Extruder-Turtle: A Library for 3D Printing Delicate, Textured, and Flexible Objects

Franklin Pezzuti Dyer
fpezzutidyer@unm.edu
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico, USA

Leah Buechley
buechley@unm.edu
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico, USA

Figure 1: Prints created using the `ExtruderTurtle` library.

ABSTRACT

This paper introduces `ExtruderTurtle`, an open-source Turtle Geometry library for 3D printing, which generates G-CODE based on the path traveled by a LOGO-inspired Turtle. We describe the functionality of our library and demonstrate how it provides an intuitive and accessible way to create objects with a range of interesting properties, using Fused Deposition Modeling (FDM) 3D printers. We also present a collection of examples–including replications of previous research as well as original investigations–to demonstrate the power and flexibility of our library and this general approach. Our examples include 3D printed textiles, textured surfaces, and delicate string-art sculptures, along with traditional Turtle Geometry forms.

TEI '22, February 13–16, 2022, Daejeon, Republic of Korea

ACM ISBN 978-1-4503-9147-4/22/02…$15.00
https://doi.org/10.1145/3490149.3501312

CCS CONCEPTS

• **Human-centered computing → Interactive systems and tools**.

KEYWORDS

3D printing, G-CODE, Turtle Geometry, LOGO, digital fabrication, computational design, computational geometry

ACM Reference Format:
Franklin Pezzuti Dyer and Leah Buechley. 2022. Extruder-Turtle: A Library for 3D Printing Delicate, Textured, and Flexible Objects. In *Sixteenth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '22), February 13–16, 2022, Daejeon, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3490149.3501312

1 INTRODUCTION

When designing objects for FDM 3D printing, the traditional workflow proceeds as follows: CAD software, like Rhino or Blender, is used to design a shape which is then exported to a file format, like STL, that encodes its geometry. Once generated, an stl file is passed to slicer software, which slices the geometry into layers, and outputs a sequence of G-CODE commands, which specifies the path to be traveled by the 3D printer's extruder. Then, each horizontal cross-section of the object is printed one layer at a time.

Computational design and geometric programming tools like OpenSCAD and Grasshopper can also generate 3D models (and

corresponding .stl files), but they too output geometries rather than machine paths. It should be stressed that neither CAD software nor geometric programming tools were designed to output G-CODE. G-CODE is almost always generated by separate slicing software.

The pairing of 3D modelling (or computational design) and slicer software is powerful but restrictive. There is a range of interesting shapes, surfaces, and textures that cannot be generated by slicers. This shortcoming has been examined by many researchers, who have developed creative approaches to generate and characterize individual designs that cannot be produced through the traditional process (cf. [10, 29, 30]).

This paper introduces an open-source library that we intend to function as a *general-purpose* tool that enables the exploration of custom G-CODE generation. Instead of developing and characterizing a single design, like previous researchers have done, we present our tool and then explore how it can be used to produce a wide range of designs. Our explorations include the 3D printing of hairs, textiles, string-art, and non-planar surfaces. The contribution of this work is: 1) a robust and user friendly open-source library for GCODE generation, in conjunction with 2) a large collection of examples that demonstrate the utility and flexibility of the library. We hope the paper also serves as 3) an illustration of the unique and compelling possibilities afforded by custom G-CODE generation. Our intention is to "lower the floor" of technical expertise required to experiment with custom G-CODE generation, and "widen the walls" of perceived possibilities in this space [20].

2 PREVIOUS WORK

2.1 Turtle Geometry and 3D Printing

Though closely associated with LOGO and most well known as a programming paradigm for children, Turtle Geometry is best understood as a powerful general-purpose alternative to Euclidean Geometry. It has many advantages as a framework for exploring computational geometry. These are covered extensively and beautifully in *Turtle Geometry: the Computer as a Medium for Exploring Mathematics* [1].

Turtle Geometry describes geometrical shapes not in terms of the static points and curves of Euclidean Geometry, but as paths that are dynamically traced out by a "turtle", holding a "pen", that moves across the plane. The turtle is directed via forward, backward, left, and right as well as pen-up and pen-down commands [1, 16]. Shapes are created as the turtle drags the pen around in different patterns. Although it was originally used only for 2D figures, the turtle paradigm has since been extended to describe curves in 3D space by allowing rotation about multiple coordinate axes [32]. The turtle paradigm enables people to describe and generate complex shapes in an intuitive way, without using trigonometry. Turtle Geometry and LOGO have been widely used in mathematics and computer science education [1, 16, 28].

The resemblance between a turtle that traces geometrical shapes in its path and an extruder that leaves strands of plastic filament trailing behind it suggests an analogy between Turtle Geometry and 3D printing. There are several instances of past research combining Turtle Geometry and digital fabrication. "Turtle Stitch" is an application that uses 2D Turtle Geometry to generate embroidery patterns [34]. "Beetle Blocks" is a programming environment that uses 3D Turtle Geometry to design 3D objects that can be printed [21–23]. This youth-oriented software generates large scale geometry and must be used in conjunction with slicing software. In an earlier precedent, Eisenberg explored mathematical knot structures by tracing out their paths with a 3D turtle and 3D printing the results [5].

In a project that most closely resembles the work we describe here, Kanada et al. wrote a basic library that generates G-CODE with turtle movements [7]. Our library expands significantly on this earlier work by introducing a set of new features including: turtle pen up and pen down commands, which enable the printing of structures that require non-continuous extrusion; commands that enable the blending of Euclidean geometry and Turtle Geometry; commands that control the rate of filament extrusion; and commands that allow the printer to pause in the middle of a print and (optionally) respond to user interaction. These new features are critical to producing the range of examples presented in this paper. For example, printing hairs requires the pen up and pen down commands, textured prints require filament extrusion control, and textile structures require the pause command.

Our library can also be seamlessly integrated into Grasshopper/Rhino projects, and has features, including the drawTurtle() method, that allow users to easily visualize turtle movement and location in Rhino. This allows users to visualize their 3D prints and turtle paths in real time as they code.

We also demonstrate a much larger and more diverse set of examples than Kanada et al. Our aim is to do a more thorough job of illuminating the range of possibilities. Importantly, most of our examples rely on features–especially pen up, pen down, and extrusion control–that are unique to our library.

2.2 Custom-Generated G-CODE

Custom G-CODE can be created in many different ways. A program written in any general purpose language can output a .gcode file, but it is common for G-CODE generators to be coded in programming environments designed for 3D modeling, like Grasshopper. (Note that the popular geometric programming language OpenSCAD does not have the language features required to generate G-CODE.) There are plugins (c.f. Droid [31]) that enable a programmer to generate G-CODE directly in Grasshopper, but these plugins mostly replicate the behavior of slicers. The fine-grained control of print head behavior that is required to produce the examples we discuss in this section and the rest of the paper entails the writing of custom programs.

When G-CODE is generated directly for an FDM printer, it is possible to print structures that are as thin as a single strand of extruded filament. Several researchers have leveraged this ability to generate hair-like 3D printed structures [3, 10]. Patches of 3D printed hair can be used to decorate 3D printed figures and to create a variety of brushes. They can be used to increase the mechanical adhesion between two surfaces (creating Velcro-like structures), to control objects' movements across a surface using a "stick-slip" mechanism, can serve as tactile and acoustic sensors when coupled with electronic components [15], and can enhance virtual reality experiences with haptic feedback [3].

Wavy, curly, and bumpy surface textures can be built by changing the extrusion process with custom G-CODE. Takahashi & Miyashita developed a system where a shape is printed layer by layer but the print head is lifted slightly above the previous layer, instead of extruding directly onto it. The filament curls as it is extruded onto the previous layer, creating an object with a ruffled surface. They conducted a detailed exploration of the parameter space, cataloguing which combinations of height and extrusion rate result in which textural patterns [29]. The artist LIA also created a series of striking and beautiful 3D printed sculptures that explored a range of G-CODE generation techniques, some of which seem to be the inspiration for Takahashi & Miyashita's work [12].

Surface texture is useful as a medium for storing information, as an alternative to or extension of visual media. In another paper, Takahashi & Miyashita discuss a framework where different patterns can be "embossed" onto surfaces using 3D printing [30]. 3D printing has been widely used to create braille or braille-like textures on the surface of objects (cf. [2]). In similar fashion, Shin et al. explored how colors can be mapped to 3D printed textures to enable people to appreciate visual artwork by experiencing colors through texture [27].

Creatively printed materials can also have a variety of mechanical and tensile properties. The use of metamaterials, or small-scale structures composed of the same material, can be used to control the elasticity of a 3D print, or even impart varying levels of elasticity to different regions inside the print [25].

Another challenge of nontraditional 3D printing is to produce textile-like structures, which could potentially be applied to the fabrication of clothing or other wearable objects. Techniques for mimicking textiles include using the extruder head to "weave" filament between printed pillars and even combining extruded plastic with actual textiles and fabrics to enhance tensile strength [24, 29].

In addition to interesting and useful tactile and tensile properties, manipulating 3D printed objects' local properties can give rise to interesting optical effects. For instance, controlling the direction in which filament is extruded can change how light is reflected off of a printed surface [8], and techniques such as helical 3D printing can give rise to complex Moiré patterns [9].

Custom software can also be used to print wireframe mesh structures consisting of segments that are only one stroke of filament thick. Not only do these skeletal constructions have a unique appearance, but they can be used to rapidly "preview" prints without wasting an unnecessary amount of time and filament [14]. In short, custom G-CODE generation opens up a wide range of expressive and technical possibilities.

It is worth emphasizing that all of the research described in this section involved writing an application-specific G-CODE generator. Each generator was capable of generating only one specific type of shape. `ExtruderTurtle` takes a different approach; it is a *general-purpose* library that can be used to generate G-CODE for a wide range of shapes and applications, including many of those described above. Our intention is to make explorations like these much easier to code and 3D print.

3 THE EXTRUDER-TURTLE LIBRARY

Our open-source library consists of a single class, `ExtruderTurtle`, which has methods that combine 3D Turtle Geometry with the functionality of a 3D printer. The source code for our library, along with extended documentation of all of the examples described in this paper, can be found on the Extuder Turtle website [4]. As with traditional Turtle Geometry, we keep track of the turtle's heading and position. We have added state variables to the turtle that capture information relevant to 3D printing. For instance, our turtle has variables that keep track of the temperature and speed of the print head.

Many turtle actions, including all movements through space, correspond to G-CODE commands [19]. As methods are called, these commands are written to a G-CODE file which, when executed by a 3D printer, instructs the extruder's nozzle to trace out the path traveled by the turtle. In the spirit of Turtle Geometry, G-CODE is written in relative mode, meaning that all XYZ-coordinates are measured relative to the extruder's current position, rather than a fixed origin.

`ExtruderTurtle` can be used as a stand alone python tool or can be incorporated into Grasshopper/Rhino projects. Grasshopper and Rhino can be used to visualize the turtle's path, including the approximate thickness of the filament that will be extruded and the current position and heading of the turtle. See Figure 2.

To begin an `ExtruderTurtle` program, the method `ExtruderTurtle()` must be called to instantiate a turtle object. (In the rest of the paper, we will refer to a turtle object with the name "t".) The `t.setup(x,y,z,filename)` method must be called to specify the turtle's starting position and the file to which G-CODE will be written. The rest of the library's methods can be sorted into two groups: 1) those that control the geometry of the path traced by the turtle and 2) those that control the way in which filament is deposited.

3.1 Methods Controlling Path Geometry

In 3D space, the turtle's heading is determined by three unit vectors - the "forward" vector, which can be visualized as pointing out the front of the turtle's head; the "left" vector, which points out of the turtle's left side; and the "up" vector, which is outwardly normal to the turtle's shell. This allows for the following commands for controlling rotation:

Command	Description	GCODE
`t.yaw(angle)`	Rotates the Turtle's body about the "up" vector	N/A
`t.pitch(angle)`	Rotates the Turtle's body about the "left" vector	N/A
`t.roll(angle)`	Rotates the Turtle's body about the "forward" vector	N/A
`t.right(angle)`	Equivalent to `t.yaw(-angle)`	N/A
`t.left(angle)`	Equivalent to `t.yaw(angle)`	N/A

We also provide a `t.draw_Turtle()` method that is useful for visualizing the position and heading of the turtle in Grasshopper/Rhino. This command returns a triangular Rhino surface that depicts the plane the turtle is currently in. The sharpest point of the triangle indicates the direction of the turtle's forward vector (see Figure 2, c).

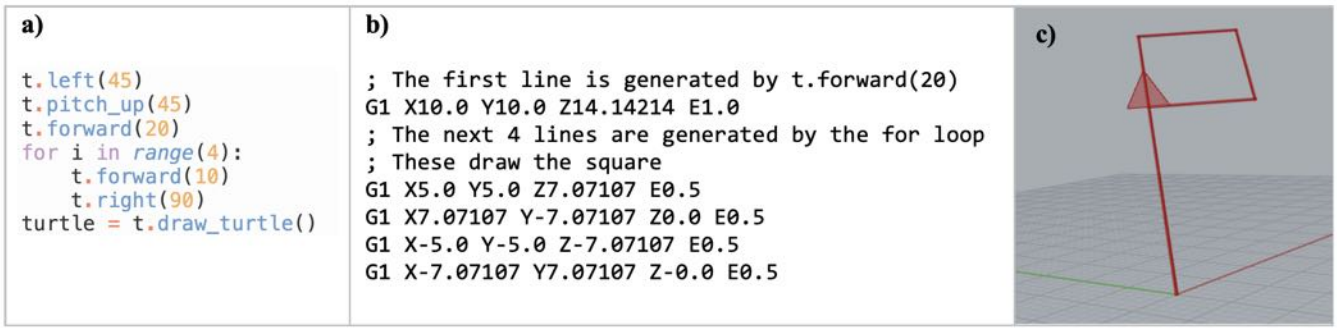

Figure 2: A sample `ExtruderTurtle` program: a) the python code, b) the corresponding G-CODE, c) the visualization of the path in Rhino. The shaded triangle indicates the turtle's heading and position at the end of the program, when `t.draw_Turtle()` is called. Note that this path would not result in a successful 3D print.

The following commands control the turtle's movement:

Command	Description	GCODE
`t.forward(distance)`	Moves the Turtle a specified distance in the direction of its "forward" vector	G1
`t.backward(distance)`	Equivalent to `t.forward(-distance)`	G1
`t.lift(height)`	Moves the Turtle a specified distance in the direction of its "up" vector	G1

Our library also includes methods that allow a programmer to access and control the turtle's state more directly, including `t.getX()`, `t.set_position()`, `t.get_yaw()`, `t.set_heading()`, etc. These functions allow a programmer to combine Turtle and Euclidean Geometry, which can be useful in many situations. Figure 2 shows an example series of turtle commands accompanied by the G-CODE they generate and a visualization of the path in Rhino.

3.2 Methods controlling Filament Deposition

The simplest commands controlling filament deposition are `t.penup()` and `t.pendown()`, borrowed from traditional 2D turtle commands. The former "lifts the pen", meaning that no filament is extruded until the pen is "put down" again using the latter command. To generate cleaner printed transitions, the `t.penup()` command adds a G-CODE command (E-3) to the file specifying a negative extrusion of 3mm and the `t.pendown()` adds a command with the corresponding positive extrusion (E3).

When the pen is down, filament is extruded at a constant rate, so that the same amount of filament is extruded per unit distance as the extruder moves. For most FDM printers, using a nozzle with a diameter of .4mm, filament with a diameter of 1.75mm, and a layer height of .2mm, the standard value is .03. In practice, we have found best results using an extrude rate between .01 and .10. The default extrude rate for our library is 0.05, but this can be changed using the command `t.set_density(extrude_rate)`. Note that the unit of extrusion rate is: mm of filament extruded per mm of print head movement (mm/mm). The command `t.extrude(amount_mm)` can be used to extrude filament in-place without moving the turtle.

The command `t.rate(rate_mm/s)` sets the speed of the print head, in millimetes traveled per second. The default speed is 1000 mm/s. The commands `t.extruder_temp(temp_C)` and `t.bed_temp(temp_C)` control the temperature of the extruder and print bed respectively. These variables can impact extrusion stiffness and bed adhesion. The `t.dwell(ms)` and `t.pause_and_wait()` commands allow us to control the timing of the print. The `t.dwell(ms)` command pauses for the given number of milliseconds and can be useful when very precise control of extrusion is needed. The `t.pause_and_wait()` command allows us to generate prints that involve user interaction. A listing of all printer-related commands, including the G-CODE generated by each command, can be found on the Extruder Turtle website [4].

Our library also includes a set of header and footer files that write G-CODE initialization and finishing sequences to the beginning and end of our generated G-CODE files. A program can also read information from these files, including the size of the print bed, the location of the center of the bed and so on. This structure enables us to work with different 3D printers quickly and easily; we can include machine-specific initialization sequences and other information in these files.

4 EXTRUDER TURTLE PRINTS

We used our `ExtruderTurtle` library to explore a range of applications. We aimed to: 1) replicate work of previous researchers, demonstrating the library's usefulness and versatility, and 2) generate novel forms, illustrating how it can support and encourage new research. Novel forms described in this section include the pine tree structures described in section 4.3 and the textile structures described in section 4.5. To our knowledge, this is the first time that a Turtle Graphics approach has been used to generate all other forms as well, with the exception of the Hilbert Curve discussed in section 4.2. All of the examples we present were printed with 1.75mm white PLA filament, using a .4mm nozzle on either a Prusa i3 MK3S or a Creality Ender 3 printer.

4.1 Basic Shapes

Before we discuss more complex examples, it is worth noting that there are many advantages to using our library to generate simple

geometrical figures. Shapes that have walls that are the width of a single strand of extruded filament can be printed very quickly, and by adjusting the speed and extrusion rate for these prints, it is possible to create objects with a range of stiffnesses. Figure 3 shows two hexagonal prisms. The thick and sturdy prism on the right was printed with an extrude rate of .1mm/mm. The delicate, flexible, and slightly translucent prism on the left was printed with an extrude rate of .02mm/mm. Both were printed at the default speed of 1000mm/s, and each 28 x 30 x 12mm prism took under seven minutes to print. Figure 3 also shows the very simple program that generates the hexagonal prisms. Here, our code mimics a slicer, stacking sixty layers of thin hexagons to build the shape.

```
for i in range(60):
    for j in range(6):
        t.forward(15)
        t.right(60)
    t.lift(.2)
```

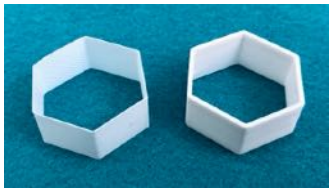

Figure 3: Left: Simple code that generates a hexagonal prism. Right: Two prisms printed with different extrude rates.

The procedure above generates hollow tubes. We often want to print shapes with a bottom. There are two straightforward ways of generating a solid layer using a turtle. We can employ either a zig-zag or concentric path to fill a shape outline. Figure 4 shows a few of these basic shapes. For more extensive discussion of all of our examples, including example code, see the Extruder Turtle website [4].

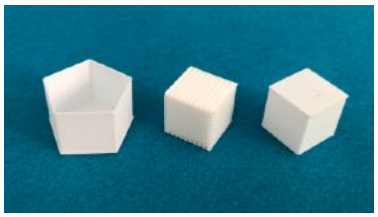

Figure 4: A pentagonal prism with a bottom and solid cubes with different fill densities.

Functions like these are, of course, built into slicing software, but it is useful to be able to carry out basic slicing/stacking tasks in conjunction with more exotic G-CODE generation. We present these examples to introduce our library and suggest its potential, and also because we will use some of this functionality in the examples we discuss below.

4.2 Fractals

One affordance of the turtle representation is that it facilitates the design of fractal structures that would be cumbersome to describe in other ways. By interpreting an L-System as a set of turtle instructions, intricate figures can be generated [17]. L-system rules define a set of substitutions–typically a single character is substituted by a collection of several characters–to be undertaken during one iteration of the system. An L-System "word", a series of characters, can be translated into a turtle shape; each character in the L-System is interpreted as a turtle command, like forward or right. With each successive iteration of the L-System, simple turtle paths are replaced with more complex ones. Figure 5, shows an example structure, a sculpture showcasing successive iterations of the Koch Snowflake curve stacked atop each other. In this L-system, a straight path is substituted for a strait path with a triangle jutting out of the middle at each iteration.

Figure 5: A sculpture generated by stacking successive iterations of Koch curves.

We have found that 3D printed space-filling curves, which can also be generated using L-Systems, have interesting mechanical properties. These curves fill the plane with a single non-intersecting line (see [17], page 12). A 3D print of a 2D space-filling curve, extruded in the z direction, retains its 2D shape at rest, but can stretch to over twice its resting width when extended. Figure 6 shows a Hilbert curve in its resting and extended states.

Figure 6: A 3D-printed Hilbert curve relaxed and extended.

Slicers are capable of producing the examples we have shown so far. However, the Turtle Geometry framework encourages a different and we would argue useful way of working and thinking. Furthermore, for many simple design tasks, `ExtruderTurtle` facilitates the writing of simple and elegant code, a streamlined workflow, and very quick printing.

4.3 Hairs and Webs

`ExtruderTurtle` can be used to generate hair like-structures that are impossible to generate with a slicer. We have found that the easiest way to create hair is to print it parallel to the print bed and to anchor strands to solid structures on both ends, following the fiber-bridging technique described in [10] and [26]. The hair will remain permanently attached to a structure on one end. The other end-structure serves as a support, insuring that the filament dries properly and does not interfere with subsequent paths taken by the 3D printer. The support can be removed with scissors or a mat knife when the print is finished.

To generate a basic fiber-bridging print, the turtle first builds a support consisting of two solid vertical structures, a hair-length apart. This keeps the hairs off the print bed. Next, the turtle takes a zig-zagging path, back and forth across the supports. On each pass, a single strand of filament is extruded. Another layer of support is printed before a second hair layer is added.

We were also able to print fine hairs using a technique described by Laput et al., where the print head moves quickly away from an extrusion blob to generate a more fiber-like strand [10]. This approach, employed in conjunction with printed supports, was especially successful. Figure 7 shows a variety of "hairy" prints.

Figure 7: A collection of hairs, brushes, and bristles.

We can use similar techniques to print structures with closed-loop hairs. For instance, in the pine tree structures that are shown in Figure 8, right, the branches consist of closed loops with endpoints lying on the trunk. Notice that the branches of the printed tree are not perpendicular to the trunk, nor do they have sharp corners. After being extruded, these loops of warm filament deform and sag under their own weight, giving them a more natural appearance. The contrast between the visualization, as generated by the code, and the printed shape can be seen in Figure 8, center.

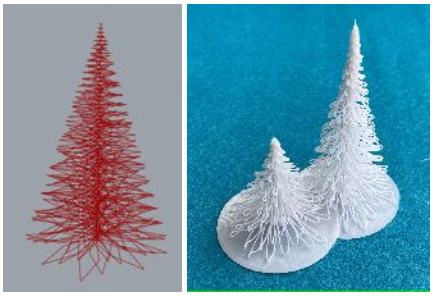

Figure 8: Pine Trees. Left: A visualization in Rhino. Right: Printed trees.

We can also use single-filament extrusions to create string art sculptures. Lovely mathematical models can be constructed in this fashion [13, 35]. For example, Figure 9 left shows a classic mathematical figure, where a parabola is created from a series of straight lines.

Figure 9 right shows a string art cardiod. This was produced using the following basic algorithm: a number, N, of equally spaced points are placed around the circumference of a circle, and each point in position n is joined to the point in position $2n$ by a piece of string. The way in which the strings overlap each other creates an emergent image of a cardioid. The example in Figure 9 is printed with $N = 67$ points, 66 layers and 8 chords per layer.

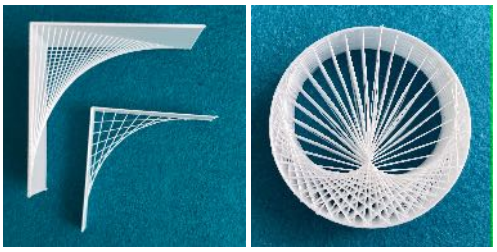

Figure 9: String art sculptures. Left: Classic geometry. Right: A cardioid.

4.4 Surface Texture

By adding small perturbations to the turtle's path, bumps can be added to smooth surfaces, giving them a braille-like texture. Here, we again employ the powerful general principle of replacing a simple turtle path with a complex one that ends in the same location. Two paths are equivalent with respect to the larger geometric procedure as long as the turtle eventually ends up in the same location; a bumpy path can be substituted for any straight one. Note that this is the same principle we employed to generate the fractal patterns described earlier. Turtle Geometry provides an especially simple and elegant way of describing these kinds of patterns.

```
def bumpy_polygon_prism (size, number_sides, height):
    for i in range(int(height/layer_height)):
        for j in range(number_sides):
            x = random.randint(1,size-3)
            t.forward(x)
            # create 2mm bump
            bump(2,90)
            t.forward(size-x-2)
            t.right(360/number_sides)
        t.lift(layer_height)
```

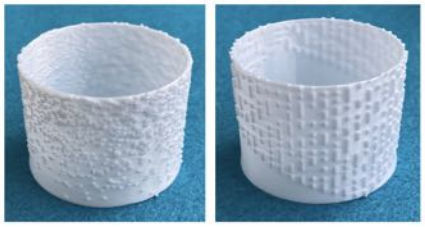

Figure 10: Top: A program that generates a prism with a randomly placed 2mm bumps. Bottom, left: bump density increases from bottom to top, right: bumps generated by a 2D cellular automaton using Wolfram's "Rule 30" [33].

Adjusting the size and angle of bumps in our program can give rise to different textures: for example, printing layers of bumps that protrude from the surface at an angle creates a texture that feels smooth when rubbed in one direction, and sharp when rubbed in the opposite direction. The distribution of bumps can be patterned or random, or some combination of the two. Figure 10 shows examples of bumpy artifacts.

Using a technique described by Takahashi & Miyashita, we can also imbue 3D prints with irregular ruffled textures. Increasing the size of the gap between the extruder nozzle and the previous layer of a print leaves room for filament to "wiggle" before adhering to the previous layer, introducing some irregularity. Increasing the extrusion rate forces strands of molten filament to "curl up" as they are deposited. The interesting textures shown in Figure 11 resulted from applying an increase in extruder height and extrusion rate to a simple seven-pointed star.

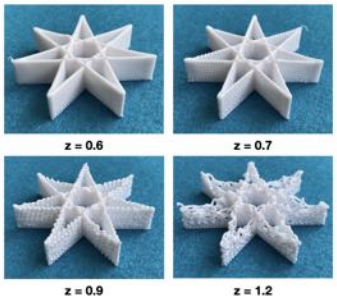

Figure 11: Star-shaped prints demonstrating how layer separation affects texture in prints with curly lines of filament.

The size of the initial "gap" between the nozzle and the print bed is not the only important parameter for these prints. In order to maintain a consistent texture throughout, the layer height h also needs to be tuned. For the prints we have discussed so far, $h = 0.2$mm. However, when the filament is curly, each layer is vertically thicker. If h is much smaller than the thickness of a bumpy layer, the increasing height of the print will outpace the vertical movement of the extruder, "closing the gap" between each layer and the next and causing the layers to become flatter towards the top of the print, as in the case of $z = 0.6$, Figure 11 top left. On the other hand, if h is much larger than the thickness of a bumpy layer, the gap between the nozzle and the print will grow, causing layers to be increasingly separated and irregular, as in the cases $z = 0.9, 1.2$, Figure 11 bottom. These phenomena can be utilized deliberately to create a print with a texture gradient from bottom to top, or h can be tuned to ensure a constant texture throughout, as in the case of $z = 0.7$, Figure 11 top right.

4.5 Textile-Like Structures

While experimenting with our library we discovered that flexible textile-like structures can be produced with very simple turtle programs. While different 3D printed textiles have been explored by previous researchers [29], our approach to creating such structures is original.

We described earlier how flexible structures can be generated by simply decreasing the printer's extrude rate. However, lowering the extrude rate too much can cause print layers to become fragile and separate from each other, and this technique is often insufficient to create prints that have the true flexibility of textiles.

By repeating patterns of local movements, similar to those used to generate novel textures, we can create textile-like surfaces. We instruct the turtle to make many short movements perpendicular

Figure 12: Flexible circles with different extrusion densities, affecting their flexibility and transparency.

to the direction of the surface as it traces out the shape. When the turtle makes a quick movement perpendicular to the surface, a small amount of filament is dragged away from the surface, and onto a protrusion. This approach gives rise to structures consisting of many thick "pillars" connected by thin wisps of plastic. Despite the fragile appearance of the structures shown in Figure 12, the overall prints are surprisingly strong. It is also possible to print flat sheets of flexible material using the same technique.

4.6 Nonplanar Slicing

In prints with very gently graded surfaces, the use of slicers can often lead to conspicuously low print quality due to a "stair-stepping" effect. However, quality can be improved by printing layers that are slightly curved instead of perfectly flat, which is beyond the functionality of most slicer software. The ability to print non-planar paths can also be leveraged for other purposes.

Note that collisions, with either the extruder (Figure 13 right) or the box surrounding the print-head (Figure 13 left), are a significant issue for non-planar prints. Collision points differ for different printers and nozzles.

Figure 13: Two kinds of collisions that must be avoided on non-planar paths.

Consider a variation of the seven-pointed star introduced earlier, in which the middle of the star bulges upwards (Figure 14). In this case, the strands of filament joining two vertices of the star are not straight lines, but slightly concave in the z-direction. A parabolic path can be drawn for each edge of the shape. Again, we can leverage the principle of replacing a simple path with a more complex one. As long as the parabolic path ends in the same place the straight path did (as long as the total z-displacement along the path is 0) it can be substituted for the straight path.

If we let $a = h/X$ for each turtle step along a parabolic path, where h is the desired height of the parabola and X is 1/2 of the total path distance, the maximum z-displacement will be given by $Xh/2$

```
for layers in range(30):
    v = layers/30
    for side in range(7):
        for x in range(-X, X+1):
            a = v*h/X
            t.forward_lift(1, -x*a)
        t.right(360*4/7)
```

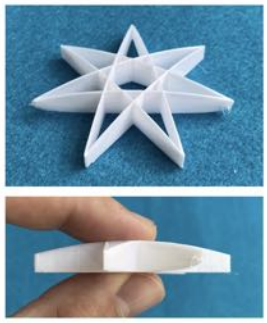

Figure 14: Nonplanar seven-pointed star printed with shallowly parabolically curved layers.

(in mm). If we wanted to draw a parabolic layer with a specific maximum z-displacement, we can use this equation to solve for appropriate values of X and h. Z-displacements that are too large can cause collisions, so this formula can be used to choose X and h in such a way that ensures no collisions will occur.

Since the bottom layer of the star-shaped solid should be flat, and the top layer should be the most concave, we employ a gradient to vary the steepness of the layers linearly from the bottom to the top layer. The final program to generate the finished star is also pictured in Figure 14.

Figure 15: Nonplanar vase with a sinusoidal oscillation along the rim.

Similar techniques can be used for more complex curves, as long as they are shallow enough to avoid collisions. Despite this constraint, interesting prints are possible. Figure 15, for example, shows a vase with a sinusoidal oscillation in height along the rim.

5 DISCUSSION

Despite its utility for generating the designs described above, the 3D turtle paradigm has a few drawbacks. One limitation is that Turtle Geometry's emphasis on local movement makes it difficult to design prints with complex geometry that lack convenient symmetries or simple low-level descriptions. For instance, it would be infeasible to use `ExtruderTurtle` to generate, say, a figurine in the shape of a human or animal, since they do not admit a simple geometrical description.

In general, using a slicer to generate G-CODE insures that most prints will be successful. When generating custom G-CODE, with our library or through other means, there are countless opportunities for failure. The workflow process is often more time consuming, since printing success is not guaranteed. Some prints require close supervision because inappropriate G-CODE can damage the printer. The extruder can be moved outside the bounds of the print area in the x, y, or z direction. (Crashing the print head into the build plate can be particularly damaging.) Other routines can clog the extruder by extruding too much filament when the extruder is close to the print-bed.

Avoiding collisions on a print can also be a challenge. Even if the extruder tip follows a path that doesn't collide with any previously printed paths, the nozzle or the print-head box may, as shown in Figure 13. Collision avoidance is particularly challenging in the Turtle Geometry framework, with its emphasis on purely local geometry. Collisions can mostly be avoided by making careful calculations, such as those described in the section on nonplanar slicing, and by working in the Grasshopper/Rhino environment so that paths can be visualized prior to printing.

When generating delicate structures, we have found that some parameters–like extrude rate, speed, and temperature–may need to be re-tuned for different filaments as well as different 3D printers. Tuning can be a painstaking process that involves generating and printing many iterations of a design. Different filaments produce better results for different prints. For instance, Wood PLA–a filament that is a combination of plastic and wood particles– is stiffer than pure PLA and produces especially nice brushes and string sculptures.

Some of these drawbacks could be addressed by adding more slicer-like functionality to our software. For instance, it could present a warning whenever a turtle moves outside of the print-area, or prevent the writing of G-CODE for such paths. Though a significant undertaking, it should also be possible to add collision detection to the library. We could also conduct a series of structured tests on different printers, filaments and parameter settings to develop guidelines for this aspect of G-CODE generation.

We have found `ExtruderTurtle` to be a powerful tool for exploring the possibilities of custom G-CODE generation. With it, we have been able to recreate designs proposed by other researchers and, more importantly, explore a wide range of new forms. We anticipate using the library regularly, both in research and teaching.

We are also excited to explore using `ExtruderTurtle` in conjunction with a wider range of printers. In particular, we are interested in working with printers that employ unconventional materials, such as ceramics and food [6, 11, 18]. `ExtruderTurtle` provides much more flexibility and control over the printer and, thereby, finer control over unique material properties that may be leveraged on such machines.

REFERENCES

[1] Harold Abelson and Andrea diSessa. 1980. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics.* The MIT Press.

[2] Atheer Awad, Aliya Yao, Sarah J. Trenfield, Alvaro Goyanes, Simon Gaisford, and Abdul W. Basit. 2020. 3D Printed Tablets (Printlets) with Braille and Moon Patterns for Visually Impaired Patients. *Pharmaceutics* 12, 2 (Feb. 2020), 172. https://doi.org/10.3390/pharmaceutics12020172

[3] Donald Degraen, André Zenner, and Antonio Krüger. 2019. Enhancing Texture Perception in Virtual Reality Using 3D-Printed Hair Structures. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems.* ACM, Glasgow Scotland Uk, 1–12. https://doi.org/10.1145/3290605.3300479

[4] Franklin Dyer. 2021. Extruder Turtle Website. https://handandmachine.org/projects/extruder_turtle/index.html

[5] Michael Eisenberg. 2005. Technology and the Future of Educational Crafts. *Educational Technology* 45, 3 (2005), 3–11. https://www.jstor.org/stable/44429206 Publisher: Educational Technology Publications, Inc.

[6] Chang He, Min Zhang, and Zhongxiang Fang. 2020. 3D printing of food: pretreatment and post-treatment of materials. *Critical Reviews in Food Science and Nutrition* 60, 14 (Aug. 2020), 2379–2392. https://doi.org/10.1080/10408398.2019.1641065

[7] Yasusi Kanada. 2015. "3D Turtle Graphics" by using a 3D Printer. *Journal of Engineering Research and Applications* 5 (April 2015), 70–77.

[8] Yasusi Kanada. 2016. Method for Procedural 3D Printing Using a Python Library. *Journal of Information Processing* 24, 6 (2016), 908–916. https://doi.org/10.2197/ipsjjip.24.908

[9] Yasusi Kanada. 2018. Complex Moiré Patterns Generated by Helical 3D Printing with Three Waves. (2018), 2.

[10] Gierad Laput, Xiang 'Anthony' Chen, and Chris Harrison. 2015. 3D Printed Hair: Fused Deposition Modeling of Soft Strands, Fibers, and Bristles. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, Charlotte NC USA, 593–597. https://doi.org/10.1145/2807442.2807484

[11] Alain Le-Bail, Bianca Chieregato Maniglia, and Patricia Le-Bail. 2020. Recent advances and future perspective in additive manufacturing of foods based on 3D printing. *Current Opinion in Food Science* 35 (Oct. 2020), 54–64. https://doi.org/10.1016/j.cofs.2020.01.009

[12] LIA. [n.d.]. Filament Sculptures – LIA. https://www.liaworks.com/theprojects/filament-sculptures/

[13] Jon Millington. 1999. *Curve Stitching: Art of Sewing Beautiful Mathematical Patterns* (2007th edition ed.). Tarquin Group, Diss. String art, string-art.

[14] Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, François Guimbretière, and Patrick Baudisch. 2014. WirePrint: 3D printed previews for fast prototyping. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, Honolulu Hawaii USA, 273–280. https://doi.org/10.1145/2642918.2647359

[15] Jifei Ou, Gershon Dublon, Chin-Yi Cheng, Felix Heibeck, Karl Willis, and Hiroshi Ishii. 2016. Cilllia: 3D Printed Micro-Pillar Structures for Surface Texture, Actuation and Sensing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, San Jose California USA, 5753–5764. https://doi.org/10.1145/2858036.2858257

[16] Seymour Papert. 1980. *Mindstorms: children, computers, and powerful ideas*. Basic Books, New York, NY.

[17] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer. 1990. *The Algorithmic Beauty of Plants* (first edition ed.). Springer, New York Heidelberg.

[18] Ronald Rael, Virginia San Fratello, Barrak Darweesh, Constantina Tsiara, and Alexander Curth. 2020. POTTERWARE. https://www.potterware.com/docs

[19] RepRap. [n.d.]. G-code Reference Wiki. https://reprap.org/wiki/G-code

[20] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. 2005. Design principles for tools to support creative thinking. (2005).

[21] Manuel Riel and Ralf Romeike. 2020. Fundamental Concepts of 3D Turtle Geometry. *CONSTRUCTIONISM 2020* (2020), 332.

[22] Manuel Riel and Ralf Romeike. 2021. 3D Print your Artifacts–3D Turtle Geometry as an Introduction to Programming. In *2021 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1454–1461.

[23] Rosenbaum, Eric. [n.d.]. Beetle Blocks - Visual code for 3D design. http://www.beetleblocks.com/

[24] L. Sabantina, F. Kinzel, A. Ehrmann, and K. Finsterbusch. 2015. Combining 3D printed forms with textile structures - mechanical and geometrical properties of multi-material systems. *IOP Conference Series: Materials Science and Engineering* 87 (July 2015), 012005. https://doi.org/10.1088/1757-899X/87/1/012005 Publisher: IOP Publishing.

[25] Christian Schumacher, Bernd Bickel, Jan Rys, Steve Marschner, Chiara Daraio, and Markus Gross. 2015. Microstructures to control elasticity in 3D printing. *ACM Transactions on Graphics* 34, 4 (July 2015), 1–13. https://doi.org/10.1145/2766926

[26] Sevenson, Brittney. 2014. Incredible 3D Printed Paintbrush, Broom, & More Created with Fiber Bridging Technique. https://3dprint.com/32480/3d-print-paintbrush-bridging/

[27] Jaeeun Shin, Jundong Cho, and Sangwon Lee. 2020. Please Touch Color: Tactile-Color Texture Design for The Visually Impaired. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–7. https://doi.org/10.1145/3334480.3383003

[28] Cynthia J. Solomon and Seymour Papert. 1976. A case study of a young child doing turtle graphics in LOGO. In *Proceedings of the June 7-10, 1976, national computer conference and exposition on - AFIPS '76*. ACM Press, New York, New York, 1049. https://doi.org/10.1145/1499799.1499945

[29] Haruki Takahashi and Jeeeun Kim. 2019. 3D Printed Fabric: Techniques for Design and 3D Weaving Programmable Textiles. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 43–51. https://doi.org/10.1145/3332165.3347896

[30] Haruki Takahashi and Homei Miyashita. 2017. Expressive Fused Deposition Modeling by Controlling Extruder Height and Extrusion Amount. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, Denver Colorado USA, 5065–5074. https://doi.org/10.1145/3025453.3025933

[31] Sebastian Teo. 2018. Droid - 3d Print Slicer and Path Plotter. https://www.food4rhino.com/en/app/droid-3d-print-slicer-and-path-plotter

[32] Tom Verhoeff. 2010. 3D turtle geometry: artwork, theory, program equivalence and symmetry. *International Journal of Arts and Technology* 3, 2/3 (2010), 288. https://doi.org/10.1504/IJART.2010.032569

[33] Eric W. Weisstein. [n.d.]. Rule 30. https://mathworld.wolfram.com/Rule30.html Publisher: Wolfram Research, Inc.

[34] Ursula Wolz, Michael Auschauer, and Andrea Mayr-Stalder. 2019. Programming embroidery with turtlestitch. In *ACM SIGGRAPH 2019 Studio (SIGGRAPH '19)*. Association for Computing Machinery, New York, NY, USA, 1–2. https://doi.org/10.1145/3306306.3328002

[35] Robert C. Yates. 1974. *Curves and their properties* (reprint edition ed.). National Council of Teachers of Mathematics.