6.8420: Computational Design and Fabrication
Charlotte Folinus
Massachusetts Institute of Technology
Spring 2024

Homework 3: Design to Machine Instructions

Section 1: Printing with a virtual printer

Problem 1.1: Set up a virtual printer
This section did not ask any specific questions.

Problem 1.2: Virtually print an object
I sliced a 3D model for a topo map of Franconia Notch in the White Mountains of New Hampshire.

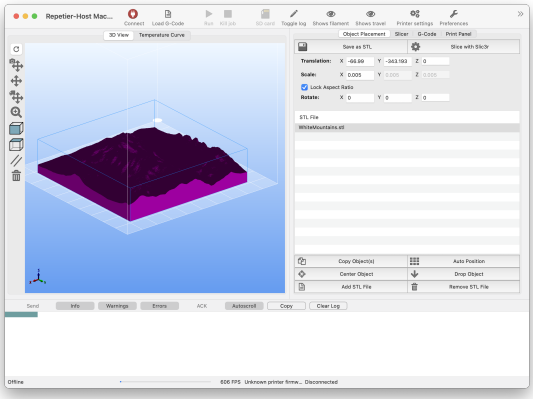

Figure 1: Full 3D model

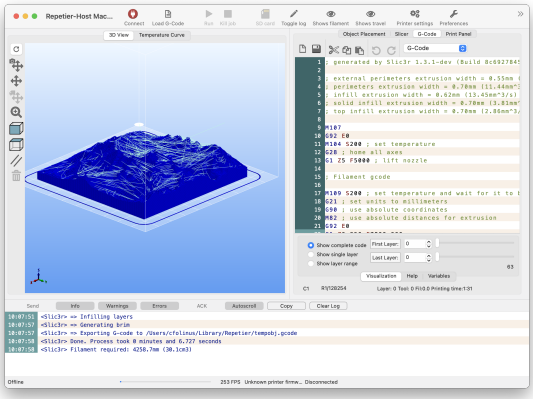

Figure 2: Full sliced representation (blue), including print head traverses (green)

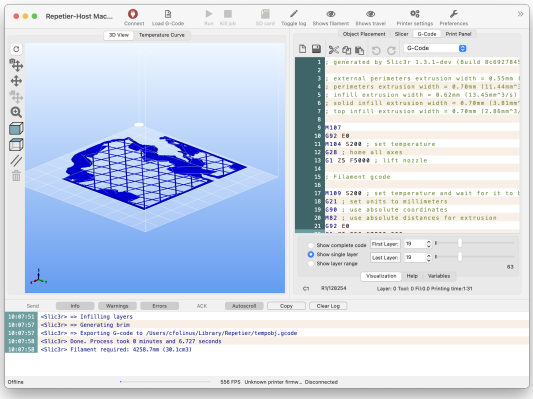

Figure 3: Path for an individual layer (layer 12)

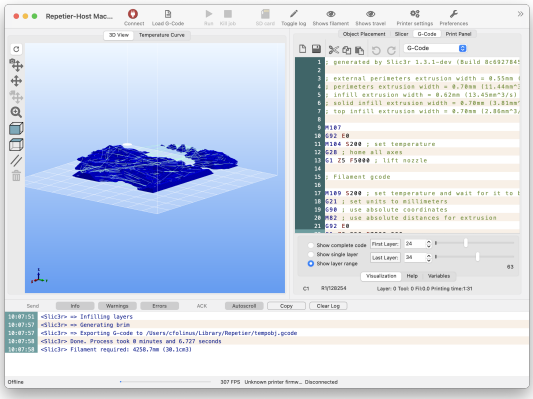

Figure 4: Paths for a range of layers

Section 2: Mesh slicing

Problem 2.1: Implement `create_contours`

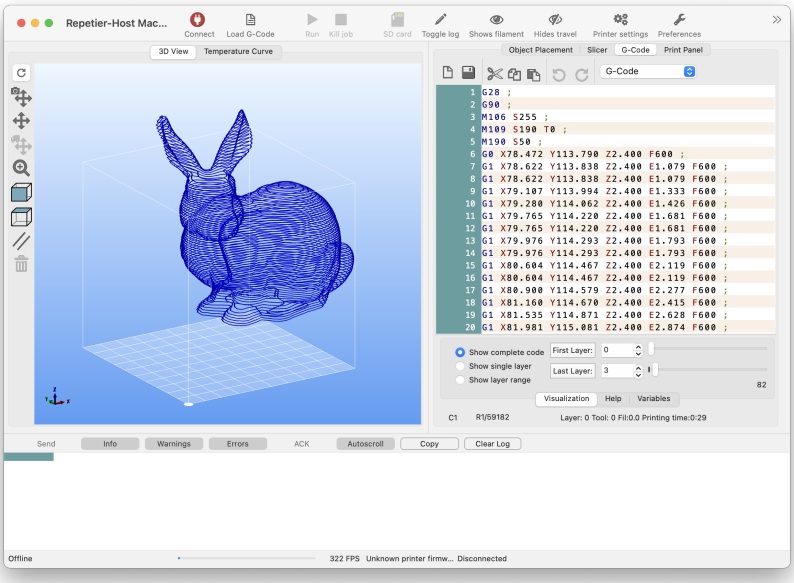

Figure 5: The sliced stanford bunny!

Problem 2.2: Limited robustness with "triangle soup" Right now, my slicer is not capable of handling missing facets, which would produce open contours (instead of closed contours) and not be added to the list of contours.

- Detection: I currently stop adding points to a contour once it has been flagged as closed (the next node is the same as the current start/end node). Currently, if the contour is open, the code continues trying to add points until there are no more edges/points to add. Open contours could be detected as contours that are both open and not able to be connected to any other nodes (the next closest node is farther than our permissible tolerance).
- Handling: After detecting an open contour, I could repair this defect by adding an additional edge to connect the end point to the start point.

Section 3: Abstractions for sheet metal design

Problem 3.1: Design a DSL *This section did not ask any specific questions.*
Problem 3.2: Implement an interpreter
This section did not ask any specific questions.

Problem 3.3: Design a cube

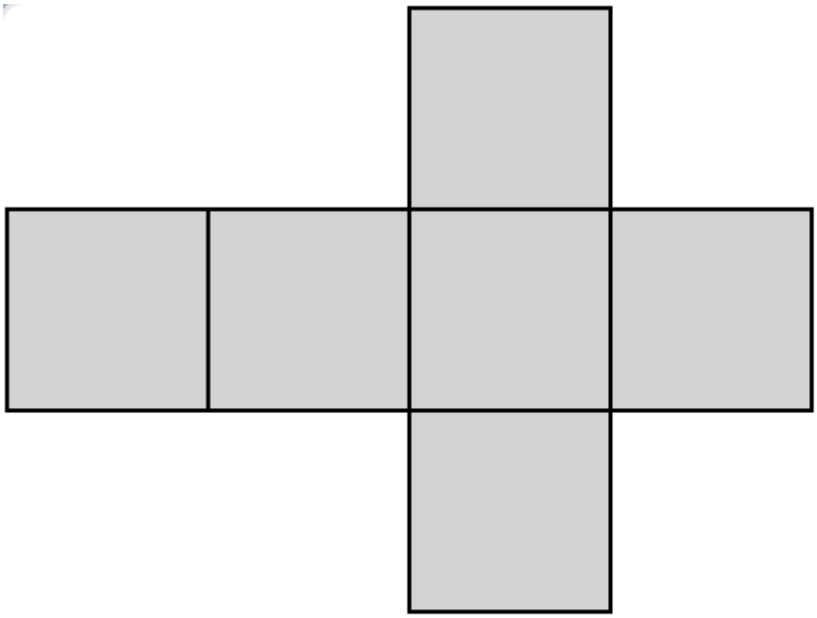

Figure 6: Cube pattern (screenshot of SVG output)

Problem 3.4: Make a more complex shape I created a pattern for a linear rail guide. The rail guide has smaller tabs along its base, which could be used to rivet/fasten/attach it to a base plate.

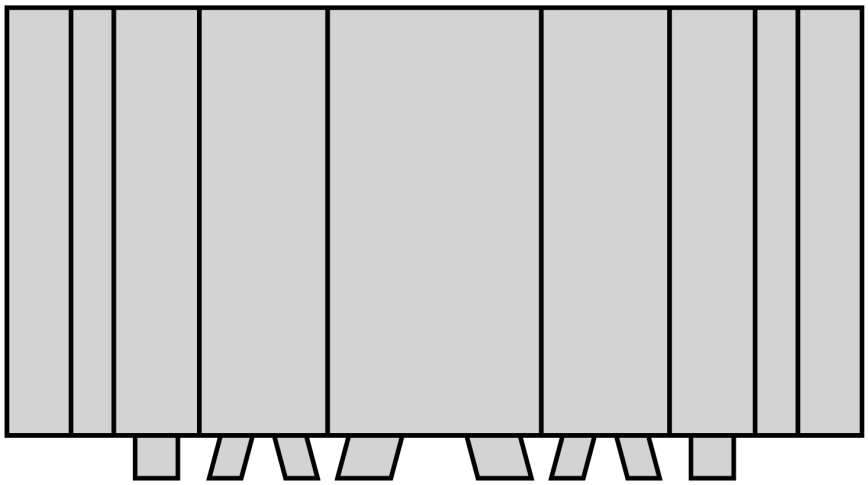

Figure 7: Pattern for slide rail geometry (screenshot of SVG output)

Figure 8: Side view of the assembled slide rail.

Figure 9: Top view of the assembled slide rail (please pardon my non 90 degree angles!).

Section 4: High-level discussion about crafting DSLs

Problem 4.1: Using a language that describes a fabrication process can be interesting because it ensures that everything created in this language can be manufactured using this process. Consider, for example, a language for carpentry described in class. Give two concrete applications you can imagine using this language for.

Application 1: Designing commercial (office) furniture for low financial and carbon cost

- Many companies are increasingly concerned with the enviromental impact of the goods they sell or that they purchase for furnishing their offices. However, they may still want their furniture to be creative and expressive (with some personality, not just boring pieces!).
- A DSL for carpentry could be used in the fabrication/manufacturing of these pieces
- The overall fabrication cost could be computed, which might vary depending on factors like the geographic region of production (i.e., Switzerland vs. China). However, manufacturing in low-manufacturing cost regions may incur a higher shipping cost (and higher associated carbon cost of shipping) to regions in Europe and the United States.
- Shipping and production may also have design- and location-specific carbon or embodied energy costs, which might be of interest to companies seeking to have a low embodied carbon while also having a low financial cost and highly creative furniture.
- The DSL could be used to navigate tradeoffs about design and manufacturing location to help designers understand how the fabrication process, total production cost, and embodied energy might differe depending on the production location/warehouse location (and the design!).

Application 2: Artisans

- There are many educational/artistic applications where designers may seek to create visually expressive/interesting pieces in a way that is financially tractable (your time is money, and you may not have the highest-end equipment).
- Cottage-scale makers (or schools) might be limited by factors like equipment size, and these constraints could be incorporated into the DSL-suggested fabrication strategies.
- This type of tool could also be useful for project bidding or early-stage costing, which is often an important of running a successful design/art studio.

Problem 4.2: However, there are challenges with the idea of crafting a DSL that describes all that can be manufactured with a given device. First, it may be very hard to express a DSL that ensures that everything you express in it is manufacturable with a specific process, is sufficiently expressive to ensure that you can actually express everything that can be manufactured using this language, and is also simple enough so that writing programs in this language is not too daunting. With this in mind, we often need to make some compromises. The DSL we proposed earlier in this assignment, tailored to describe the fabrication process of creating tabs and then bending them, has its limitations. Identify and describe at least one issue that might arise in the fabrication process that this language does not account for. Propose a verifier to check if this issue occurs in the language.

The DSL we used for sheet metal design does not directly compute the geometry of the bent part, whether this geometry is physically feasible (e.g. not self-intersecting), or whether this geometry can be realistically fabricated with available tools. Sheet metal tooling/equipment has finite-sized tool radii, and the tools need to be able to enter/leave (you likely cannot bend a perfect cube without using additional components/features, because the sheet metal die needs a way to access the part). It would be relatively straightforward to detect whether the bent sheet metal part is self-intersecting by computing the 3D geometry of the deformed shape – we have already provided the DSL with information about the bend angles of each tab, and we could use this to compute each tab's position in 3D space. If the parallelogram defining one tab intersects with the parallelogram defining another tab, the two tabs intersect, and the part geometry would need to be corrected.