Bendy Print

October 19, 2019

1 Abstract

In this work I explore applications for flat, bending filament prints. Any convex mesh can be rigidly unwrapped, sliced and printed using generated G-Code. Furthermore, if one wishes to fold the print to its original shape, triangle extrusions can be generated that give guidance and provide flat surfaces for gluing.

2 Introduction

Printing thin objects with FDM printers is quite convenient: They can be printed in little time and the surface facing the print bed tends to be exceptionally smooth. A custom tool that provides the ability to leverage these properties for

more general cases seems worth consideration. Additionally, custom C-Code generators give more freedom to achieve desired properties like how easily final prints can bend.

In this project I want to find out what G-Code properties produce flat prints with desired properties like flexibility and sturdiness. Then I want to provide a possible application of such prints beyond 2D meshes: Generating flat prints that can be folded back into a 3D shape. Additionally I want to use an FEM simulation to deform and measure properties of the input mesh.

3 Related Work

The Youtube Channel "Make Anything" made a video on FDM printed textiles [1]. While some of the shown methods are also based on bending thin prints, a tool that creates G-Code automatically from an input mesh would make these methods more accessible.

4 Unwrapping

Provided 3D meshes need to be unwrapped rigidly. For this algorithm I assume getting a convex mesh which doesn't run into intersections, but it can be easily extended by doing collision checks and splitting the mesh into multiple wraps.

The unwrapping algorithm, starting at face 0, places each face in sequence until all faces are part of the unwrapped mesh. A transformation matrix is computed such that the two edge vertices that ought to be connected match the already placed triangle. Since the unwrapped mesh is used as a shell by the slicer, the same face from a scaled down version of the same source mesh is moved using the same transformation 1.

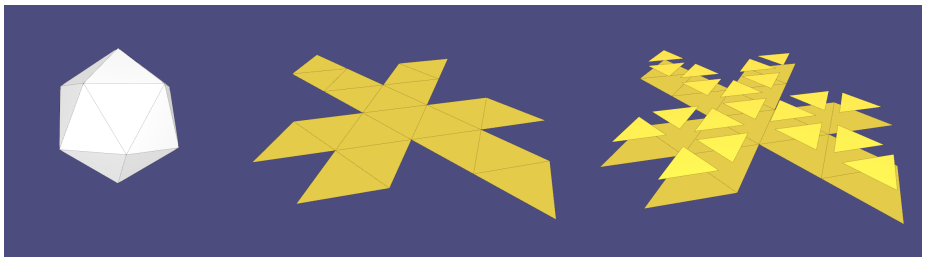

Figure 1: Unwrapping of an iso-sphere, with included shell triangles to the right.

[1] Make Anything video: https://www.youtube.com/watch?v=ge-q6iXDAoc

5 Slicer

5.1 Quad slicer

The first slicer I implemented is a rectangle shape with an axis aligned grid fill. The slicer itself can be implemented with a few for-loops, the difficult part is figuring out the semantics and range of the G-Code attributes. Repetier [2] helped a lot, since not only does it allow cross-referencing with its own slicer, it can also import G-Code and show the path of the printer head. This allows testing generated prints in a timely manner 2.

One important attribute in G-Code is extrude E. Extrude is an axis that describes the total amount of filament extruded in mm from the beginning of the print. This attribute is not well suited for printer parameters. Instead, one can define filament density D to compute how much filament needs to be extruded between points p and q like so: $|p - q| \cdot D$

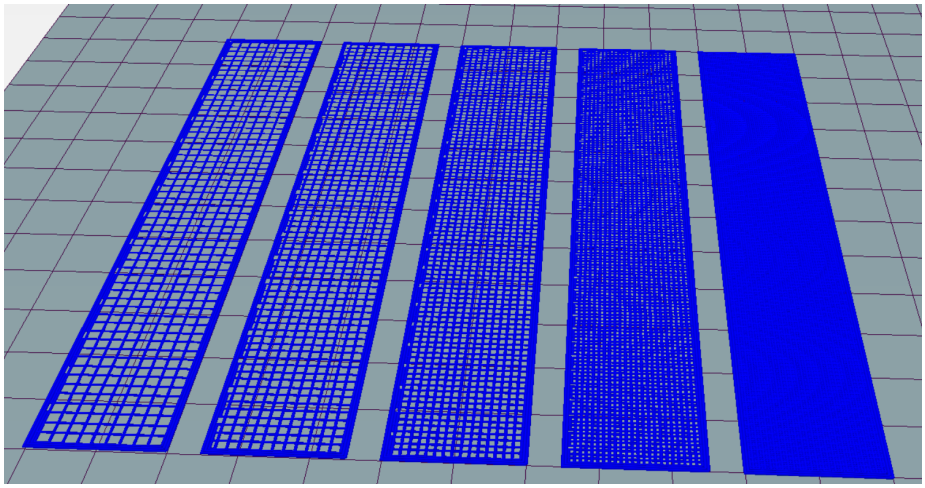

Figure 2: Generated rectangles with differing density to test which values are best suited for printing.

It is clear that best results in sturdiness are achieved with as little gaps in the filament as possible (the more dense the better), while the flexibility seems to almost solely depend on the layer height. Best prints are therefore achieved with as thin and as dense prints as possible.

[2] https://www.repetier.com/

5.2 Interval slicer

To slice a general 2D shape I implemented a scan-line algorithm using slices of intervals. Given an input direction $v \in \mathbb{R}^2$ that is supposed to be orthogonal to the print path, a rotation matrix $R = \begin{bmatrix} v_x & v_y \\ v_y & -v_x \end{bmatrix}$ is constructed and all vertex positions transformed into slicer space. Line-intersections with print paths can be done efficiently by checking for a specific y value whether it is between a line's two vertices. Intersecting triangles will result in two intersections defining an interval $(a, b) \in \mathbb{R}^2$.

Interval-slices are data-structures that hold a list of intervals. When inserting an interval it will combine all overlapping intervals. This way, when using a scan-line algorithm along v with a given stride, all interval-slices span the mesh.

After the slices are computed a sequence of intervals are extracted by stacking intervals that have a lot of overlap to their previous interval along v. This way neighbouring intervals can be connected so the printer head doesn't need to jump every line 3.

5.3 Outlines

Outlines are computed using a triangle adjacency list. This way it can be determined which edges are connected to only one triangle and therefore are part of the outline. Since all triangle edges/vertices have the same rotation all these outline edges can be attached end-to-end to create closed outlines around the whole mesh.

To accommodate for the line width of the 3D printer the outlines need to be displaced to have a margin. Since again all outlines are rotated the same way, the 2D orthogonal to any given outline segment will always point inside the mesh and can be used for offsetting. Sharp corners have to be taken into consideration by extending the outline with an additional segment, otherwise the outline extends far beyond the corner even with small margins 4.

Having computed the outlines, instead of intersecting triangles for the fill the outlines can be sliced in a similar manner to prevent overlapping. Instead of creating intervals straight away from two line intersedctions, all line intersections for a given $y - value$ need to be computed and sorted. From this list intervals can be spanned every two entries.

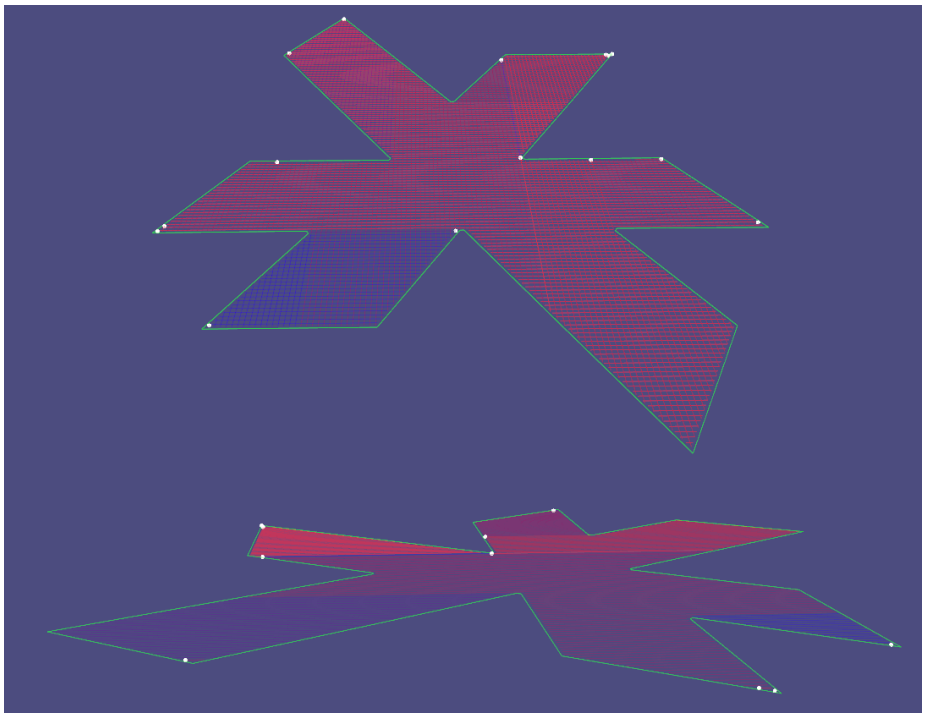

Figure 3: Interval sequences are coloured differently. Subsequent layers can simply define another v to make a grid-like fill.

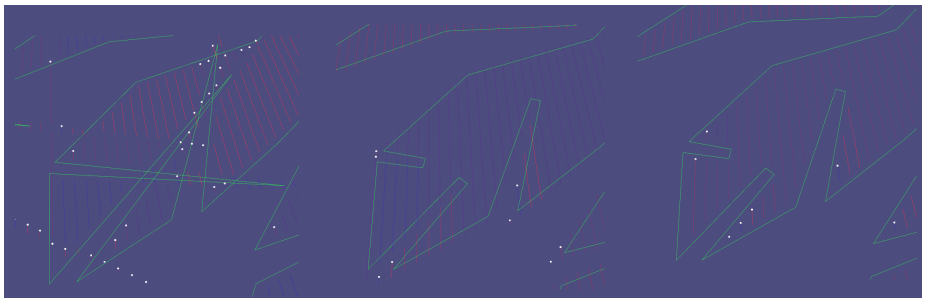

Figure 4: 1) Outlines without corner detection. 2) Outlines combined with triangle fill. 3) Slicing outlines instead of triangles.

6 FEM simulation

6.1 Derivations

As one of my primary goals for this project was to properly implement an FEM simulation, I decided to not auto-generate the solver code in Maple [3] but in-

[3] https://maplesoft.com/

stead, using the chain-rule, implement the different derivations (Deformation matrix, Strain, Energy density) independently.

Given the deformation gradient and its derivatives $F(x)$, $\frac{dF}{dx}$, $\frac{d^2F}{d^2x}$ as well as the energy density $U(F)$ and strain $E(F)$, for example St. Venant-Kirchhoff and Green strain:

$$U(F) = \frac{1}{2}\lambda \cdot tr(E)^2 + \mu \cdot tr(E \cdot E)$$
$$E(F) = \frac{1}{2}(F^T F - I)$$

The gradient and Hessian with respect to F can be computed as follows:

$$\frac{dU}{dF} = \frac{dE}{dF}^T \cdot \frac{dU}{dE}$$
$$\frac{d^2U}{dF \cdot dF^T} = \frac{dE}{dF}^T \cdot \frac{d^2U}{dE \cdot dF^T} + \frac{d^2E}{dF \cdot dF^T}^T \cdot \frac{dU}{dE}$$
$$\frac{d^2U}{dE \cdot dF^T} = \frac{dE}{dF}^T \cdot \frac{d^2U}{dE \cdot dE^T}$$

Similarly, the gradient and Hessian with respect to x:

$$\frac{dU}{dx} = \frac{dF}{dx}^T \cdot \frac{dU}{dF}$$
$$\frac{d^2U}{d^2x} = \frac{dF}{dx}^T \cdot \frac{d^2U}{dF \cdot dx} + \frac{d^2F}{d^2x}^T \cdot \frac{dU}{dF}$$
$$\frac{d^2U}{dF \cdot dx} = \frac{dF}{dx}^T \cdot \frac{d^2U}{dF \cdot dF^T}$$

With these parts being implemented individually, the type of strain/energy density and deformation matrix can be changed independently and only the direct derivatives $\frac{dE}{dF}$, $\frac{d^2E}{dF \cdot dF^T}$ and $\frac{dU}{dE}$, $\frac{d^2U}{dE \cdot dE^T}$ need to be given by the programmer. The validity of all computations are checked via finite difference.

6.2 Implementation

The FEM solver currently supports fixed vertices and external forces, which can be applied to selected vertices 6. The solver supports gradient descent with adaptive step-size.

Figure 5: FEM derivations and tests using finite difference

7 Outlook

While a lot has been achieved, almost all pieces of this project have great potential to be expanded.

1. The FEM solver could be done with Newtons Method, but the Hessian computation doesn't work correctly yet.
2. The mesh unwrapping could support non-convex meshes by doing collision detection when placing triangles.
3. The number of jumps in the generated G-Code could be reduced.

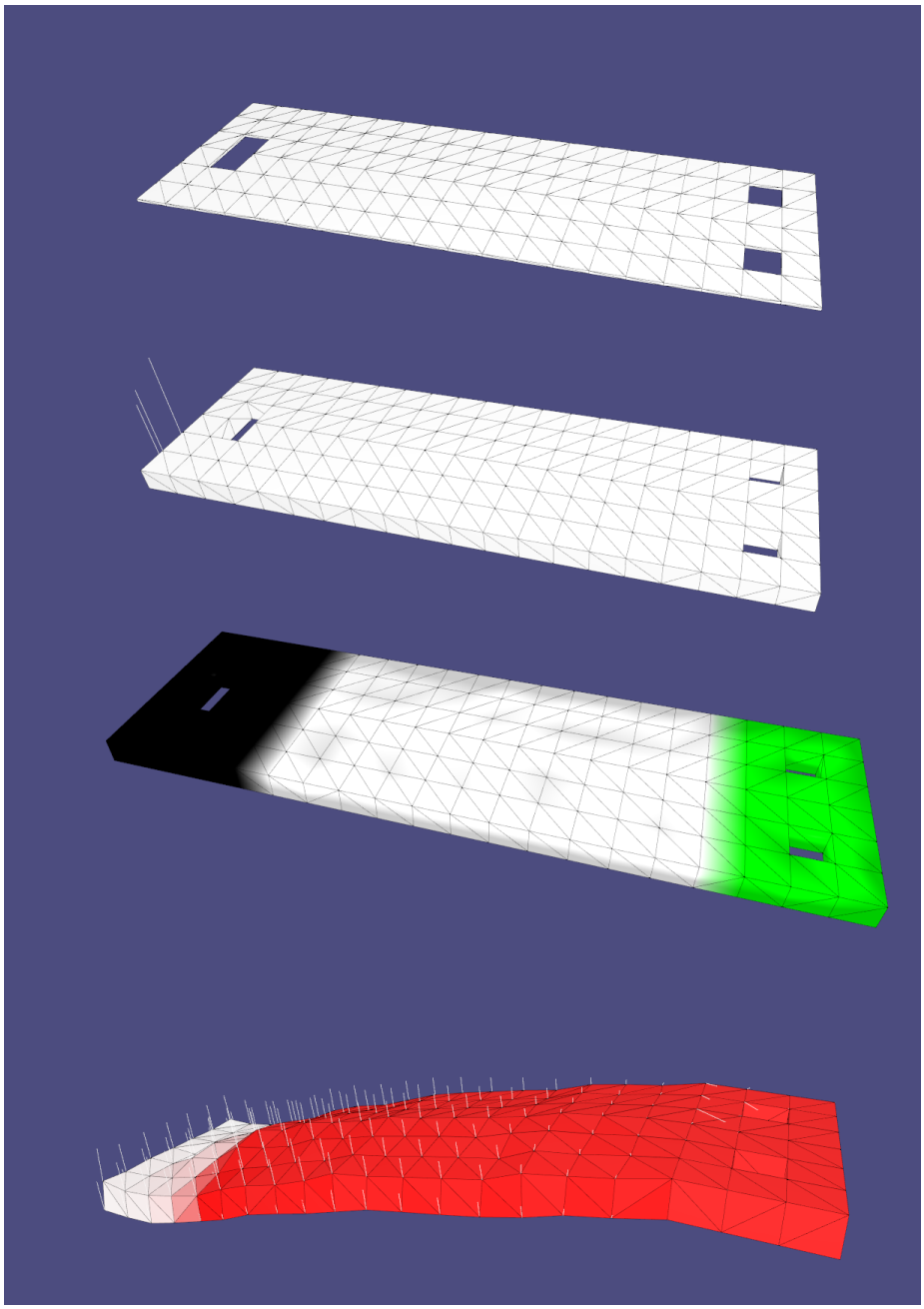

Figure 6: 1) Flat mesh. 2) Extruded mesh for better behaviour. 3) Selection of fixed vertices (black) and external force application (green). 4) Iteration result.