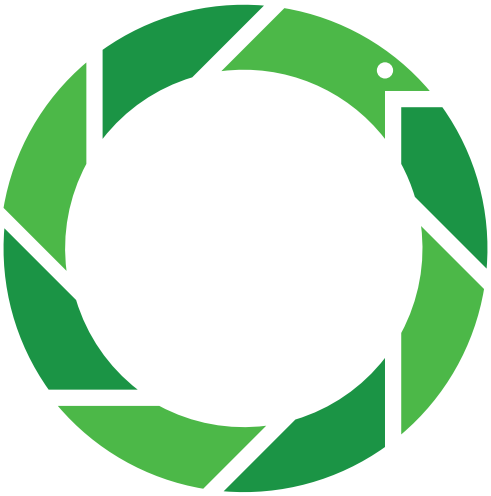


Ouroboros
Handbook

Using Grasshopper, Ouroboros and Xylinus to develop 3D-prints without travel movements
V1.0

A diploma thesis by Daniel Westhof

Kunsthochschule Kassel 2021

{0;0} Introduction

foreword

This is not only the initial release of the Ouroboros Handbook but also my diploma thesis.
I will extend it as Ouroboros develops.

Although basic knowledge about Rhino and Grasshopper is expected from the reader, I hope that this document is also interesting to other technologically savvy people using different CAD packages.

This first revision will only cover Fused Deposition Manufacturing (FDM) printing. Ouroboros has a lot of potential for ceramic 3D-Printing with syringes but for now I'll stick with things I'm familiar with.

aknowledge-ments

Thanks to the IKEA Stiftung for funding my work on this project by granting me a scholarship.

To Markus Schein and Oliver Vogt for mentoring me.

To Ryan Hoover for laying the foundation on which I build Ouroboros by developing Xylinus.

To my parents for believing in me throughout my long time at university.

abstract

The goal of Ouroboros is aiding the development of 3D-printed products, using a big nozzle, by allowing to print in one go without travel movements. This results in short print times while achieving a nice surface finish even with 1 mm nozzles and 0.8 mm layer height. This makes the 3D-printer a viable tool for small scale fabrication of products. Even furniture parts can be printed in a reasonable time.

tools used

Software:

Rhinoceros 3D version "7 SR8"
https://www.rhino3d.com

Grasshopper version "Thursday, 15 July 2021 05:00"
https://www.grasshopper3d.com

Xylinus version 0.33.00
https://www.food4rhino.com/en/app/xylinus-novel-control-3d-printing

Ultimaker Cura version 4.10.0
https://ultimaker.com/software/ultimaker-cura

3D-Printer:

Heavily modified Anet A8 from 2017 With fake Volcano Hot end from china and 1mm nozzle.

https://anet3d.com/pages/a8

Filament:

PLA (Polylactic acid) refills from Das Filament
https://www.dasfilament.de

PLA is a bioplastic made from cornstarch.

declaration of academic integrity

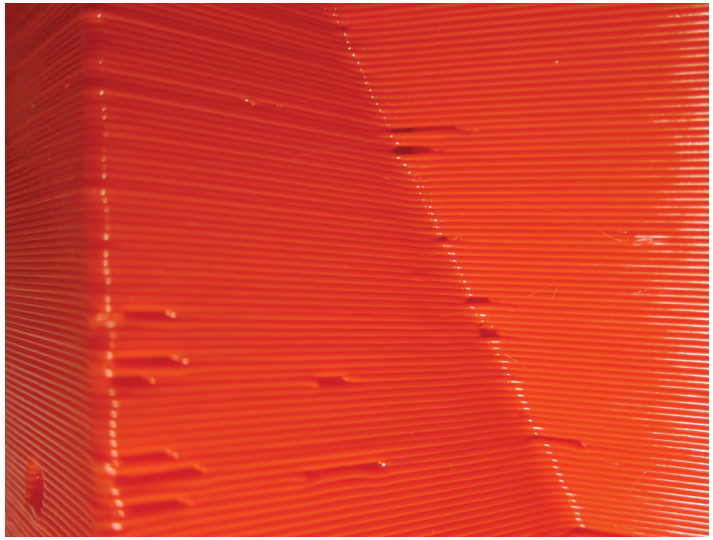
I, Daniel Westhof, declare that this thesis is my own work and that I didn't use any texts or graphics I didn't create myself except when it is acknowledged as such and properly attributed.
I also declare that all code and components made by others that I used in my grasshopper definitions are either part of the software listed above or are marked as such and credit is given.

_____ _____
date signature

{0;1} Objective

the issue to solve

The major issue with printing with big nozzles is that at some point the stringing can not be handled by tuning the retraction and temperature settings anymore. This results in messy surfaces with spikes and holes.

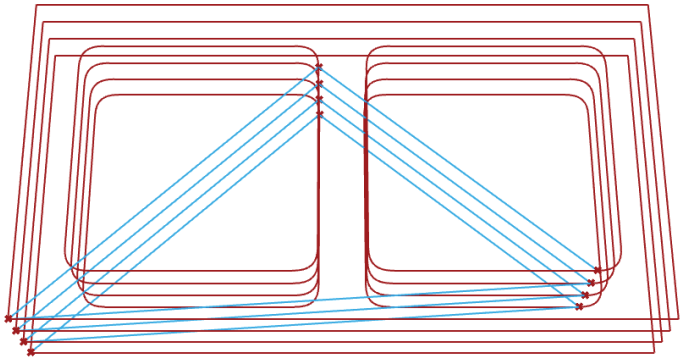


Combing can only mitigate the problems and results in even longer travel times. But, didn't we buy our printers to print and not to travel? And didn't we choose the 1 mm Nozzle to print big parts crazily fast?

That is why I became obsessed with creating 3D-prints that can be printed in one go without traveling at all.

When your slicer slices an object it generates a set of curves at the intersection of an XY-plane and the object for each layer. These are offset to compensate for the extrusion width. Usually you have no control over the start and the end of the curves, and the slicer wants to print each curve completely before going to the next. This means that even touching curves have travels between them.

I was not willing to accept that.

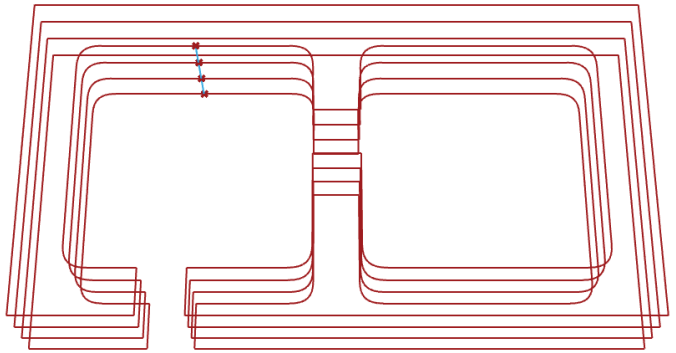


red dots: start/end points
red lines: print paths
blue lines: travel paths

the theory behind my solution

What if you could tell the slicer to print one curve to a specific point, switch to another touching curve, print that first and then to continue with the first one? If you could stitch all the curves into one big curve? If you now could also align the end points, preferably to somewhere out of sight, the only travel movements that are left are the layer changes.

You can do all of that! I call it stitching and this handbook will teach you how to do it.



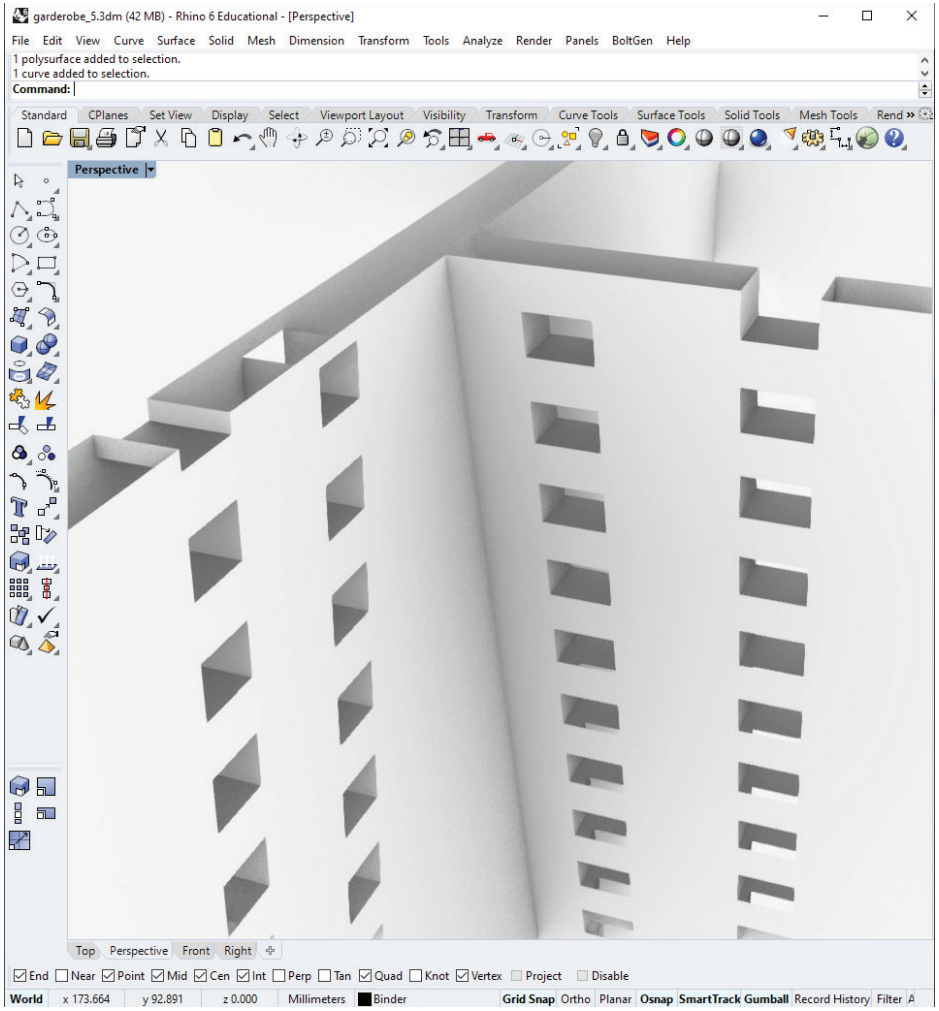
This, of course, has some major limitations. All closed curves have to be touching each other on at least one spot.
But inside these limitations is a lot of room for creative solutions and functional aesthetics.

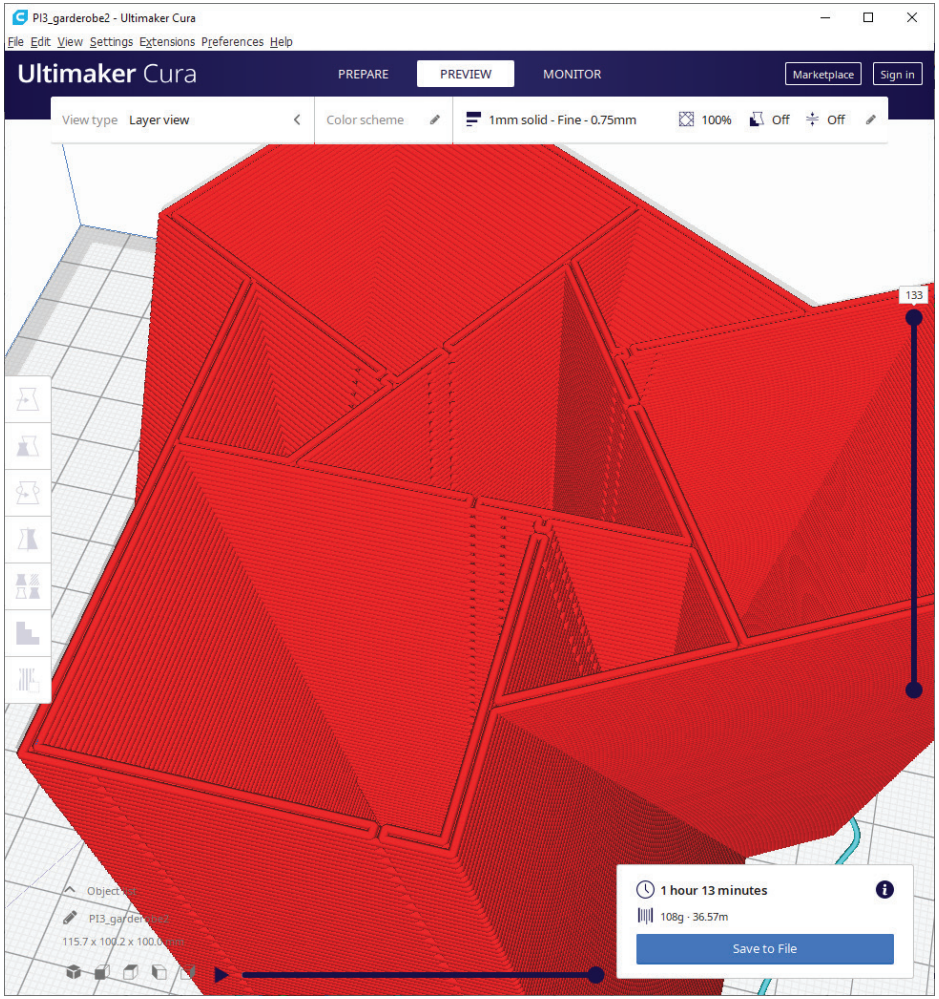
{0;2} My pre Ouroboros work flow

basic setup

Before I had started Ouroboros, I achieved satisfying results with Curas **Surface Mode**. It does not fill out the sliced curves but uses them directly as paths.

To show Cura where I want the stitches to connect the compartments, I had to cut holes into the double walls and to connect the surfaces in a staggered pattern as shown below.





Print paths generated by Cura with this technique.

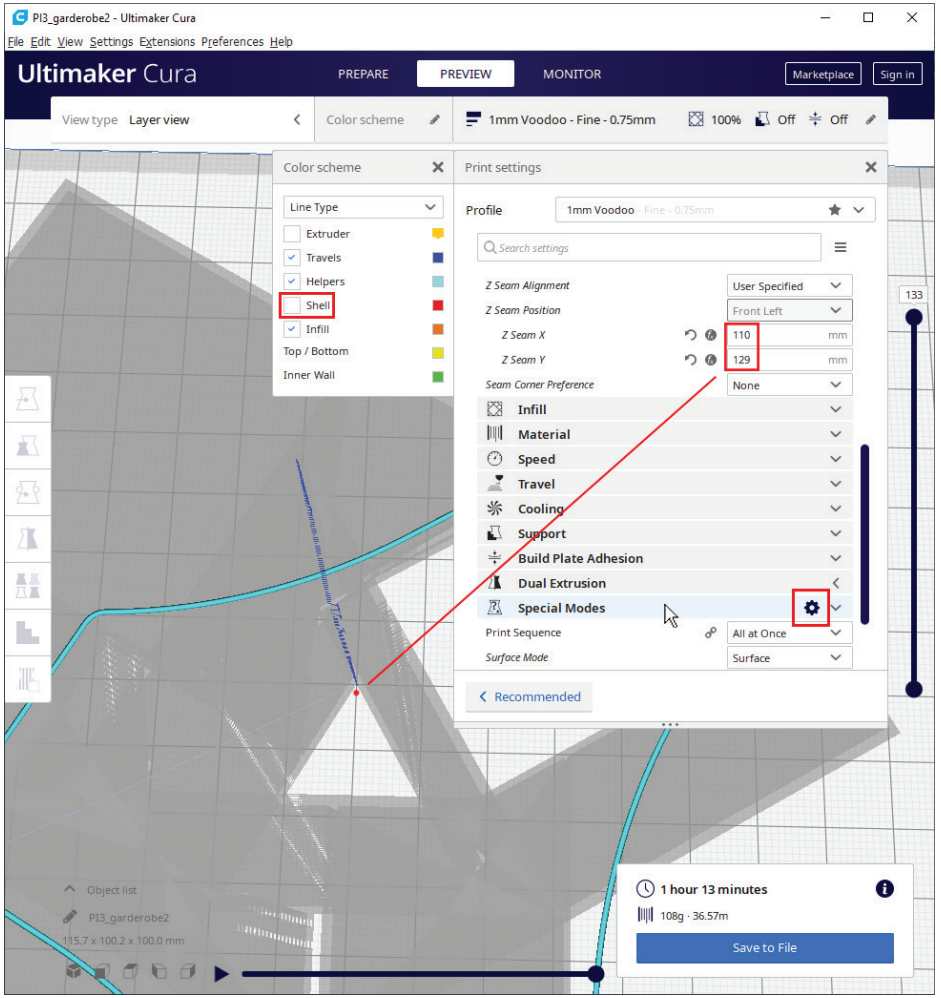
Unfortunately, this only works most of the time, but finding the reason can be a nightmare, because Cura is basically a black box. Another frustrating issue was that crafting these stitches can be a tedious task and when I wanted to change the layer height I had to start all over again.

Although it basically worked fine, there had to be a better solution. When I realised that I could transfer this work flow directly into Rhino and Grasshopper, Ouroboros was born.

Cura settings

If you want to replicate this technique, you have to edit the following settings:

Set **Surface Mode** to **Surface** and **Z Seam Alignment** to **User Specified**. Then you can either select a preset from **Z Seam Position** or define the seam more precisely with **Z Seam X** and **Z Seam Y**. The latter has the advantage that you can hide the seam somewhere inside where it doesn't affect the outer surface. In some cases **Shortest** works fine as well but the seam will then be on the outside though.



To check if everything worked, uncheck **Color Scheme > Shell** to reveal the travel paths. They should be neatly stacked. Sub mm variations are allright.

Keep in mind that the seam can only be on a corner and not in the middle of a line.

Also a hint to Cura beginners:
Most settings are hidden by default. To get to the **Setting Visibility** preferences, click on the gear that appears when you hover over a setting category.

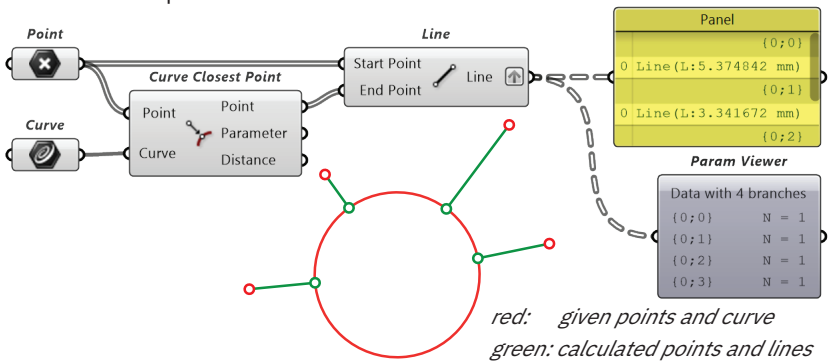
The other settings heavily depend on your printer and your geometry. But I had good results with my Anet A8 printer, with Volcano hot end and a 1 mm nozzle, with layer heights between 0.5 mm and 0.75 mm and an outer wall speed between 20 mm/s and 30 mm/s.

{1;0} Grasshopper Basics

what is Grasshopper

Grasshopper is a visual scripting interface for Rhino, a Computer Aided Design (CAD) application developed by McNeel & Associates.

It has a work flow similar to modular audio synthesizers which allows the creation of parametric geometry by connecting components with “cables”.



red: given points and curve
green: calculated points and lines

A network of Grasshopper components as shown above is called a definition. They have inputs on the left and outputs on the right. The data is organized in so-called data trees which can be thought of as nested lists.

data trees

The contained data is like leaves - attached to a branch that is part of a bigger branch and so on. The addresses of the branches are called data paths whose path elements are separated by a semicolon and surrounded by curly brackets.

You can view the contents of a data tree in a **Panel** or just the tree structure itself in a **Param Viewer**.

The tree structure of the inputs and outputs can be manipulated by setting a series of flags. In the example above you can see the **Graft** (⇧) flag which puts every item into its own branch.
So instead of all being in {0}, they are in {0;0}, {0;1} etc.
There are also **Flatten** (⇩) to put all items of all branches into a single branch, **Reverse** (⇄) to reverse the order of the items in each branch and **Simplify** (⧖) to remove all path elements that are the same on all paths.

parameters

Like variables in programming languages, parameters contain data. They can reference Rhino geometry or contain data that is generated by Grasshopper components. In addition to the **Point** and **Curve** parameters you can see in the example, there are a lot of other data types. For example, **Boundary Representations (Breps)** which can contain surfaces and solids.

components

There are components for all kinds of operations, for example geometry, manipulating the data tree and mathematics.

Grasshopper has a very active community that provides a multitude of mostly free plug-ins that contain additional components. They can be pre-compiled from C# or contain user editable Grasshopper definitions called **User Objects**.

{1;1} Xylinus Basics

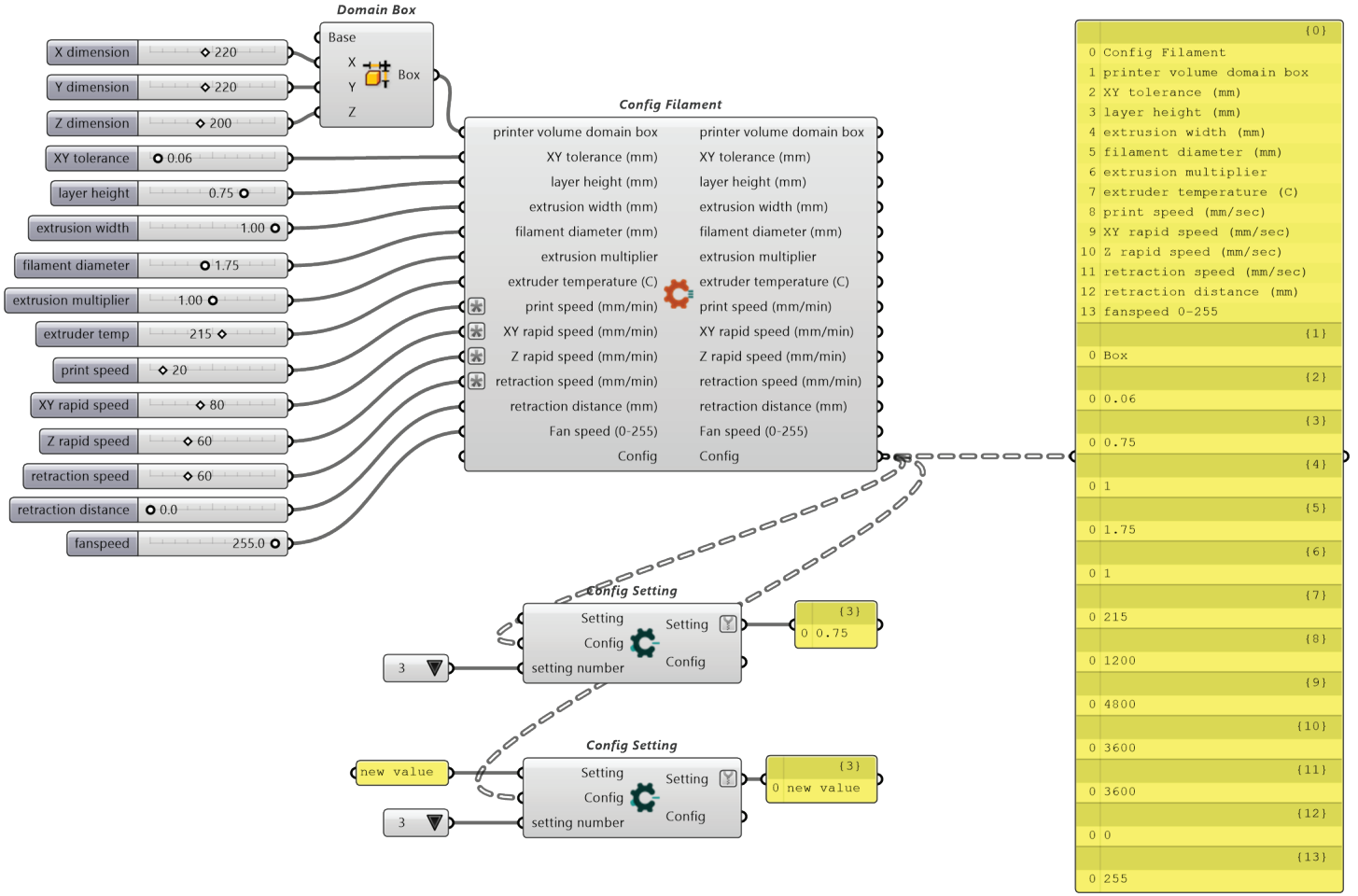
what is Xylinus?

Xylinus is a set of Grasshopper **User Objects** developed by Ryan Hoover. It can generate G-code, the control instructions for the printer, directly from Rhino and Grasshopper geometry.

It works with FDM printers, syringe printers and resin printers.

Ryan Hoover created Xylinus to control bioprinters and other experimental 3D printers he has developed at the Baltimore Under Ground Science Space and the Maryland Institute College of Art.

Within the scope of my diploma project I made quite a few contributions to Xylinus that will be part of the next release. More on that later.



the config system

All settings are held in a data tree.

They can be inserted and extracted with the config components either in bulk or individually.

While the **Config Setting** component is universal a specialised bulk component is needed for each printer type.

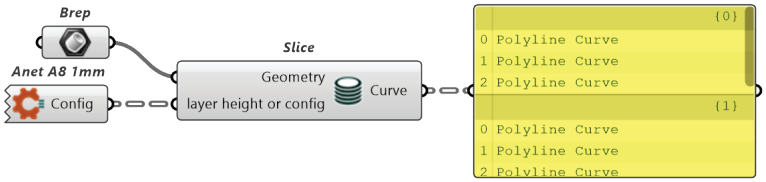
The table of contents in the {0} branch is only for human readability. To access the settings only the id is needed.

Most Xylinus components can accept the specific setting or the whole config bundle.

These configs can be clustered and saved as a user object for future use.

slicing

To be able to print something we first need curves.
You can generate them anyway you want but Xylinus offers some tools for that.

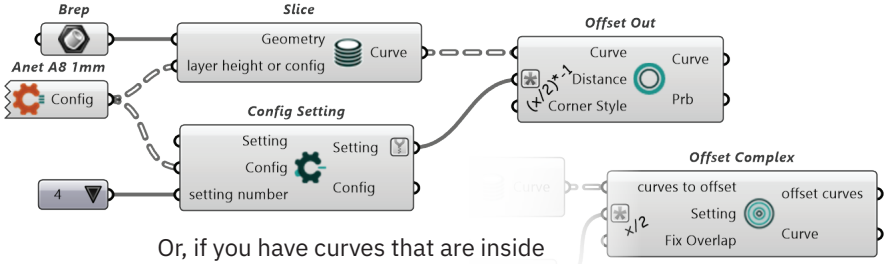


Slice, for example, takes a surface or a solid and generates the contour curves for each layer.

manipulating paths

Before you print those curves you can do with them whatever you want, with all the might of Grasshopper at your fingertips.

Xylinus also brings some handy components that make handling the curves easier. **Offset Out** for example.

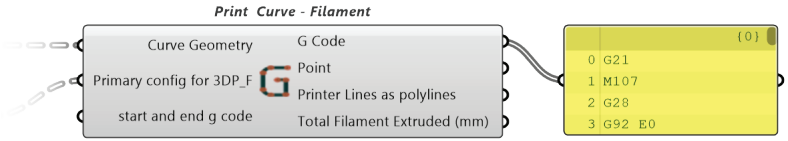


Or, if you have curves that are inside other curves and you want the inner ones to offset to the outside, **Offset Complex** is your friend.

There are also rudimentary tools to generate infill.

generating G-Code

Print Curve Filament converts the curves to G-code. If not supplied, it generates some basic start and end G-code but you can and should customize it to suit your printer.



Ouroboros Basics {2;0}

what is Ouroboros?

Ouroboros is a set of user objects intended to be used in conjunction with Xylinus. The heart of it are two components that stitch curves together in the way I described in {0;1}.

the origin of the name

Ouroboros is the Greek name of the serpent devouring its own tail. Originally The symbol is from Egypt and is found in the mythology of many other ancient peoples. It is interpreted as the eternal cycle of life, death and rebirth and the oneness of everything.

By unifying curves and always returning to the starting point Ouroboros reflects in its name the essence of its function.

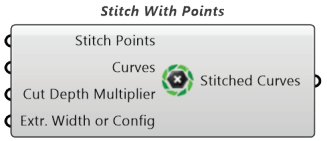
source: Wikimedia, public domain
(https://commons.wikimedia.org/wiki/File:Serpiente_alquimica.jpg)

{2;1} Ouroboros Components

In this chapter I will describe the Ouroboros components followed by the containing definitions displayed in fold-outs.

Stitch With Points

Stitch With Points combines a list of closed curves into one closed curve using points to position the stitch.

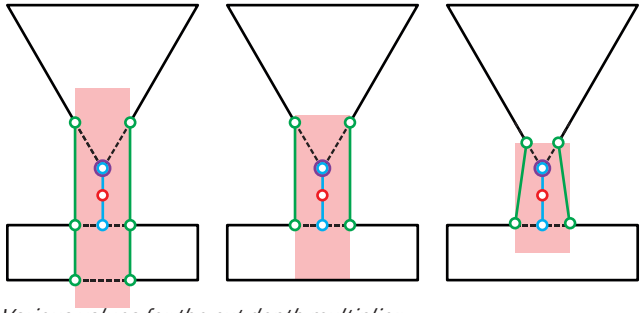


The **Stitch Points** have to be between or near the curves that have to be stitched. Of course you can stitch more than two curves at once. Just make sure that you have one point for each stitch.

Their order is irrelevant - they find their curves by ignoring everything that is more than twice the **Extrusion Width** away.

It works by constructing a rectangle to cut into the curves, deleting the intersection and connecting the curves by their exposed end points.

The cutting rectangle is constructed by first finding the closest points to the **Stitch Point** on the curves. These points are connected by a line. Its midpoint is the center of the rectangle which is aligned to that line. The rectangle has a width of the **Extrusion Width** and the height of twice that, multiplied by the **Cut Depth Multiplier**.



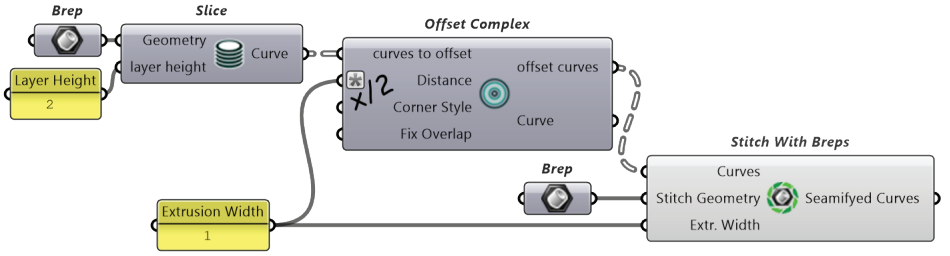
Various values for the cut depth multiplier
Black: curves
Purple: stitch point
Blue: closest points on curve to stitch points
Red: cutting rectangle and its center point.
Green: resulting stitch lines

The **Cut Depth Multiplier** determines how far the rectangle will cut into the curves. If its value is too high, it might cut into other parts of the curve (shown in the left sketch). If it is too low, the rectangle might not cut with its sides and the end points would end up to be too close to each other (shown in the right sketch).

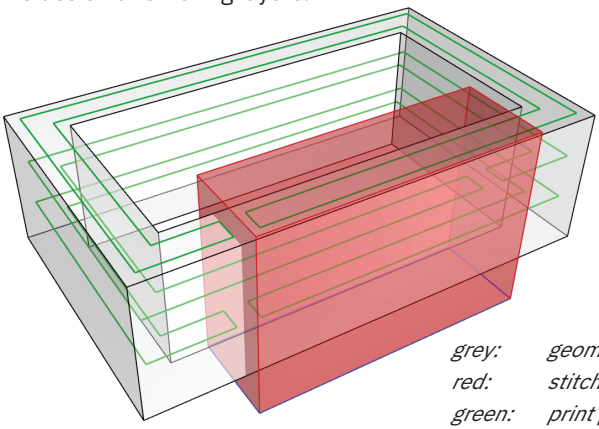
The stitches are possibly a weak point. I would advise not to put them on the same spot in each layer.

Stitch With Breps

Stitch With Breps merges a list of closed curves into one closed curve using solids to generate an alternating stitch pattern.



The intersection between the curves and the **Stitch Geometry** marks the position of the stitches. The stitch aligns to opposite sides on alternating layers.



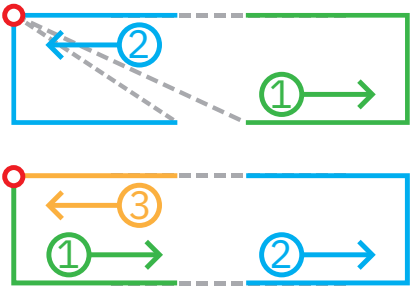
grey: *geometry to slice*
red: *stitch geometry*
green: *print paths*

Hole Punch

Hole Punch is, for now, the only utility component included in Ouroboros.

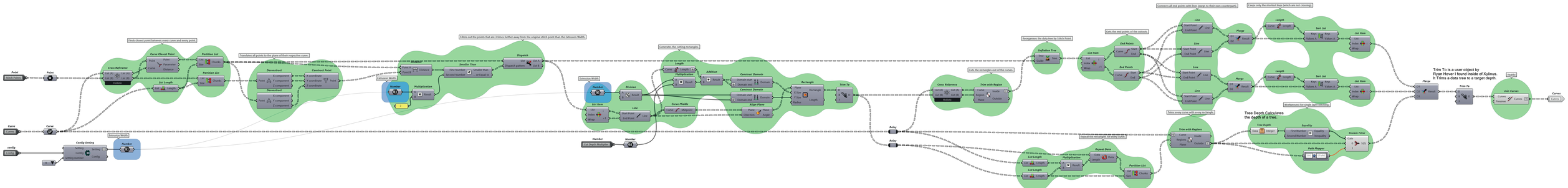
Sometimes it is unavoidable to break the circle and cut a hole into a curve. Having open curves can result in travel movements but if you need them, they should be right where the hole is.

But just cutting a hole into the closed curve is not going to cut it. We want the layer to be started at the former seam point. If you cut a closed curve in Grasshopper, the seam point will be in the last segment. That segment has to be cut at the seam point and the off cut belongs to the beginning of the layer.

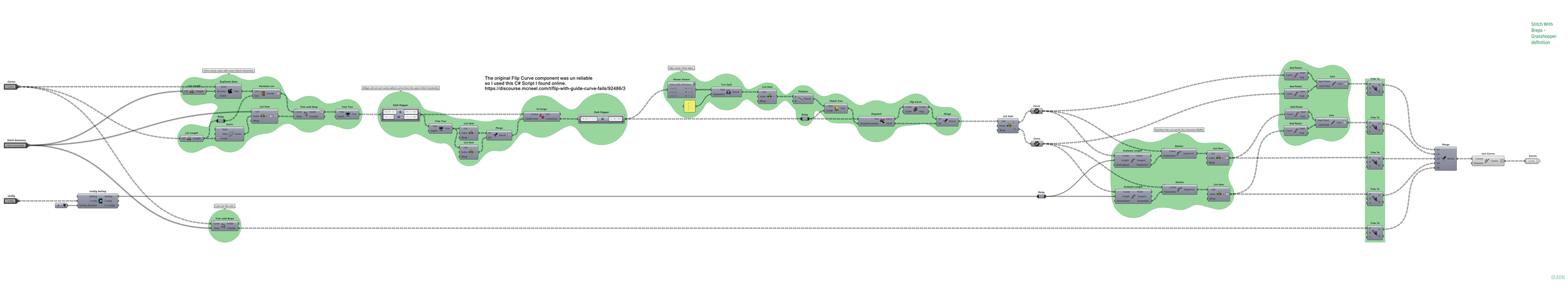


circle: *former seam point*
lines: *print path*
arrows: *print direction and order*
dashes: *travel movements*

That is what **Hole Punch** is taking care of for you.

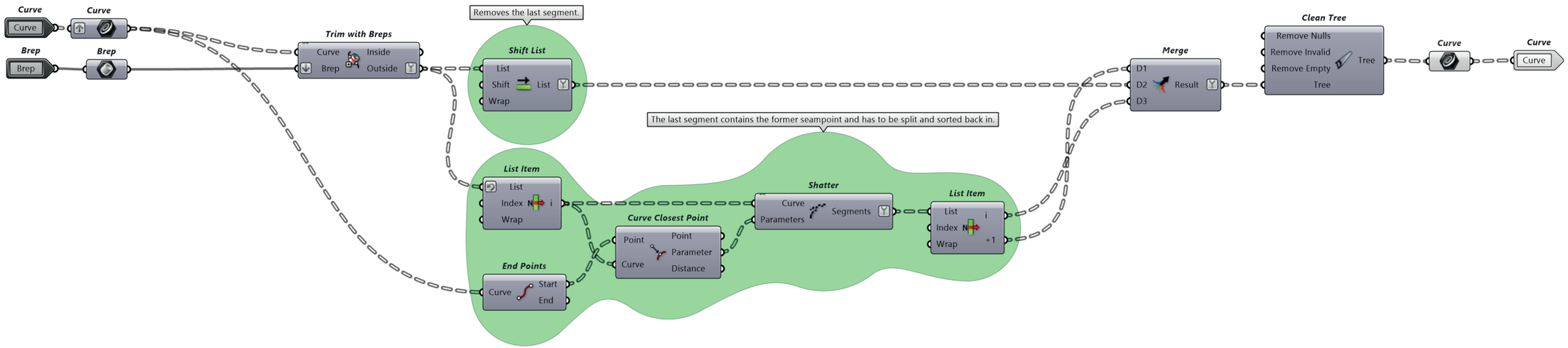

Stitch With Points - Grasshopper definition
Finds closest point between every curve and every point.
Translates all points to the plane of their respective curves.
Filters out the points that are 1 times further away from the original stitch point than the Extrusion Width.
Generates the cutting rectangles.
Reorganizes the data tree by Stitch Point.
Cuts the rectangles out of the curves.
Gets the end points of the cutouts.
Connects all end points with lines (swaps to their own counterpart).
Keeps only the shortest lines (which are not crossing).
Trims every curve with every rectangle.
Repeat the rectangles for every curve.
Workaround for single input patterns.
Tree Depth Calculates the depth of a tree.
Trim To is a user object by Ryan Hover I found inside of Xylinus. It Trims a data tree to a target depth.
Paint
Curve
Config
Extrusion Width
Cross Reference
Curve Closest Point
Partition List
List Length
Deconstruct
Construct Point
Multiplication
Smaller Than
Dispatch
Number
Division
List Item
Line
Length
Addition
Construct Domain
Domain Start
Domain End
Align Planes
Rectangle
Trim To
Relay
Trim with Region
Unflatten Tree
End Points
Line
Merge
Sort List
Tree Depth
Equality
Stream Filter
Path Mapper
Repeat Data
Join Curves

(2:1)(3)


Stitch With Breps - Grasshopper definition
Curves
Stitch Geometry
config
Config Setting
Trims every curve with every Stitch Geometry
Duplicate Data
List Length
Partition List
List Item
Trim with Brep
Trim Tree
Series
Cuts out the rest
Flips all cut out curves which come from the same Stitch Geometry
Path Mapper
Trim Tree
List Item
Merge
C# Script
The original Flip Curve component was un reliable
so I used this C# Script I found online.
https://discourse.mcneel.com/t/flip-with-guide-curve-fails/92486/3
Flips every offset layer
Param Viewer
Text Split
List Item
Modulus
Match Tree
Dispatch
Flip Curve
Merge
List Item
Curve
Curve
Divides the cut out by the Extrusion Widths
Divider
Evaluate Length
List Item
End Points
Line
Trim Tx
Merge
Join Curves
Curves

(2:1)(5)

Hole Punch
Grasshopper
definition



{3;0} Examples

Voodoo Face Shield

During the beginning of the Covid-19 pandemic in 2020 when protective gear like face shields were in high demand and in short supply, I used my pre-Ouroboros workflow to help out.

Prusa Research published 3D-models of a face shield that became the de facto standard, with many compatible variants from the community. It initially took about 3 hours to print but there were several “slim versions” with reduced print times. With a 1 mm nozzle and 0.5 mm layer height I could print in 18 minutes what others printed in 1½ hours.

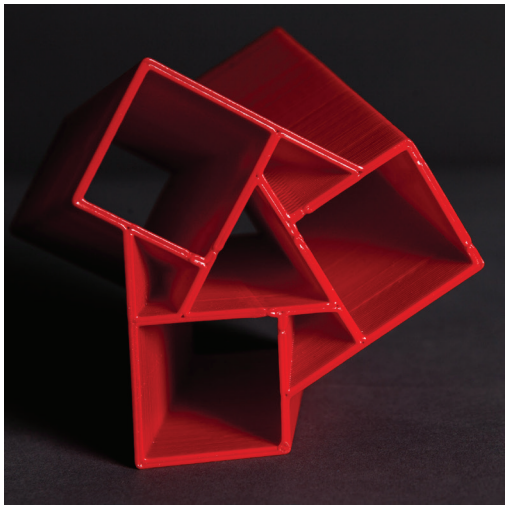
I printed about 500 pcs. of my Voodoo Face Shields and donated them all to Maker vs Virus who took care of distribution.

Garberobe

The Garberobe was the first project I implemented using this technique.

The name is a mix of Garderobe, the German word for coat rack, and Garbe meaning bundle of hay.

One set of parts is printed in about four hours which is quite fast for 300 grams of material.

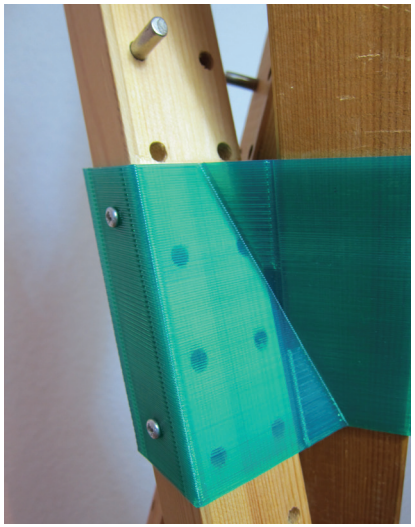


Photos by Andreas L. Berg

IVARobe

I wanted to sell the Garberobe as a kit. However, including the poles would be too bulky for shipping and letting people source the sticks themselves was tricky as well. Therefore I've build a parametric version of it to be able to print a Garberobe that fits to any old sticks that my customers find in their barn.

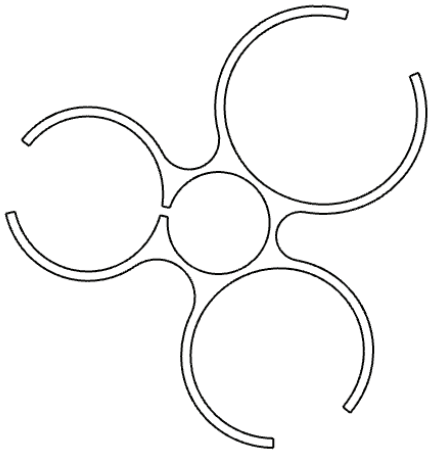
As an example and as a homage to IKEA I made one out of old IVAR shelf parts. The pegs can be used as extra hooks.



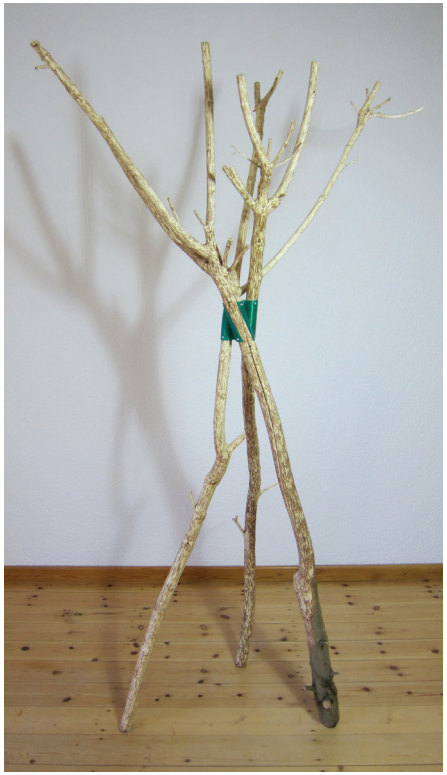
Tree Branch Coatrack

I also made a variant for tree branches.

The geometry is quite different and while making it I realized that **Stitch With Brep** would not work so well in this case. That is why I created **Stitch With Points**.



The stitches are made in alternating sequence between the inner circle and the clamps.

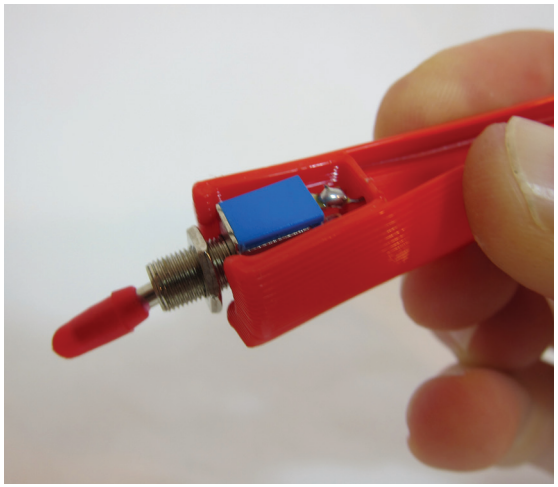


Electronic Candle

This is a minimalistic electronic candle I made for Christmas in a small series of about 30 pieces. In addition to the printed parts it only consists of a warm-white flickering LED, a Switch and a wire.

And the batteries, of course. They are snugly held by the side walls but pop out when pushed.

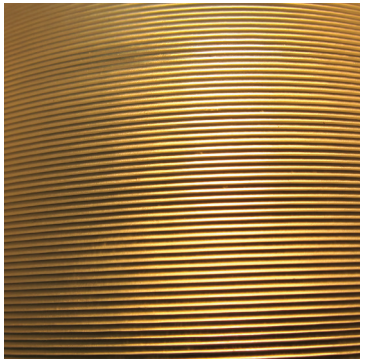
One leg of the LED is stuck through a hole and bent over a kink on the inside to ensure good battery contact. The other one is attached to a wire going down to the switch on the inside.



Lampshade

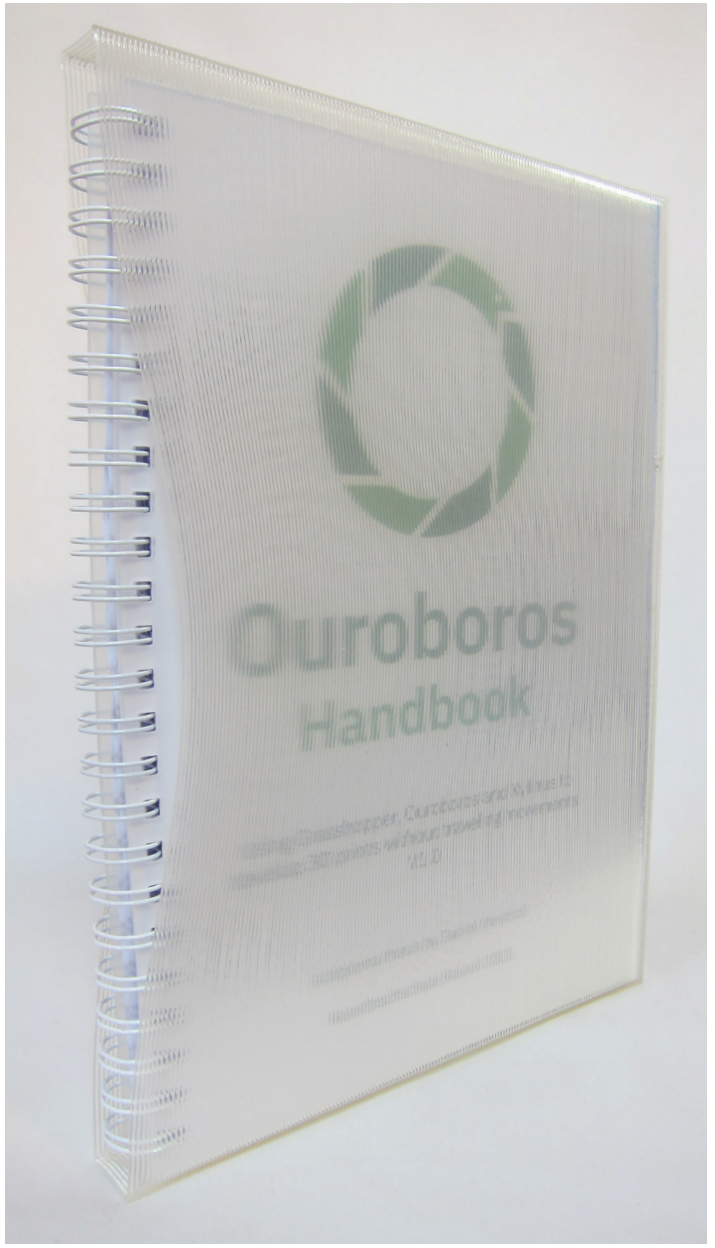
This was basically a test to see what happens when you combine a stitched, planar section with a spiralized top.

The light is surprisingly diffuse and it does that trippy thing where it slips into the column closest to you while leaving the rest dark.



Slipcase

To give the reader a proper impression of the look and feel of the unique texture, I made slipcases for the printed version of the Ouroboros Handbook.



{4;0} Xylinus Changelog

While working with Xylinus I stumbled upon some missing features. But because it consists of user objects, the inner workings are always just a double click away.

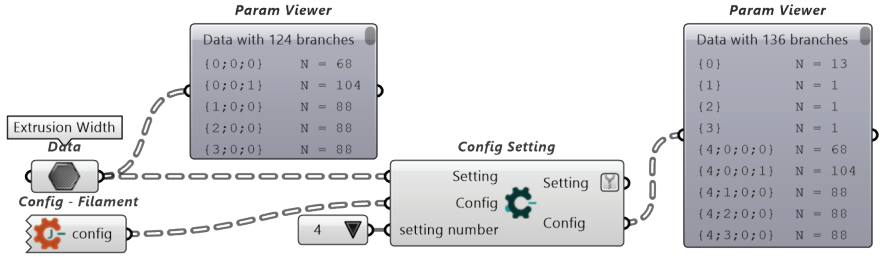
After hacking a few workarounds for very specific tasks I worked on generalizing my modifications up to a point where I could submit them to the Xylinus project where they will be part of the next release.

Like before, I will tell you about the components first and show you the Grasshopper descriptions later in the fold-outs.
For the sake of transparency I will include also the originals by Ryan Hoover.

Config Setting

The old **Config System** was designed to only have one value for each setting but was able to contain lists as well. However, I wanted to control, for example, the **Extrusion Width** for each point on each layer. That is why I wanted the settings to be able to work with whole data trees.

My solution is to “wrap” the tree inside the setting ID by prefixing the path with it like shown below.



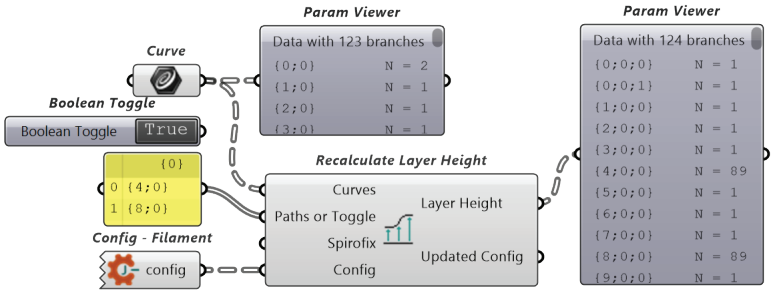
Simplify (▼) at the **Settings** output gives you the original data tree without the ID and is there by default.

Having that much control over the settings makes possible what, before, required modding of the **Print Curve** component with special inputs and custom logic, thus loosing a lot of modularity. The only thing that had to be changed in **Print Curve** is the **Config Settings** component, and everything can happen outside it.

A new class of Xylinus components has become possible.

Recalculate Layer Height

Recalculate Layer Height (RLH) is the first in a planned series of components that shift the paradigm of what the config system does. It’s no longer just a bunch of settings you define in the beginning and that get distributed to all the components that need it, but it’s itself subject to change and development.

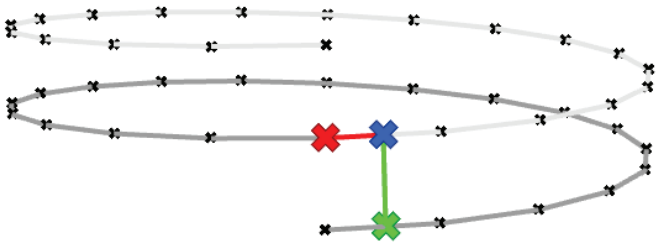


RLH measures the Z distance between the mid-point of each segment to the next curve one layer down and outputs an **Updated Config**.

To achieve this **RLH** first has to anticipate what the **Print Curve** component will do, by converting the curves to poly lines with the same tolerance. The resulting tree structure matches the one expected by the **Print Curve** component. That is why, in the example above, the paths are grafted.

The **Path or Toggle** input can be used to process specific layers only, while the rest is padded out with the standard value. Or to turn the recalculation off while testing things, because it is really computing intensive and in most cases only needed for G-code generation.

The **Spirofix** input is a workaround for working with spirals. Depending on the tolerance setting and other factors, the end of the lower curve might be closer to the first point on the upper one than to the start. To mitigate that issue it replaces the first n values with the value of n+1.

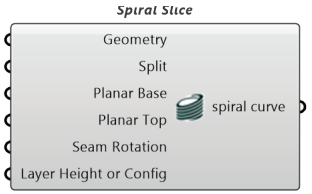


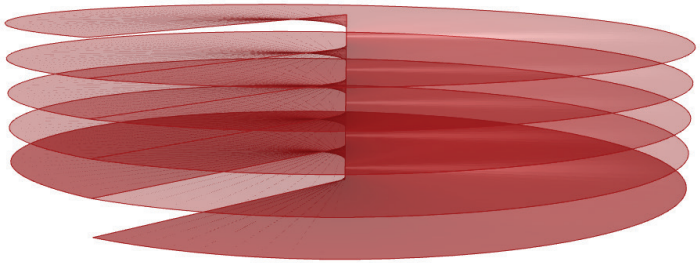
blue: point from where the distance is measured
red: point that RLH determined as the closest
green: actually desired point

RLH is essential for working more creative with the Z axis, for vertical wave patterns, the ramps of a starting spiral etc.

Spiral Slice

The previous Xylinus release introduced a **Spiral Slice** component that was very slow. Because execution time and memory usage grew exponentially, it was basically unusable for even slightly bigger parts.





The culprit was, that it used a continuous spiral to cut the whole object in one go while my version uses a small spiral for each layer.

This allows parallel computation and is much lighter on memory usage. I had performance boosts of multiple orders of magnitude, depending on the size of the part.

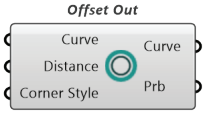
This also made the implementation of multiple values for **Layer Height** possible, a feature Ryan Hoover also integrated into the normal **Slice** component.

You can select if a **Planar Top** and **Planar Base** are generated with the corresponding inputs. The base replaces the first layer while the top is just merged in and grafted. This makes it a separate layer that will be recognized as such by the **RLH** component. The **Planar Base** is important for the first layer because the nozzle might crash into the print bed when starting the spiral and might not adhere to the bed at all at the end. The **Planar Top** is just nicer when it is flat, and is also easier to interface with other curves that might be on top of it.

By deactivating **Split** you can join all the curves into one big spiral, but be aware, that you will loose compatibility with **RLH**.

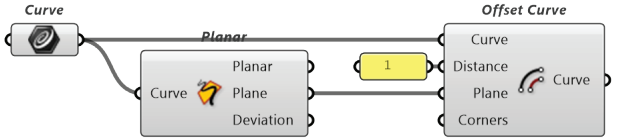
Offset Out

Offset out is one of the core components of Xylinus. It was always a bit unreliable and offsets some curves into the wrong direction and sometimes even gave no output at all.



It worked by comparing the area of the offset of x and x*-1 and chose the bigger one. That means that for each curve two offsets have to be calculated. **Offset Curve** is a slow component to begin with, so just getting around needing two would double the speed.

Looking online I found this solution and had no problems with curves offsetting in the wrong direction since.

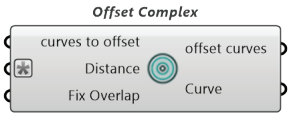


Solution found at:
https://www.grasshopper3d.com/forum/topics/offset-inwards
Recommended by user ng5 Alex

The **Probably Problematic Curves (Prb)** output holds the original input curves, whose offset resulted in a curve that was less than 50% of the original length, except when it is a small curve to begin with. This helps with workarounds to automate the handling of failing offsets. This is the part I kept unchanged from the previous version.

Offset Complex

Offset Complex offsets outer curves to the inside and curves that are contained in another curve to the outside.

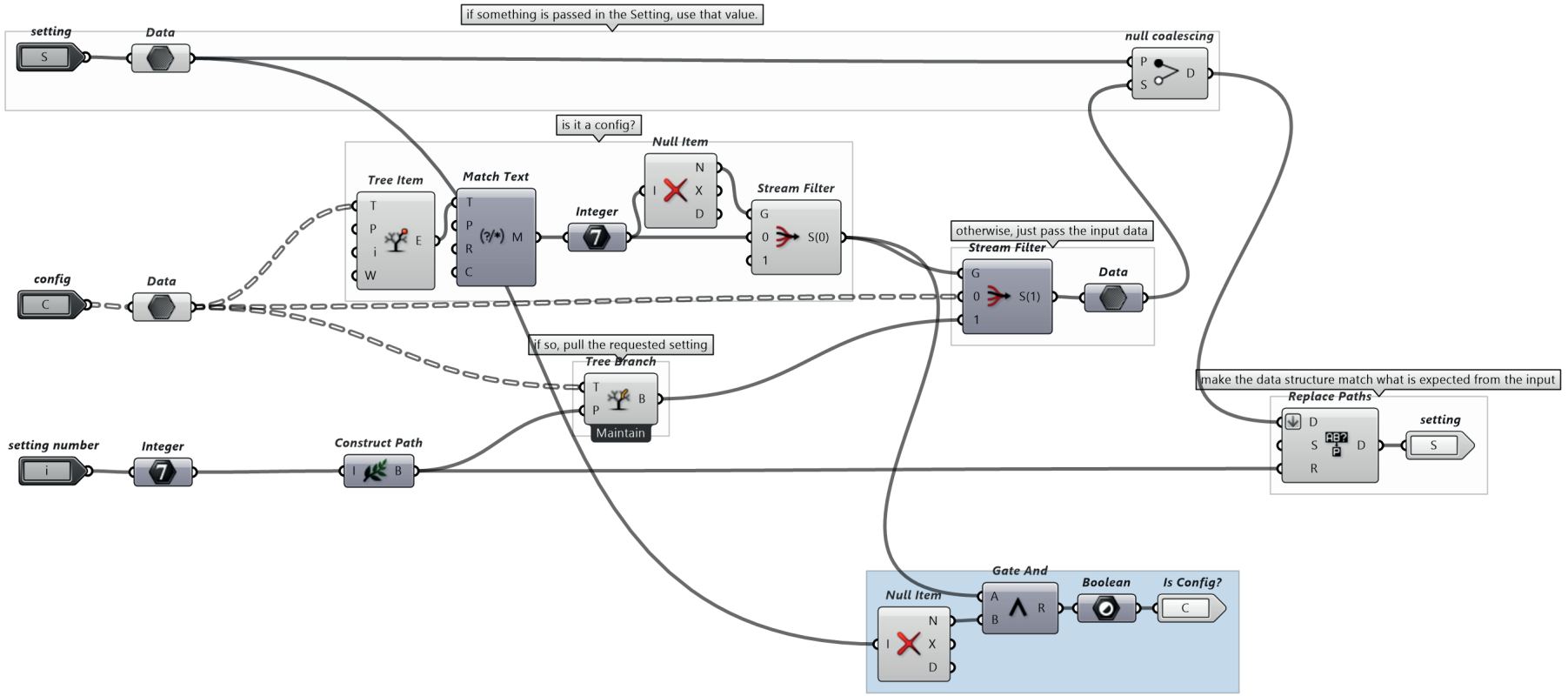


It contains three instances of **Offset Out** which means that my version of it should have made a huge impact on its speed.

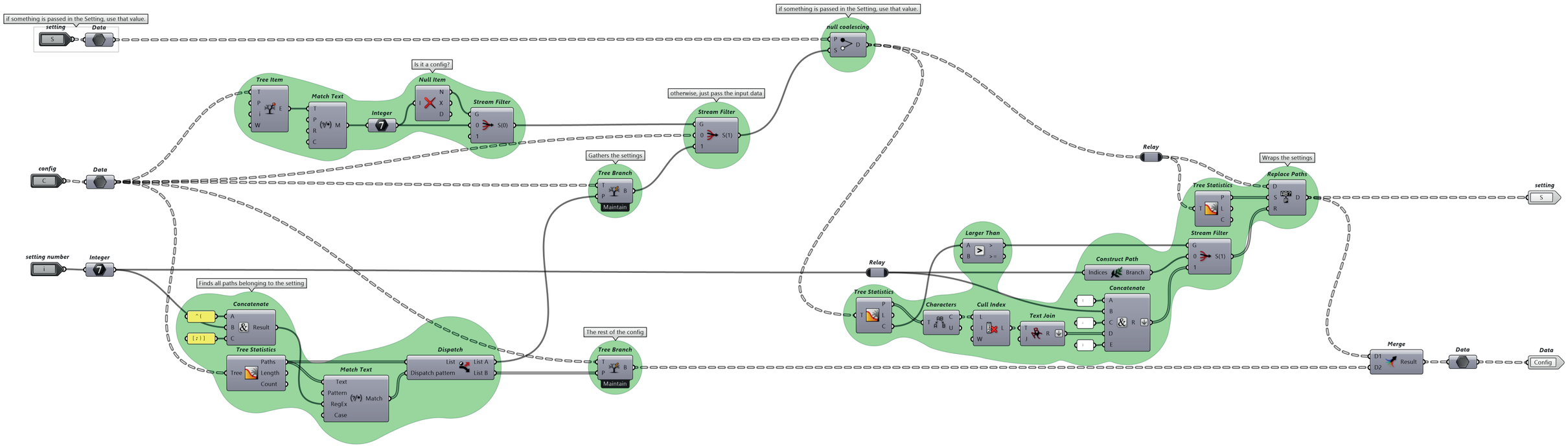
But there was another thing holding it down. In case the offsets lead to overlapping curve, the outer curves go through a **Region Union** component and get the inner curves cut out with a **Region Difference**.

This takes a lot of time and is not needed in many cases. When used with Ouroboros having overlapping curves is a clear sign that something went very wrong and to fix the overlap in that way doesn't help.

Therefore, making it optional with a **Fix Overlap** input resulted in a tenfold increase in speed.


if something is passed in the Setting, use that value.
setting
S
Data
null coalescing
P
S
D
is it a config?
Null Item
N
X
D
Tree Item
T
P
i
W
E
Match Text
T
P
R
C
(?/*)
M
Integer
7
Stream Filter
G
0
1
S(0)
config
C
Data
otherwise, just pass the input data
Stream Filter
G
0
1
S(1)
Data
if so, pull the requested setting
Tree Branch
T
P
B
Maintain
make the data structure match what is expected from the input
Replace Paths
D
S
R
D
setting
S
setting number
i
Integer
7
Construct Path
I
B
Gate And
A
B
R
Boolean
Is Config?
C
Null Item
N
X
D

Config Setting [modified] - Grasshopper definition

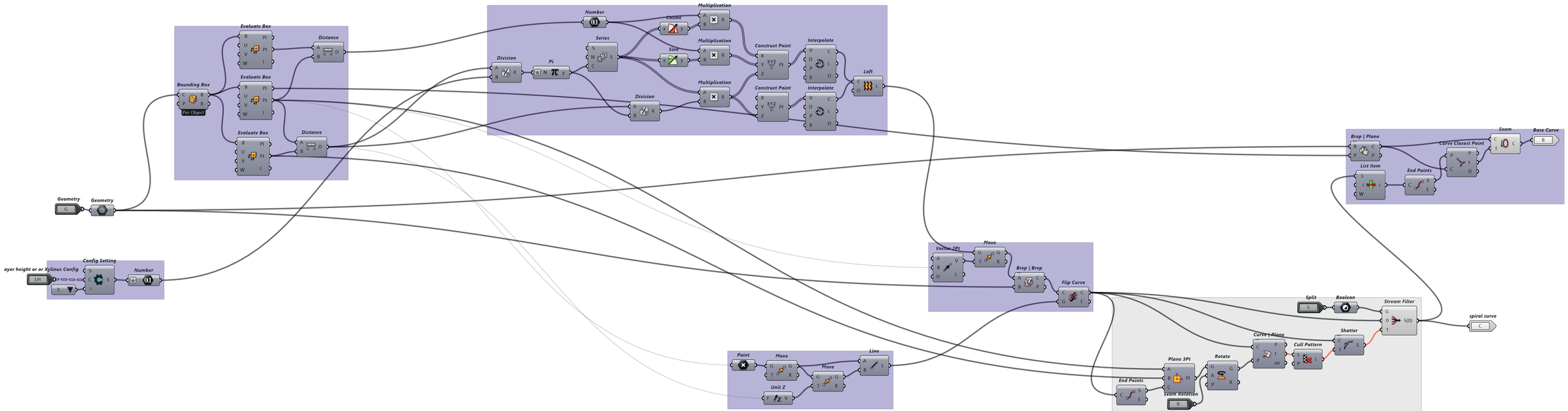


{4;0}(7)

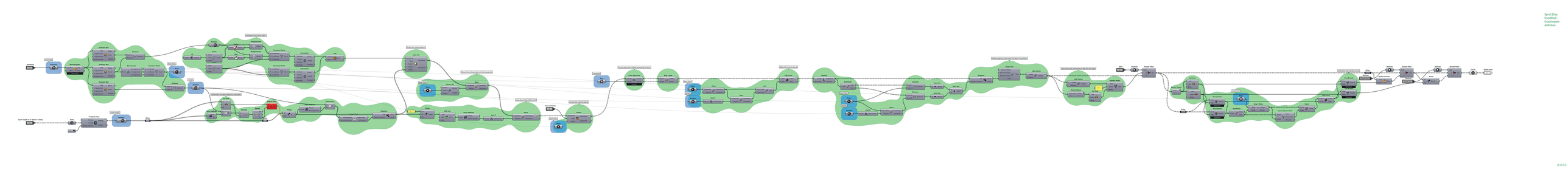
Recalculate
Layer Height
Grasshopper
definition

{4;0}(9)

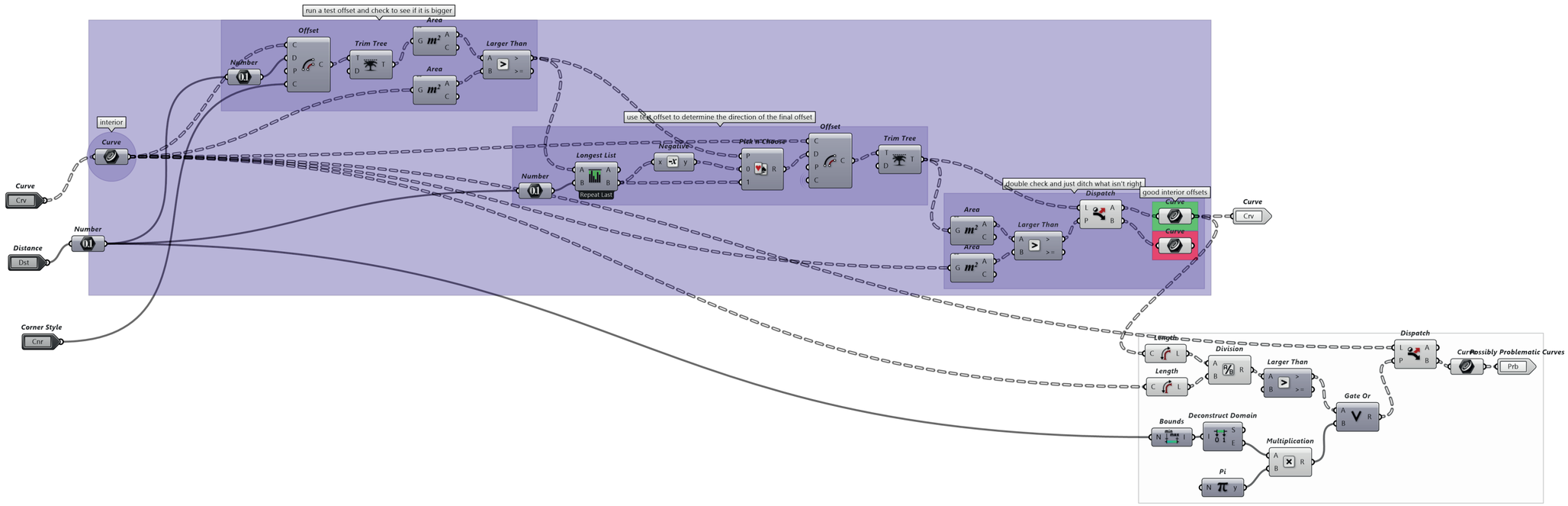
Spiral Slice [original] - Grasshopper definition



{4;0}(11)


Spiral Slice (modified) - Grasshopper definition

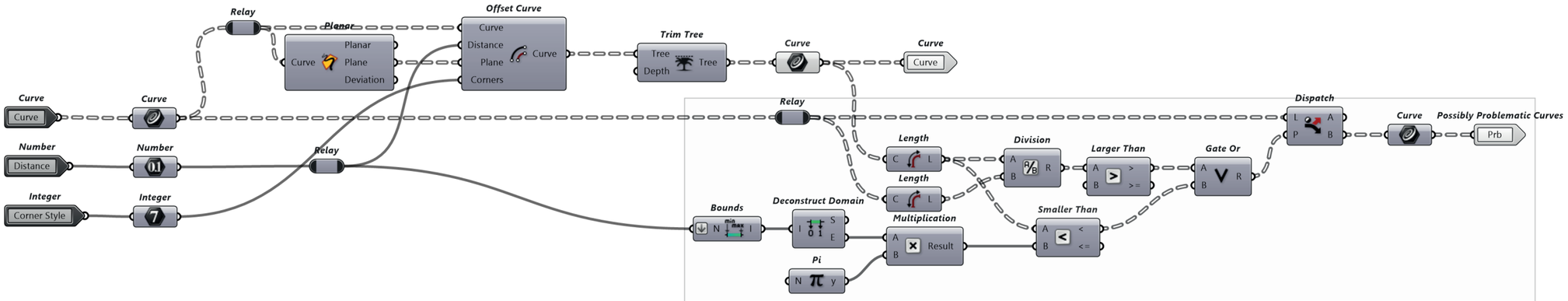
Offset Out [original] - Grasshopper definition



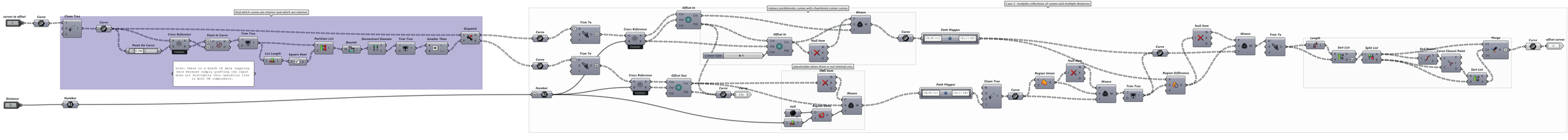
{4;0}(15)

Offset Out [modified] - Grasshopper definition

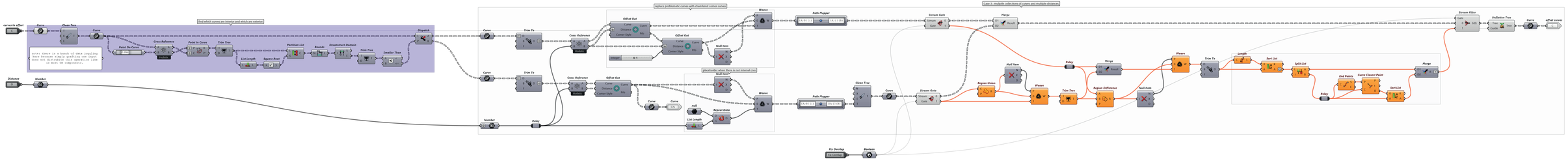
I used this solution as a guide to find the right plane.
https://www.grasshopper3d.com/forum/topics/offset-inwards



Offset Complex [original] - Grasshopper definition



Offset Complex [modified] - Grasshopper definition



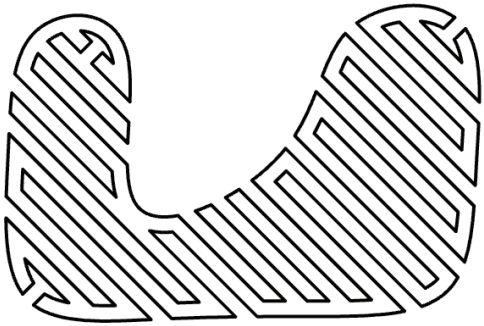
Work in progress {5;0}

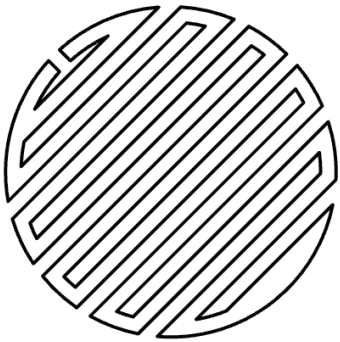
There are two components that didn't quite make it into this release, and I have also some ideas for the future.

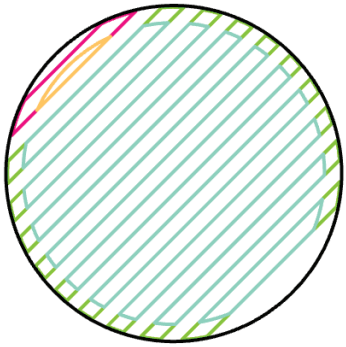
Continuous Infill

The most important feature Ouroboros is still missing is the printing of areas. Again, the goal is to end up with only one closed curve per layer.

This problem is surprisingly tricky to solve but I'm on my way to a promising approach. The curves below are no mock-ups but generated by a prototype.







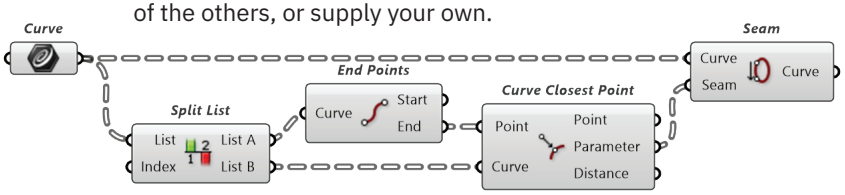
The width of the turquoise stripes depends on the desired **Infill Density.** They are connected to the outline (black) by one of its green

neighbors. The red region was sorted out because it touches the outline in more than one place. Thus, the orange Stripe is orphaned which gets fixed with the help of **Stitch with Breps** automatically.

There are many special cases that need to be considered, and I haven't figured it out completely yet. However, I am confident that **Continuous Infill** will soon be a versatile and robust tool.

Z-Seam Alignment

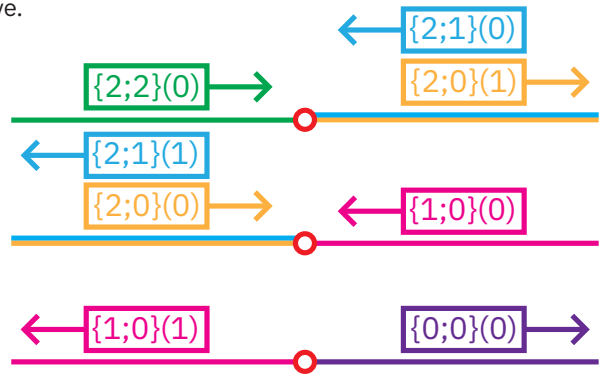
Aligning the seam point of closed curves is easy. You can either use the seam point of the first curve to find the new seam points of the others, or supply your own.



If you have some open curves in the mix, it becomes more complicated. That is why I built the **Punch Hole** component in the first place. However this results in travel movements in the place where it was punched. That is okay for small holes for screws but not for big openings where the quality of the edge has to look nice and smooth.

The Z-seam has to be in a more or less vertical arrangement and, it's not pretty, so aligning it with the opening is not desirable. You want to be able to hide it somewhere discrete.

I have a solution for that, at least for layers with only one open curve.



The curve is cut at the seam point (red circle in our example above), and one side is pushed one branch up and becomes so part of the next layer. Then everything is sorted and flipped so that it zigzags diagonally.

If there is an odd amount of consecutive open layers, the second to last layer (orange) has to retrace its path (blue). This is similar to the combing option in other slicing software and at least better than chaotic travelling.

It is important that the back trace gets its own branch because for that print paths the layer height has to be recalculated. For that purpose it would be handy to have an output that collects all these data paths to send it on to a **RLH** component.

The first data path segment of the last curve segment (green) has to match its original. It could otherwise collide with eventual layers above it. If our example only had two layers, the first segment of the orange curve would be the last one. It would then be called {1;1}(0).

Outlook

I will continue the development of Ouroboros and will make more contributions to the Xylinus project.

One major goal for the future is transferring my techniques to clay printers or syringe printers in general.

Ouroboros is a great tool for realizing my dream of designing and manufacturing my own products.

{6;0} Table of contents