Prototyping a Custom 3D Printing Slicer

Derek Gokstorp

Implementation of a Prototype Modular Infill, Tool Path and Gcode Generator in Houdini.

Table of Contents

1. Abstract

3-D printing has ballooned from a niche area of research into an industry unto itself. It is now possible for anyone who is interested in the area to obtain (or even build) 3-D printers and print arbitrary objects. However most slicing software has relatively few infill options that are fairly limited. In this paper we introduce a novel prototype slicer that can be customized to include any infill method.

We introduce a modular prototype that can be extended with multiple infill algorithms. We implement an existing infill algorithm from Lu et al (2014), as well as a novel hybrid method by combining Lu et al.'s work with Wang et al (2013). We also add two novel infill patterns, as well as a common infill pattern used in commercial slicers to show the versatility of the system. We have also implement a slicer to produce G-code for an Ultimaker 2+, and show work that was done to extend this functionality to a Makerbot 2x printer to demonstrate that it is extensible to different G-code flavours as well. We then measure the strength of the printed objects compared to the amount of material used compared to each other for analysis.

2. Introduction

There has been a plethora of research into 3D printing techniques in recent years. From Kodama's (1980) seminal paper, the industry has burgeoned into an industry unto itself, as well as being used in many other industries (D'aveni, 2015), (Huang et al, 2013). It has also been seen as a harbinger of a new manufacturing paradigm dubbed the 'third industrial revolution' (Rifkin, 2016), (the Economist, 2014), (Hull, 2015), and a new disruptive technology that will change the manufacturing landscape as we know it (D'aveni, 2015), (Hyman, 2011).
There are several opportunities to reduce the amount of material used in the printing process and also to optimize the time spent actually printing the object. Geometric considerations can not only reduce the cost/amount of material but may also be used to assess the strength and stability of an object before even printing it.

As Monzón et al. demonstrate [2016], modern 3D printing, or more appropriately additive manufacturing, consists of several different types of printing processes:

A. Vat Photo-polymerization (e.g., stereo lithography, SLA)
B. Material jetting (e.g. Poly-jet)
C. Binder Jetting (e.g., some 3D printers by powder and binder)
D. Material extrusion (e.g., fused deposit modeling, FDM)
E. Powder bed fusion (e.g. selective laser sintering, SLS)
F. Sheet lamination (e.g., Sheet Forming)
G. Directed energy deposition(e.g., laser cladding)

For the purposes of this paper, material extrusion or fused deposit modeling (FDM) with a single extruder is the type of additive manufacturing that we will be focusing on (however we will demonstrate that our method can also be extended to multiple extruders).

3. Problem Statement

Although there have been some recent advances in the area of creating cost effective 3D shapes while improving cost and maintaining strength in the past couple of years, there is still much progress to be made.

Looking at the available literature it should be possible to analyze the different techniques that have already been published and see if there is any benefit of combining techniques in novel ways to create even lighter, stronger 3D objects. Barring that possibility it would still be useful to see how combinations of techniques or comparing techniques can be beneficial depending on how biased toward minimizing material vs. maximizing strength of the object.

The main goals of this project are to:

I. Research and implement a variation of infill methods and/or combine existing methods in novel ways (implemented a modular way to allow extension or inclusion of future algorithms).

II. Simulate the strength of the object when applied forces to see the simulated strength.

III. Develop a slicer to create G-code for the print cycle and convert to the printers native instruction set (in a modular way to allow for different types of printers).

IV. Print and test the models and methods with most promise and see how the actual artifacts hold up to physical stress compared to simulated, and other authors results.

4. Related Work

3-D Printing has been a popular topic of research in several fields recently, including in the computer graphics community. Particularly research involving fused material deposited printing due to it accessibility to the public with the advent of affordable commercial printers and the maker community.

We are mostly interested research into geometric analysis of the shapes to be printed and implementation of G-code generation in a modular way that would allow for high levels of customization. Lu et al (2014) provide an approach to creating strong yet lightweight models via a novel infill algorithm and served as a foundation for the infill portion of this project. We also utilize work from Wang et al (2013) and Guerrero-de-Mier et al (2015).

Lu et al (2014) propose a novel hollowing algorithm based on the concept of honeycomb like cell structures. Honeycomb structures have proven to provide a high strength to weight ratios via structural tension (Wilson, 1990). Their work is inspired by several earlier works, notably:

Kou et al (2010), present an approach to design irregular artifacts with controllable pore shapes and distributions. They indicate an inspiration from a colloid-aggregation model, and utilize random voronoi cell merging to obtain control points for closed B-Spline curves to carve interiors of the voronoi cells.

Le et al (2014) also reference (Wilson, 1990) work with honeycomb structures. Wilson's work focuses on strong yet light honeycomb structures for aerospace applications. Honeycomb structures are inherently material efficient and can be found in beehives, rock formations, tripe and bone. Lu et al (2014), also point out that "In its closed-packed form, the hexagonal-patterned lattice is the steady-state configuration of all Voronoi diagrams".

Lu et al (2014) also reference Wang et al. (2013) who introduced their approach to cost effective 3-D printing by filling the interior edges with a skin truss scaffolding. Initially we dismissed Wang et al. as their approach focused on minimizing the truss structure while maintaining and accounting for certain physical and geometric constraints, whereas we were more focused on the same global properties of Lu et al.'s focus of strength to weight. However, as our research progressed we discovered that There were some very important similarities to these two approaches that could be combined to produce a novel infill algorithm.

Guerrero-de-Mier et al. focus on minimizing thin film distortion by controlling the print path and 'blocking' the print path to minimize heat diffusion build up. While reading their work we came across other papers where print path control was central to achieving varied results (Brooks et al., 2011), (Chakraborty et al., 2008), (Jin et al, 2011).

Brown et al. (2013), provided an excellent breakdown of creating a tool path and converting that tool path into G-Code.

5. Technical Background

In order to better understand the impetus and background of some of the reason behind our design decisions, we would like to take the opportunity to briefly discuss three of the main concepts of our project.

5.1 Tool Path Generation

Defining a tool path is a relatively straight forward process and is historically derived from machine tooling. It is basically defining a path relative to the material that it is operating on and is defined by mechanical methods (rather than hand held). One of the earliest examples of this is a path cutting lathe designed for screw cutting, dating to the 1480's (Moore, 1970).

Luckily, for 3-D printing the tool path is simple in that it is additive and cumulative with paths being traced layer by layer, and the material is being deposited as the print head moves. Below is an example of a very simple tool path to print an open square shape.

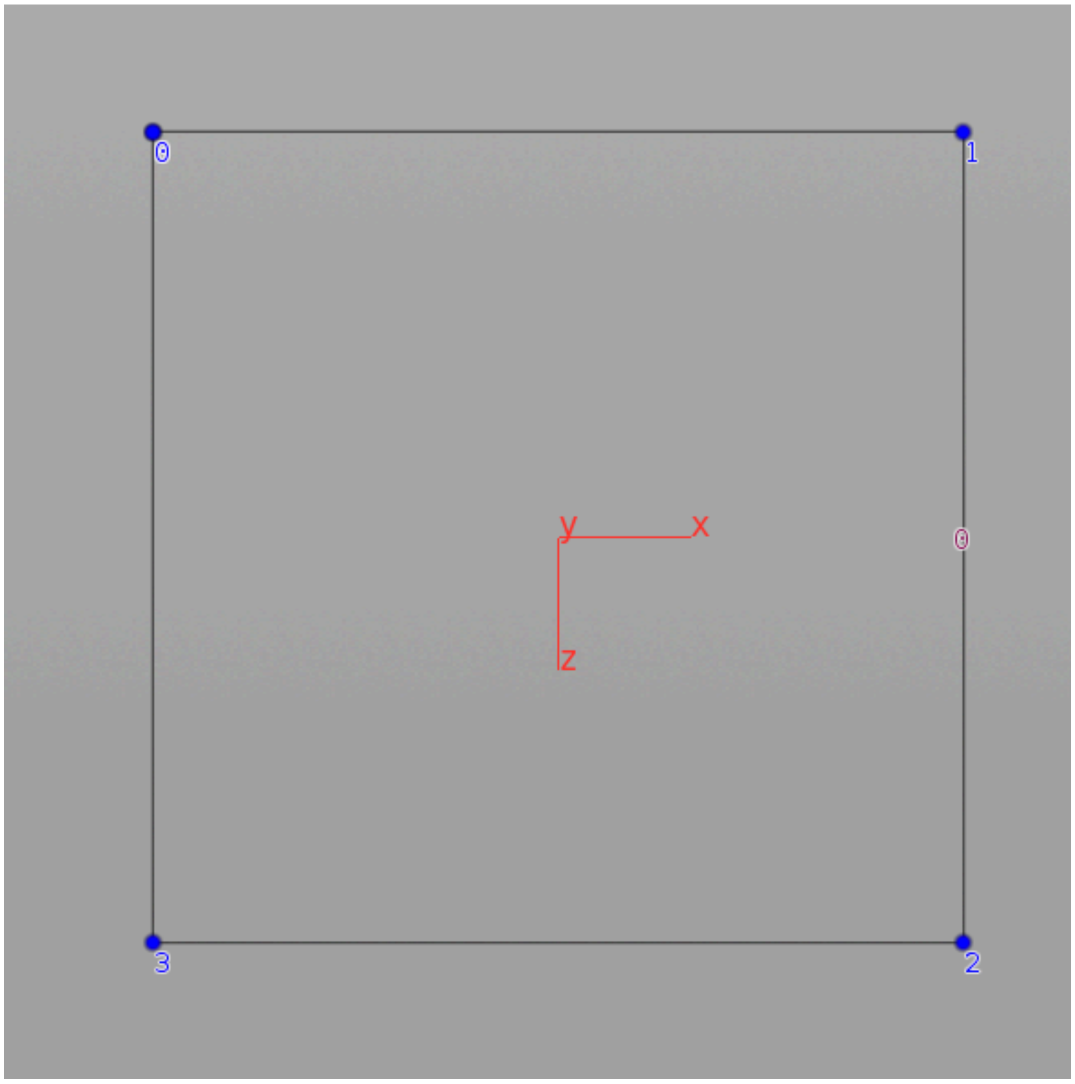

Figure 5.1.1: *The blue dots indicate points of the tool path (the blue number indicate point number, in this diagram we have p0, p1, p2 and p3). The four points constitute four edges that are all part of primitive 0 (the red number indicates primitive number). The tool path in this case is p0 to p1 to p2 to p3 to p0.*

The tool path indicated in figure 5.1.1 could be considered a boundary of a solid square – or the outer edge. In order to fill the solid square we would also need to have an infill tool path as well as the boundary tool path. Figure 5.1.2 includes the boundary and infill tool paths.

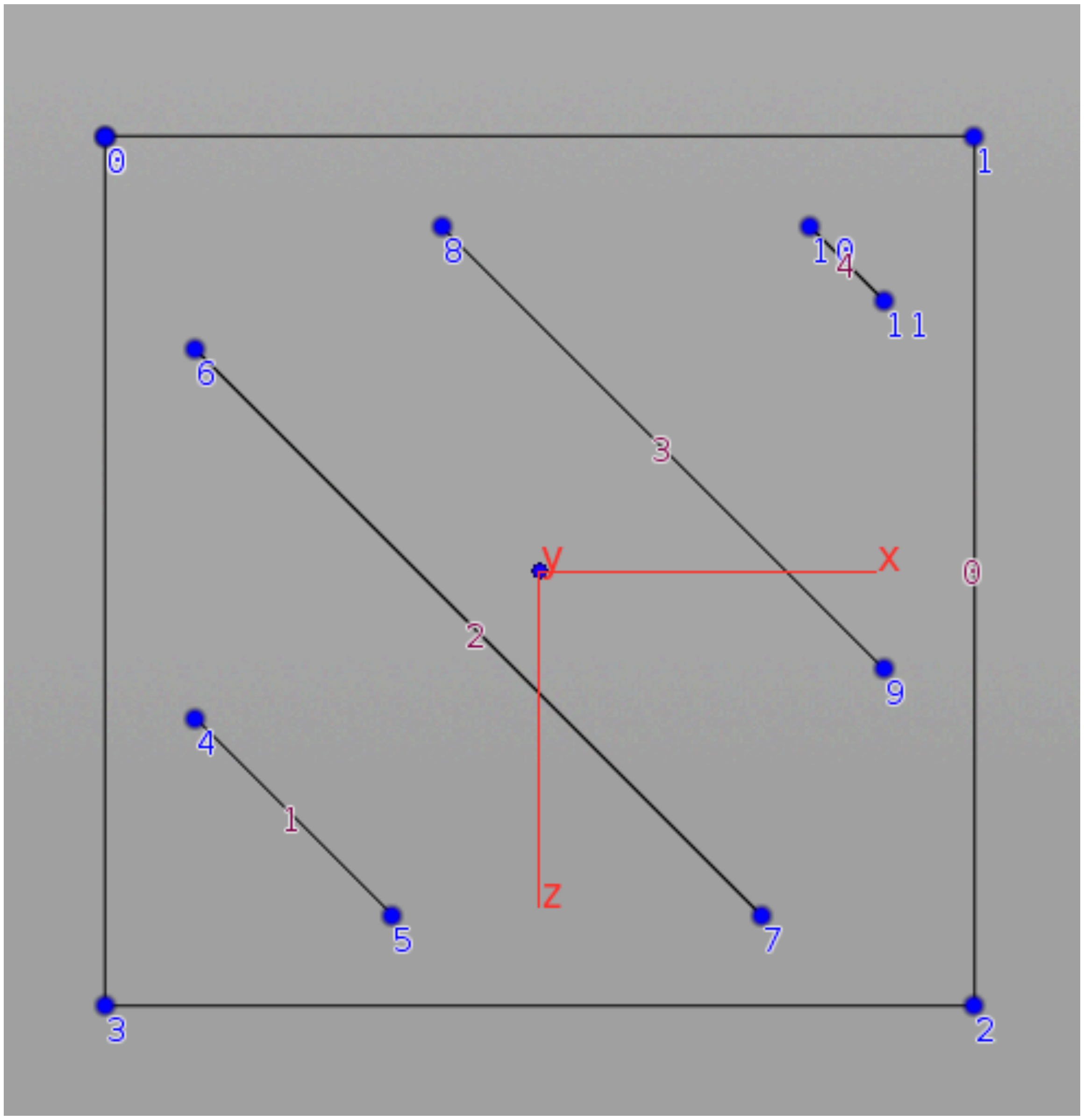

Figure 5.1.2: *The blue dots again represent points and the blue numbers the corresponding point numbers. This time we also have multiple primitives, represented by the corresponding red numbers. In this case the tool path is slightly more complicated, where there is a change in primitives there is an implied move however **without** a print, here we indicate it with parenthesis around the non-print moves. In the above diagram the tool path would be p0 to p1 to p2 to p3 (to p4) to p5 (to p6) to p7 (to p8) to p9 (to p10) to p11.*

Just as in figure 5.1.2, if we had another layer to print above this one the next layer would continue with an implied move from p11 (to p12) and then continue as this layer printed with the points continuing on to p13 to p14 … etc.

So when we discuss slicing or creating a tool path for 3-D printing, we are generally concerned with a geometric representation of the path that the print head will take with several implied assumptions: a move without a print from primitive to primitive as well as another move without a print from one layer to the next. This geometric representation can then be traversed by iterating over all layers, iterating over all primitives in each layer and iterating over all edges (defined by the connected points) in each primitive. Below is a pseudo-code representation of the process:

layers = l_0 - l_n
primitives (per layer) = pr_0 - pr_n
edges (per primitive) = e_0 - e_n
points (per edge) = $p_0 - p_1$

```
for layer ( $l_i$ ) in layers ( $l_{0-n}$ ):
   for primitive ( $pr_i$ ) in primitives in layer ( $pr_{0-n}$ ):
      for edge ( $e_i$ ) in edges in primitive ( $e_{0-n}$ ):
         for point ( $p_i$ ) points in edge ( $e_i$ ):
            if $e_0$ and $p_0$ :
               move to $p_0$ without printing
            else if $p_1$ :
               print $p_1$
```

5.2 G-code

G-code is the instruction set of most servo-mechanisms and stepping motors used for machining. However, because it is not a highly standardized language and each manufacturer may use it's own unique 'flavour' of G-code it is essential to know the available commands for the flavour that the manufacturer uses in it's instruction set.

G-code for 3-D Printing is a fairly simple instruction set for most commercial printers and there are only several commands that are essential to know:

';' - At the beginning of a line is a comment that is not processed by the machine – but is for readability for people.

'M*' - M codes are miscellaneous – in the case of the Ultimaker 2+ series the code 'M107' is for controlling the fan to cool the printer to turn on and 'M106' to turn the fan off.

'G1' - Indicates a print move, that is to say a coordinated X Y Z and E movement.

‘G0’ - Indicates a non-print move.
‘G11’ - indicates a material feed retraction.

‘F*’ - F-codes indicate the speed of the move (with the * being substituted with the speed).

‘E*’ - E-codes represent the speed of extrusion of material (usually used in conjunction with G1 commands and with the * being replaced with the total amount of material extruded since the beginning of the print – that is it is a cumulative value).

5.3 Digital Assets and Operator Type Libraries

In our implementation, we utilize Side Effects Software’s Houdini for reasons outlined below. ‘Digital assets’ and ‘operator type libraries’ are a common terms that we will be using throughout the Method Overview.

Digital assets in the context of Houdini are a collection of one or more built in nodes in Houdini and/or other digital assets that are wrapped up into a single node with parameters that are expected to be manipulated by the user promoted to the top level. This can be also be called a custom ‘operator type’. Another way of thinking of digital assets is that it is an encapsulation of a very particular and usually reusable workflow – with well defined inputs in terms of geometry as well as parameterization of that workflow and well defined outputs.

Operator type libraries (or OTL’s as they are commonly referred) are files on disk that contain the definition of one or more digital asset. Having OTL’s in Houdini’s OTL scan path means that you are able to use the operator types (or digital assets) in that session of Houdini.

All created digital assets for this thesis have been bundled in the OTL ‘custom_slicer_tools.otl’ and will need to be installed to be utilized using the ‘Assets > Install Asset Library’ menu in Houdini.

These two concepts are well defined and highly utilized in the visual effects industry as a method of tool creation and deployment as well as collaborative tool building.

6. Method Overview

Implementation in Side Effects Software’s Houdini was a natural choice for several key main reasons, including: it is highly extendible, it has a very open architecture, it has an excellent volume toolset, it has a built in dynamics toolset with a finite element solver for stress analysis and has a highly procedural intuitive node based workflow. However further to this it is a fairly mature software, and has been around for the past 25 years. Due to the length of time in the industry and the number of users and the nature of its open and highly extendable nature – there has become reusable techniques and workflows in visual effects which very similar and are applicable to solving problems mentioned above.

One thing that we were very interested in solving while implementing Lu et al (2014), was the ability to be able to generate G-code commands in a custom way for whoever was using the prototype. This was a key aspect of the work we were interested in achieving as if we were to simply create a hollowed object and feed it into a commercial slicer – the commercial slicer would determine the print path and the infill pattern for the solid areas. Whereas we wanted the user to be free to determine the tool path (or print path) and have the printer respect that print path decision. Our inspiration for wanting to be able to control the print path was Guerrero-de-Mier et al. (2015) and their work with 'blocking' or isolating areas per level.

6.1 Generating a Custom Tool Path Generator in Houdini

Our first step in being able to prototype a modular slicer in Houdini was the ability to generate a geometric tool path. We work from the assumption that we will have manifold geometry (see section 6.3 for more information).

We then created a digital asset with the type name 'tool_path' and the label 'Tool Path', with a minimal set of two promoted parameters. The parameter named 'view' and labeled 'View Object/Slice' allows you to choose to view just a single slice of the tool path or to view the entire object's tool path (this parameter defaults to a single slice as the slicing may take several minutes depending on the size of the object). The digital asset also consists of two input geometries, first input is the original geometry (or the outer shell) and the second input consists of the hollowed internal areas (see infill implementations in section 6.3). We assume lengths of measurements such that 1 Houdini unit = 1cm.

Internally the digital asset is comprised of several sections. The original geometry is bounded and then used to generate vertical line from the 'Y' minimum value the the 'Y' maximum value. This vertical line is then divided by resampling the line into sections that are 1 mm long (0.1 Houdini units) as this is the height of the print layer for the Ultimaker 2x using a 0.4mm extrusion nozzle. The print layer height was determined by using Ultimaker's Cura open source slicer to slice a simple box and reverse engineering the resultant G-code. We will refer to this line as the 'vertical layer line' as it defines the height of each layer in the resultant tool path.

We also create a vertical line of the maximum bounds of the X axis, and then resample the line into segments that are 0.349 mm apart. Even though the extrusion nozzle is 0.4mm – we want a little bit of overlap on either side to bind to previous printed segments, and the print bead is slightly smaller than 0.4 mm due to the melted polylactic acid (the printing material being used – this is a biodegradable thermoplastic aliphatic polyester derived mainly from corn and is a usual printing material) being viscously elastic. We will refer to this line as the 'horizontal infill line' as we will use this to generate infill later on.

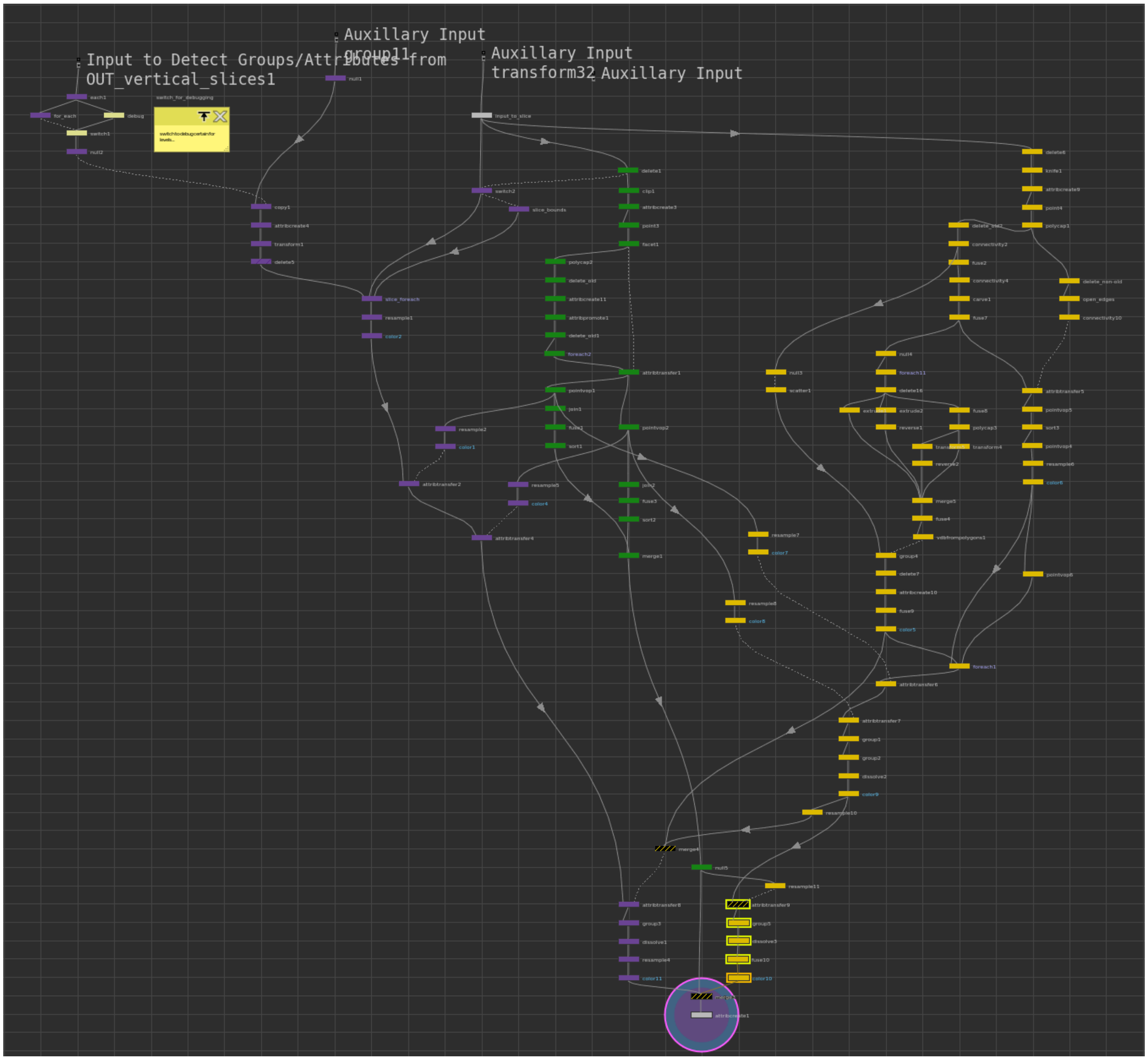

Figure 6.1.1.1: *The first outer for loop for generating a tool path per layer. The nodes in purple generate the infill, the green nodes generate the outer edges and the yellow nodes generate the interior hollow edges.*

We then merge the original geometry (or the outer shell) and the hollow interior geometry together into a single geometry using a merge node. These three geometries are the fed into a first level for each loop. This first loop is to iterate over every point on the vertical layer line.

Inside the first for each loop we have three main sections: A section to generate the infill, a section to generate boundary exterior edges and a section to create interior hollow boundary edges (see figure 6.1.1.1). This network proved to be the most complex in terms of the number of nodes as well as the complexity of the computations involved.

6.1.1 Tool Path Infill

For the first section the current point from the vertical layer line is the only one that we have access to from within the outer for each (point vl_i). Points from the horizontal infill line are then copied onto the point vl_i and then enter a secondary for each loop. In the secondary for each loop we again iterate over each point from the horizontal infill line that has been instanced onto vl_i (point hl_j). This then enters a tertiary for each loop where the point hl_j is rayed onto the combined geometry (CG) i.e. hl_j -> CGhl_{j1} and then rayed again i.e. CGhl_{j1} -> CGhl_{j2} a total of 10 times. We found that 10 intersections were usually enough for the geometry that we were testing, and that higher numbers would slow down calculations.

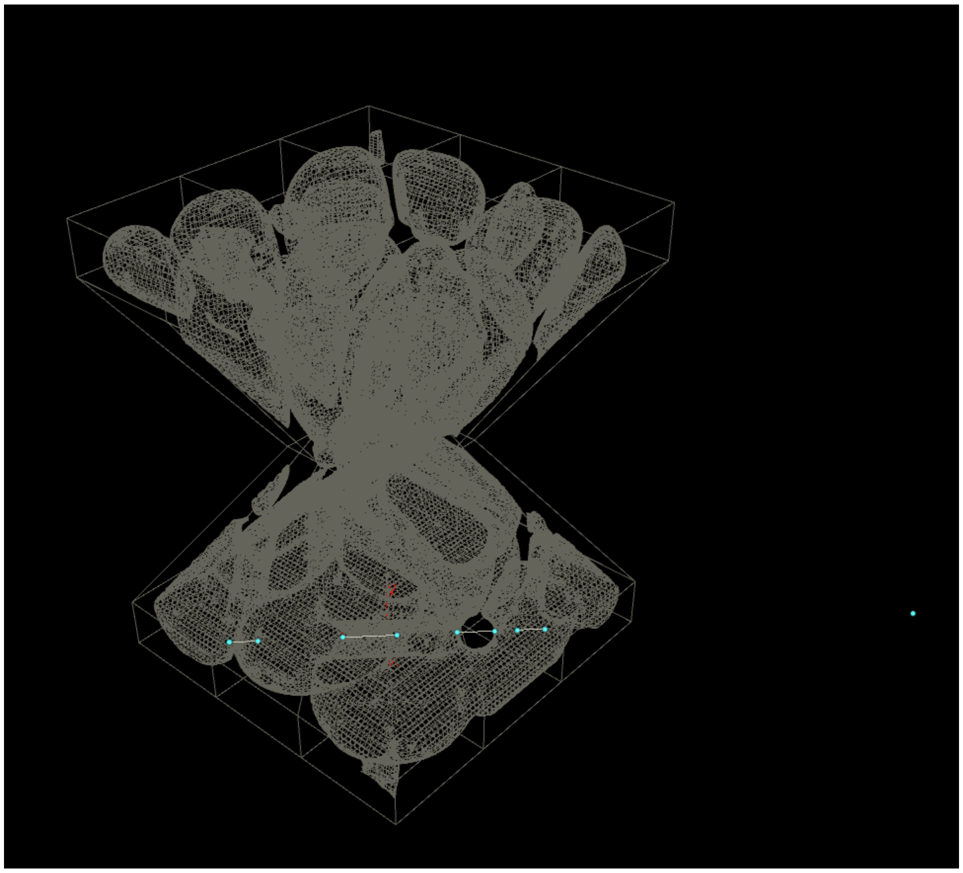

Figure 6.1.1.2: *The point hl_j can be seen on the far right hand side. Then from right to left inside the bounds of the model we have* CGhl_{j0}, CGhl_{j1}, CGhl_{j2}, CGhl_{j3}, CGhl_{j4}, CGhl_{j5}, CGhl_{j6} *and* CGhl_{j7}. *With edges created between the even and odd paired sequential numbers.*

After exiting the tertiary for loop, we would then create an edge between every even then odd sequential point. Because we are dealing with manifold geometry it is safe to assume that because our point to by rayed onto the surface (*hl_j*) is outside the geometry (CG) the first hit (*hl_{j1}*) and the second hit (*hl_{j2}*) will represent the point at which the ray first enters the geometry and the point at which the ray leaves the geometry (either a hollow area or leaving the geometry completely). Therefore we create edges on each pair of consecutive even and odd points. If there are less than 10 intersection points, the remaining points will remain on the last point exiting the solid geometry – we fuse these points into a single point as creating an edge on points in the same location is unnecessary (see figure 6.1.1.2).

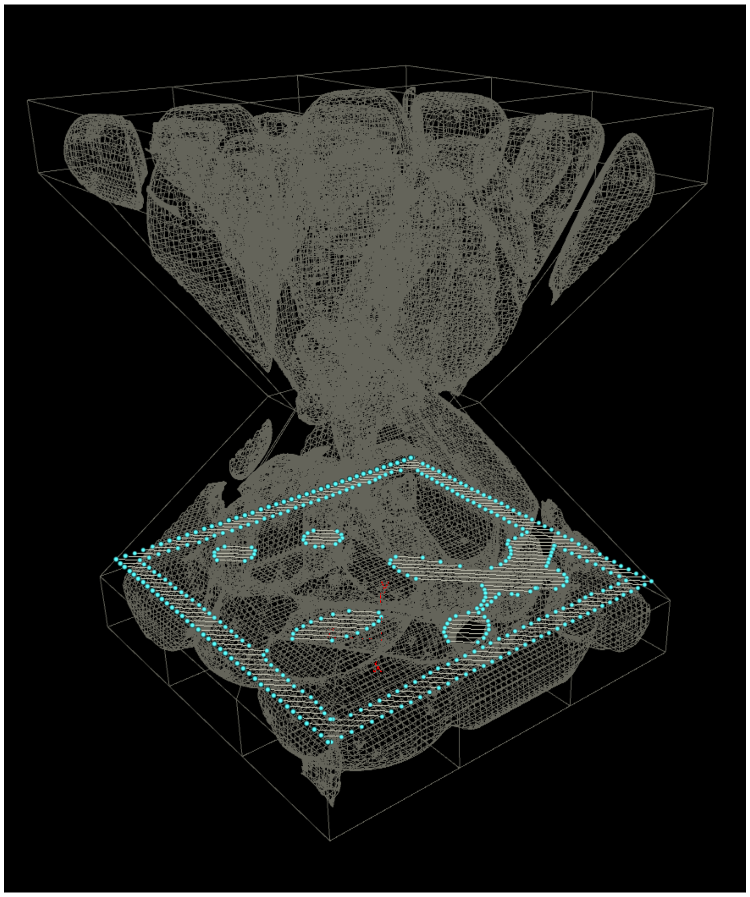

Figure 6.1.1.3: *Here we have an entire layer of infill after exiting the secondary for loop.*

Additionally after exiting the secondary for loop we have an entire layer of infill (figure 6.1.1.3). This will get further processed later on in the process.

6.1.2 Tool Path Exterior Boundary Shell

In the second section of the primary for loop we create a boundary edge of the outer surface by bi-secting the surface of the outer shell at the height of the current layer in the for loop (the Y value of the point vl_i). We then peak the geometry inward (the negated value of the surface X and Z normal), by half the distance of the print bead width value (1/2*0.349 mm) and then peak that in tern inwards once more by a full print bead width value to create a two bead thick wall throughout the X and Z surfaces (figure 6.1.2.1).

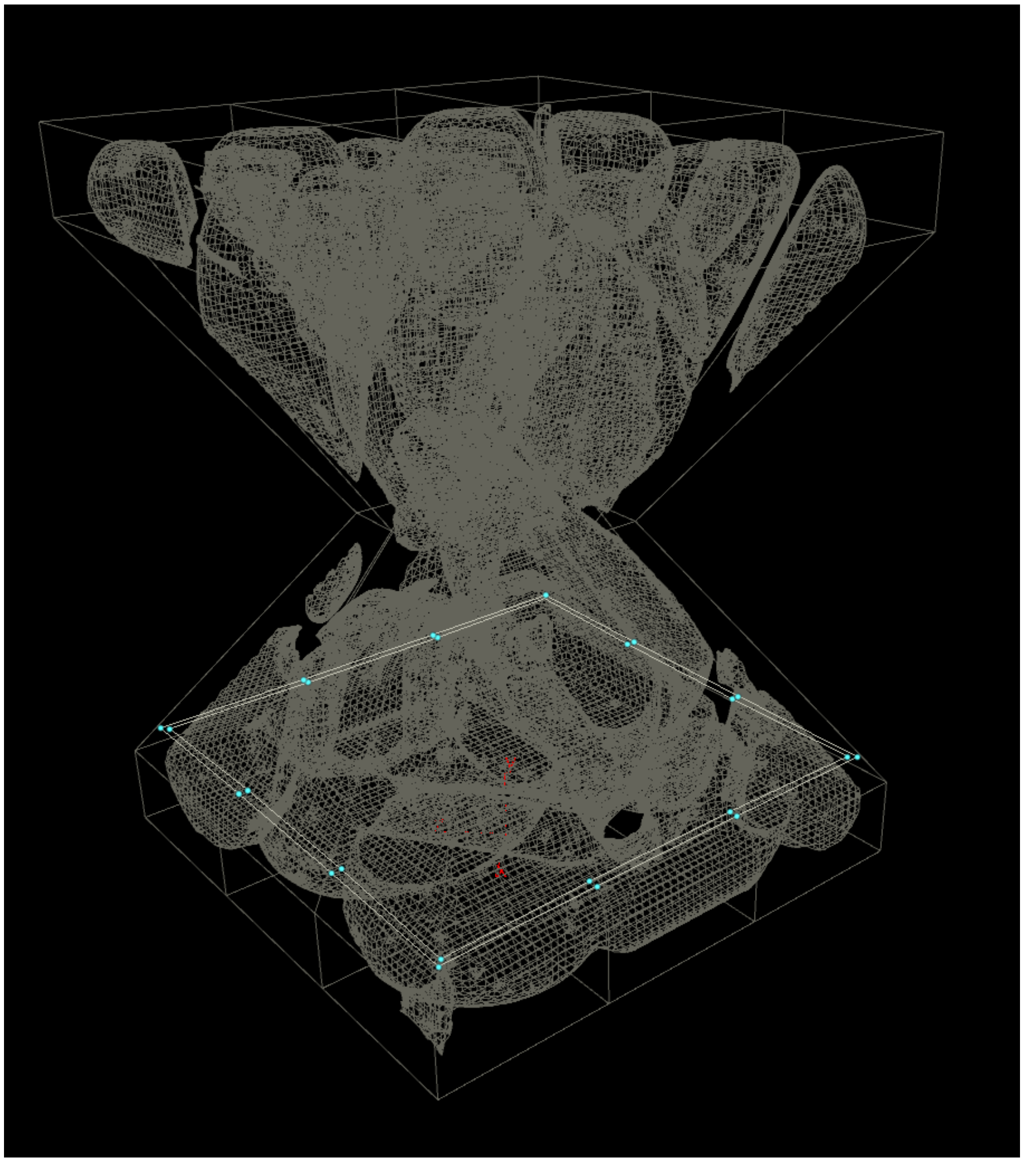
Figure 6.1.2.1: *Tool path exterior boundary shell.*

6.1.3 Tool Path Interior Boundary Shell

The third section of the primary for loop has to do with generating an interior boundary shell. As with the exterior boundary shell we also bi-sect the interior geometry at the height of the current layer height (the Y value of point vl_i). and then peaking this only once (we would only like a single layer of boundary shell on the interior as the infill will fill in the rest – figure 6.1.3.1).

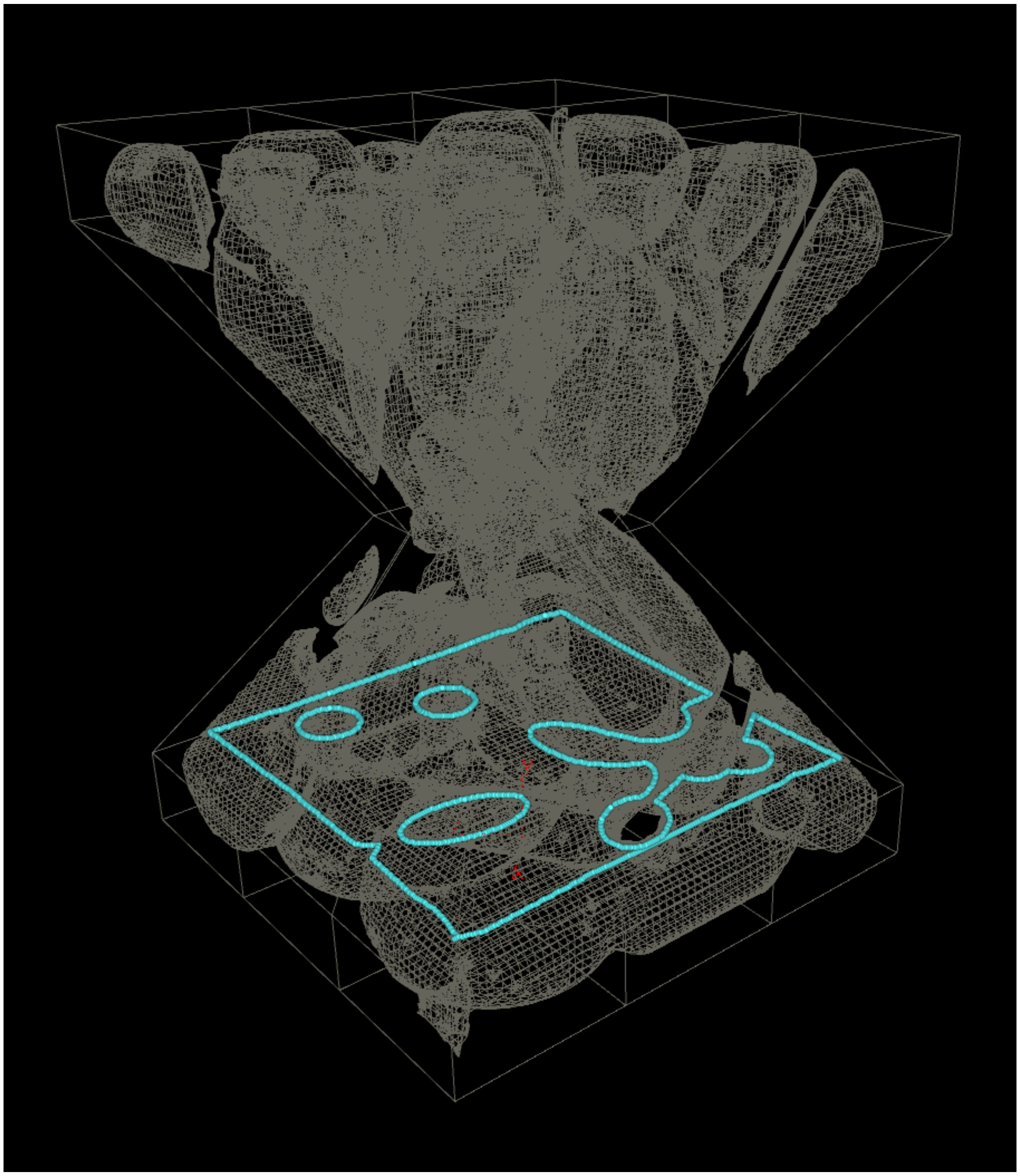
Figure 6.1.2.1: *Tool path interior boundary shell.*

6.1.4 Completing the Layer Tool Path

After generating the infill, outer and the inner boundary shells, we still need to complete some processing of these paths as there will be multiple overlaps . This would result in material being deposited in the same place by the printer multiple times and would not result in an effective print (see figures 6.1.4.1 and 6.1.4.2).

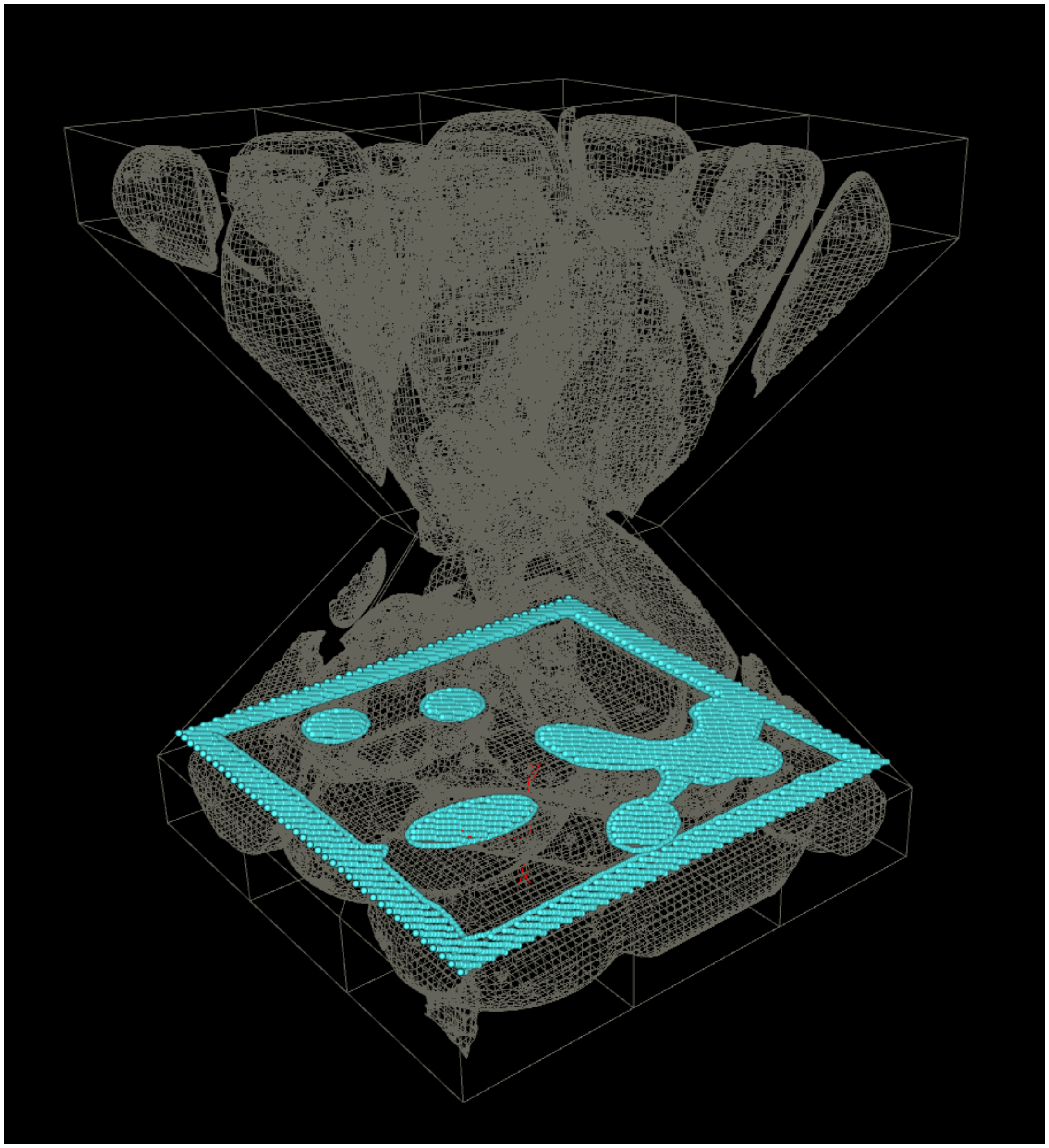

Figure 6.1.4.1: *Level combined tool path.*

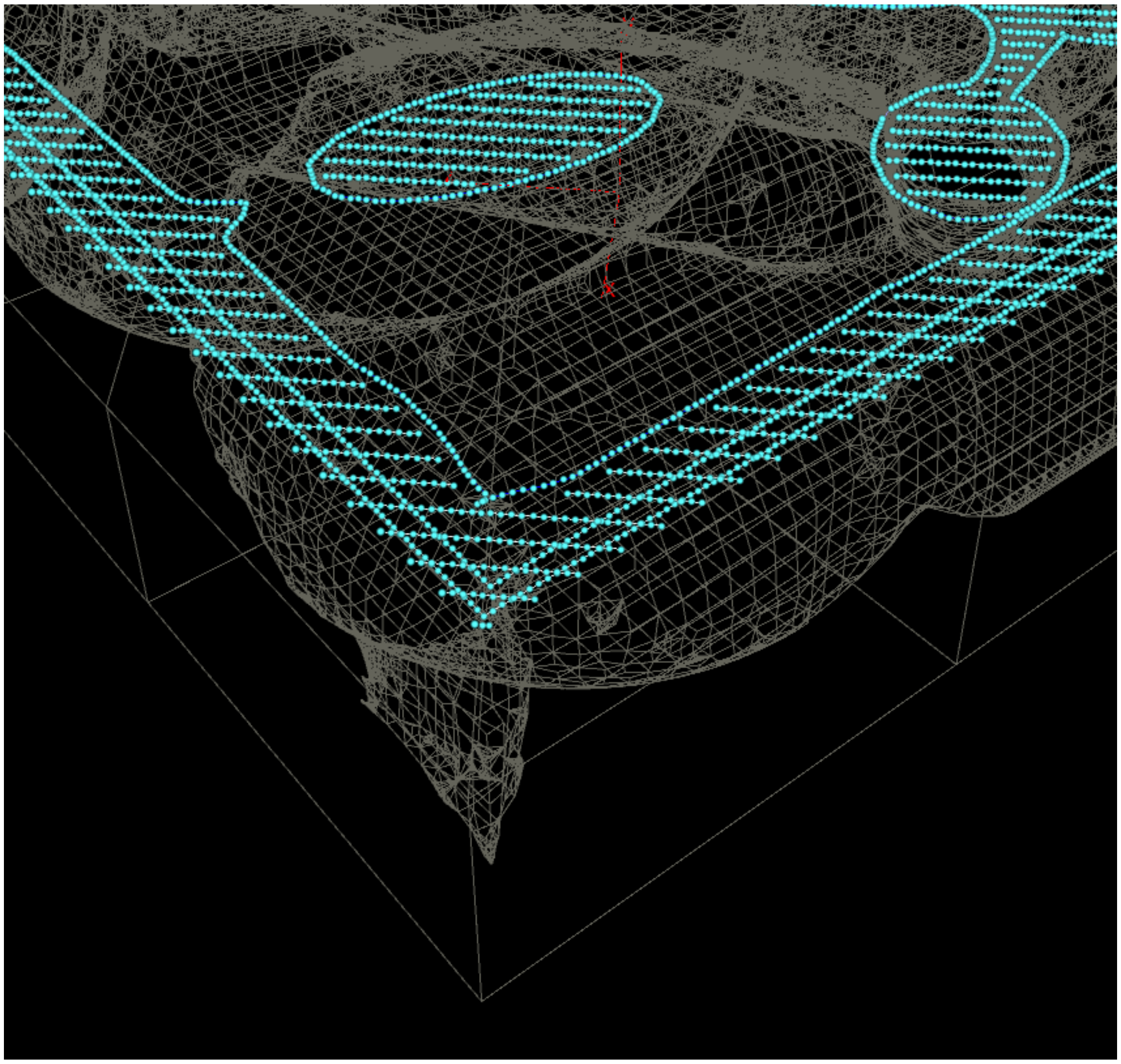
Figure 6.1.4.2: *Level combined tool path close-up.*

In order to ensure only a single print in a single location we define an order of precedence: outer boundary shell takes precedence over both inner boundary shell and infill paths. In order to obtain this we delete any points that come within half the distance of the print bead width value (1/2*0.349 mm). Inner boundary shells also take precedence over infill paths and we also delete any points that come within half the distance of a print bead with (see figure 6.1.4.3 and 6.1.4.4).

Finally because we are printing by edge we want to try to reduce the number of edges that we print. We reduce the number of points on the outside boundary by reverting to the original number of points. On the interior we reduce the number of edges by fusing points within a radius of 0.15 mm (see figure 6.1.4.5). This could be reduced even more aggressively by analysing the angle of deviation between one connected edge and the next (since they are all co-planar). However, this was beyond the scope of the current project, as we were simply looking to prototype the process to see if it was feasible – not to optimize the process.

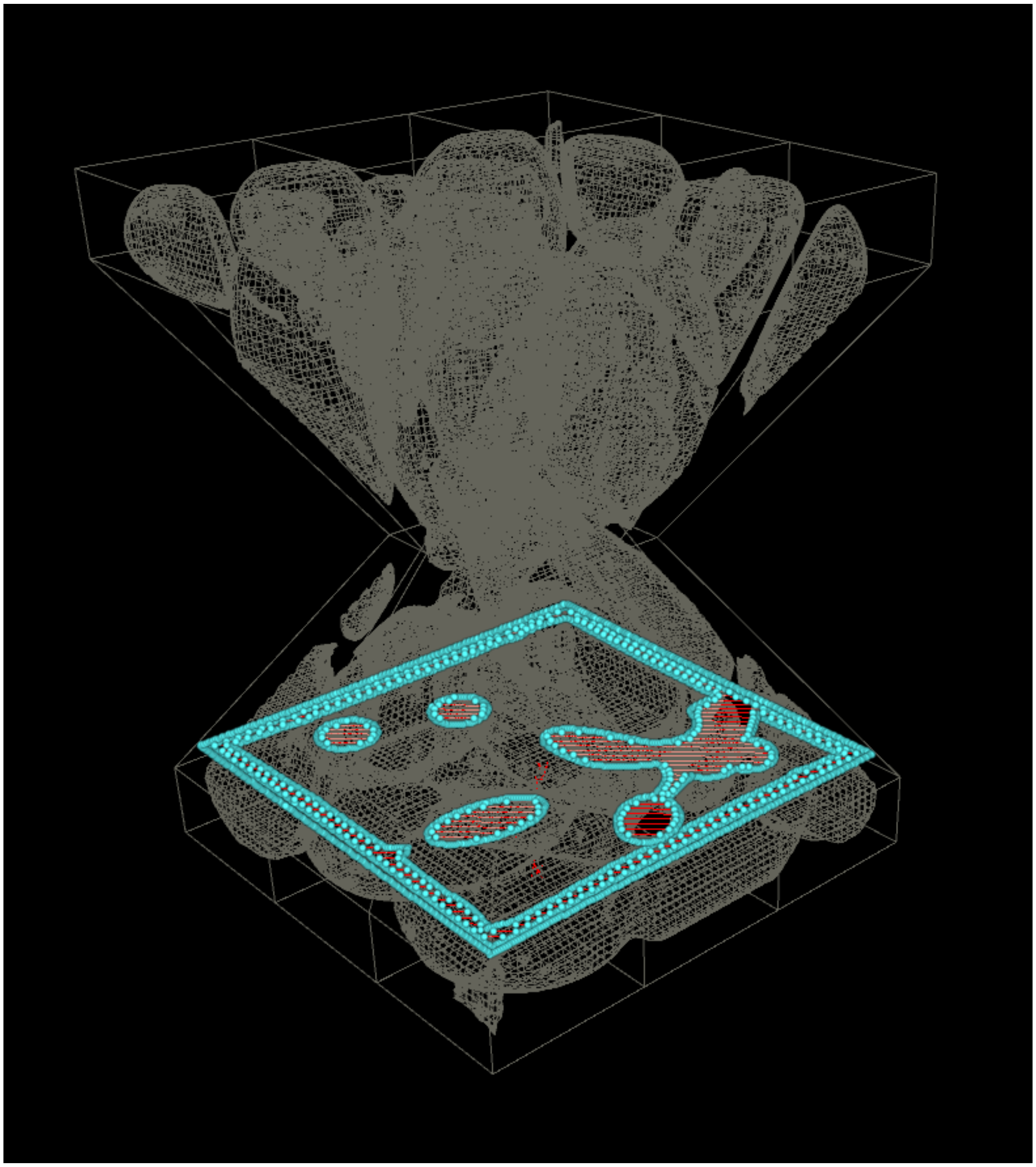

Figure 6.1.4.3: *Level tool path post processing for overlaps.*

After exiting the primary for loop, as long as the parameter named 'view' and labeled 'View Object/Slice' is set to object you should have the entire tool path for the object (see figure 6.1.4.6 for a portion of this).

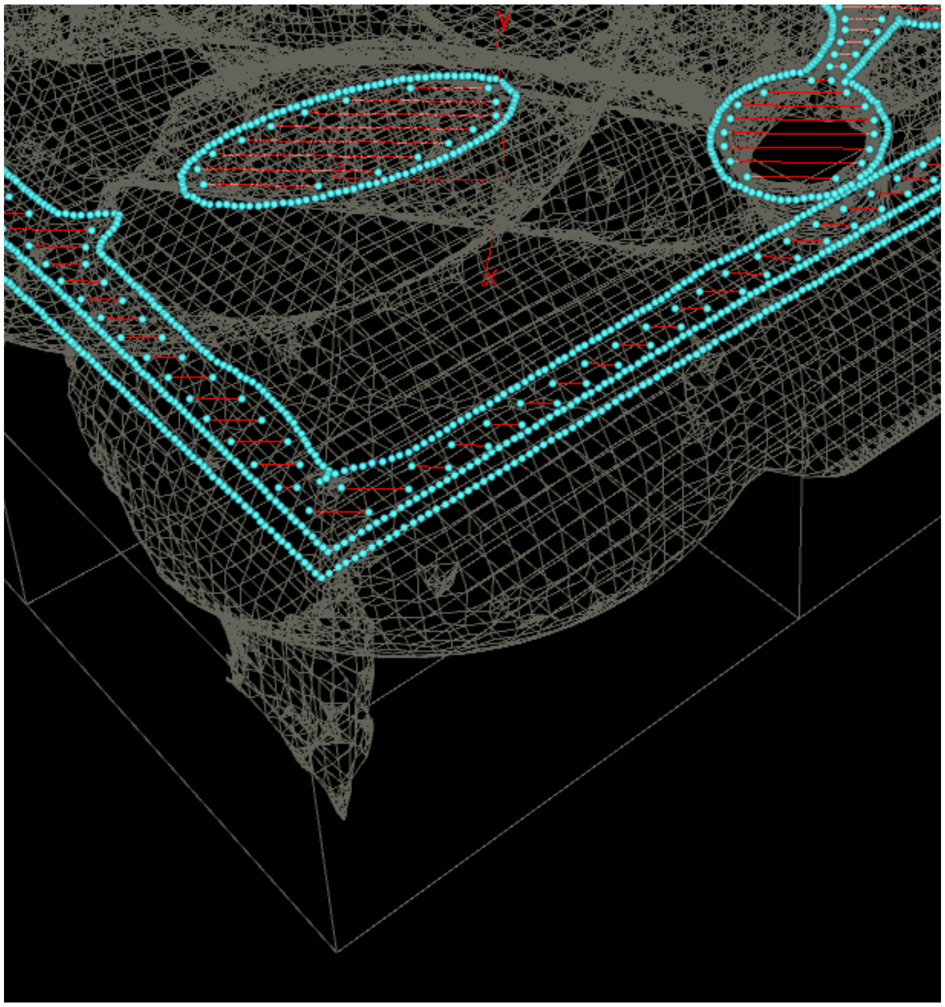
Figure 6.1.4.4: *Level tool path post processing for overlaps close-up.*

6.2 G-Code Generator

The G-code generation digital asset is partially comprised of nodes and also partially a python script which traverses the tool path and prints the subsequent G-code to a file. Initially this was written as a G-code generator for a Makerbot 2x (as this was the intended destination printer). It was also shared with David Correa and Abel Groenwolt, PhD candidates at the University of Stuttgart in the Institute for Computational Design. They then modified the original code to work with a Makerbot 2x with dual extruders.

The script was then re-written to work with an Ultimaker 2+ extended. While writing the code and testing we found an optimization by being able to swap the point order for every other primitive we could cut the print time down by almost 40 % by reducing long travel times. We created a digital asset for this operation as well as having the option to delete layers based on VEX (Houdini's C like shading and geometry manipulation language) attribute group syntax. This could also be optimized more aggressively later on by using a point cloud lookup and searching for nearest points that have not been printed yet.

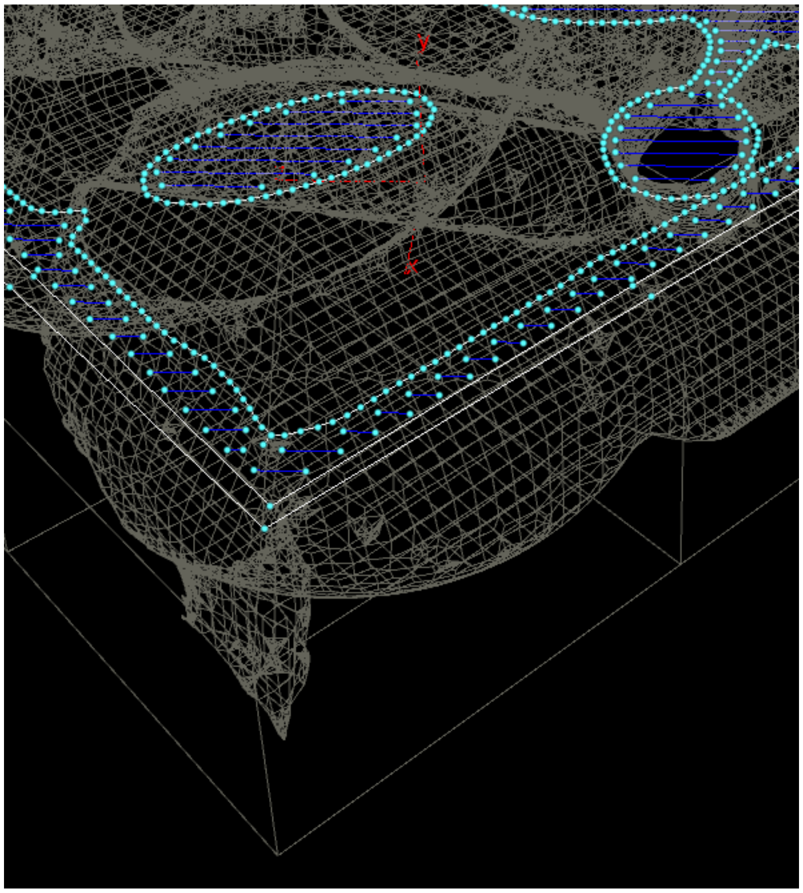

Figure 6.1.4.5: *Level tool path post processing for overlaps and edge reduction close-up.*

6.3 Infill Implementations

Initially we were looking to implement a single infill algorithm (centroidal voronoi partitioning) by Lu et al. While researching their work we came across Wang et al.'s work and realized that there was an overlap between the two. We realized that we could combine the two processes and use the truss method described by Wang et al. only instead of using a skin truss we could create a truss from the edges of the voronoi diagram and save even more material than Lu et al.

After working on this we also realized that while we could save material it would not be as strong, however if we altered Lu et al.'s approach and used a greater iso value when creating the centroidal voronoi we could have overlapping hollowing regions and combine that with the voronoi truss method to create a third alternative method.

Below we describe our implementation of there methods.

6.3.1 Manifold Geometry

Being able to confirm that you have manifold geometry is an important part of the initial steps of creating a printed object. It is important to know: Where is the inside of an object? And, where is the outside of an object in order to know where you need to print and where you do not want to print. Creating manifold geometry is also a very common process in visual effects to ensure that you have proper collision geometry when doing simulations with volume colliders.

Open VDB is an open sourced C++ library that uses a hierarchal data structure as well as tools for efficient storage and manipulation of sparse volumetric discrete data [Museth, 2010]. On top of Houdini's native volume tools, open VDB is also tightly integrated in Houdini. With Open VDB, it is relatively straight forward to convert an object to a VDB volume representation and then convert back to a geometric representation. This process guarantees a manifold object (or in some situations manifold objects depending on the methods used).

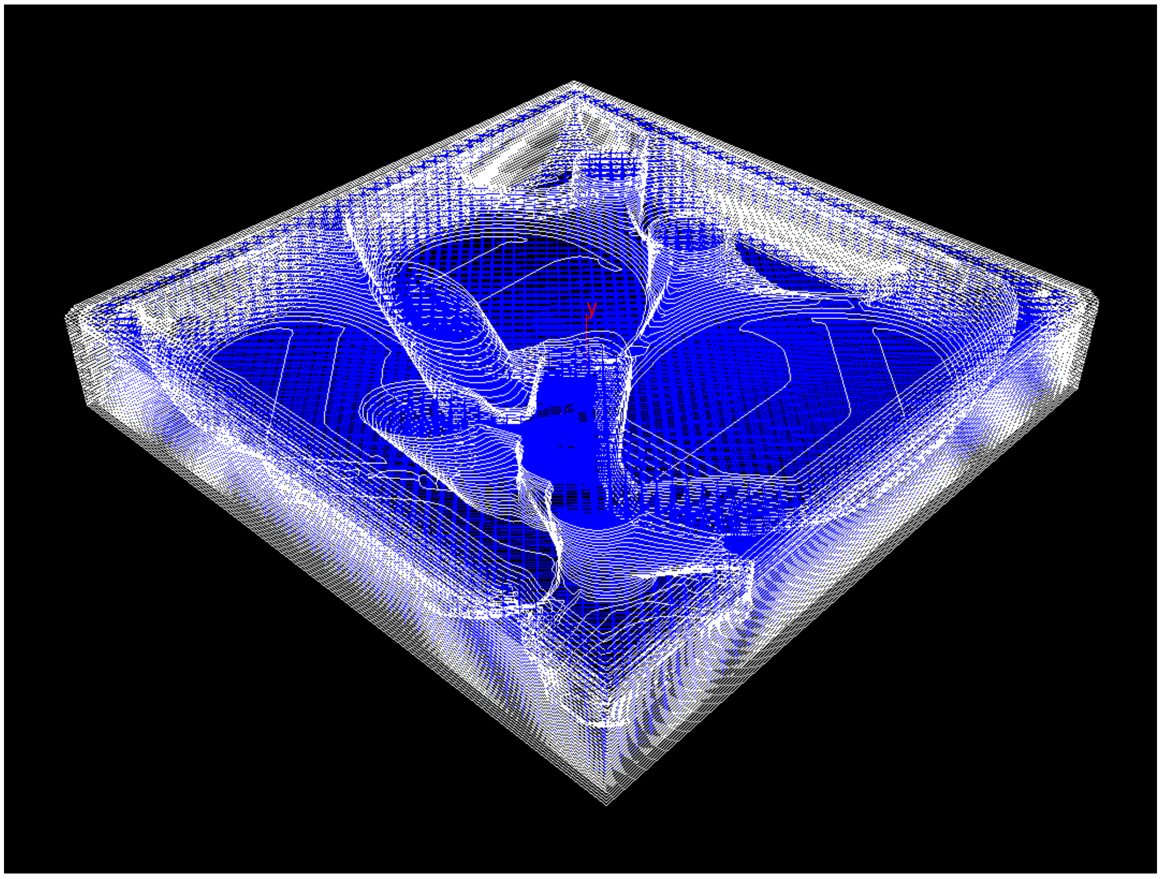

Figure 6.1.4.6: *Tool path for the first thirty three layers of a model using hybrid centroidal voronoi truss hollowing.*

However, despite the ease of creating the manifold representative geometry there are still some problematic issues. Volumes are comprised of one or more individual voxels (usually many). The number of voxels in the volume determines the volumes resolution. Just for purposes of explanation – it is sometimes helpful to think of voxels as a 3-dimensional version of a pixel. So if you have X number of vertical pixels and Y number of horizontal pixels you will have X by Y number of pixels. Same for a volume – except you will have X, Y and Z voxels. The issue here becomes if you go to too high a resolution you can very quickly use up all of your ram and lock up your computer. However through introspection of the geometry itself it is possible to determine an optimal volume resolution – either based on the bounding size of the object, the resolution of the size of the print head being used, or both. These can be options can be controlled by limiting the size of input objects to not be larger than the print area of the printer. Those defaults should *try* to match or be slightly higher quality than the printer head resolution.

In order to achieve this we make the assumption that the user must know the unit scale that they are working in metric units. We simply uniformly scale the object to the correct unit scale that we have predetermined for the printable size for the printer type selected (currently it will only support Ultimaker 2+ Extended however this could be extended to include other printers).

This step is common amoung the hollowing algorithms to ensure we have manifold geometry.

6.3.2 Centroidal Voronoi Partitioning

As mentioned earlier, Lu et al. [2015] proposed a hollowing optimization problem based on adaptive centroidal voronoi partitioning. Although, we are not necessarily interested in the adaptive section (we would prefer to leave this up to the user whether they are more interested in strength or effective use of material). In order to approximate the steps that Lu et al. took to achieve their method we break it down into smaller steps that can easily be replicated in Houdini:

1. Ensure manifold representation of the geometry.
2. Tetrahedralize the manifold geometry prior to simulating.
3. Simulate the tetrahedralized geometry using the finite elements method.
4. Utilize the results of the finite elements simulation to create a stress map.
5. Create a volumetric representation of the stress map.
6. Scatter points in the stress mapped volume.
7. Create a voronoi decomposition.
8. Create centroidal hollowed areas.

For steps 1 through 5 we created a digital asset to preform these operations, as they will be reused in future hollowing methods. This digital asset's type name is 'fem_analysis' and it's label is 'FEM Analysis'. First, from an input shape or one of the test geometries we create a manifold object by using a 'VDB from Polygon' node in Houdini. The resolution of the grid is fixed assuming a maximum resolution of the build area and the object size. This ensures a manifold geometry (even if the input geometry is two dimensional). The volumetric representation is then re-converted back to a geometric representation using a 'Convert VDB' node.

Next, a "high quality" fixed length (nearly equilateral) remeshed surface is created which contains only triangles using a 'Remesh' node. With this computed surface mesh a tetrahedralized interior mesh is added using a 'tetrahedralize' node.

This tetrahedralized geometry is now fed into a finite element method simulation, and we compute an initial stress map. The type of stress map that is created can be determined by the user they have the options of creating a stress map from either: 'dissipation density', 'kinetic density' or 'potential density'. Taking this point attribute accumulated over all of the points in the tetrahedralized mesh, we calculate the maximum and minimum values and normalize this data. For our case we remap the lowest value to 0.1 and the highest value of 1 additionally remapping every value in between accordingly. This is to try and maximize the hollowness of areas of low densities.

Sometimes when performing FEM simulations there are either extreme values or values that the user may want to ignore or bound. There are two methods that are promoted to the top level of the digital asset for doing this: 'Ignore Min/Max Values' which will ignore values below the specified minimum value and values and values above the specified maximum values, and 'Set Min/Max Values' which will effectively floor values below the specified minimum value to the minimum value and ceiling the values above the maximum values to the maximum value.

The stress map is then used to create a density volume utilizing a 'Volume from Attribute' node. Then we scatter points relative to the density of the volume.

The digital asset outputs 5 different types of geometry (in order of outputs):

- The simulated geometry
- The stress map as a volume
- The stress map as points from the FEM Simulation (displayed with colour values from 0.1 – 1)
- Points scattered in the stress map by density (the number of point can be controlled using the 'Number of Points to Scatter in Stress Map' parameter)
- Original geometry (not remeshed as the simulated geometry is)

We then created a second digital asset which would do the centroidal voronoi hollowing. This digital asset's type name is 'centroidal_voronoi' and it's label is 'Centroidal Voronoi'.

This digital asset takes three inputs (usually from the FEM Analysis node): 'Simulated Geometry', 'Voronoi Points' and 'Original Geometry'.

There are limited parameterized options that can be changed by the user at the top level:

'Output' – either 'Hollow area only' or 'Hollow area and Original Geometry'
'Deletion Index' – This is the primitive number to delete to get the hollow area only (usually 0 – the primitive with the highest area or 1 the primitive with the second highest area).
'Voronoi Hollowing Type' – Either 'Spherical Harmonic' or 'ISO Value'
'Iso Value' – This is the iso value to use when converting from a volume to a polygonal mesh.

We did not implement this adaptively as we would prefer to give control to the user to adaptively adjust manually as they see fit – however there is nothing to that it cannot be implemented adaptively.

Depending on the options listed above the original geometry is then passed inside where a voronoi distribution is created and the cells are hollowed. Once this process is complete, the output geometry can be fed into the 'Tool Path' digital asset, and then the 'Tool Path Optimization' digital asset and finally the 'G-code' digital asset to print the resultant G-code to print on an Ultimaker 2x printer (see example 'centroidal voronoi' in the submitted .hip file).

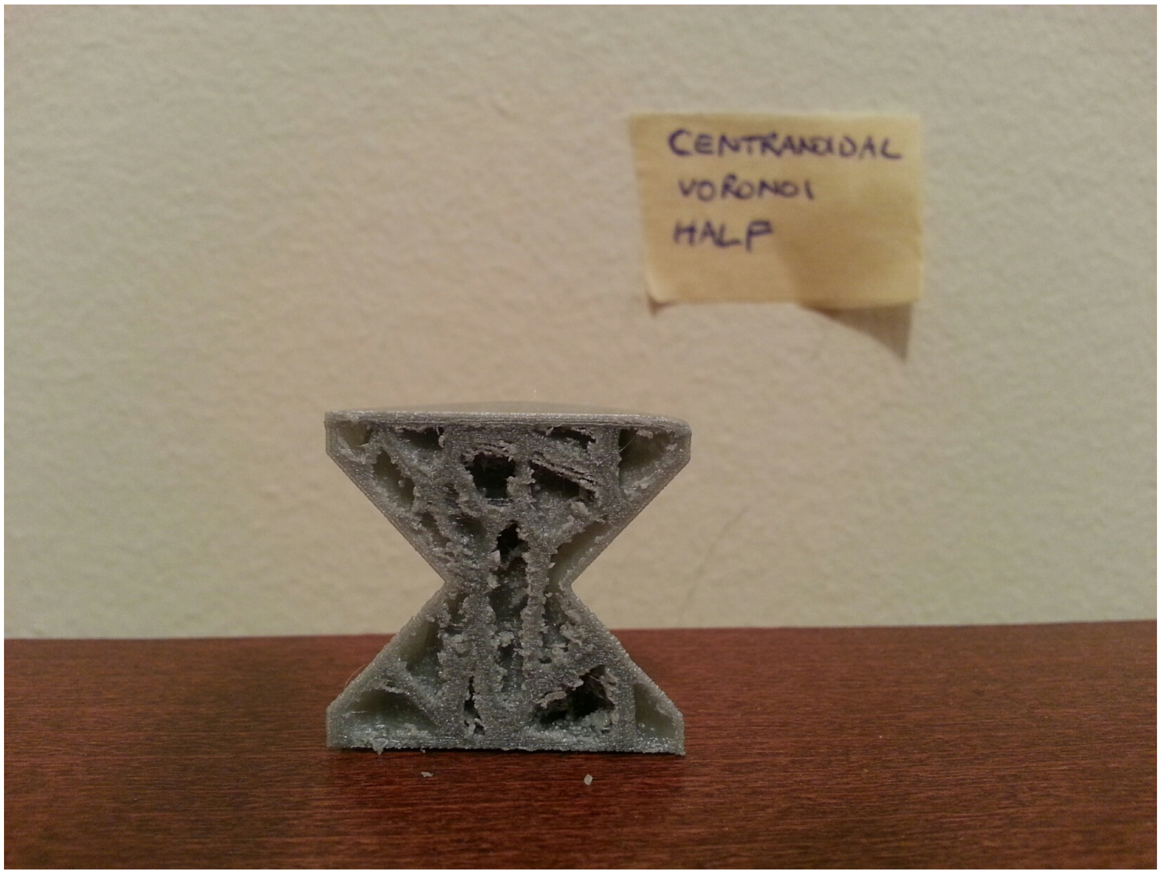

Figure 6.3.1.1: *A cross section of the centroidal voronoi method.*

6.3.3 Voronoi Truss

In order to implement the voronoi truss method we created a digital asset with the type name 'voronoi_truss' and the label 'Voronoi Truss'. For this method we create a voronoi fracture from the input voronoi points and then hollow the entire object except the edges from the voronoi edges. Out of all of our implementations this has the option to be the most material saving, however the strength may suffer.

The user has the option to control the width of the voronoi truss via a promoted parameter on the top level of the digital asset called 'truss width'.

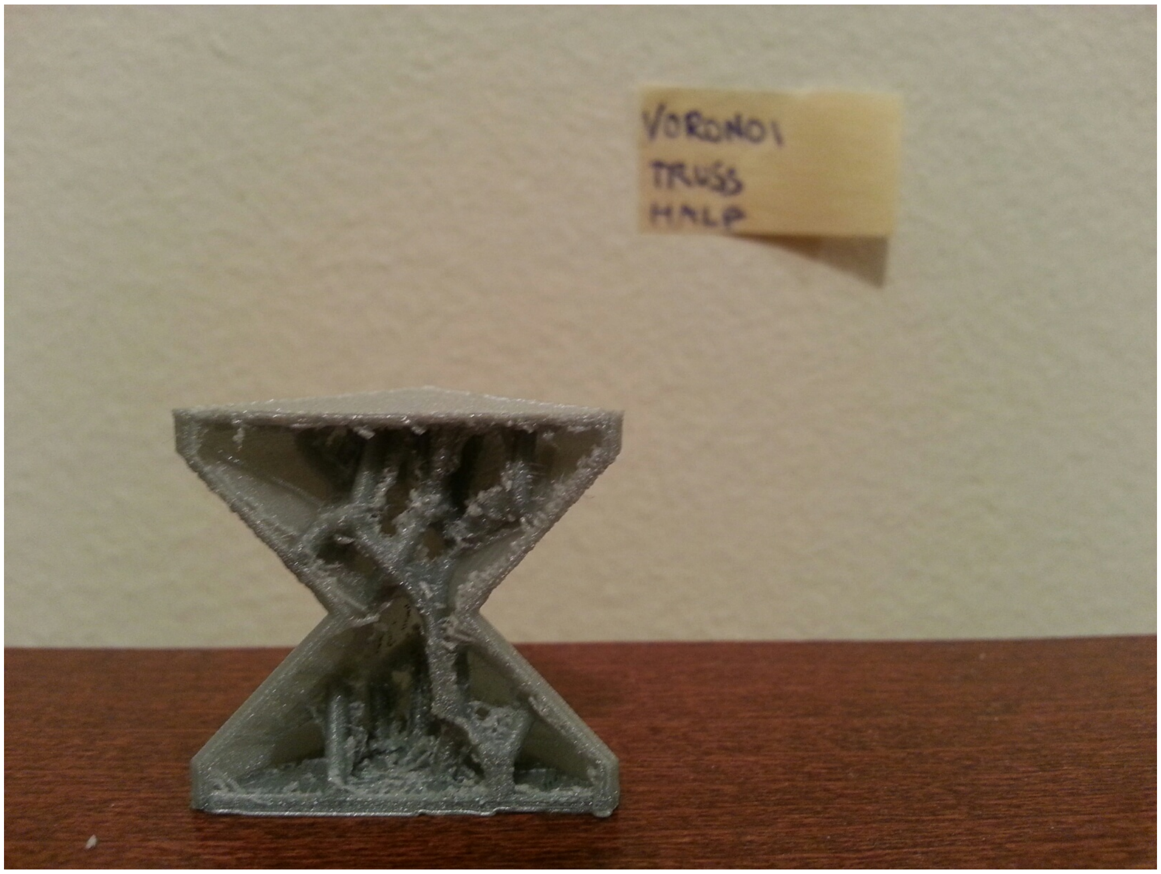

Figure 6.3.2.1: *Voronoi truss method cross section.*

6.3.4 Hybrid Voronoi Truss Partitioning

This method is a combination of the centroidal voronoi hollowing and the voronoi truss. This allows us to use a larger iso value when recreating the hollowed region in the voronoi to go outside the original boundary and even overlap with regions nearby. We then use this combined volume in conjunction with the voronoi truss method to ensure that we have a minimum of at least the truss, and the user can alter the amount of material (and the resultant strength of the printed model).

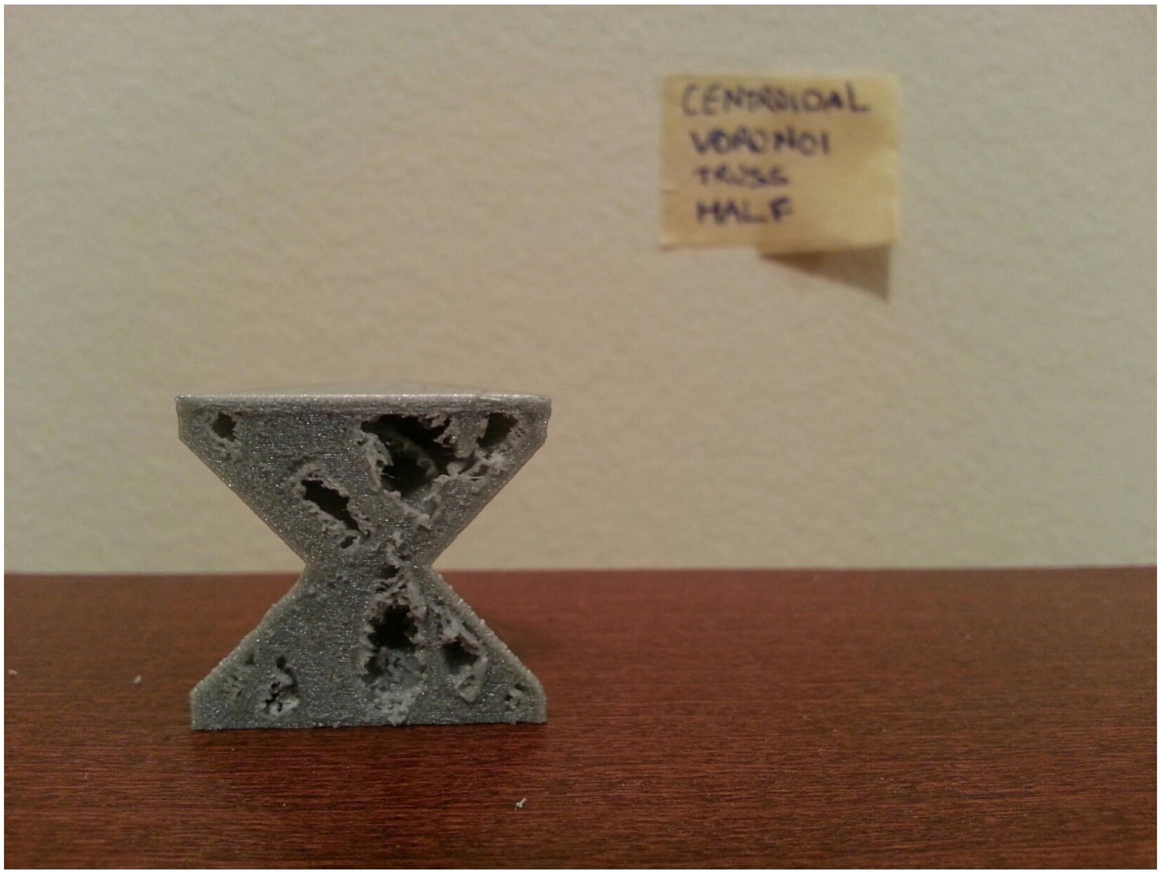

Figure 6.1.4.6: *Hybrid centroidal voronoi truss method cross section.*

Conclusions

Weighing the different models after printing them we were able to determine approximately the amount of material used:

Solid model - ~9 grams
Centroidal voronoi hollowing - ~ 5 grams
Voronoi struts - ~ 3 grams
Hybrid voronoi struts - ~ 6 grams

After printing the models we attempted to stress test them to see at what point they would break. We used a bench vice and a weight scale to test the printed models. The only model that we were able to fracture was the voronoi struts model at 30 Kg's. All of the other models would break up to 50 Kg's of pressure.

Overall we consider the project to be a success. We were able to achieve most of the objectives in the problem statement. However, there were some that we were not able to accomplish. The most major of which is testing the simulated strength. This was simply not possible in Houdini as the remeshing took much too long.

There were several other issue with the project including:
- Currently this only works well with smaller geometry.
- Since we use normals to inset outer boundaries, the inset does not inset equally.
- The speed of the current system should be improved (most should be rewritten in VEX for multithreading).

Future Work

There are many different areas that this work could continue in. It could prove to be quite useful for 3-D printing as a research tool. As it has been written in a fairly modular fashion expanding the different hollowing models should be straighforward.

Additionally improving the inset algorithm for the boundary edges would be a good improvement. As would rewriting a majority of the tools in VEX to improve the time to print.

References:

Brooks, H., Rennie, A., Abram, T., McGovern, J., 2012. Variable Fused Deposition Modelling – Analysis of Benefits, Concept Design and Tool Path Generation, *Proceedings of the 5th International Conference on Advanced Research and Rapid Prototyping, 511-517.*

Brown, A., de Beer, D., 2013. Development of a stereolithography (STL) slicing and G-Code generation algorithm for an entry level 3-D printer, *AFRICON 2013.*

Cao, W., and Miyamoto, Y. 2003. Direct Slicing from AutoCAD Solid Models for Rapid Prototyping. *International Journal of Advanced Manufacturing Technology,* 21:739-742.

Chakraborty, D., Reddy, B., Choudhury, A., 2008. Extruder path generation for Curved Layer Fused Deposition Modelling, *Computer Aided Design, 20 2,235-243.*

D'Aveni, R. 2015. The 3-D Printing Revolution, The Big Idea, *Harvard Business Review*, May 2015

The Economist, 2015. A third industrial revolution, *The Economist.* May 21st, 2014. Retrieved September 30th, 2016.

Guerrero-de-Mier, A., Espinosa, M., Domínguez, M., 2015. Bricking: A New Slicing Method to Reduce Warping, *MESIC Manufacturing Engineering Society International Conference 2015,* Volume 132: 126-131

Huang, S., Liu, P., Mokasdar, A. and Hou, L. 2013. Additive manufacturing and its societal impact: a literature review. *International Journal of Advanced Manufacturing Technology,* 67:1191-1203.

Hyman, P. 2011. Ten Disruptive Technologies, *Communications of the ACM*. Vol. 54, Issue 9, p20-20.

Kodama, H. 1981. Automatic method for fabricating a three-dimensional plastic model with photo-hardening polymer, *Review of Scientific Instruments,* Vol. 52, No 11, 1770-1773.

Kou, X., Tan, S., 2010. A simple and effective geometric representation for irregular porous structure modelling, *Computer Aided Design*, volume 42, issue 10:930-941

Lee, K., H., and Choi, K. 2000. Generating Optimal Slice Data for Layered Manufacturing. *International Journal of Advanced Manufacturing Technology,* 16:277-284.

Lu, L., Sharf, A., Zhao, H., Wei, Y., Fan, Q., Xuelin, C., Savoye, Y., Changhe, T., Cohen-Or, D., & Baoquan, C., 2014. Build-to-Last: Strength to Weight 3D Printed Objects. *ACM Transactions On Graphics*, 33, 4.

Luo, N. and Q. Wang. 2016. Fast slicing orientation determining and optomizing algorithm for least volumetric error in rapid protyping. *International Journal of Advanced Manufacturing Technology,* 83:1297-1313.

Moore, W., 1970. *Foundations of Mechanical Accuracy* (1st edition), Bridgeport, Connecticut, USA: Moore Special Tool Co.

Monzón, M. D., Ortega, Z. and Ortega, F., 2014. Standardization in additive manufacting: activities carried out by international organizations and projects. *International Journal of Advanced Manufacturing Technology,* 76:1111-1121.

Museth, K., 2013. VDB: High-resolution sparse volumes with dynamic topology, *ACM Transactions on Graphics (TOG),* volume 32, issue 3, article No. 27.

Rifkin, J. 2011. *The Third Industrial Revolution*. Palgrave McMillan
Vaezi, M. and Chua, C., K. 2010. Effects of layer thickness and binder saturation level parameters on 3D printing process. *International Journal of Advanced Manufacturing Technology,* 53:275-284.

Wang, W., Wang, T., Yang, Z., Liu, L., Tong, X., Tong, W., Deng, J., Chen, F., and Liu, X. 2013. 'Cost-effective printing of 3D objects with skin-frame structures' *ACM Transaction On Graphics*, 32, 5.

Wilson, S., 1990. A new face of aerospace honeycomb, *Materials and Design*, volume 11, issue 6:323-326.

Xinghua, L., Shengpeng, L., Zhou, L., Xianhua, Z., Xiaohu, C., Zhongbin, W. 2015. An investigation on distortion of PLA thin-plate part in the FDM process. *International Journal of Advanced Manufacturing Technology,* 79:1117-1126.