step to the inside to fill the whole surface. This filling has to be done on the 3D surface without the projection used in this work. Another method is to separate the surface into different sub surfaces and fill each region with a rectilinear pattern with a different density like [Lim et al., 2016] did. Figure 4.19(b) shows an alternative pattern to prevent underfilling. This method also has to be generated on the 3D surface. On low maximum angles, the over and under-filling can be ignored since it is not such a big problem. Three-axis printers do not print high maximum angles well and the stair-stepping is way worse on low angles so it might be a good idea to set the maximum angle to a value where the stair-stepping is better and the over and under-stuffing is acceptable.

5 Evaluation

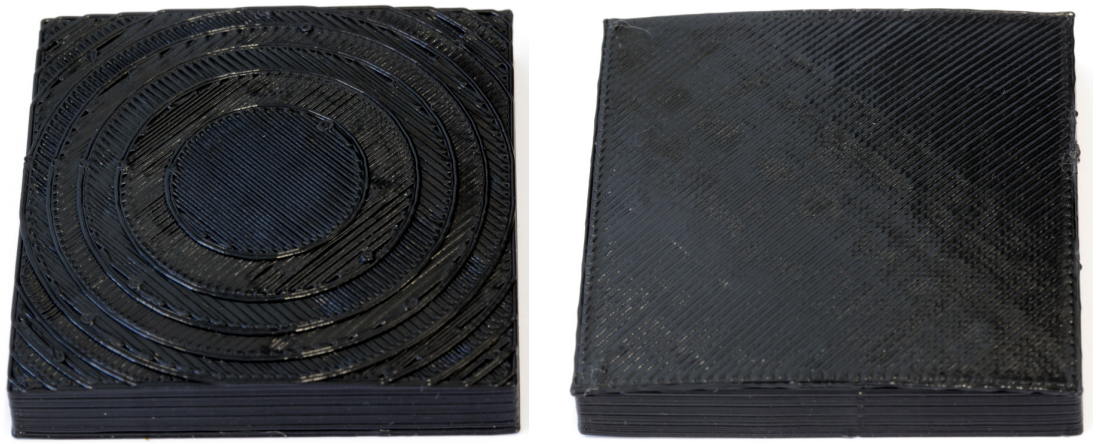
This chapter shows different tests to evaluate the implementation and the prints generated with it. The Tests that are performed include surfaces which heavily suffer from stair-stepping and complex surfaces. The printability of different angles is also tested with a three-axis printer. The slice and print speed is exterminated as well as the approximation error from the designed model. All test objects are printed on an Ultimaker 2 with a 0.4 mm Olssen Block nozzle.

5.1 Stair-Stepping

In this test, the ability to remove stair-stepping from surfaces is evaluated. Two objects which suffer from stair-stepping are compared to the same objects with nonplanar top layers. The first object in figure 5.1 contains one surface which is tilted by 5°. This object shows the ability to remove stair-stepping on tilted flat surfaces. The second object in figure 5.2 is the top 50 by 50 mm area of a sphere with a radius of 220 mm. This object shows the ability to remove stair-stepping from a curved surface with multiple facets. All objects are printed with 0.3 mm layer height. On the test prints, the objects with nonplanar layers (figure 5.1(b)

(a) (b)

Figure 5.1: The 5° tilted surface to test for stair-stepping printed with (a) planar layers and (b) nonplanar layers.



(a) (b)

Figure 5.2: The top 50 by 50 mm area of a sphere with a radius of 220 mm to test for stair-stepping printed with (a) planar layers and (b) nonplanar layers.

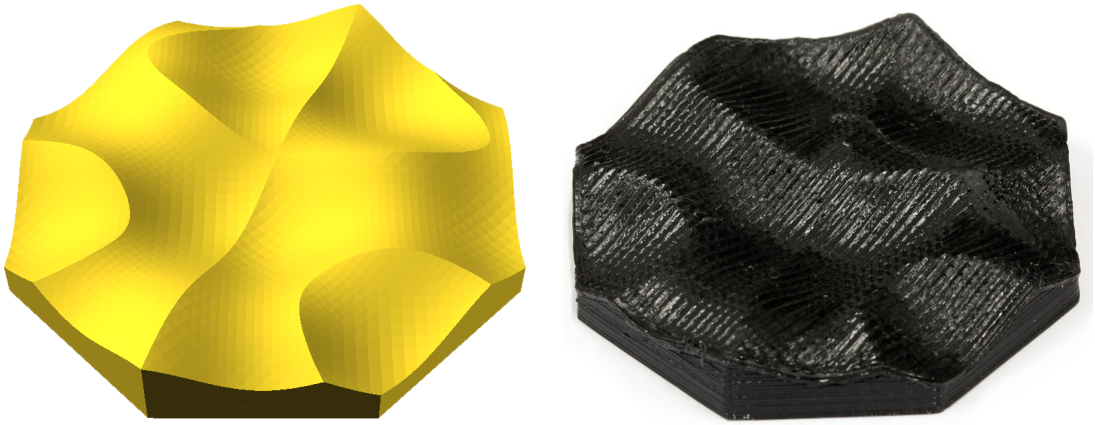
and 5.2(b)) have a much smoother surface than the objects with planar layers (figure 5.1(a) and 5.2(a)). The stair-stepping is removed entirely with nonplanar layers and the surfaces appear very smooth.

5.2 Complex Surfaces

In this test, a complex surface geometry is printed to evaluate the ability of the implementation to generate a proper toolpath on this surface without collisions. The surface is created by the following formula:

$$z = (\sin x \cdot \cos y) \cdot 4 \tag{5.1}$$

The surface is rotated by 45° and intersected with itself, which results in the model shown in figure 5.3(a). The printed model with nonplanar surfaces is shown in figure 5.3(b). The model is printed with 0.2 mm layer height. The toolpath that was generated, follows the contour and caused no collisions while printing the object. On the nonplanar layers, the print speed was slower than usual. The speed of the z-axis was limiting the overall print speed in the surface. The slower speed is mainly because of the relatively low acceleration and jerk values that are configured for the z-axis. Since usually the z-axis just moves on layer changes, these values are set very conservatively. Tuning the acceleration an jerk of the z-axis should increase the print speed in such complex surfaces. The surface of the



(a) (b)

Figure 5.3: (a) The model of the complex surface to check for proper toolpath generation without collisions and (b) the printed complex surface.

print is relatively rough. This is because the nozzle penetrates the surface while printing. The surface angle is on most facets of the surface above 20°. Lastly, slight over- and under-filling, especially on slopes perpendicular to the printing angle, can be observed. This can be compensated by generating variable extrusion widths which react to those variations in the line distance.

5.3 Printability of Different Angles

As seen in the previous test, the nozzle can penetrate the printed surfaces especially when a high angle is printed. To check if a three-axis printer would generally be able to print surfaces of a given angle and where the limitations are, seven test objects were printed with different angles. The printed angles are 5°, 10°, 15°, 20°, 25°, 30°, and 40°. All objects are also printed with planar layers to make the problems with stair-stepping visible. The printed objects are shown in figure 5.4. All angles can be printed without delamination problems or collisions with the printer. Missing filament retractions while moving inside the object over different heights caused some oozing which is visible on low corners of the object. The oozing can be compensated by changing the travel moves on nonplanar layers as mentioned before. The surfaces of the nonplanar printed objects get worse with rising ramp angle while the surface quality of the planar printed objects gets better. It seems that with a 20° ramp angle, the optical smoothness of the surface is even on both the planar and nonplanar object. On the objects with a lower angle,

Figure 5.4: The printability of surfaces with different ramp angles. The objects in the top row are printed with planar layers and in the bottom row with nonplanar layers. The printed angles are from left to right 5°, 10°, 15°, 20°, 25°, 30°, and 40°. Above 20° the surface of the planar objects is smoother and below the nonplanar surface is smoother.

the nonplanar surface is smoother while on the higher angles the planar surface is smoother. It might be a rough estimation to only print nonplanar surfaces with a maximum angle of 20° to get the smoothest possible object.

5.4 Print and Slicing Speed

The printing itself consumes most of the time in the whole 3D printing process. The actual slicing time is only relevant when the user adjusts settings to check if the toolpath looks as intended. In this test, either the time a slicing process needs to complete, as well as the printing time a model needs to print is measured. Both are measured with planar, nonplanar, and adaptive layers on the quarter sphere from section 5.5. The sphere model has 3778 facets and a nonplanar printable area. The object is sliced with 0.3 mm layer height while the adaptive layers vary between 0.1 mm and 0.3 mm. For printing, the nozzle and print bed are preheated to remove this variable from the time measurement. The time each slicing method took to slice and print the model can be seen in table 5.1. The print time is rounded to full minutes while the slicing time is rounded to a tenth of a second. The slicing time of the nonplanar method is significantly higher but not unusable

	slicing	printing
planar	7.7 sec	91 min
nonplanar	25.8 sec	93 min
adaptive	11.7 sec	125 min

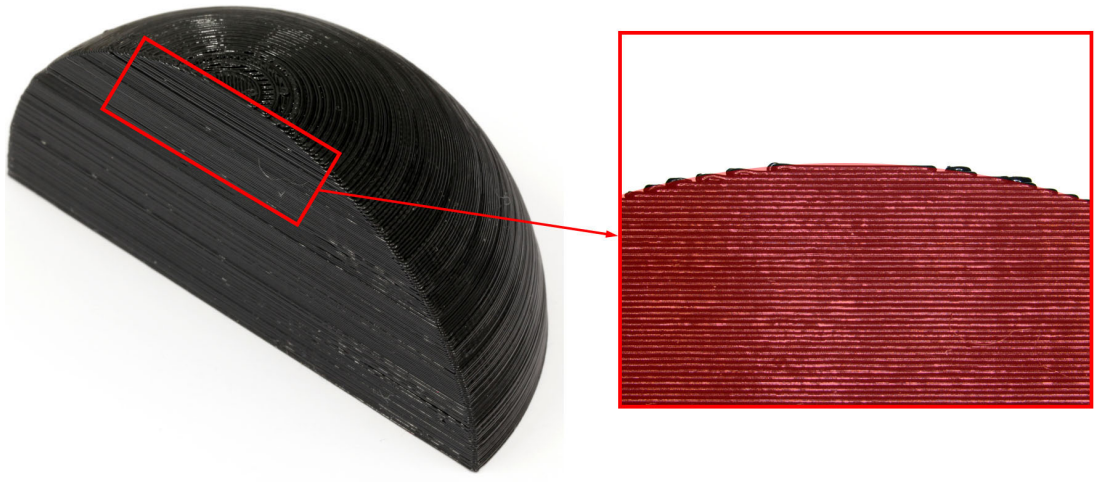
Table 5.1: The measured time to slice and print the quarter sphere the previous section. While the slicing of the nonplanar layers is significantly longer, the print time stays about the same. With adaptive layers, the print time is significantly longer.

high. It might be possible to improve the speed of the process by optimizing and parallelizing it. But, the slicing with nonplanar layers will never be as fast as planar slicing since the implementation adds additional steps to the process and some of them are rather complex. The print time of the nonplanar object is similar to planar printing but with an improved surface quality. A nonplanar printed object outperforms an object printed with adaptive layers. Overall the print time of a nonplanar printed object does not rise significantly compared to the planar printed object.

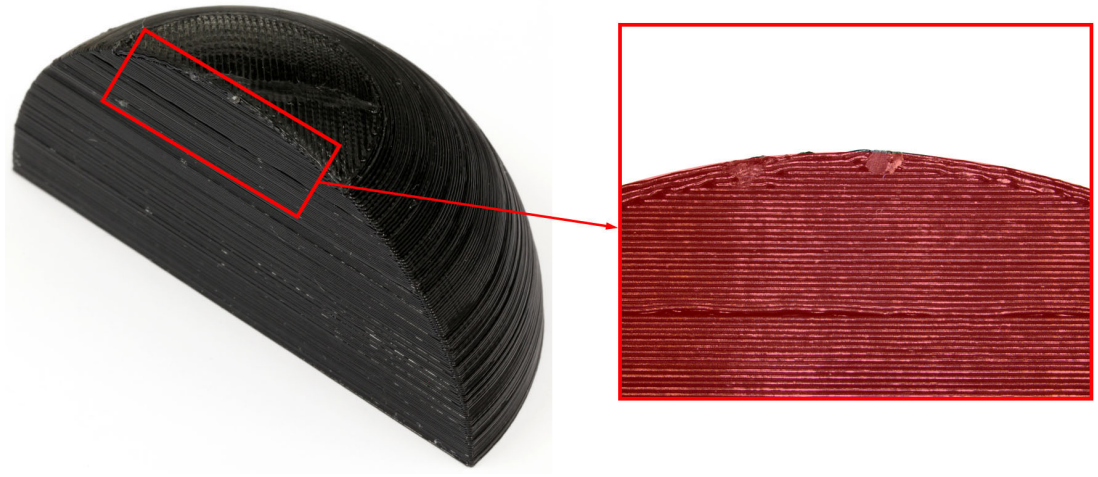
5.5 Approximation Error

Approximation errors are the difference between the modeled object and the actual printed object. Positive approximation errors are when the object is larger than the model in one region while with negative approximation errors these regions are smaller than designed. A quarter sphere with a radius of 40 mm is printed with planar (figure 5.5(a)), nonplanar (figure 5.5(b)) and adaptive layers (figure 5.5(c)) to evaluate the approximation error. The side profile of each object is magnified and the modeled sphere is overlaid in red. Each region that is above the sphere is a positive approximation error and every region below is a negative approximation error. The planar and the nonplanar objects are printed with 0.3 mm layer height while the adaptive layers vary between 0.1 mm and 0.3 mm.

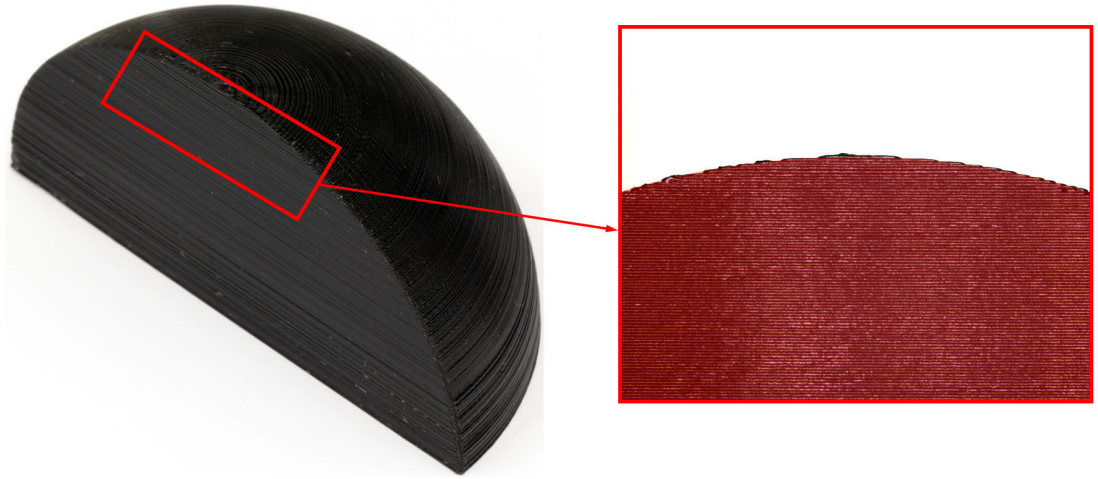
The planar object has a relatively high approximation error compared to the adaptive or the nonplanar printed object. The approximation error on the nonplanar object in the area of the nonplanar layer is in theory zero because the toolpath follows the surface geometry exactly. Practically, the surface roughness that the penetrating extruder produces creates an approximation error from the actual object. This error rises with the surface angle. Looking at the full object, the adaptive printed object looks closest to the model due to its consistent appearance. The different looking surfaces on the nonplanar object stands out. Furthermore, the transition between the nonplanar and the planar region is a bit rough. Looking only on the magnified side view, the nonplanar surface is closer to the model and has a more smooth surface finish. It cannot be generally determined if nonplanar printed objects have the best approximation error since this heavily depends on the objects geometry and the possibility to generate collision-free nonplanar surfaces.



(a)



(b)



(c)

Figure 5.5: The quarter sphere printed with (a) planar layers, (b) nonplanar layers, and (c) adaptive layers. On each quarter sphere, the side profile of the surface is magnified to see the approximation error. The modeled sphere is overlaid in red.

6 Conclusion

The generation of nonplanar layers for smoother surfaces works as intended. The surfaces are extracted from the triangle mesh of the model by their angle relative to the z-axis. They are grouped and filtered, so each surface meets the two criteria of maximum angle and maximum height. Collision prevention ensures that the printhead does not crash into previously printed structures while printing nonplanar layers. The surfaces are generated by moving parts of the planar layers upwards to form a new nonplanar layer. The toolpaths are generated as 2D toolpaths and are then projected downwards according to the geometry of the facet mesh. This forms a 3D toolpath that can be visualized in Slic3r to ensure that the path looks as intended. The G-code is generated from the toolpath, causing no collisions while traveling. The printed objects looked much better with the smooth nonplanar surface than with stair-stepping artifacts from planar surfaces. The print time for an object with nonplanar layers does not increase significantly compared to the object with planar layers. The mechanical properties are closer to the designed model. Although the bonding between the layers should increase when a nonplanar layer is printed above them, this was not tested. Test prints showed that a high maximum printable angle does not mean that all surfaces should be printed that way. But on lower surface angles the large stair-stepping artifacts can be compensated. On complex objects, the nonplanar surfaces are often not printable due to possible collisions with the printhead. This could be compensated by designing a special printhead made for nonplanar printing that can reach angles over a great height. The implementation in this work is open-source and can be found on GitHub on the following URL `https://github.com/Zip-o-mat/Slic3r/tree/nonplanar-thesis`.

6.1 Outlook

The software implemented in this work is usable and stable. But since some features might not work as intended, nonplanar surfaces must be used with caution. The missing compatibility with the support generation is the main problem since this feature is essential for some objects printed with an FDM printer. Further tests will show if more features are incompatible with nonplanar layers. The toolpath generation can be parallelized to increase the speed since this is not always the case yet. The current implementation might be ineffective at some steps. Better collision prevention on travel moves and overall better path planning which reduce oozing would increase the print quality even further. Stair-stepping also occurs on bottom facing surfaces, this was completely ignored in this work. It would be generally possible to print nonplanar layers on top of support layers and then generate the planar layers above. While printing the nonplanar extrusion heights, the nozzle height is currently not adjusted according to the printing direction. This could be compensated in future releases of the software. The test prints showed that surfaces with a high angle are squished onto each other. It is to be evaluated whether the offset for the different layers should be generated along the facet normals rather than along the z-axis to reduce this effect. Last but not least, the emerging gaps on convex surfaces stay to be examined and compensated in future releases of the software.

Glossary

2D	Two Dimensional
3D	Three Dimensional
AM	Additive Manufacturing
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CNC	Computerized Numerical Control
STL	STereoLithograhy
AMF	Additive manufacturing file format
3MF	3D Manufacturing Format
ASCII	American Standard Code for Information Interchange
FDM	Fused Deposition Modeling
CLFDM	Curved Layer Fused Deposition Modeling
PID	Proportional Integral Derivative
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
LGPL	GNU Lesser General Public License
AGPL	GNU Affero General Public License

Bibliography

[3D Systems, 1988] 3D Systems (July 1988). Stereolithography interface specification.

[Ahlers, 2015] Ahlers, D. (2015). *Development of a Software for the Design of Electronic Circuits in 3D-Printable Objects*. Universität Hamburg.

[Bread slicer] Bread slicer. https://github.com/nick-parker/bread.

[Burns, 1993] Burns, M. (1993). *Automated fabrication: improving productivity in manufacturing*. Prentice Hall.

[Chakraborty et al., 2008] Chakraborty, D., Aneesh Reddy, B., and Roy Choudhury, A. (2008). Extruder path generation for curved layer fused deposition modeling. *Computer Aided Design*, 40(2):235–243.

[Cura] Cura. https://ultimaker.com/en/products/ultimaker-cura-software.

[Ding et al., 2015] Ding, D., Pan, Z., Cuiuri, D., Li, H., Larkin, N., and Duin, S. (2015). Multi-direction slicing of STL models for robotic wire-feed additive manufacturing. In *Solid Freeform Fabrication Symposium*.

[Douglas and Peucker, 1973] Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

[Forefront Filament, 2016] Forefront Filament (2016). `http://www.forefrontfilament.co.uk/blog/2016/11/14/how-to-print-with-flexible-filaments`. accessed on 04 oct. 2018.

[Gibson et al., 2015] Gibson, I., Rosen, D., and Stucker, B. (2015). *Additive Manufacturing Technologies: 3D Printing, Rapid Prototyping, and Direct Digital Manufacturing*. Springer, New York, 2nd edition.

[Horvath, 2014] Horvath, J. (2014). *Mastering 3D Printing*. Apress, Berkely, 1st edition.

[Huang and Singamneni, 2014] Huang, B. and Singamneni, S. (2014). A mixed-layer approach combining both flat and curved layer slicing for fused deposition modelling. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 229(12):2238–2249.

[Jin et al., 2017] Jin, Y., Du, J., He, Y., and Fu, G. (2017). Modeling and process planning for curved layer fused deposition. *The International Journal of Advanced Manufacturing Technology*, 91(1-4):273–285.

[Khurana et al., 2017] Khurana, J. B., Dinda, S., and Simpson, T. W. (2017). Active-z printing: A new approach to increasing 3D printed part strength. *Solid Freeform Fabrication Symposium*.

[Kubalak et al., 2018] Kubalak, J. R., Wicks, A. L., and Williams, C. B. (2018). Using multi-axis material extrusion to improve mechanical properties through surface reinforcement. *Virtual and Physical Prototyping*, 13(1):32–38.

[Lim et al., 2016] Lim, S., Buswell, R. A., Valentine, P. J., Piker, D., Austin, S. A., and De Kestelier, X. (2016). Modelling curved-layered printing paths for fabricating large-scale construction components. *Additive Manufacturing*, 12:216–230.

[Llewellyn-Jones et al., 2016] Llewellyn-Jones, T., Allen, R., and Trask, R. (2016). Curved layer fused filament fabrication using automated toolpath generation. *3D Printing and Additive Manufacturing*, 3(4):236–243.

[Micali and Dornfeld, 2016] Micali, M. and Dornfeld, D. (2016). Fully three-dimensional toolpath generation for point-based additive manufacturing ssystems. In *Solid Freeform Fabrication Symposium*.

[Slic3r] Slic3r. http://slic3r.org/.

[Tyberg, 1998] Tyberg, J. (1998). *Local adaptive slicing for layered manufacturing*. PhD thesis, Virginia Tech.

[Wasserfall, 2015] Wasserfall, F. (2015). Embedding of SMD populated circuits into FDM printed objects. In *Solid Freeform Fabrication Symposium*, volume 26, pages 180–189, Austin.

[Zhao et al., 2018] Zhao, H.-m., He, Y., Fu, J.-z., and Qiu, J.-j. (2018). Inclined layer printing for fused deposition modeling without assisted supporting structure. *Robotics and Computer-Integrated Manufacturing*, 51:1–13.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 19 Okt, 2018 Daniel Ahlers

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 19 Okt, 2018 Daniel Ahlers