A New Method for Reverse Engineering XYZ coordinates into Computer-Aided Design (CAD) and Stereolithography (STL) File Formats

Nishant Aswani, Barkin Simsek

16 October 2018

1. Problem Identification and Statement

The proposed challenge involved the reconstruction of a solid 3D model based on a given set of XYZ coordinates. The XYZ coordinates provided represented the nozzle movements of a 3D printer that printed at a layer thickness of 0.2 mm. The reconstruction was required to result in a stereolitography (.STL) file.

2. Gathering Information and Object Visualization

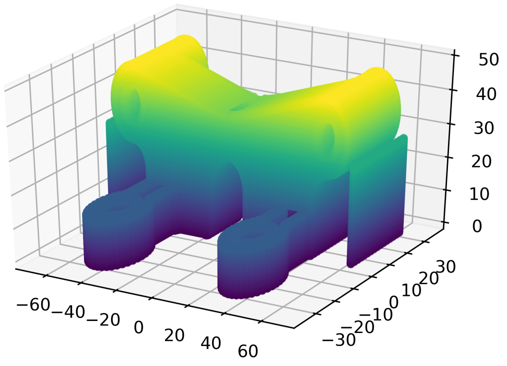

Figure 1: A scatter plot visualization of the XYZ coordinates

In order to visualize and comprehend the final structure, the numpy library and scatter plot function in Python were implemented to produce a 3D graph. The individual plotting of points provided a structure dense enough to produce the structure depicted in Figure 1.

Processing the original .txt file made it easier to work with in Python. A header row was added to the .txt file denoting each column with it's respective descriptor (i.e. "x coor" for x coordinates). The .txt file was then saved as a .csv file for use with the pandas library. The code used for the preliminary Python implementation can be founder under listing 1 in the Appendix.

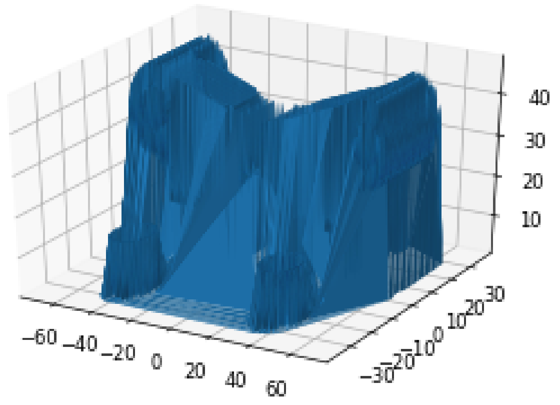

Figure 2: An attempted trisurface plot visualization of the XYZ coordinates

Having visualized the data, it was hypothesized that there was some Python library,which would transform a set of XYZ coordinates into a 3D mesh, allowing one to export the mesh as an STL file. A tutorial blog post(Hattem, 2015) described a Python library called "numpy-stl," capable of working with STL meshes. Looking at the documentation of numpy-stl (Hattem, 2018), it was discovered that STL generation, at least for the library in question, would require vertices and faces.

A numpy-stl issue ticket, found on the library's git page (jkokorian, 2016), inspired a new approach involving the use of triangulation to generate the 3D plot. This resulted in the trisurface plot depicted in Figure 2. However, due to the numerous unclean lines that were deforming the object, this approach was discarded.

Other attempts at using libraries to directly generate an STL mesh from point coordinates involved separately employing the surf2stl library in Mathematica and pygmsh library in Python; however, these approaches too were to no avail.

Further research, and a new drive to understand the process, elucidated what occurs when a STL file is generated. Figure 3 explained the notion that an STL file is essentially a high-resolution polygon mesh of the CAD model. Moreover, a Stack Overflow answer outlined how one would go about constructing triangles from a set of given coordinates. Essentially, given a coordinate $P(i, j)$, one could connect it to another coordinate $P(i + x, j)$, then connect that to $P(i + x, j + x)$ to form a clockwise triangle of three vectors. The next triangle, restricted to the clockwise orientation, could be formed by connecting the three points, $P(i, j)$, $P(i+x, j+x)$, and $P(i, j+x)$ with vector lines. Finally, the normal could be generated by using the cross product from two vectors that were part of a triangle.

3. Attempt at Using Meshlab

Meshlab's features provided a potential work-around to the issue of creating triangles from XYZ coordinates. The program was capable of computing normals for point sets, as well as supporting Poisson surface reconstruction to create an mesh file (Kazhdan, 2006), which could eventually be saved as an STL file.

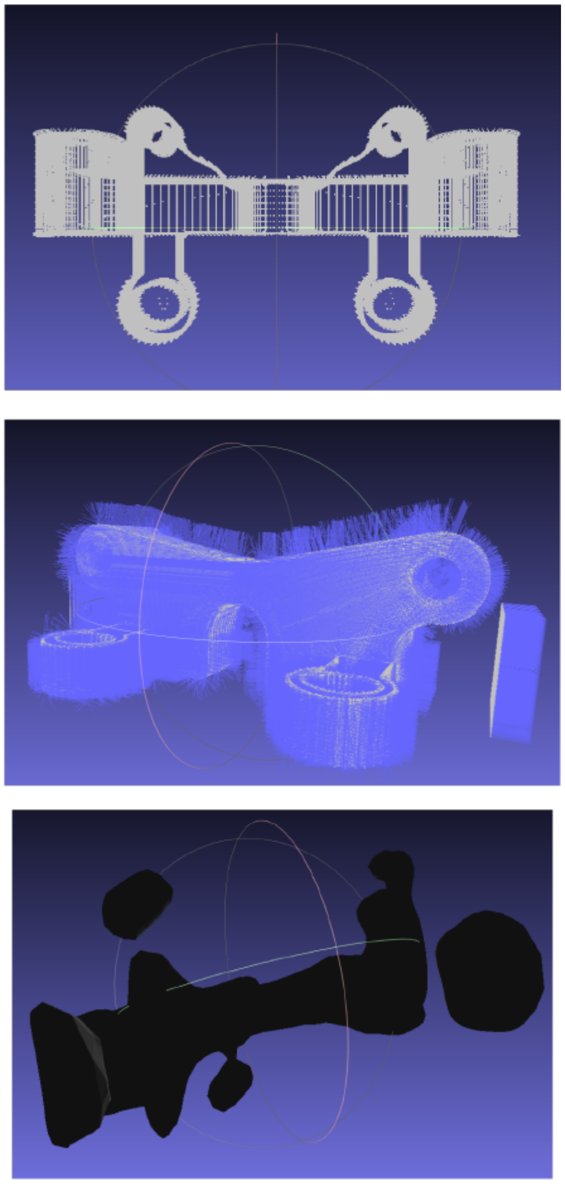

Figure 3: The top figure shows the imported point cloud. The middle figure displays the rendered normals. The final figure shows a variation of a a Poisson surface reconstruction.

Figure 4c shows one of the various attempts at the Poisson surface reconstruction, highlighting the abstract nature of and the difficulty that occurred in recreating the 3D structure in Meshlab. This process was concluded to be imprecise and not rigorous enough to generate a feasible STL file.

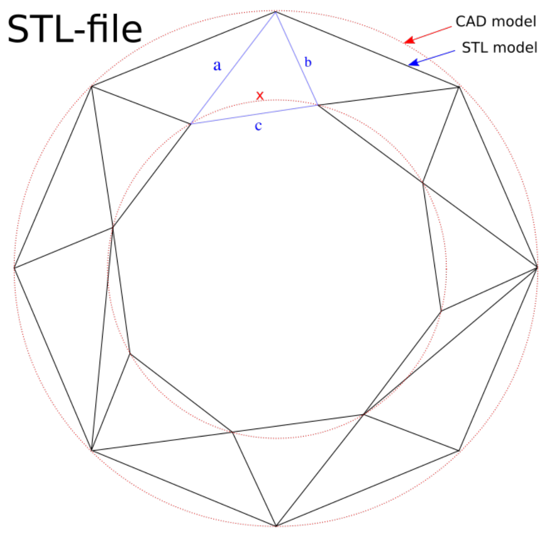

Figure 4: Two concentric circles, representing a CAD model of a doughnut shape, and a series of triangles approximating the doughnut, representing how STL modeling works.

4. Onshape

A final and successful attempt at recreating the 3D structure came about as a result of thinking through the lens of a slicer. Initially, openSCAD was used to draw polygons for each layer, extrude each layer, and shift the layers to their respective heights. Having automated this process in a few simple lines of code, it seemed the most efficient way of recreating the object. However, openSCAD does not allow for files to be imported; therefore, the coordinates had to be copied into the code, leading to a file that could not be reasonably executed.

Certain that a CAD software would work, the FeatureScript component offered by Onshape was discovered. The FeatureScript tool, which allows one to design CAD models through JavaScript and C syntax, was used to automate the process of STL generation. The actions of creating a sketch, drawing a poly-line through all of the points on a given layer, deleting unnecessary faces, and then extruding the sketch by 2 millimeters were all automated for the 234 layers.

The first attempt at using FeatureScript did not involve a mechanism to clean the unnecessary faces generated from the polyline drawing, thus a manual cleaning of the first layer was carried out.

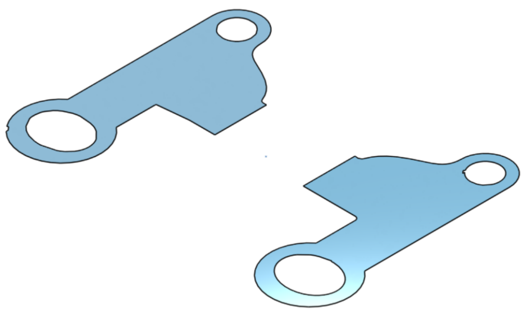

Figure 5: The first layer regenerated in Onshape and manually cleaned.

However, it was soon clear that manual cleaning and extrusion would not be feasible for a total of 234 layers.

Thus, future versions of the FeatureScript code were developed to querying all the faces in a layer, querying the largest of them in a for loop, and deleting them before carrying out extrusion. While this did not result in sketches as well cleaned as in Figure 5, it allowed for a basic, automated cleaning for 234 layers.

Onshape was unable to render and assemble all of the layers in one document for final touches and STL export. Thus, several layers were packaged and generated separately in Onshape (see Figure 6), all of which were exported and assembled in Autodesk Fusion 360. Figure 7 represents the final product.

The Onshape script can be found within the compressed folder and is not listed in the appendix.

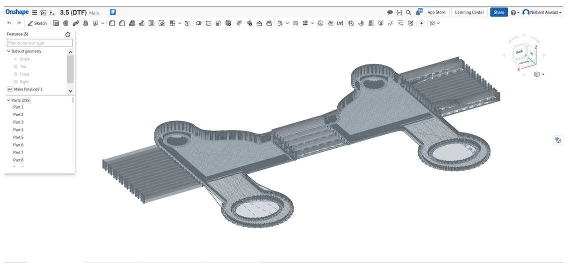

Figure 6: Package 5, depicting a selection of layers found in the first third of the structure

5. Acknowledgements

The Engineering Design Studio (EDS) for being our home, allowing us access to powerful resources, and providing a welcoming atmosphere.

Matthew Karau for his vision for engineering students and the EDS.

Daniel Chirita and H. Utku Unlu for checking the logic of our code and acting as motivators and guides.

References

Hattem, R. V. (2015). Rendering your stl files with matplotlib using numpy-stl. Retrieved from https://w.wol.ph/2015/07/10/rendering-stl-files-matplotlib-numpy-stl/

Hattem, R. V. (2018). Numpy-stl 2.7.0. Retrieved from https://pypi.org/project/numpy-stl/

jkokorian. (2016). Write matplotlib surfaceplot to stl file 19. Retrieved from https://github.com/WoLpH/numpy-stl/issues/19

Kazhdan, M. (2006). Poisson surface reconstruction.

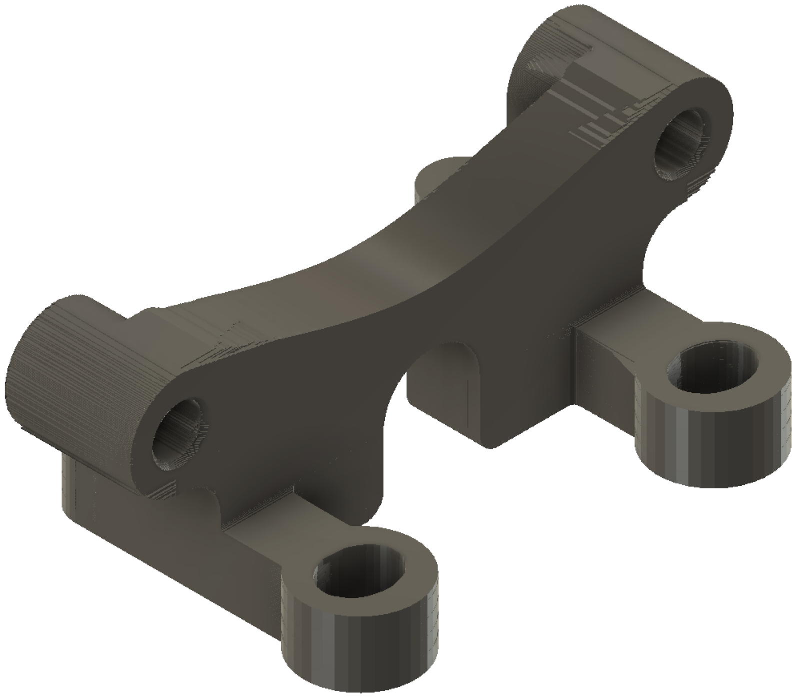

Figure 7: The final STL file

6. Appendix

Listing 1: Python Code for Visualization.

tiny numberstyle

```
from mpl_toolkits import mplot3d
import pandas as pd
import csv
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv('C:\\Users\\username\\Downloads\\xyz.txt',delimiter=\t)

a =df['x_coor'].values
x = a.tolist()

b =df['y_coor'].values
y = b.tolist()

c =df['z_coor'].values
z = c.tolist()

ax= plt.axes(projection='3d')
ax.scatter(x,y,z, c=z, cmap='virids', linewidth=0.5)
```