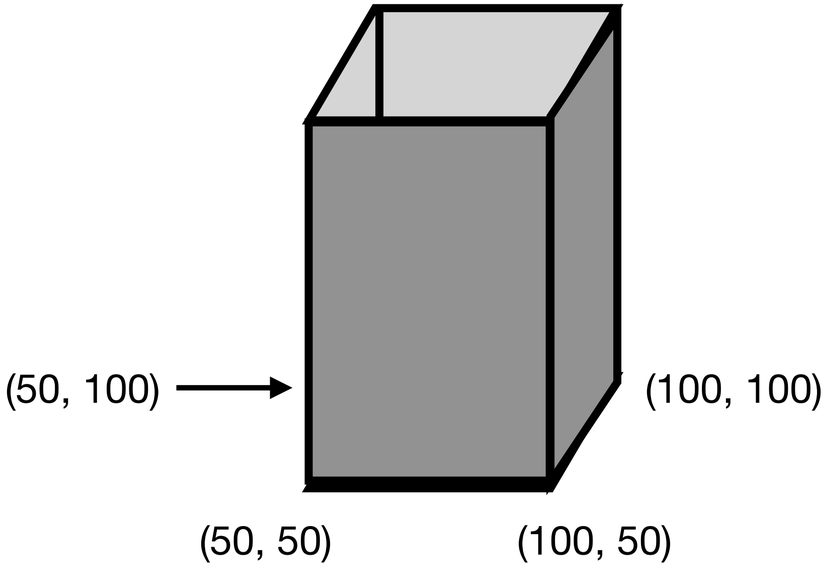
Experiments with GCODE

Franklin Dyer
CS 491

Some objects are simple enough to generate with hand-written GCODE, without even using a slicer.




```
; initialization sequence
. . .
. . .
. . .

G01 X50 Y50
G01 X50 Y100 E2.5
G01 X100 Y100 E2.5
G01 X100 Y50 E2.5
G01 X50 Y50 E2.5
G01 Z0.2

G01 X50 Y100 E2.5
G01 X100 Y100 E2.5
G01 X100 Y50 E2.5
G01 X50 Y50 E2.5
G01 Z0.4

. . .
```

First layer

Second layer

etc…

But this could take a *really long time* to write by hand!

Here's a very simple example showing how to open and read from/write to text files in Python.

`copy_file.py`

```
import sys

filename = sys.argv[1]
new_filename = sys.argv[2]

gcode = ""

with open(filename) as file:
    for line in file:
        gcode += line

new_file = open(new_filename, 'w')
new_file.write(gcode)
new_file.close()
```

Get the names of the input and output files (command-line arguments)

Loop through the lines of the input file, and add them to a string

Write that string to the output file

Running `python3 copy_file.py file1.txt file2.txt` will copy `file1.txt` to `file2.txt`.

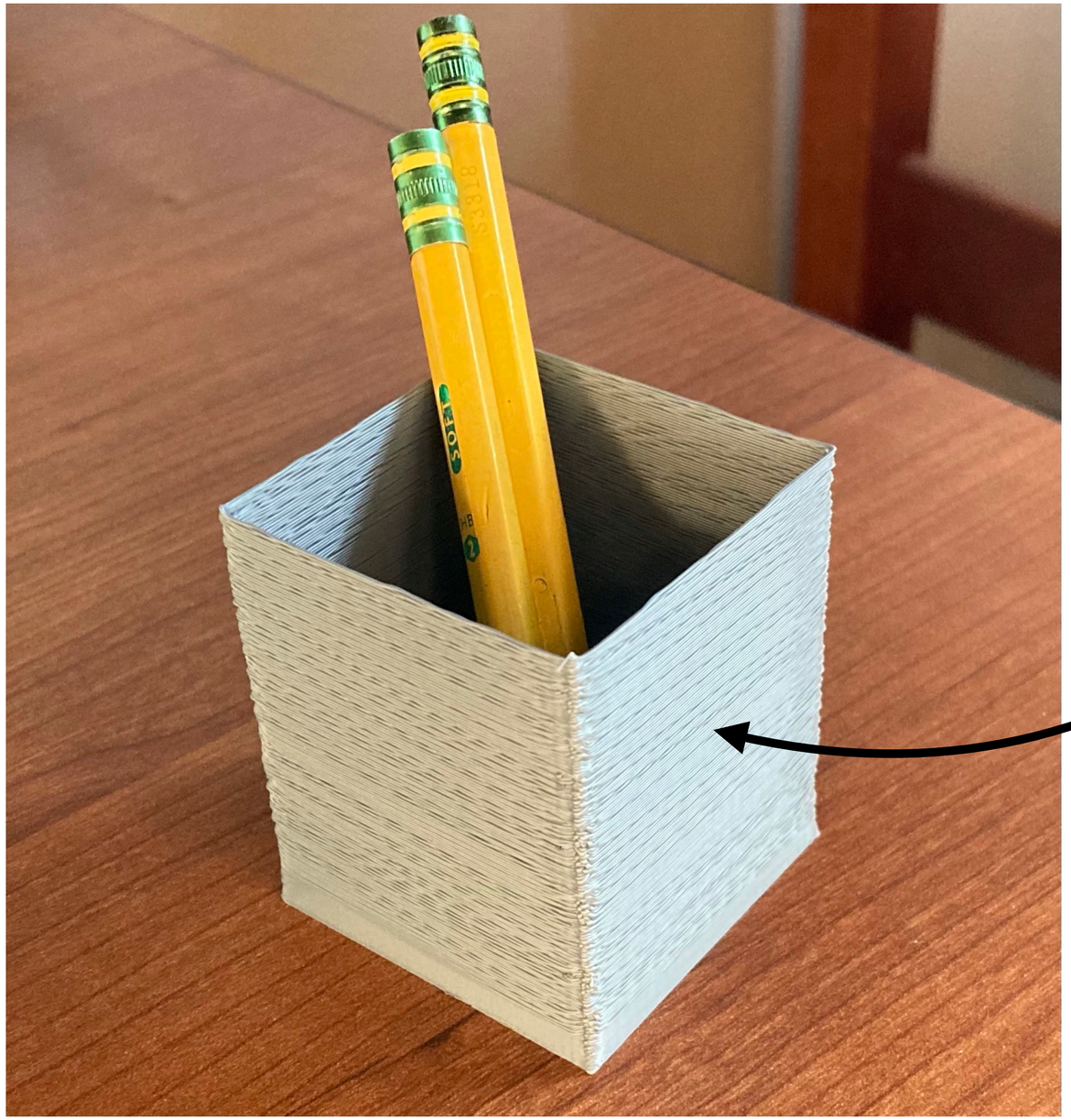
Let’s write some Python to automatically generate the GCODE for a rectangular prism!

We can copy the initialization and finalization commands from pre-written GCODE files.

In between, we want to repeatedly write the following block of GCODE:

```
G01 X50 Y100 E2.5
G01 X100 Y100 E2.5
G01 X100 Y50 E2.5
G01 X50 Y50 E2.5
G01 Z???
```

Should be replaced by 0.2 times the level number



The lines are kind of rough…
how to make them smoother?

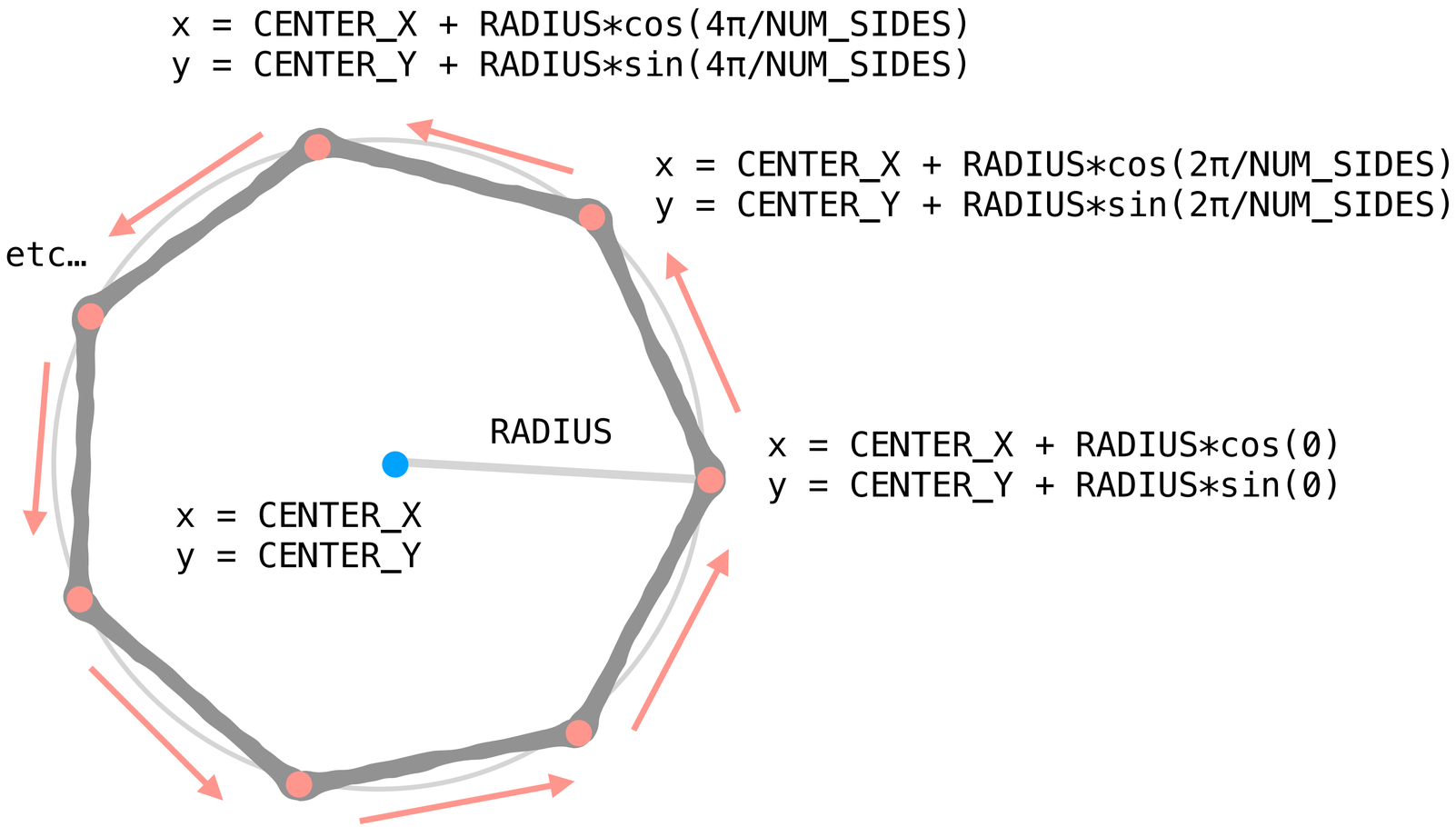
We can make the print smoother by adding “dwell” statements to slow down the extruder and prevent the plastic from dragging.

```
G01 X50 Y100 E2.5
G04 S0.2
G01 X100 Y100 E2.5
G04 S0.2
G01 X100 Y50 E2.5
G04 S0.2
G01 X50 Y50 E2.5
G04 S0.2
G01 Z???
```

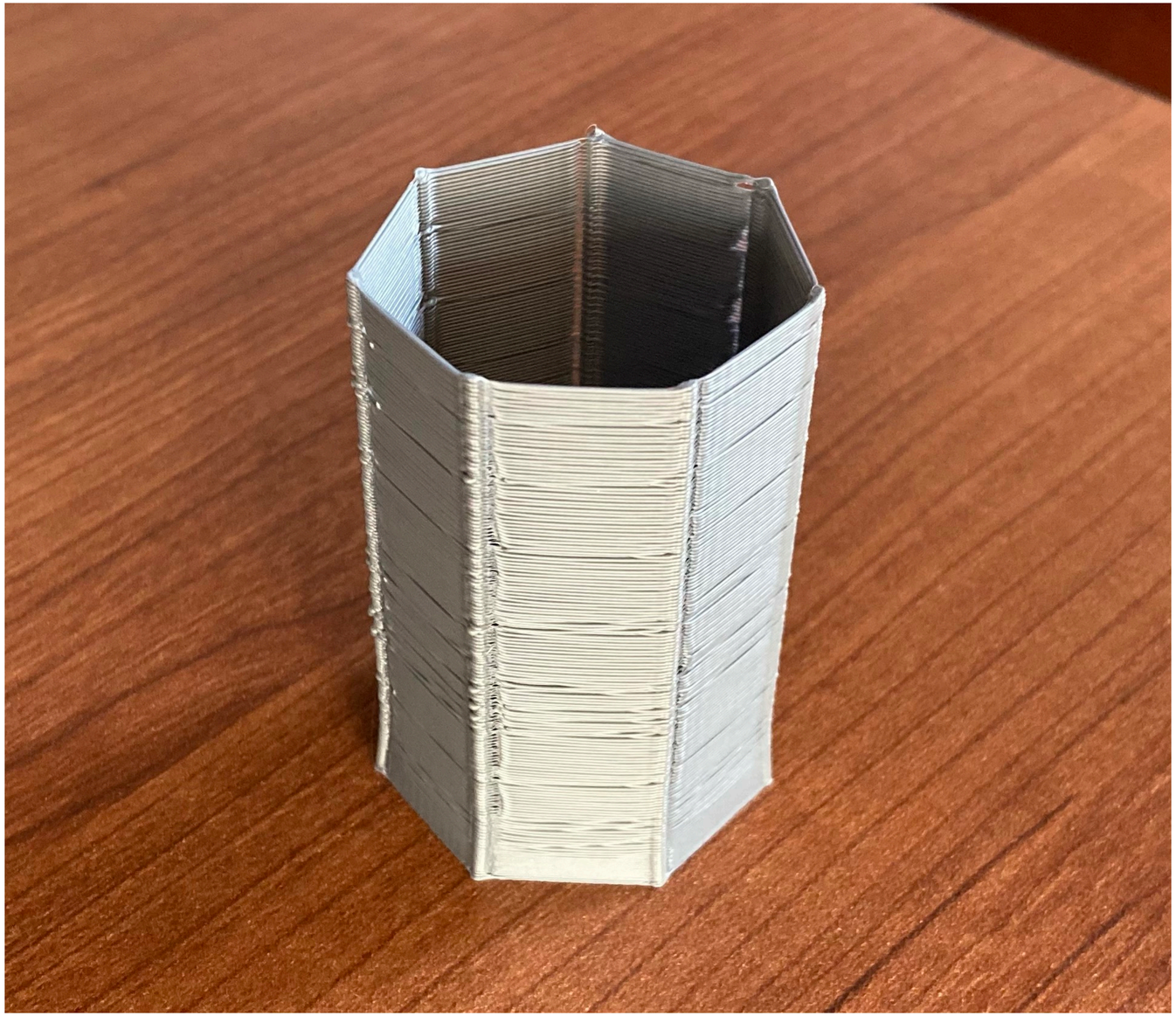
0.2 times the level number

Smoother lines because of
“dwelling” at each corner

What about drawing a rectangular polygon with many sides, instead of just a square?

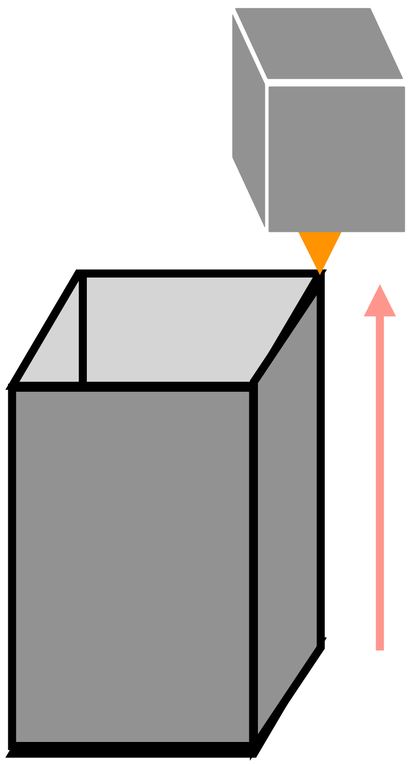


We need to loop over both the number of layers and the number of sides on each layer.



But we could have done all this with a slicer!

What cool things can we do by directly writing GCODE that can’t be done with a slicer?

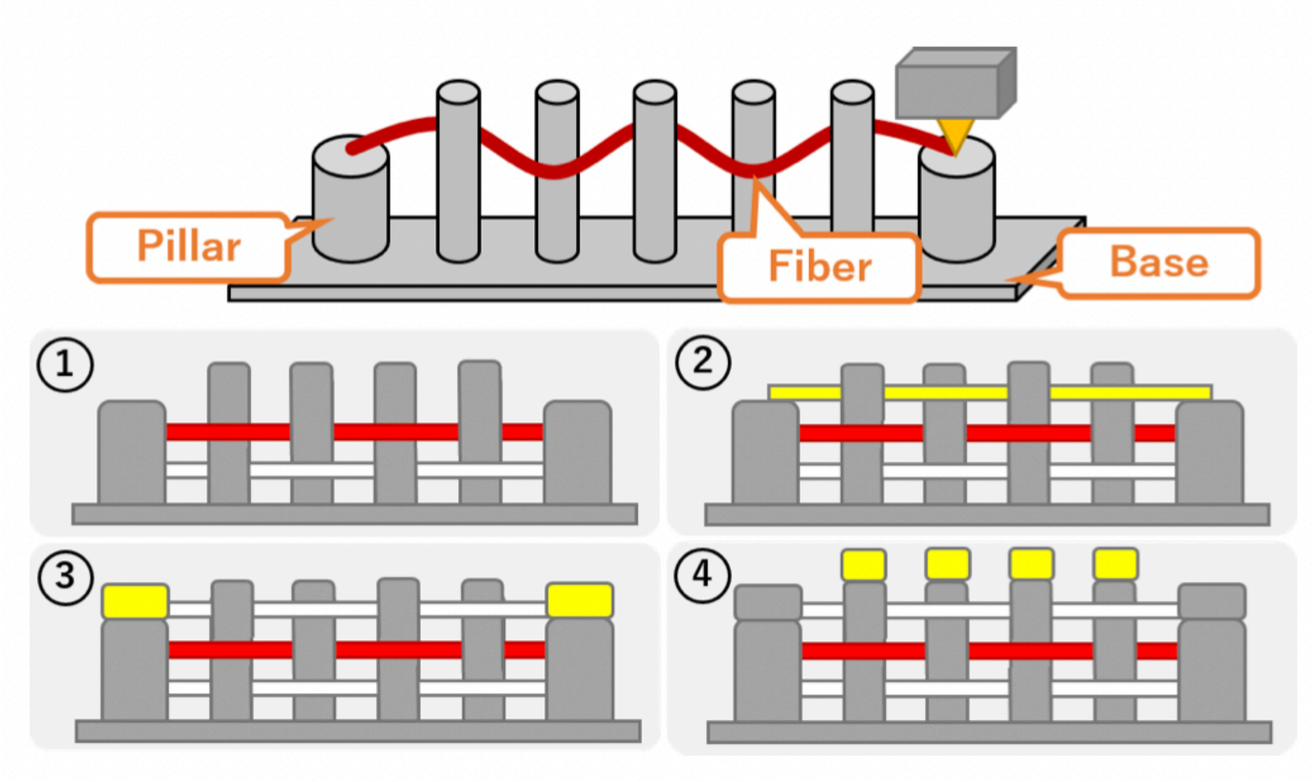


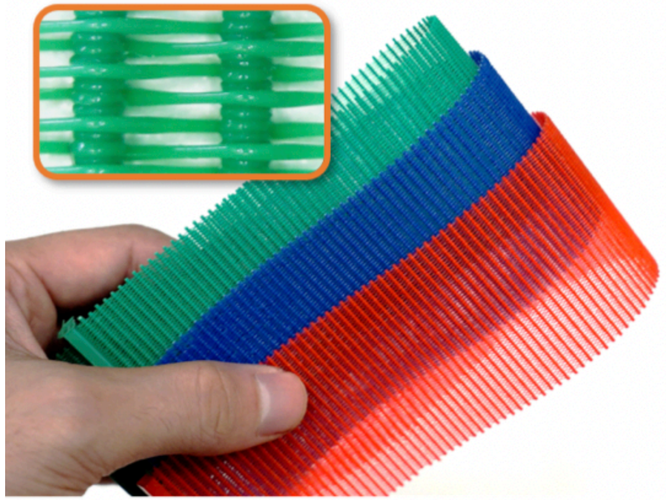
Slicers divide a solid into vertically stacked layers
and generate GCODE to print these layers from bottom to top.

After moving upwards, the extruder nozzle never goes back down.

When moving the nozzle down to heights where plastic has already been extruded, there is a risk of collision - but this can be used to generate some unique structures.

We can “weave” lengths of filament between extruded pillars, creating a fabric-like material.





Takahashi, H., Kim, J. (2019). *3D Printed Fabric: Techniques for Design and 3D Weaving Programmable Textiles.* UIST 2019: New Orleans, LA, United States.

To avoid rewriting verbose/clunky pieces of code over and over again, I wrote a custom Python class called “Extruder” that can be used to generate and save snippets of GCODE more concisely.

For example, here’s how the square prism from earlier can be generated using my Extruder class:

```
exec(open("Extruder.py").read())

ext = Extruder(50, 50, 0.2)
ext.set_density(0.05)
ext.initialize()
ext.feedrate(3000)

for l in range(300):
    ext.drawline(50, 100)
    ext.drawline(100, 100)
    ext.drawline(100, 50)
    ext.drawline(50, 50)
    ext.lift(0.2)

ext.finalize()
ext.save("square-prism.gcode")
```

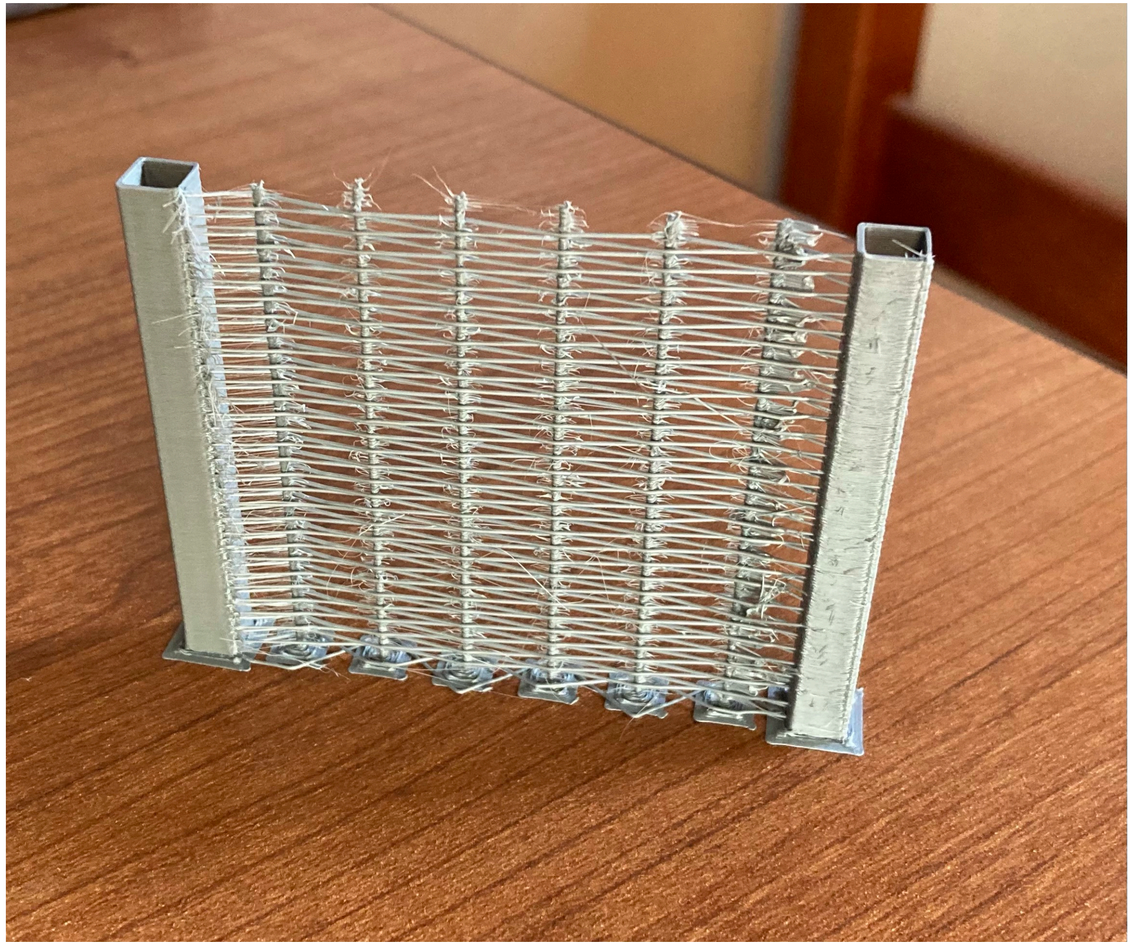
(Import Extruder classfile)

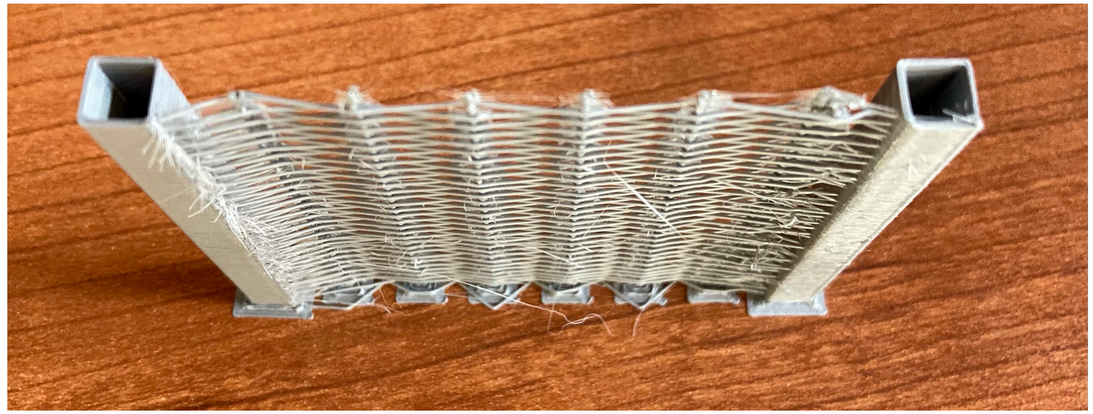
(Set up extruder)

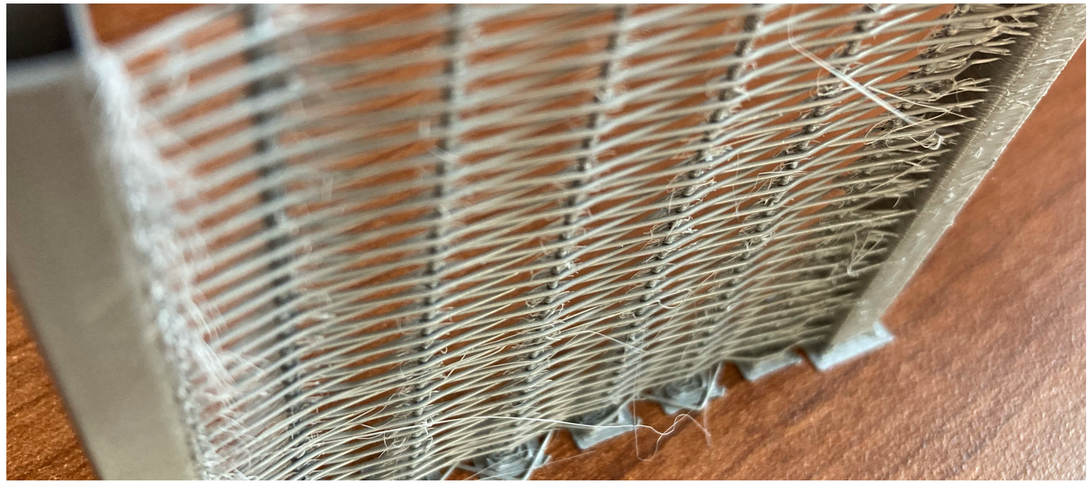
(Draw the squares on each layer)

(Finalize and save to a GCODE file)

The Extruder classfile is available on the course website.

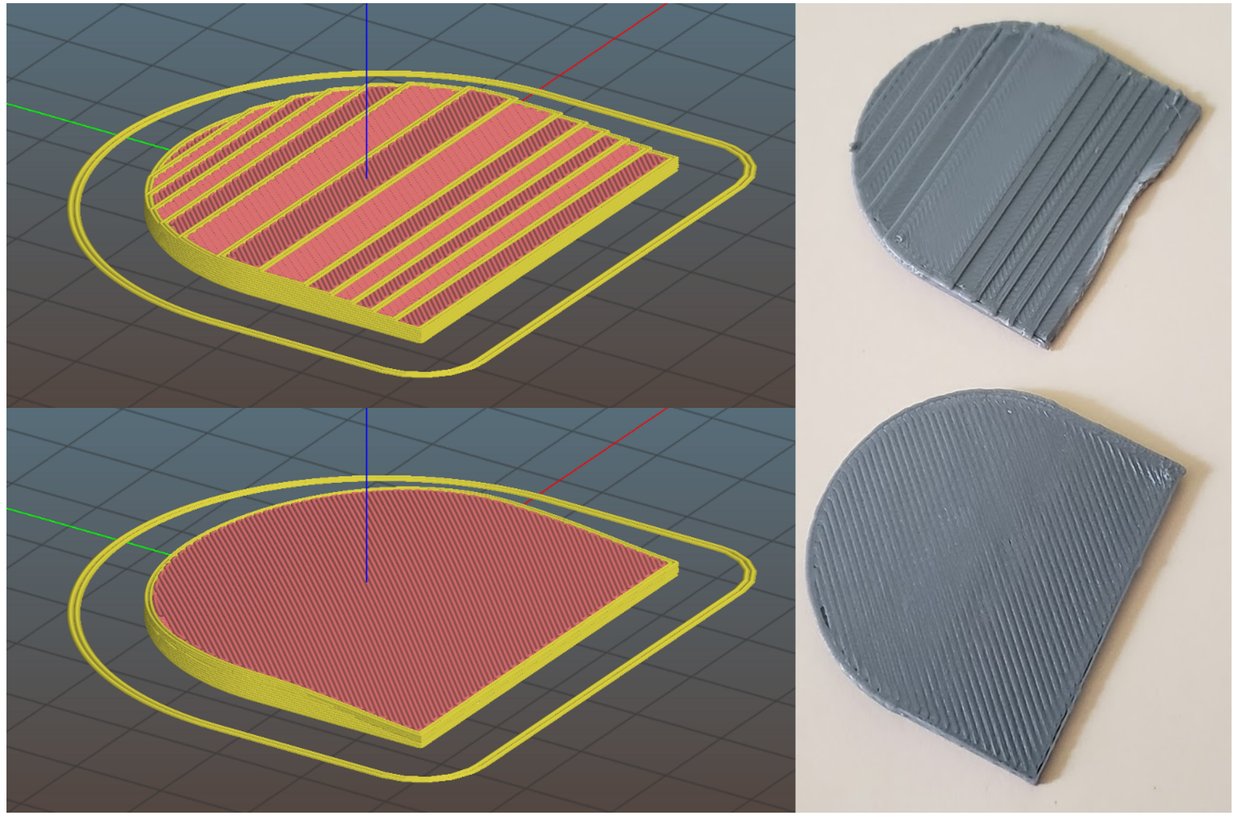






Weaving filament in a circle (regular polygon) instead of a straight line.

Very gently sloped surfaces tend not to come out very well when sliced into horizontal planes.



Nonplanar slicing can be used to avoid the “stair stepping effect” for gently sloped surfaces.

Instead of horizontal planes, the solid is sliced into slightly curved sheets.

If the curvature is slight enough, there isn’t a risk of the extruder nozzle/body colliding with other parts of the print.

There is a version of the open-source slicer Slic3r that supports nonplanar slicing.

Nonplanar Slicing. (2020, September 14). In *Appropedia*. https://www.appropedia.org/Nonplanar_Slicing

We can also use Python to alter existing GCODE files that have been generated by a slicer.

“Pug buddy” sample print

We've already seen how to read the lines of a GCODE file using Python.

We can also write functions that extract the X, Y, Z, and E values from a single line of GCODE, or replace the X, Y, Z and E values in a line of GCODE.

```
extract_coords("G1 X100 Y50 Z0 E2.5")  →  [100, 50, 0, 2.5]

extract_coords("G1 X100 Y50 E2.5")  →  [100, 50, False, 2.5]

extract_coords("G0 X100 Y50 Z0")  →  [False, False, False, False]
```

```
substitute_coords(100, False, False, False, "G1 X50")
                                                  →  "G1 X100"

substitute_coords(False, 200, 0, 5, "G1 X50 Y50 Z1 E2.5")
                                                  →  "G1 X50 Y200 Z0 E5"
```

Using these functions, we can make fine-grained textural changes that would be difficult to implement with a slicer.

For example, we can use the following code to add randomness to all of the X and Y coordinates in a GCODE file, giving the print a bumpy/“fuzzy” texture.

```
new_gcode = “”

with open(filename) as file:

  for line in file:

    match = extract_coords(line)

    if match[0] or match[1]:

      x = float(match[0])
      y = float(match[1])
      x += MAX_PERTURBANCE * (2*random.random()-1)
      y += MAX_PERTURBANCE * (2*random.random()-1)

      new_command = substitute_coordinates(x, y, False, False, line)
      new_gcode += new_command

    else:
      new_gcode += line
```

(Empty GCODE string)

(Finalize and save to a GCODE file)

(Loop over file lines)

(Draw the squares on each layer)

(Analyze GCODE command)

(Check for X/Y coords)

(Add randomness to coords)

(Create modified command)

(Otherwise, don’t change line)

Alteration of the “Pug buddy” test print in which more and more randomness is added for layers with higher Z-values

Same premise, but with less randomness,
and a more gradual increase in randomness

Another example, in which randomness is only added to the right side
(Only perturb coordinates with an X-value above the average X-value)

A different transformation: “twist” the print by rotating X and Y coordinates about a vertical axis, increasing the rotation amount for layers with greater Z-values

Some failed experiments:

NAILED IT!

Questions?