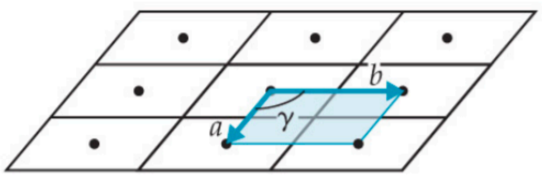
Computational Fabrication

CS 491 and 591
Professor: Leah Buechley
https://handandmachine.org/classes/computational_fabrication

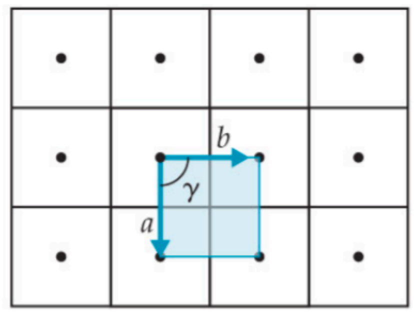
Daily Artist: Piotr Waśniowski

https://www.instagram.com/piotr_wasniowski/

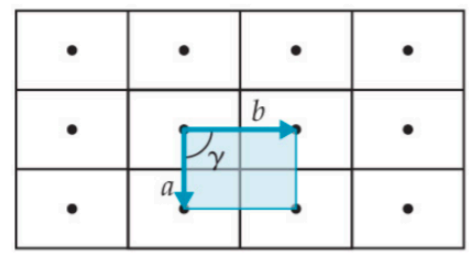
5 2D Bravais Lattice Structures



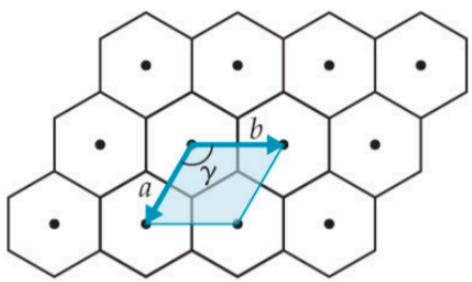
Oblique lattice ($a \neq b$, γ = arbitrary)



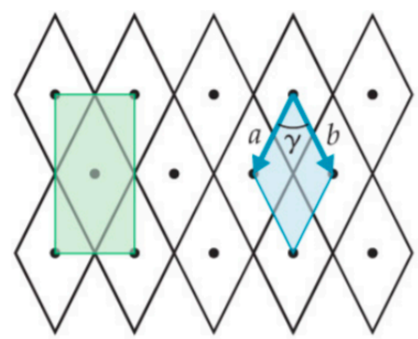
Square lattice ($a = b$, $\gamma = 90°$)



Rectangular lattice ($a \neq b$, $\gamma = 90°$)



Hexagonal lattice ($a = b$, $\gamma = 120°$)

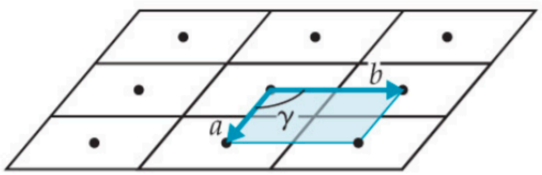


Rhombic lattice ($a = b$, γ = arbitrary)
Centered rectangular lattice

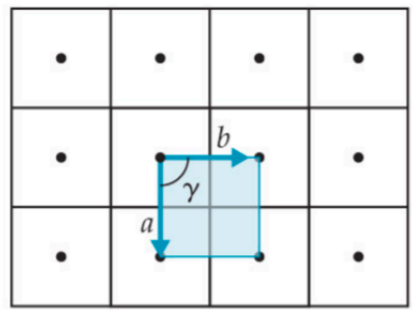
Bravais Lattice Structures

Any periodic 2D tiling maps to one of these 5 fundamental lattice structures.

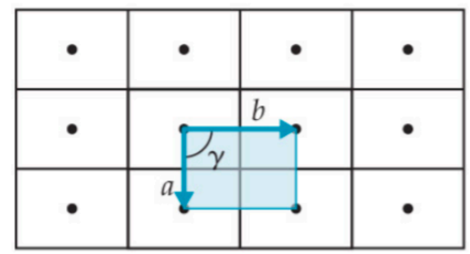
5 2D Bravais Lattice Structures



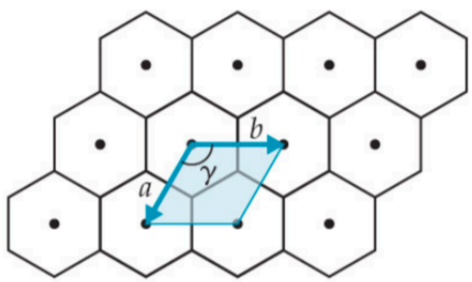
Oblique lattice ($a \neq b$, γ = arbitrary)



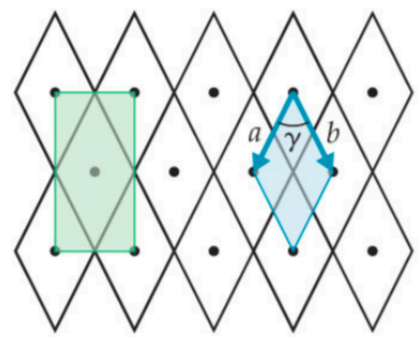
Square lattice ($a = b$, $\gamma = 90°$)



Rectangular lattice ($a \neq b$, $\gamma = 90°$)



Hexagonal lattice ($a = b$, $\gamma = 120°$)

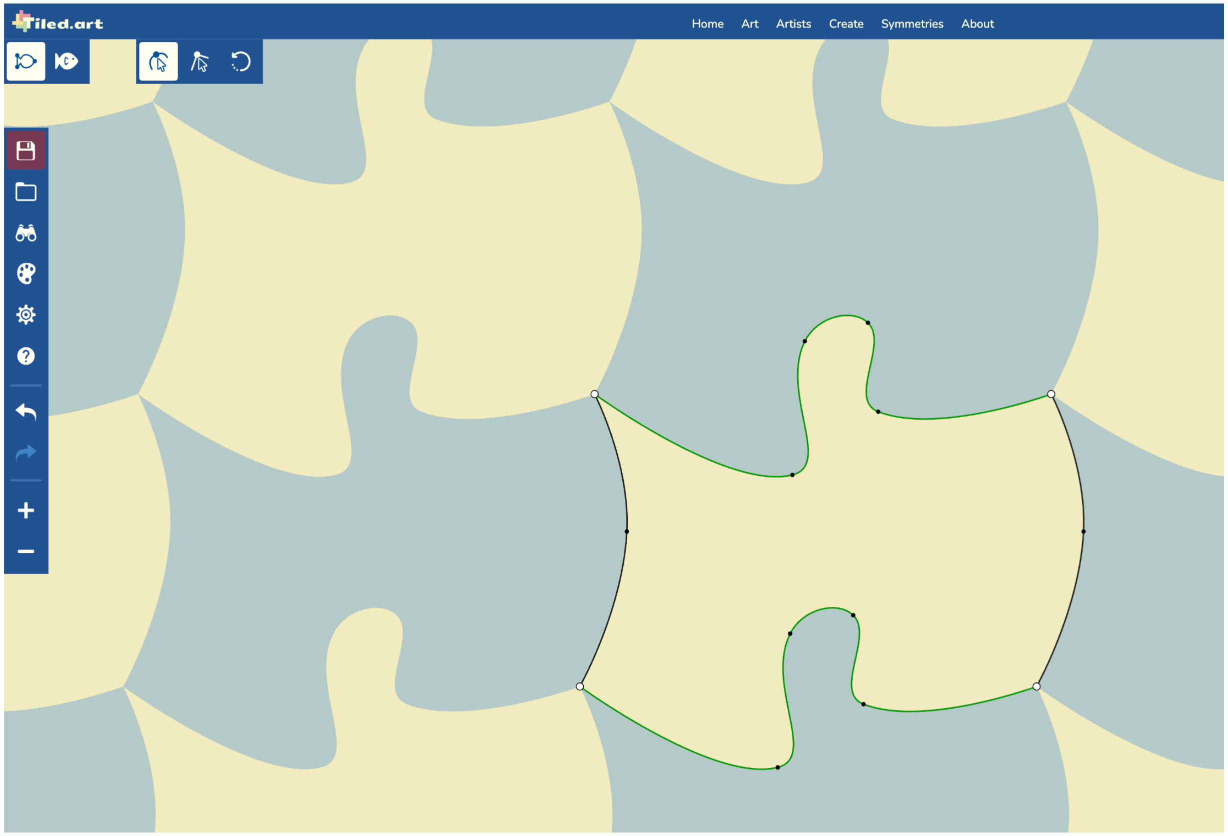


Rhombic lattice ($a = b$, γ = arbitrary)
Centered rectangular lattice

What we'll do today

1. Write code to generate these 2D lattices, illuminating some fundamental tiling geometry
2. Use our lattice generating code to generate 2D tiles and tilings

Build in GH Rhino



https://tiled.art/en/create/?id=Quad1

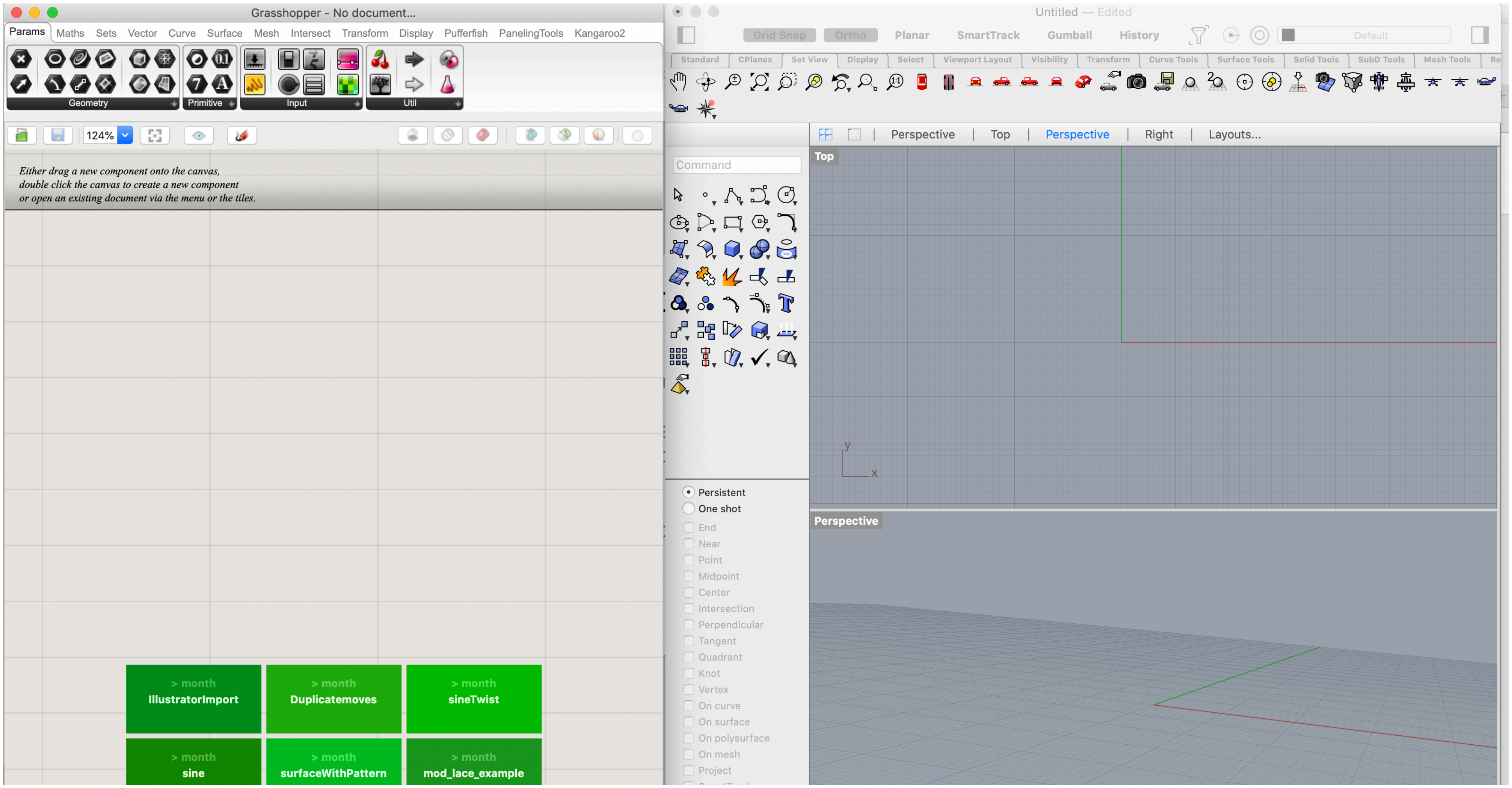
How we’ll do it

1. Generate lattice & basic tiling
    a) python block 1:
    input: a (length), b (length), and angle
    outputs: joined ab curve + list of a and b vectors
    b) python block 2, outputs:
    lattice: translate ab curve using vectors
    tiling: a closed curve (tile) for each lattice cell. outputs is list of tiles
2. Generate Escher tiling
    a) add complex (Escher) a, b line inputs to python block 1. scale and rotate these complex curves to map to a and b vectors. add complex (Escher) ab curve output to python block 1.
    b) Use new output as input to python block 2

FYI & proceed w/ caution

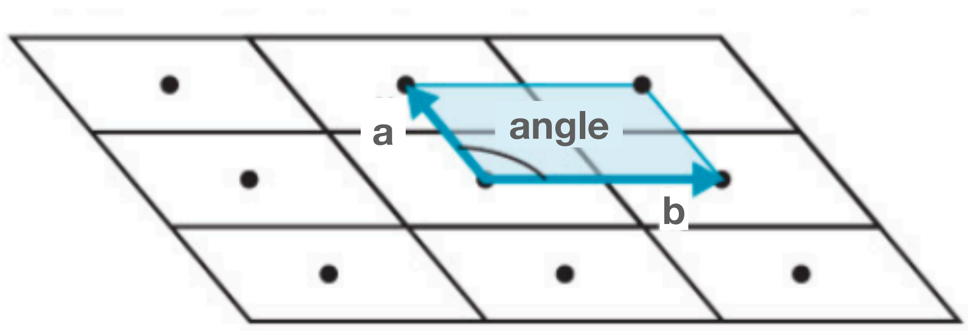
- We'll be navigating some awkward data representation issues to get python data structures to be accessible & visualizable in GH & Rhino. No 2D lists/arrays in GH!!
- Also navigating some gaps in rhinoscript implementation for python 3. In particular: no implementation of python copy and deepcopy yet.
- So, code is a bit awkward. If you get confusing compile errors or if you can't see your geometry in Rhino, refer back to these slides and check details carefully.

open up Rhino and Grasshopper



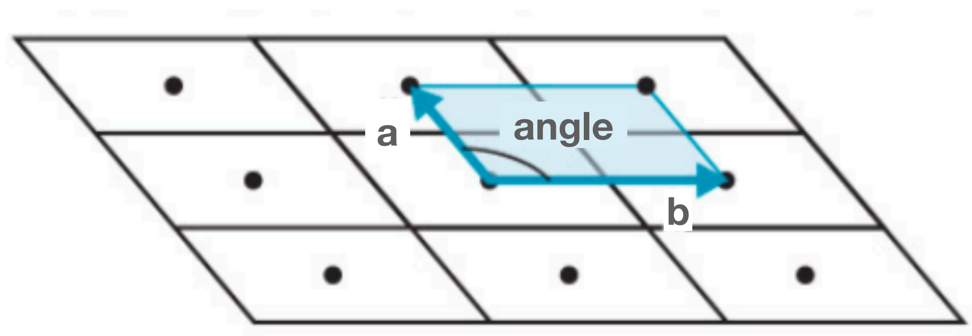
One lattice cell

Parametric lattice: 3 simple variables



- a
- b
- γ (angle)

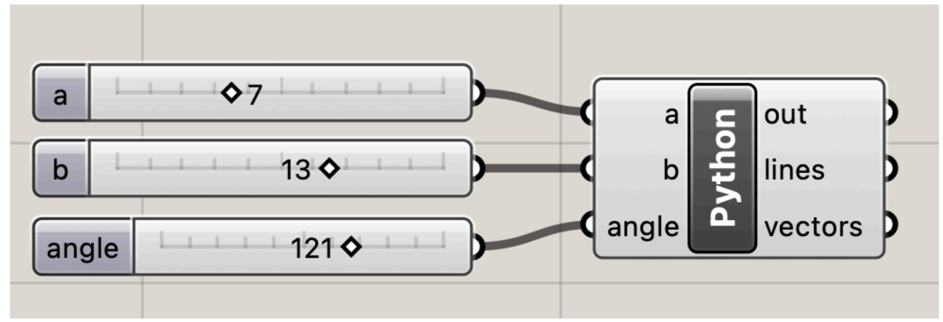
Grasshopper & Python



- Inputs:
  - a length, b length
  - angle
- Outputs:
  - joined ab curve
  - list of a and b vectors

Grasshopper & Python Code

- Inputs:
  - a length, b length
  - angle

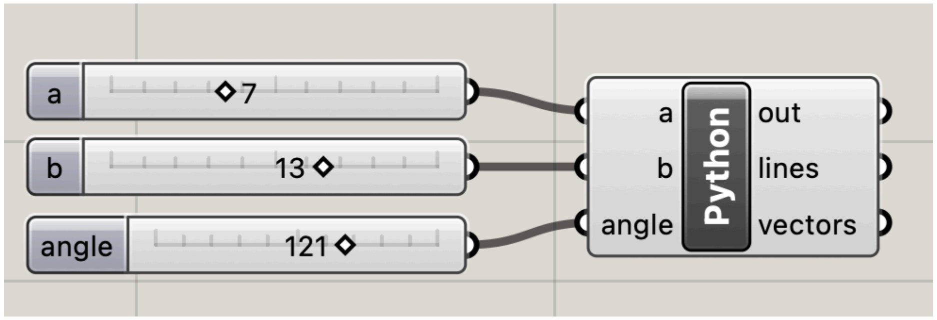


input: **Float** Type hints

- Outputs:
  - joined ab curve, "lines"
  - list of a and b vectors

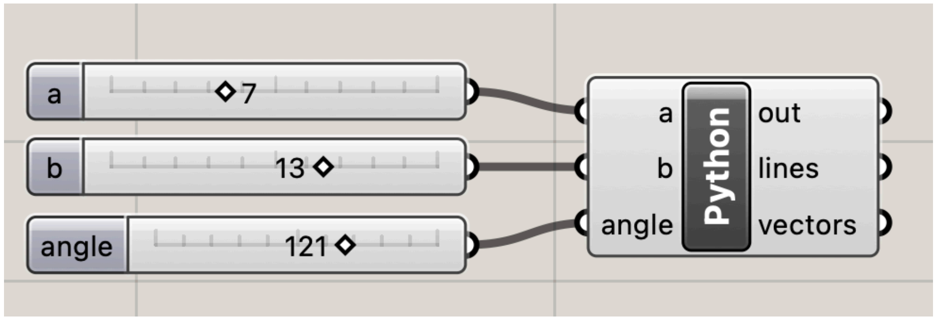
Grasshopper & Python Code

```
import rhinoscriptsyntax as rs
import math

#calculate x y points for a vector
ax = a*math.cos(math.radians(angle))
ay = a*math.sin(math.radians(angle))

# create a and b vectors
origin = [0,0,0]
a_vector = rs.CreateVector([ax,ay,0])
b_vector = rs.CreateVector([b,0,0])
vectors = [a_vector, b_vector]

# create lines so that the direction is
# counterclockwise around future tile
b_line = rs.AddLine(b_vector,origin)
a_line = rs.AddLine(origin,a_vector)
lines = rs.JoinCurves([b_line, a_line])
```



Grasshopper & Python Code

```
import rhinoscriptsyntax as rs
import math

#calculate x y points for a vector
ax = a*math.cos(math.radians(angle))
ay = a*math.sin(math.radians(angle))

# create a and b vectors
origin = [0,0,0]
a_vector = rs.CreateVector([ax,ay,0])
b_vector = rs.CreateVector([b,0,0])
vectors = [a_vector, b_vector]

# create lines so that the direction is
# counterclockwise around future tile
b_line = rs.AddLine(b_vector,origin)
a_line = rs.AddLine(origin,a_vector)
lines = rs.JoinCurves([b_line, a_line])
```

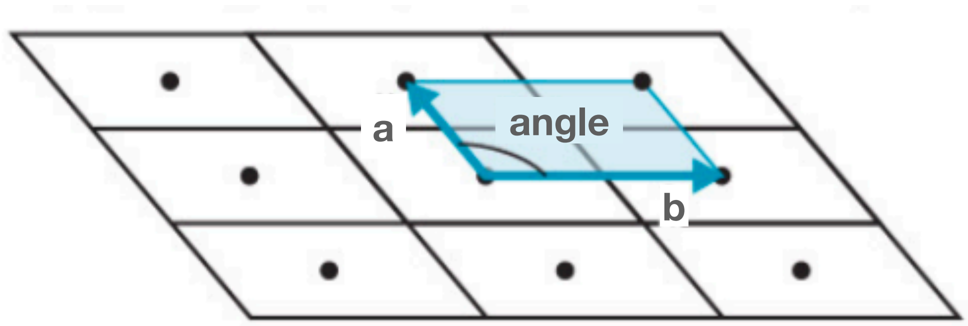


questions?

Generating the Lattice

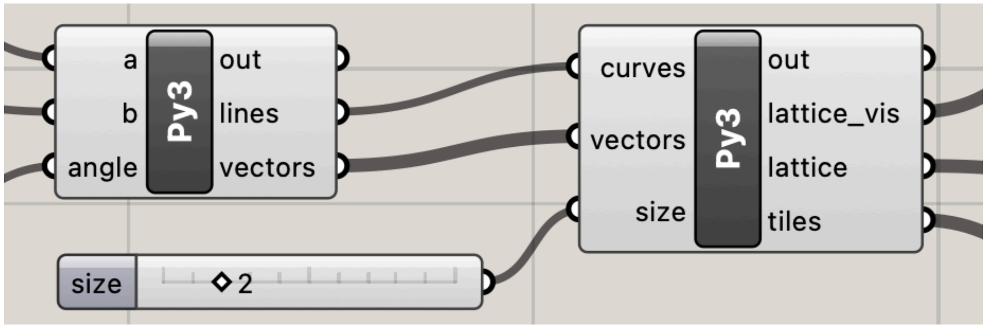
Copy and translate cell using vectors

- Inputs:
  - lines
  - vectors
  - size of lattice
- Output:
  - 2D lattice
  as list of tiles
  tile = closed curve

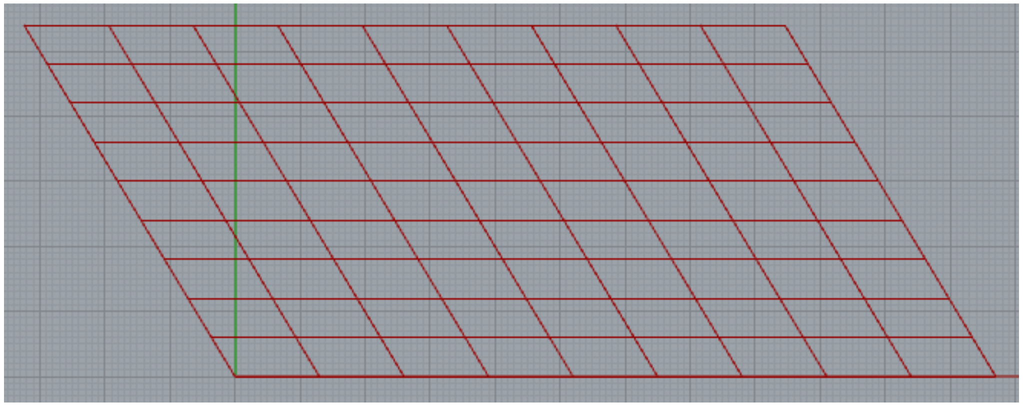


New Python Block

- Inputs:
  - curves
  - vectors
  - size of (square) lattice
- Output:
  - lattice (2D list)
  - lattice visualization (1D list)
  - tiles (1D list)
    tile = closed curve

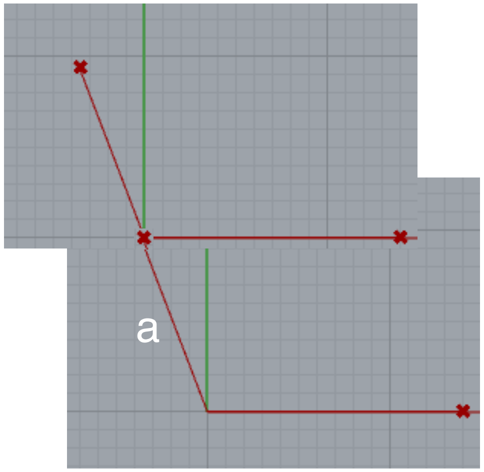


curves: **Curve** type hint
vectors: **Vector** type hint, list access
size: **int** type hint



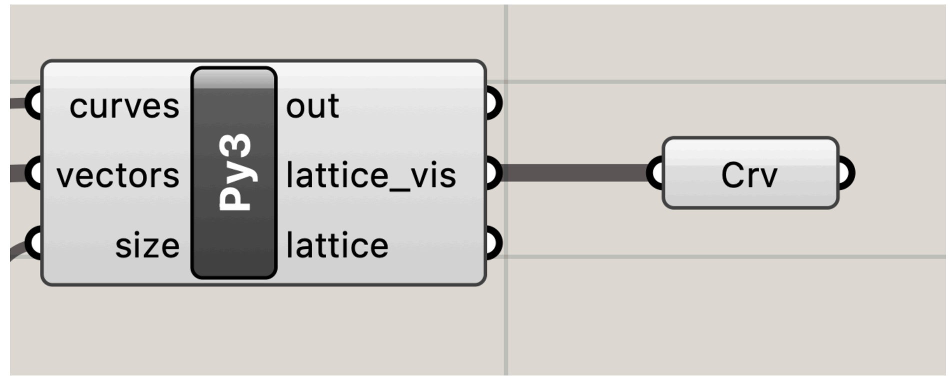
To generate 2D Lattice

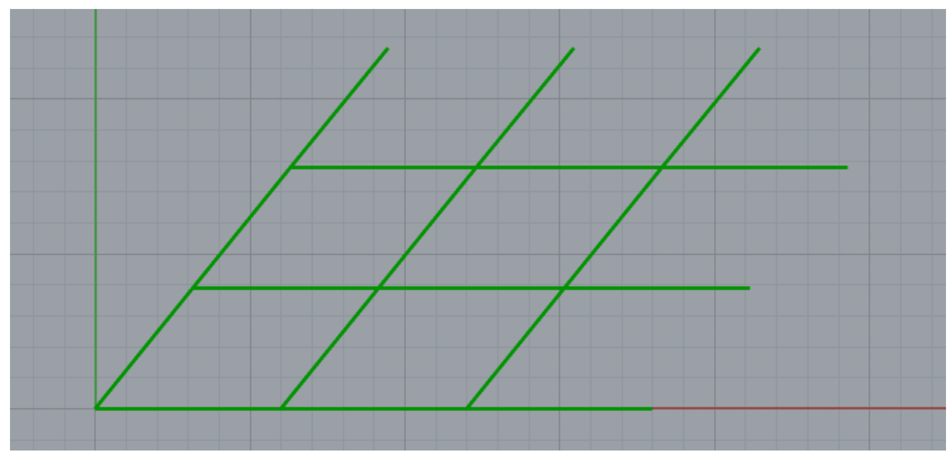
- Copy input curves and translate along **a** and **b** vectors
- Use **geom.Curve.Duplicate** to copy
- Use **rs.MoveObject()** to translate



To generate 2D Lattice

```
import rhinoscriptsyntax as rs
import Rhino.Geometry as geom
import math

# generate lattice
# copy input curves
# move them in 2D, using input vectors,
# to generate lattice
lattice = []
lattice_vis = []
for i in range (size+1):
    row = []
    for j in range (size+1):
        # copy curves
        new_curves = geom.Curve.Duplicate(curves)
        rs.MoveObject(new_curves,vectors[0]*i)
        rs.MoveObject(new_curves,vectors[1]*j)
        lattice_vis.append(new_curves)
        row.append(new_curves)
    lattice.append(row)
```

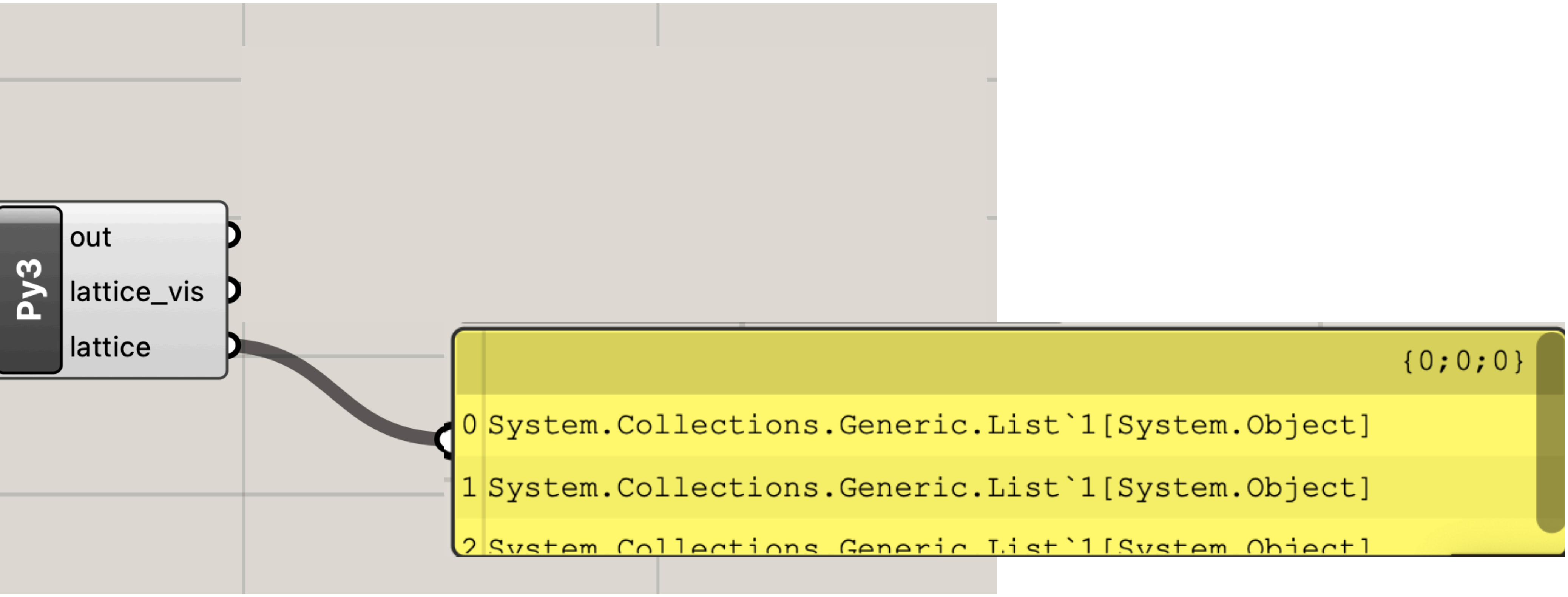




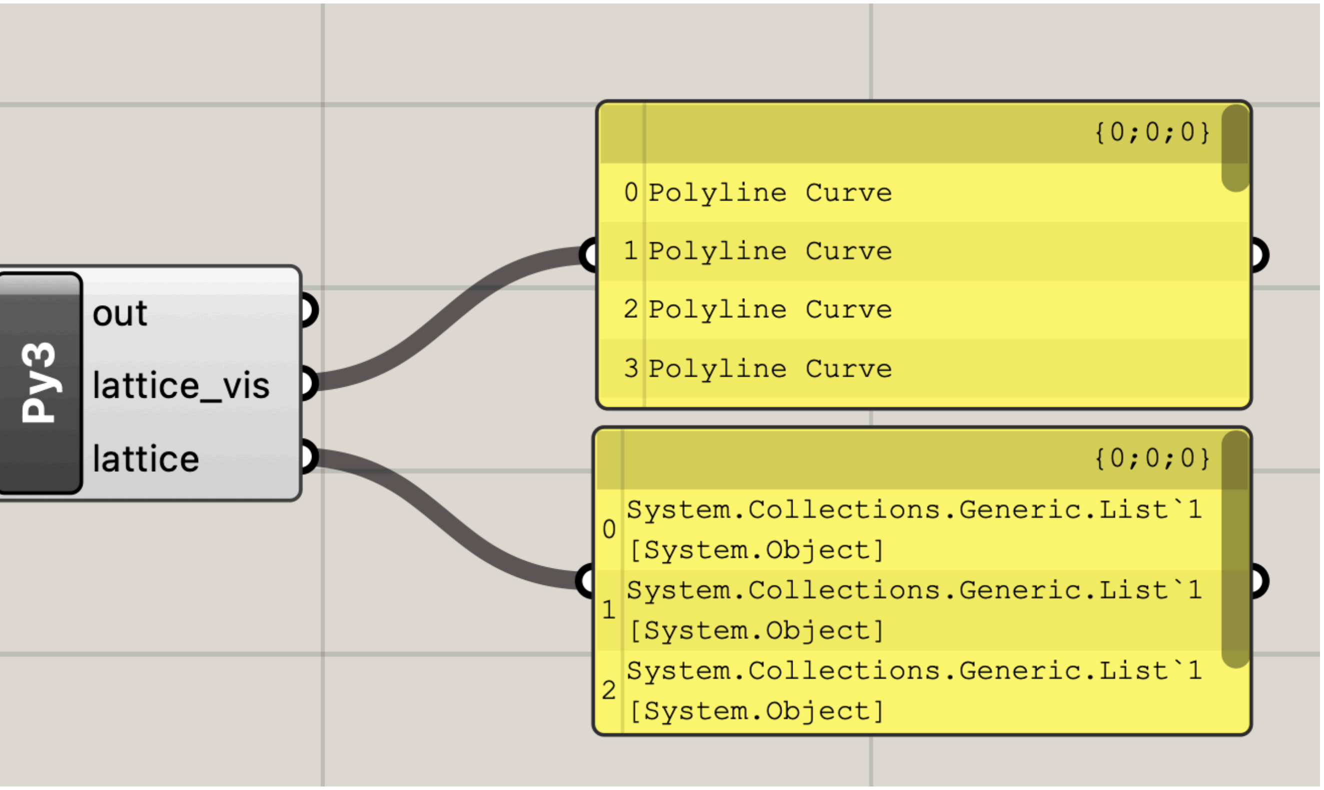
Grasshopper & Python Data Structures

- Python: lists of any dimension
- Grasshopper: 1D lists and trees only
  - Can't manipulate data from 2+D structures
  - Can't render/visualize data from 2+D structures
- Some rhinoscript geometry/GH data structures are actually (secret) lists. (ie: joined curves, polylines) So, you have to nagivate them carefully too.

2D Lattice Output: Can’t visualize

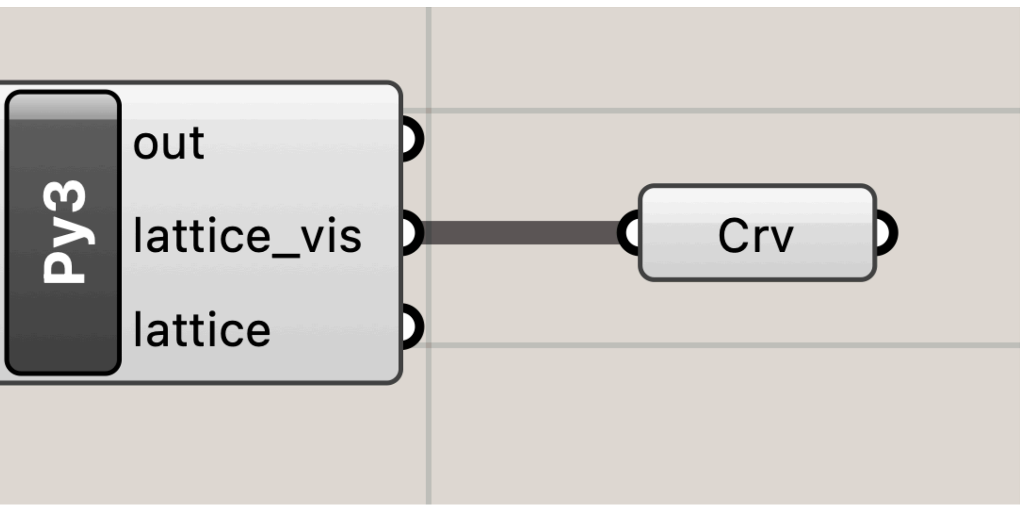


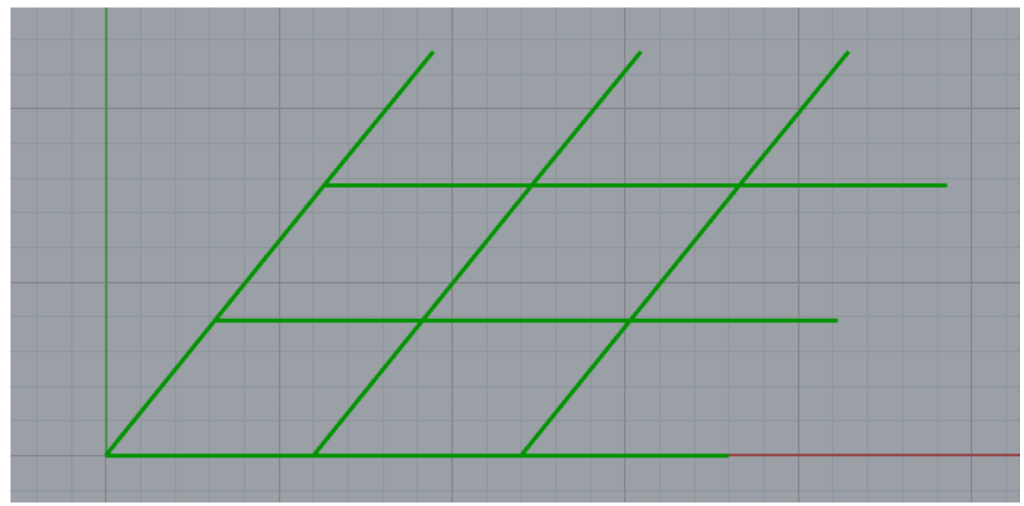
2D Lattice Output: Can't visualize



- GH doesn't work with 2D lists
- That's why we need 1D lattice_vis

Connect Curve Block to lattice_vis





questions?

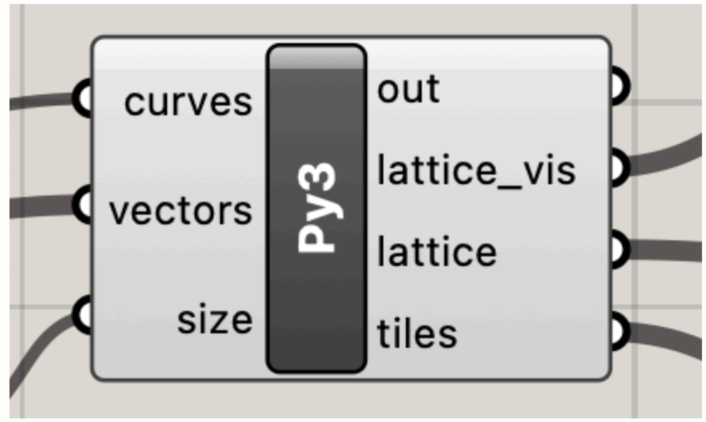
Lattice —> Tiles

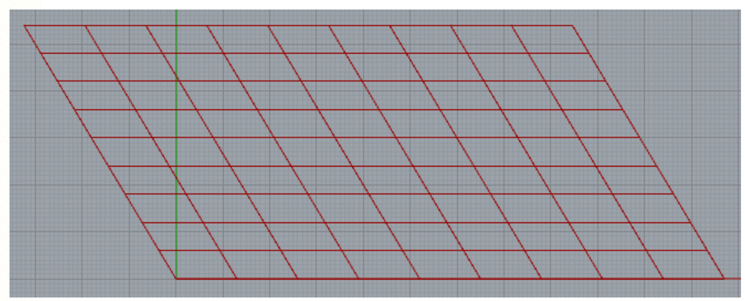
2D List of Open Curves —> 1D List of Closed Curves

- Two tasks:
  1. Generate Tiles (Closed Curves) from lattice
  2. Generate 1D List of tiles as output

Find Tile Edges & Generate Tile

```
# generate tiling
# find the edge curves for each lattice cell
# generate a closed tile shape
# add to list of tiles
tiles = []
for i in range(0,size):
    for j in range(0,size):
        bottom_left = lattice[i][j]
        top = rs.ExplodeCurves(lattice[i+1][j])[0]
        right = rs.ExplodeCurves(lattice[i][j+1])[1]
        tile = rs.JoinCurves([bottom_left, top, right])
        if (rs.CloseCurve(tile)):
            tile = rs.CloseCurve(tile)
        else:
            print("can't close tile curve")
        tiles= tiles+tile
```





Find Tile Edges & Generate Tile

```
# generate tiling
# find the edge curves for each lattice cell
# generate a closed tile shape
# add to list of tiles
tiles = []
for i in range(0,size):
    for j in range(0,size):
        bottom_left = lattice[i][j]
        top = rs.ExplodeCurves(lattice[i+1][j])[0]
        right = rs.ExplodeCurves(lattice[i][j+1])[1]
        tile = rs.JoinCurves([bottom_left, top, right])
        if (rs.CloseCurve(tile)):
            tile = rs.CloseCurve(tile)
        else:
            print("can't close tile curve")
        tiles= tiles+tile
```

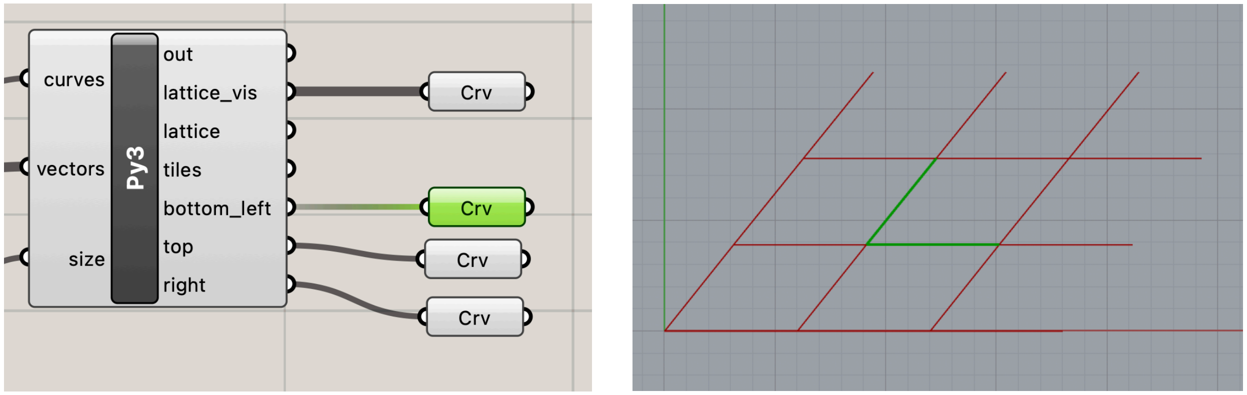
get top and right edges

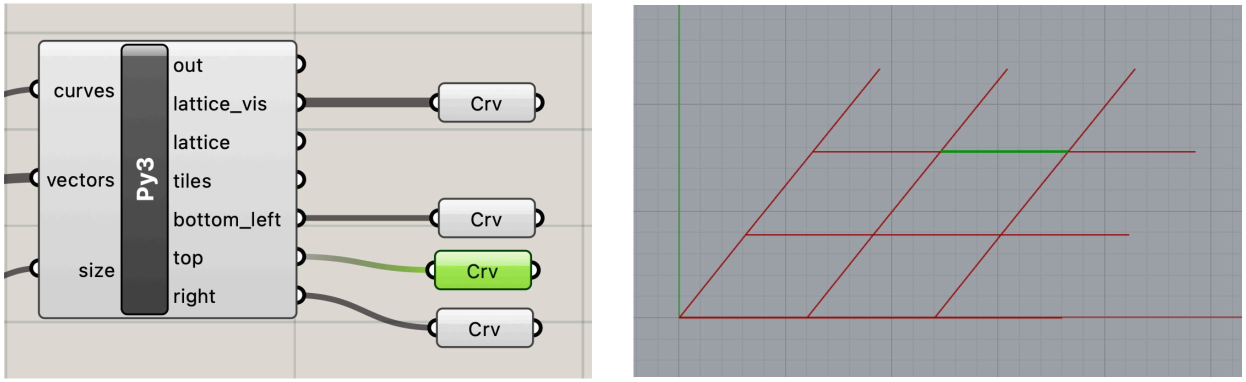
Find Tile Edges & Generate Tile

```
# generate tiling
# find the edge curves for each lattice cell
# generate a closed tile shape
# add to list of tiles
tiles = []
for i in range(0,size):
    for j in range(0,size):
        bottom_left = lattice[i][j]
        top = rs.ExplodeCurves(lattice[i+1][j])[0]
        right = rs.ExplodeCurves(lattice[i][j+1])[1]
        tile = rs.JoinCurves([bottom_left, top, right])
        if (rs.CloseCurve(tile)):
            tile = rs.CloseCurve(tile)
        else:
            print("can't close tile curve")
        tiles= tiles+tile
```

make sure you're generating closed tile

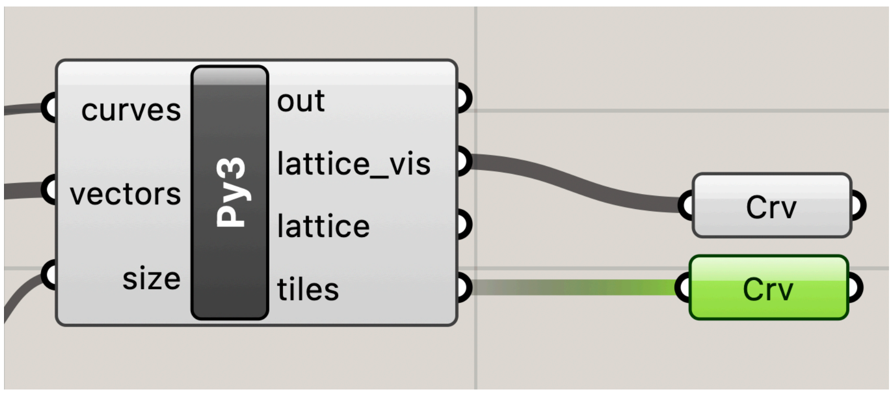
Adding outputs for tile edges may be useful for troubleshooting

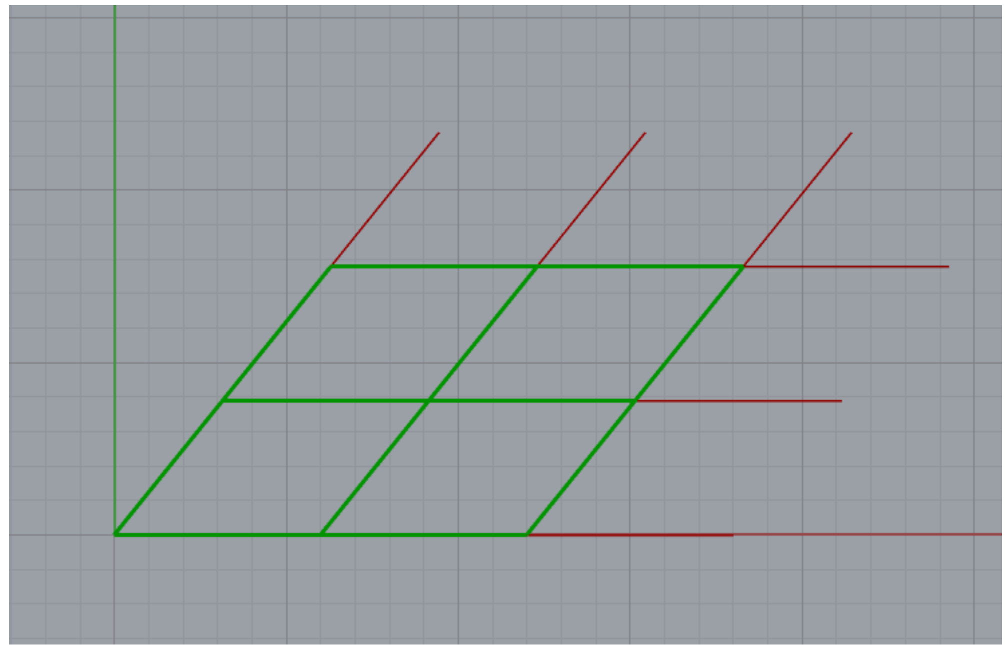




questions?

A simple tiling





What we'll do today

1. ~~Generate lattice~~
   a) ~~python block 1: takes a, b, and angle as input and outputs joined ab curve + a and b vectors~~
   b) ~~python block 2:~~
   ~~translates ab curve using vectors and outputs lattice~~
   ~~generates a closed curve (a tile) for each lattice cell and outputs a list of tiles~~
2. Generate Escher tiling
   a) add complex (Escher) a, b line inputs to python block 1. scale and rotate these complex curves to map to a and b vectors. add complex (Escher) ab curve output to python block 1.
   b) Use new output as input to python block 2

Escher Tiling

Approach

1. Allow complex Escher input curves as **ab** curves for second python block.
2. Input curve requirements:
   - **a** curve: begins at origin and ends at point on y axis
   - **b** curve: begins at origin and ends at point on x axis
3. Edit first Python block
   - Accept Escher curves as input
   - Output appropriately scaled and rotated Escher curves.

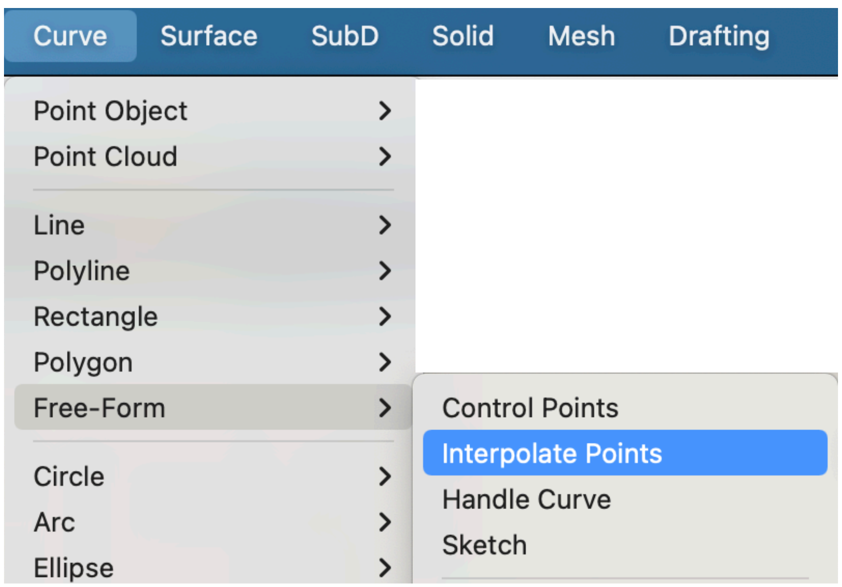
questions?

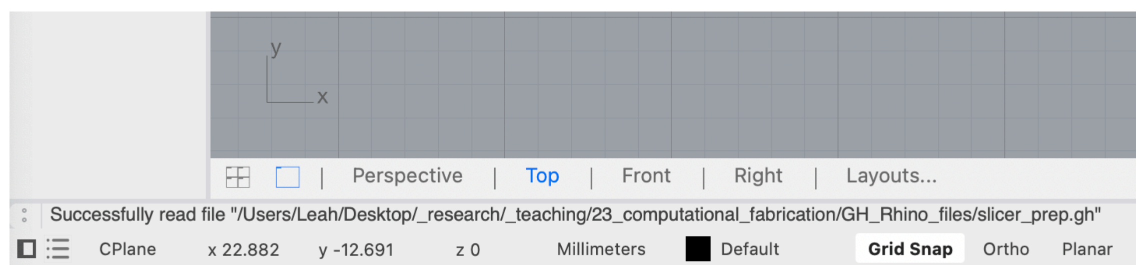
Draw Curves in Rhino

- **a** curve: begins at origin and ends at point on y axis
- **b** curve: begins at origin and ends at point on x axis

Turn on **Grid Snap** and

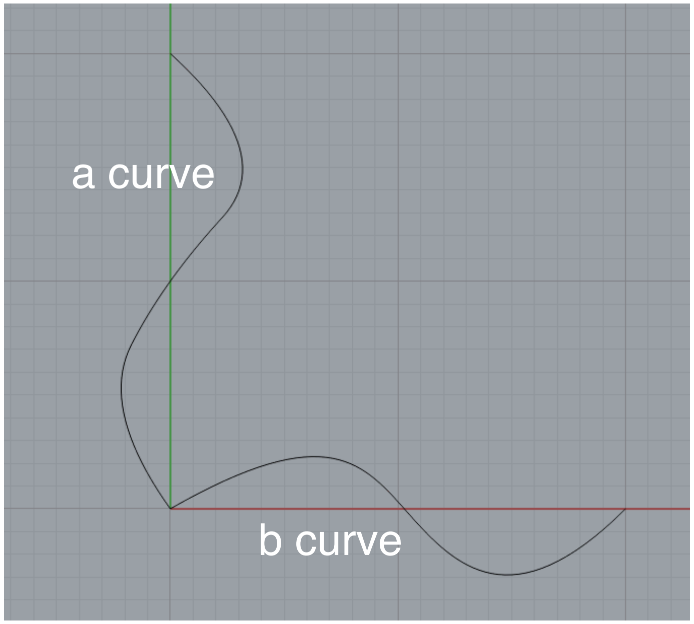
Use Curve—>Freeform—>**Interpolate Points**

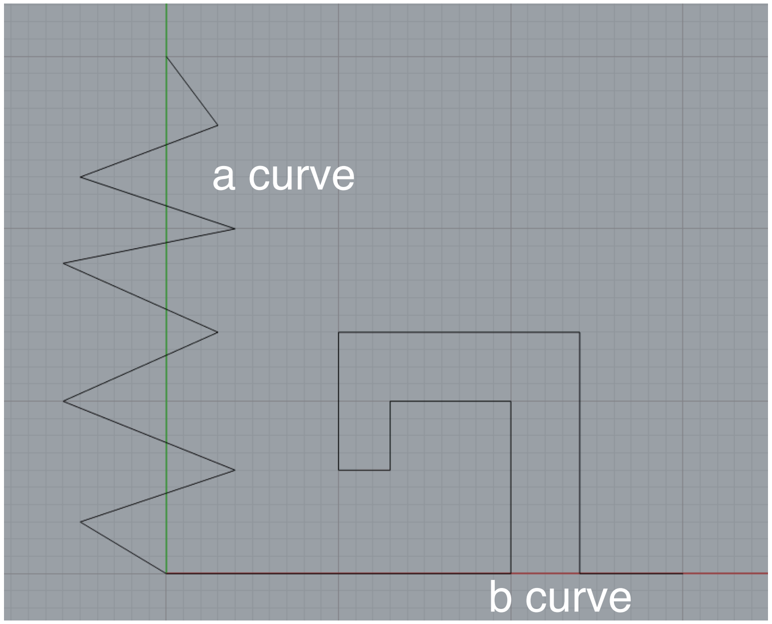




Draw Curves in Rhino

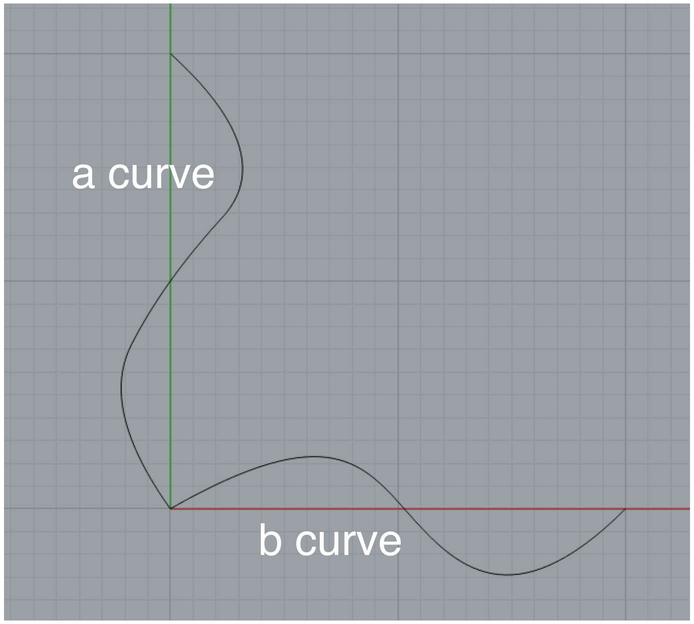
- **a** curve: begins at origin and ends at point on y axis
- **b** curve: begins at origin and ends at point on x axis

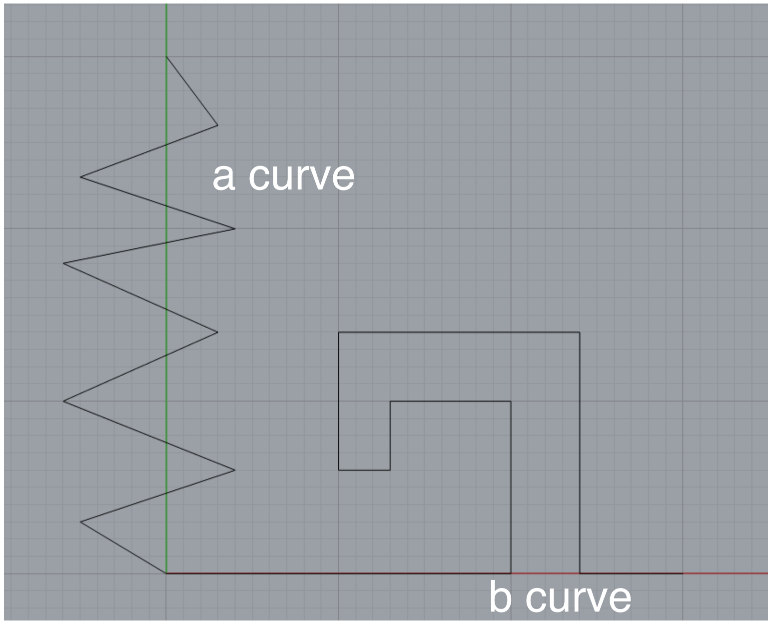




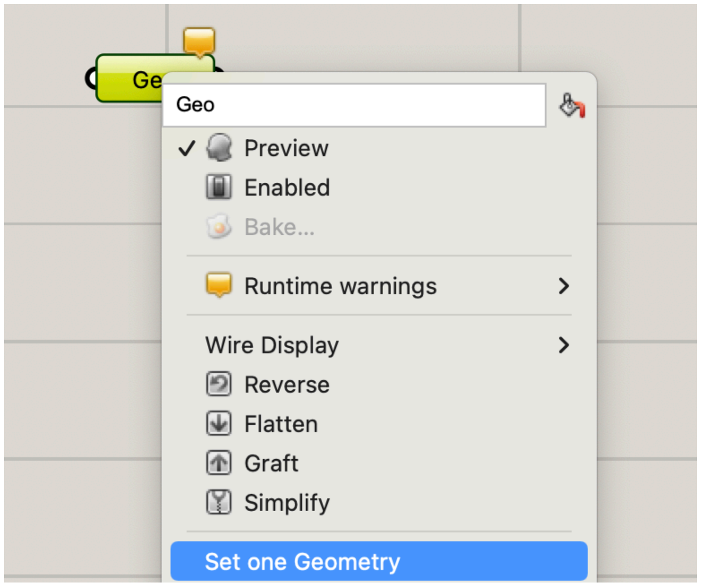
Draw Curves in Rhino

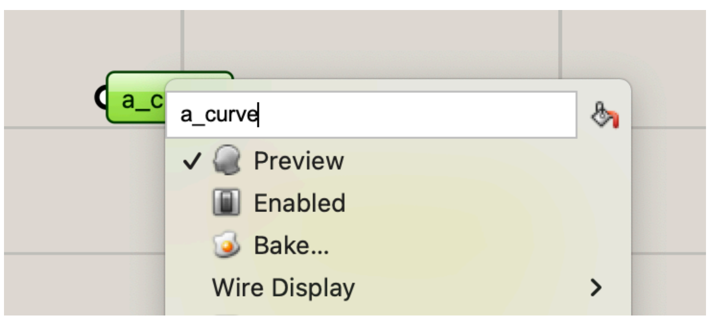
Save this Rhino file to preserve your curves.





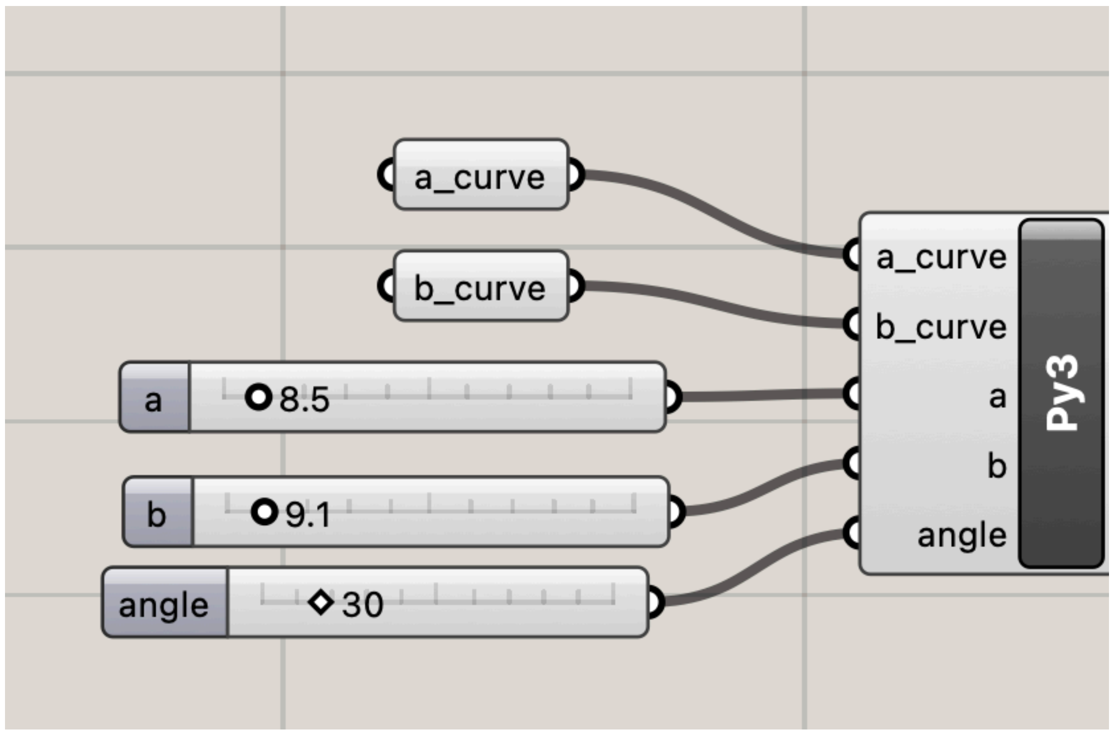
Associate Curves with GH object





1. Create a Geometry "Geom" block
2. Right click on it and choose "Set one Geometry"
3. Select the a curve you drew in Rhino. It should turn green.
4. Right click on block and rename it to a_curve
5. Do the same thing for your b curve. Name it b_curve

Add Inputs for these curves to 1st Python Block

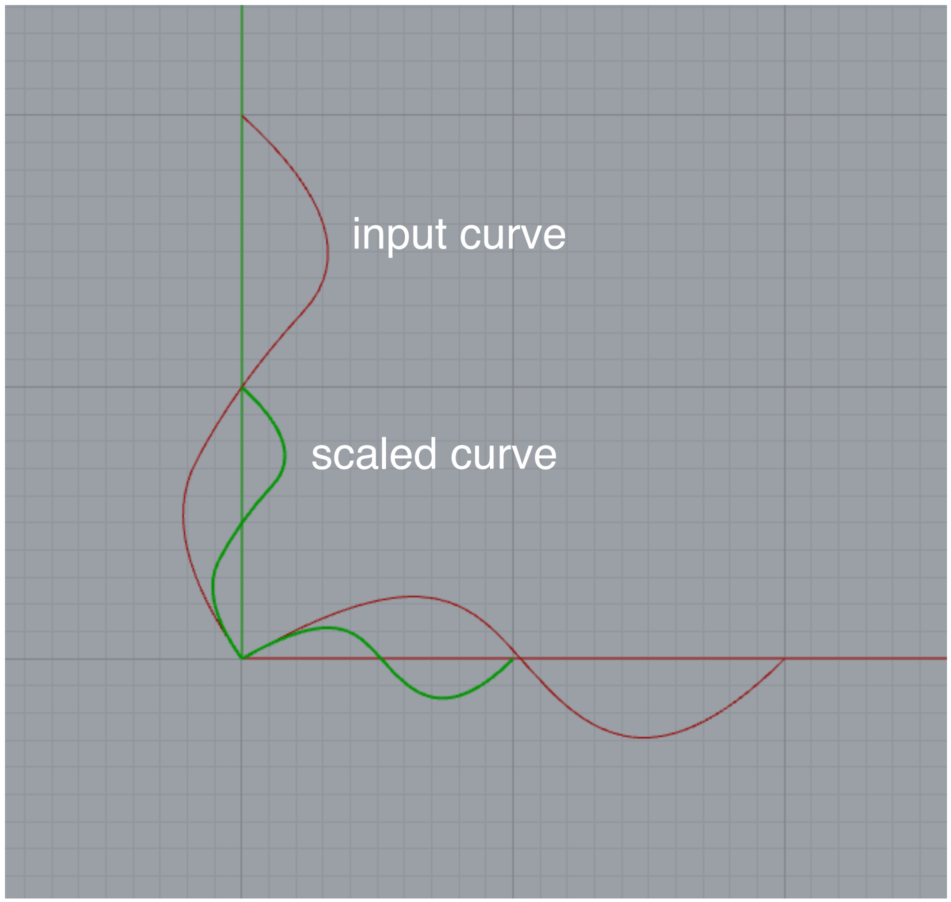


Scale Curves to fit a and b lengths

1. Use **rs.CurveEndPoint()** to find end points of curves.
2. What does the end point tell us about the length of curve **a**?
3. Use **rs.ScaleObject()** to scale each curve
4. What is the scale factor for curve **a**?

```
#scale input curves to fit specified lengths
#get current length of both curves
a_end = rs.CurveEndPoint(a_curve)
a_length = a_end.Y
a_scale = [a/a_length,a/a_length]
a_curve= rs.ScaleObject(a_curve,origin,a_scale)
```

Scale Curves to fit a and b lengths

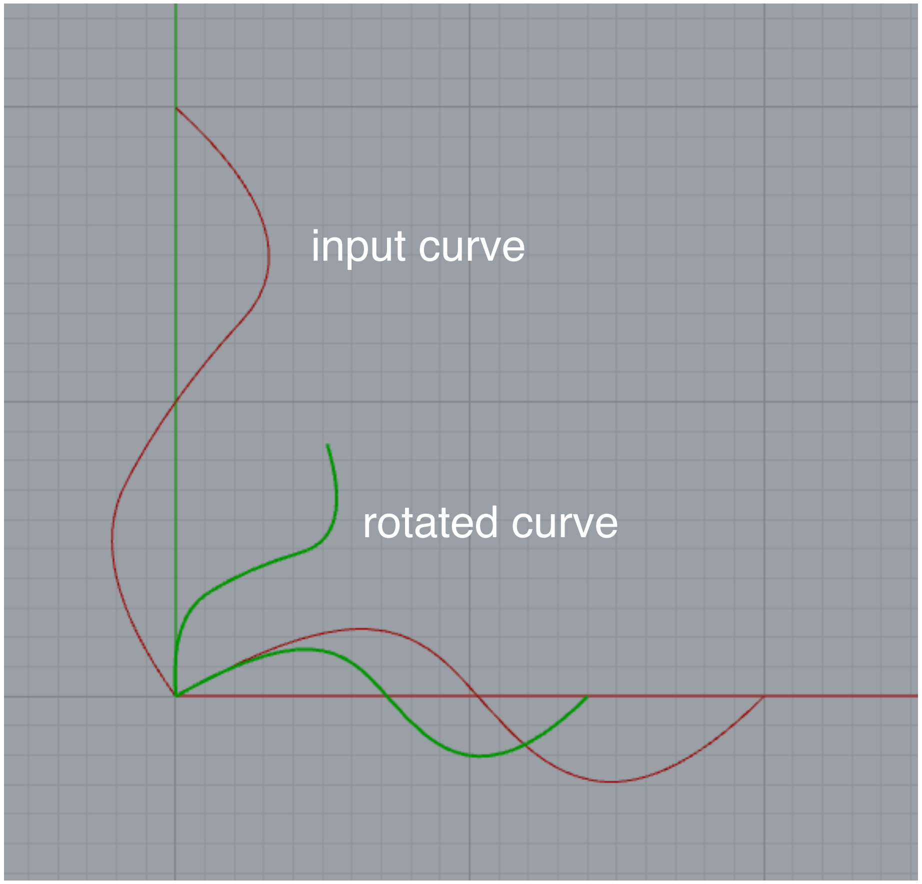


Rotate Curves to fit Lattice

1. Which curves do we have to rotate?
2. What is the rotation angle in terms of the input angle?

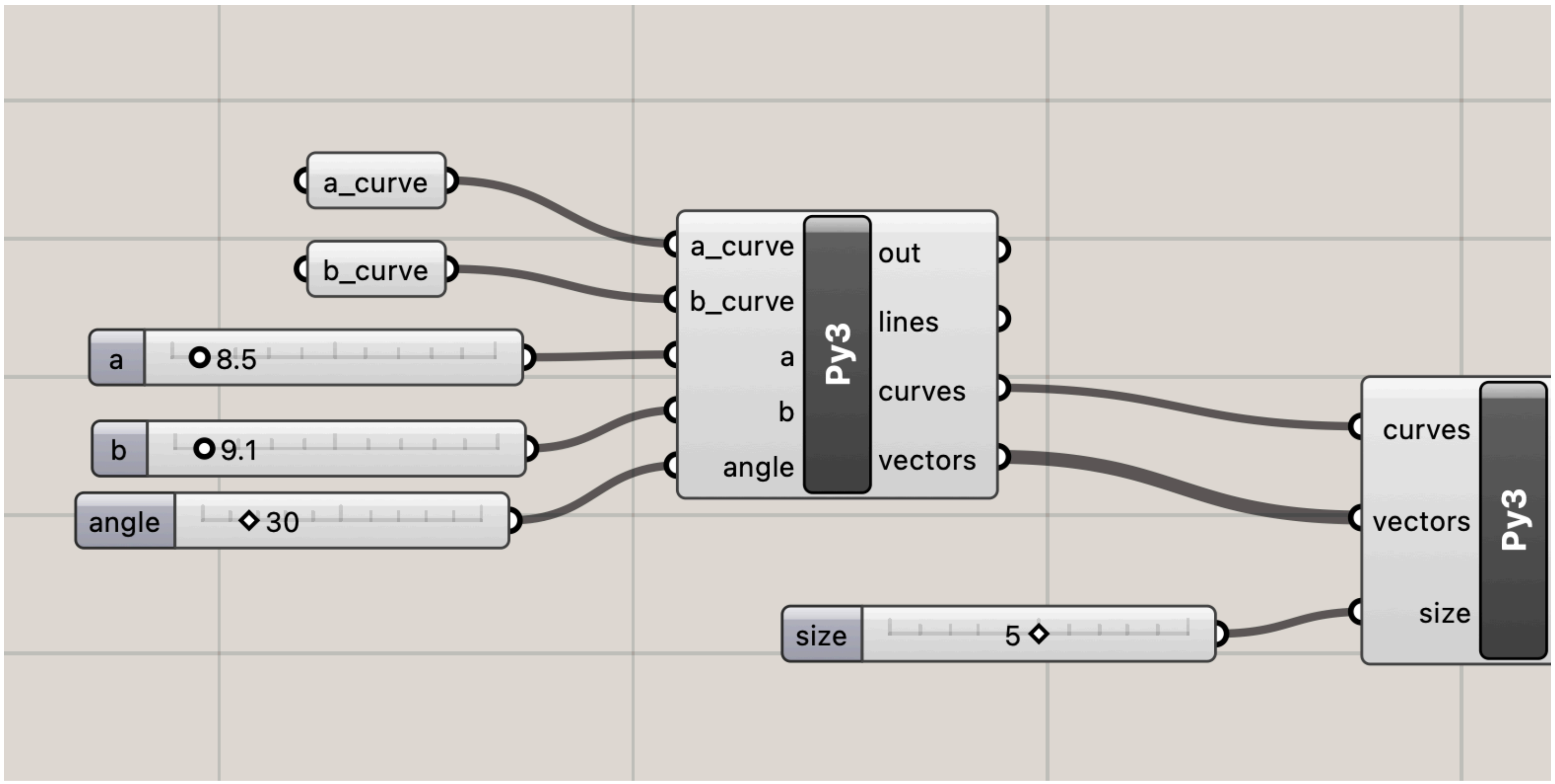
```
#rotate a_curve to correct orientation
a_curve = rs.RotateObject(a_curve,origin,angle-90)

curves = rs.JoinCurves([a_curve,b_curve])
```

Rotate Curves to fit Lattice



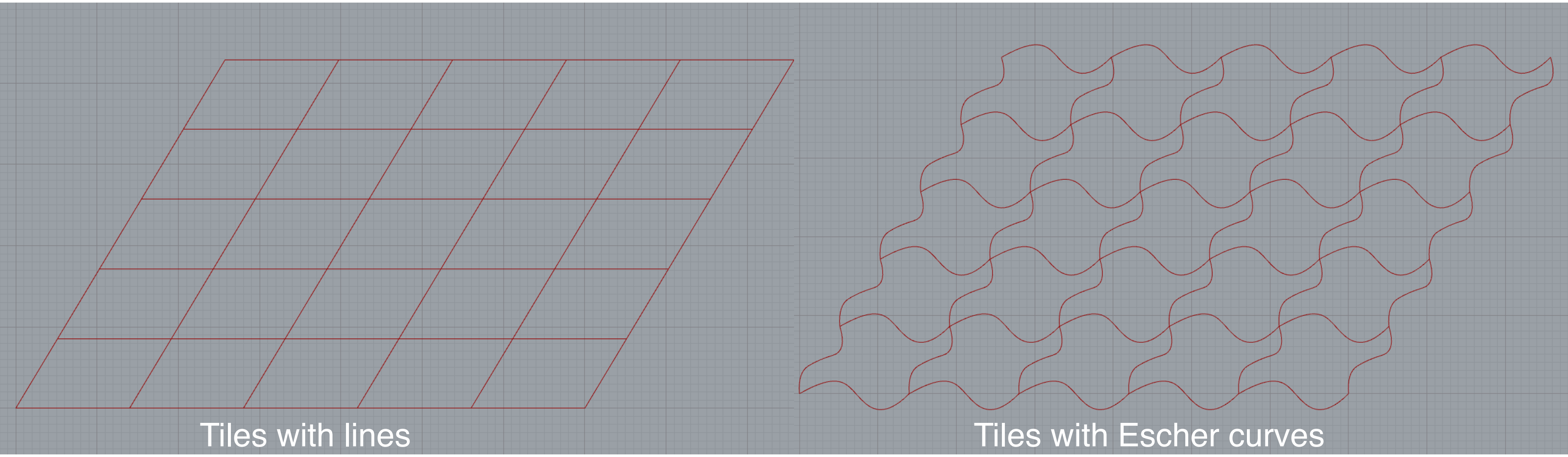
questions?

Connect Curves to Tiling Code

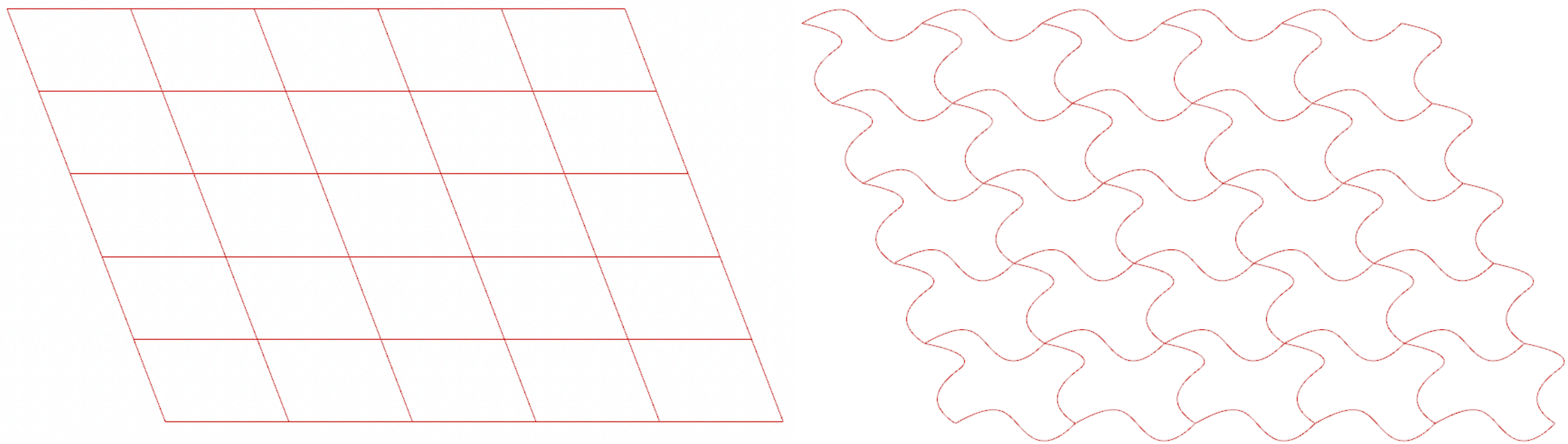


It should just work :)

Connect Curves to Tiling Code



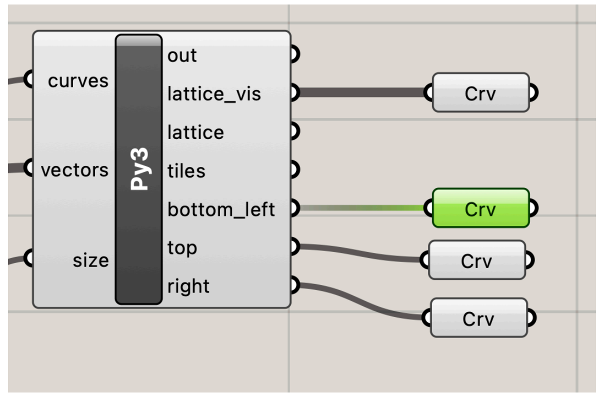
Connect Curves to Tiling Code

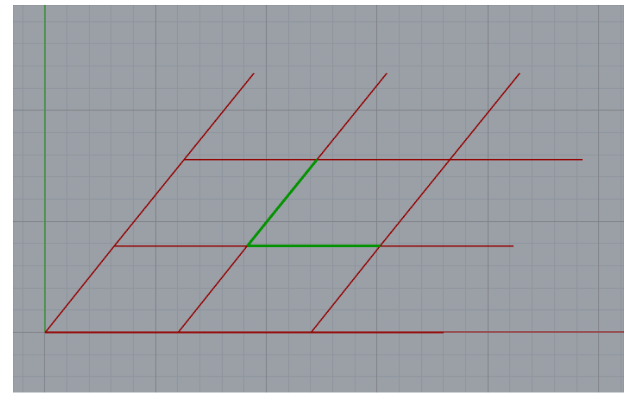


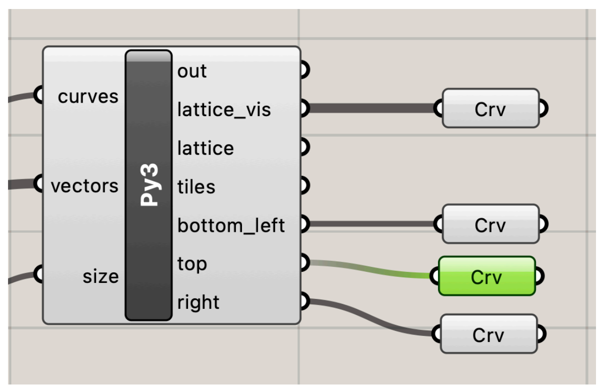
Rendered view in Rhino

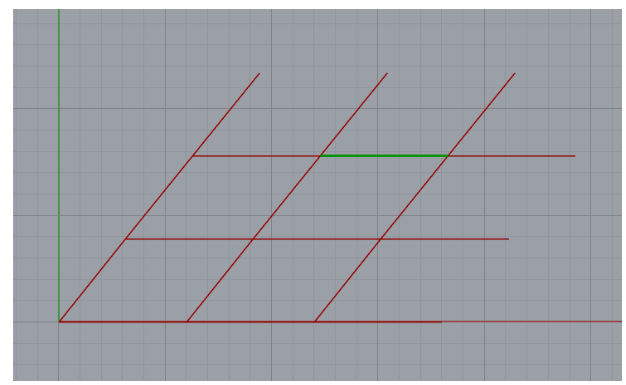
If it doesn't just work

- Check to make sure you're generating a closed tile with your new curves. Look at tiles edges.
- Tile generation will also depend on the order in which you joined curves in the first Python block.



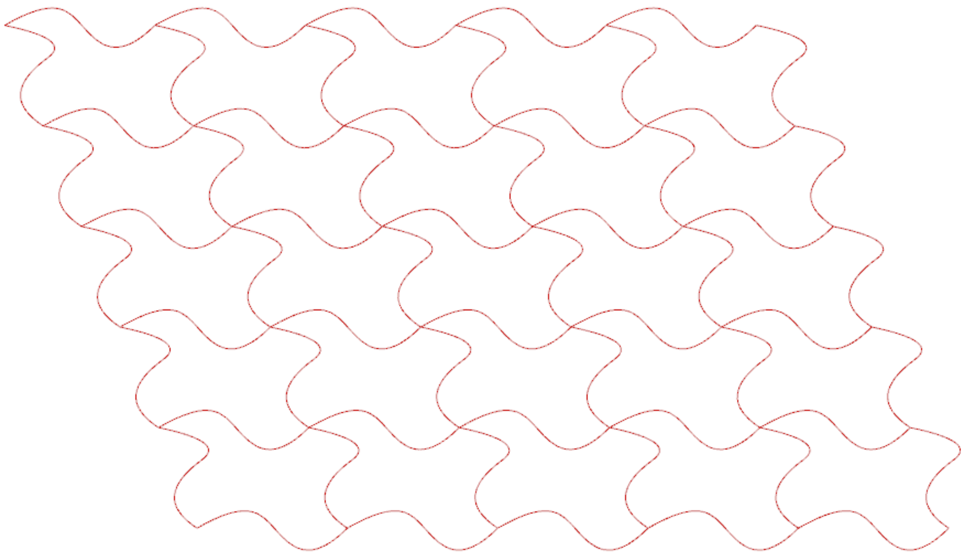


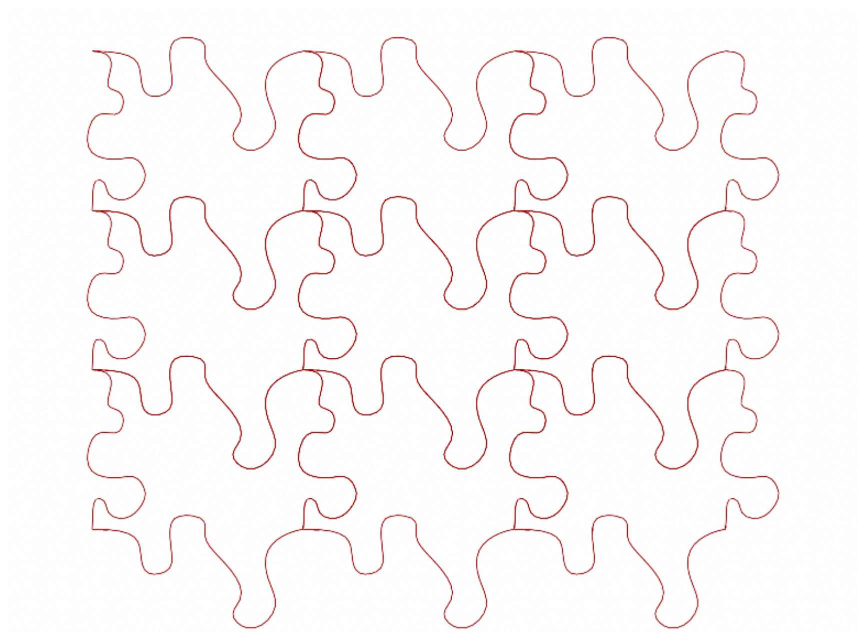




questions?

Edit Rhino curves to get different Escher tilings

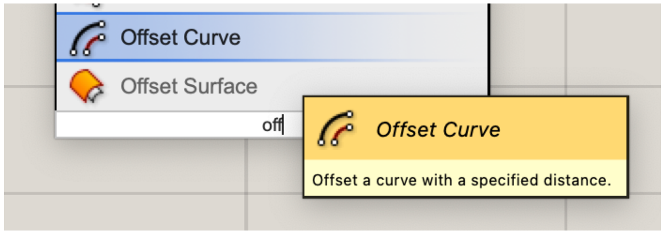


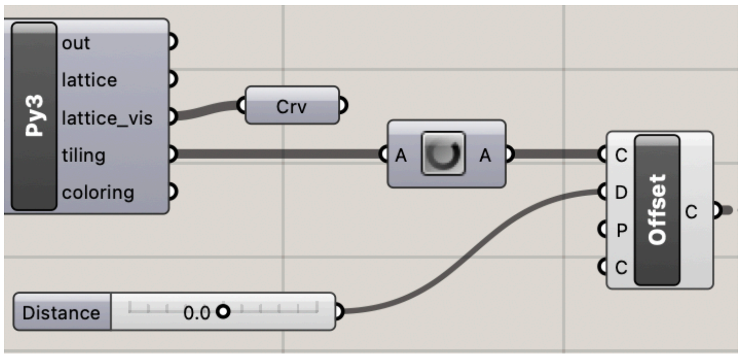


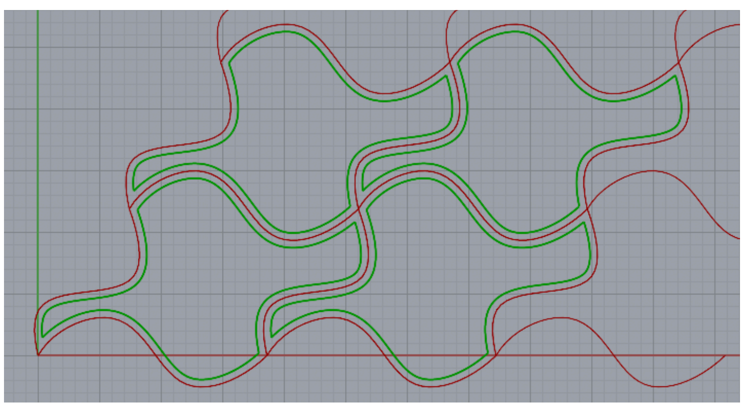
Rendered view in Rhino

Generating Printable 3D Tiles

Offset Tile Shape for Physical Tiling

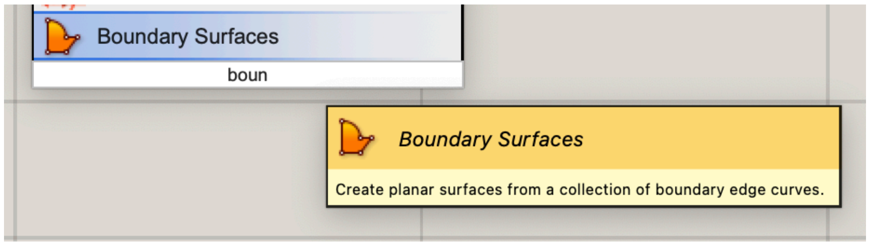


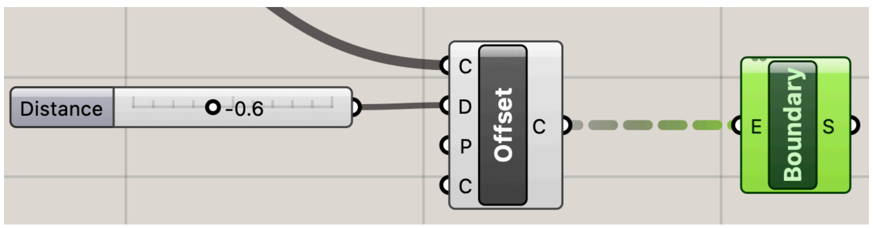


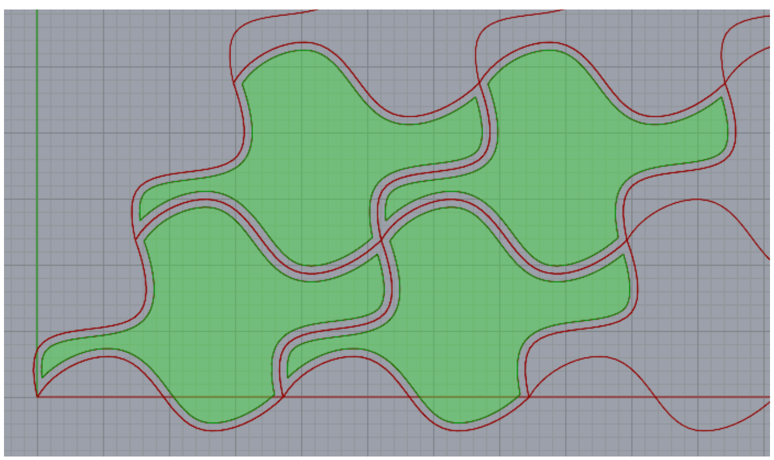


1. Create an **Offset Curve** GH block
2. Create a **Data Dam** GH block to prevent expensive computations from triggering when you change parameter values
3. Connect your tiles to the C (curve) input through the Data Dam block
4. Create a float number slider
   Range of number slider: -3.0 to 3.0
5. Connect number slider to the D (distance) input of Offset Curve block
   Negative number: offset in
   Positive number: offset out

Create Tile Surface

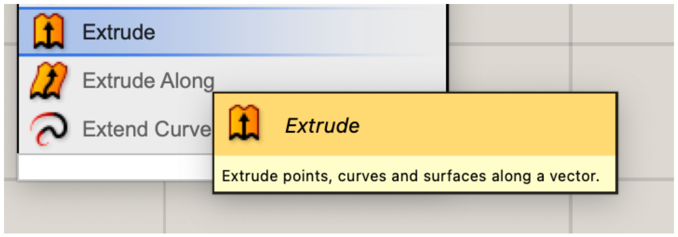


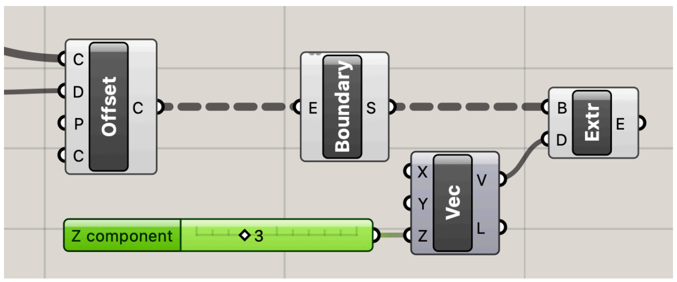


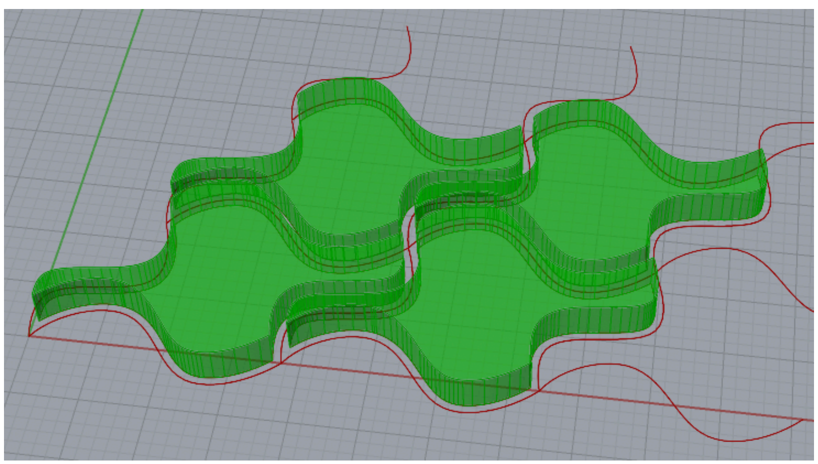


1. Create a **Boundary Surfaces** GH block
2. Connect the C output from Offset to the E input to Boundary

Extrude Surface to Generate 3D Tile



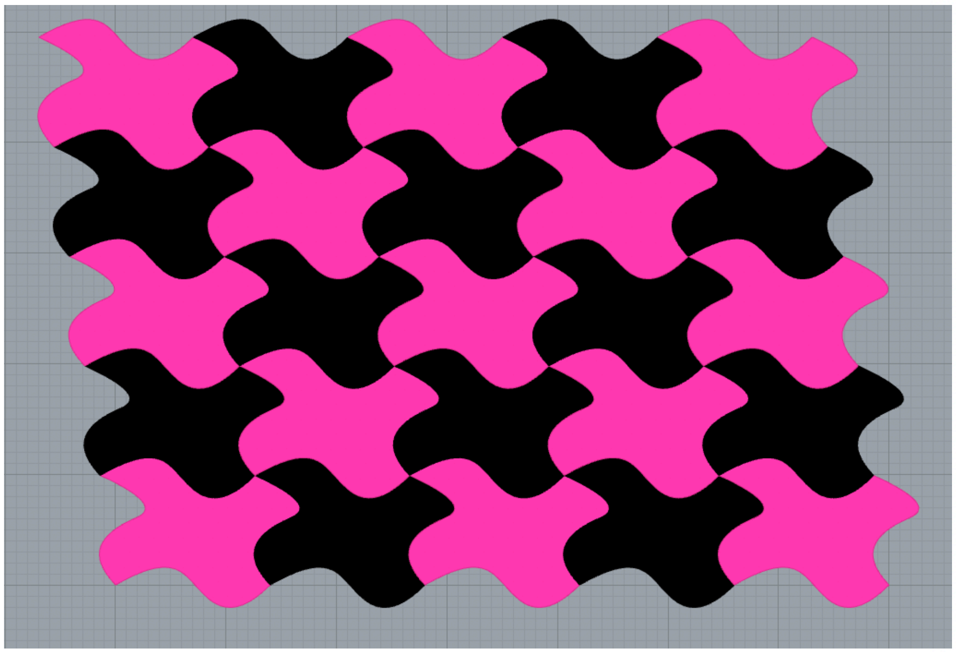




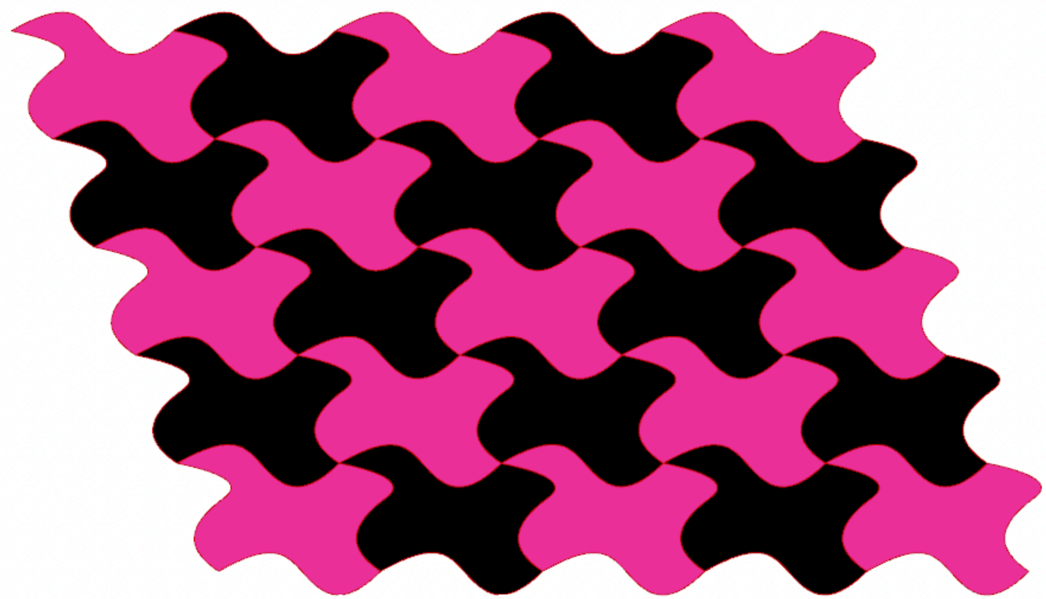
1. Create an **Extrude** GH block
2. Create a **Vector** GH block and provide a number slider input for Z.
3. Connect the S output from Boundary to B on Extrude and the V output from Vector to D

questions?

Add Some Color



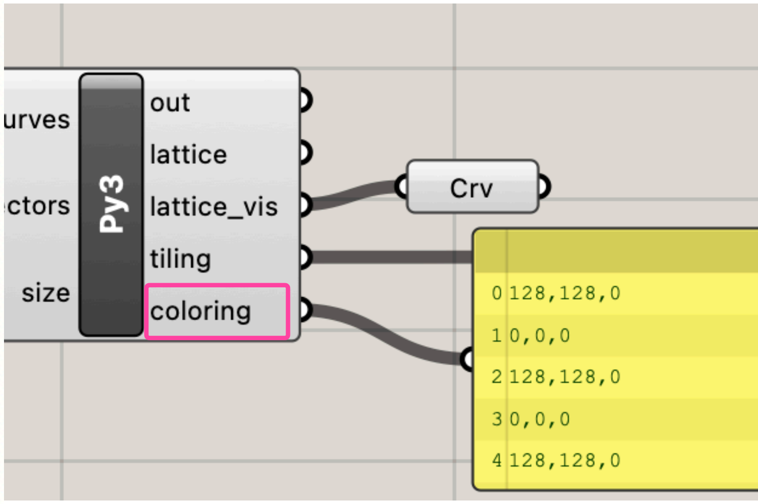
Wireframe view in Rhino



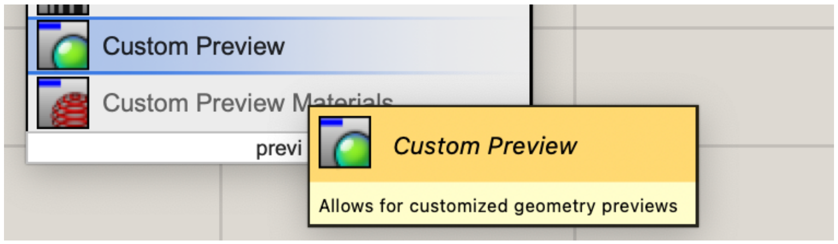
Rendered view in Rhino

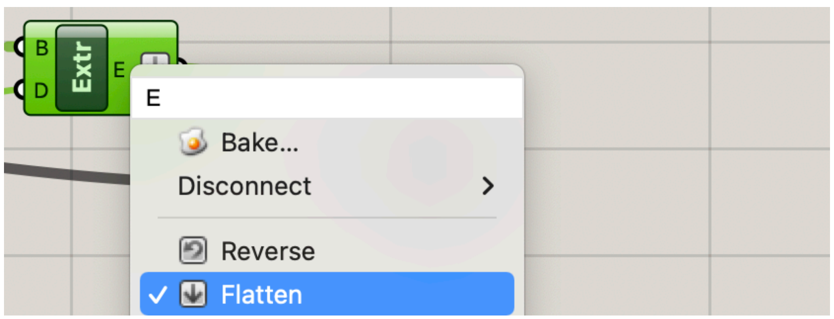
2nd Python Block

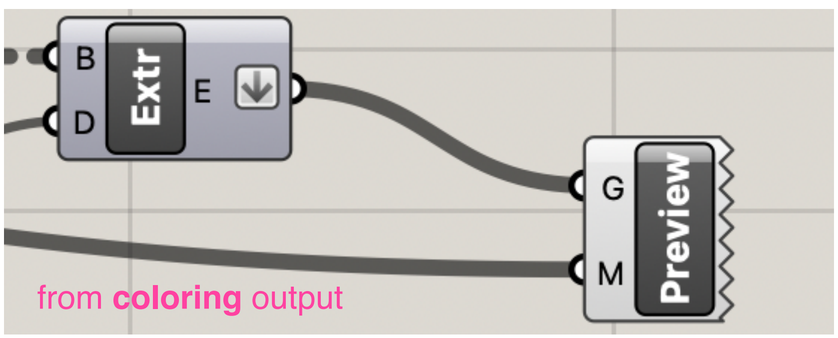
```
tiling = []
coloring = []
for i in range (len(lattice)-1):
    for j in range (len(lattice)-1):
        bottom_left = lattice[i][j]
        top = rs.ExplodeCurves(lattice[i+1][j])[0]
        right = rs.ExplodeCurves(lattice[i][j+1])[1]
        tile = rs.JoinCurves([bottom_left,top,right])
        if (rs.CloseCurve(tile)):
            tile = rs.CloseCurve(tile)
        else:
            print("can't make a closed tile")
        tiling = tiling+tile
        if (i%2==0 and j%2==0):
            coloring.append("128,128,0")
        elif (i%2==1 and j%2==1):
            coloring.append("128,128,0")
        else:
            coloring.append("0,0,0")
```



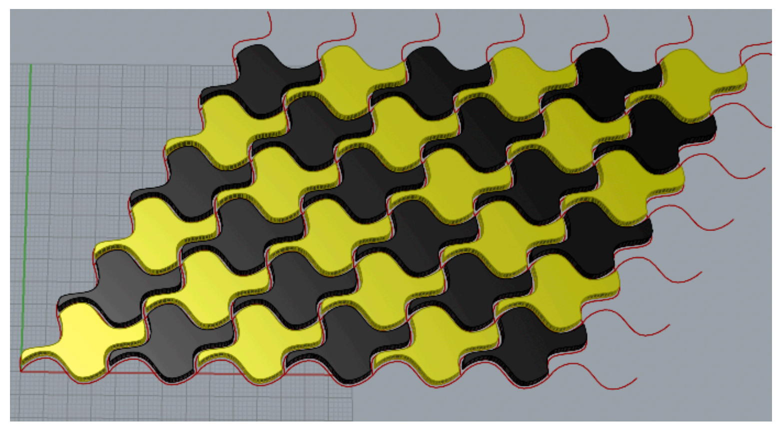
Add Some Color



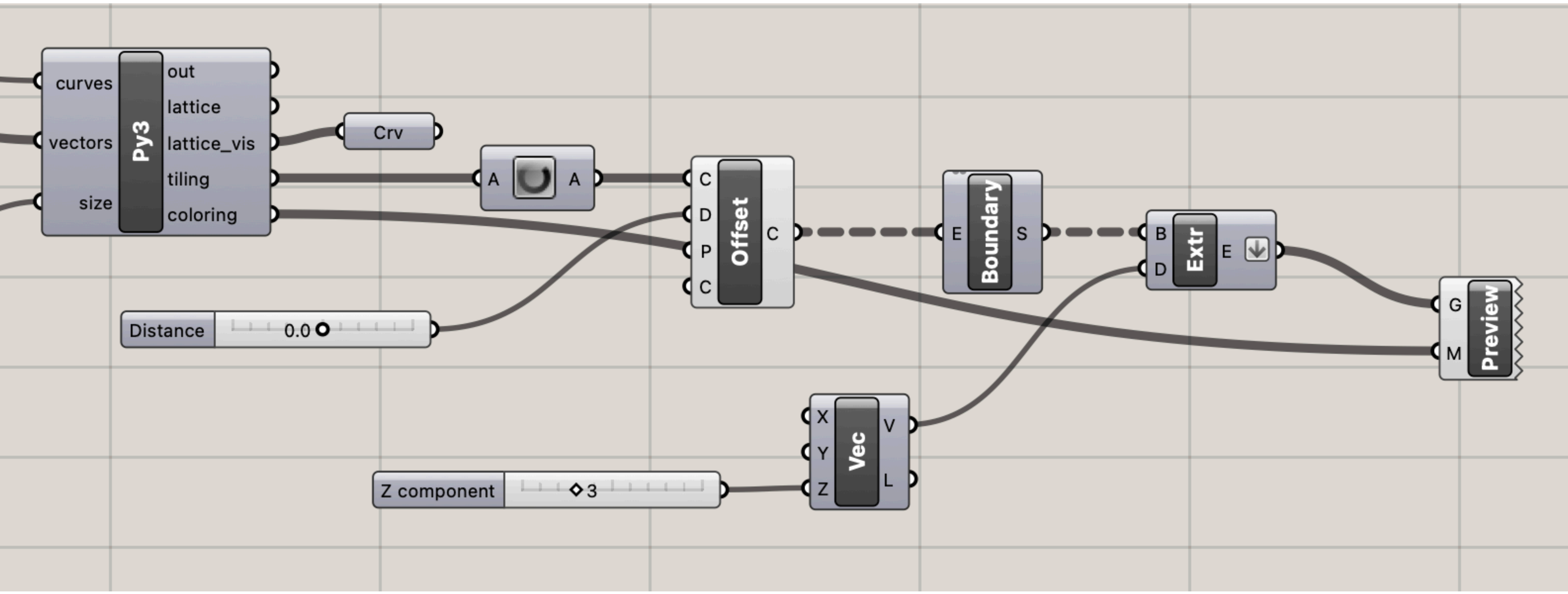




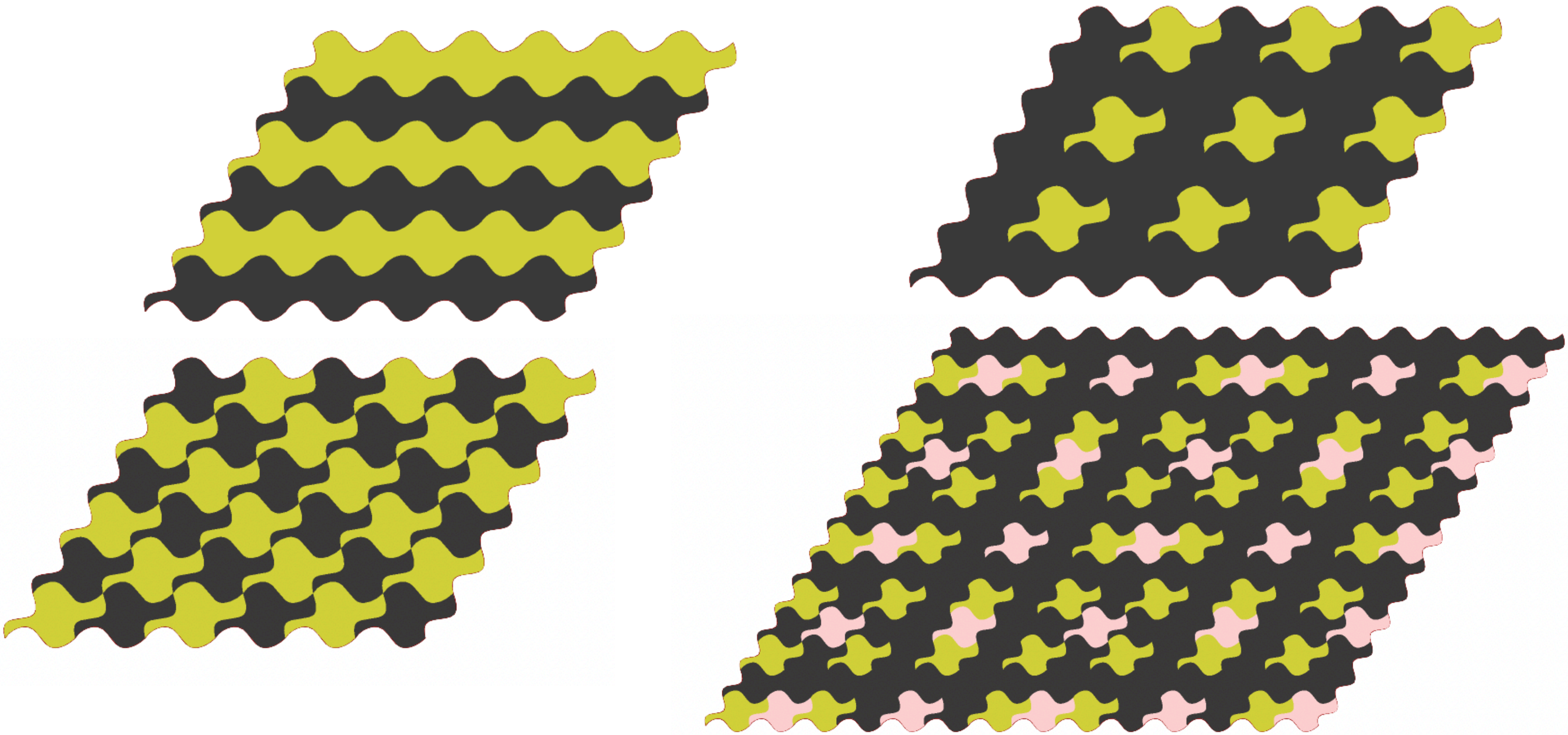
1. Create an **Custom Preview** GH block
2. Connect the output of the Extrude block to G (geometry) input. **Flatten** the output from Extrude.
3. Connect the coloring output to the M (materials) input.



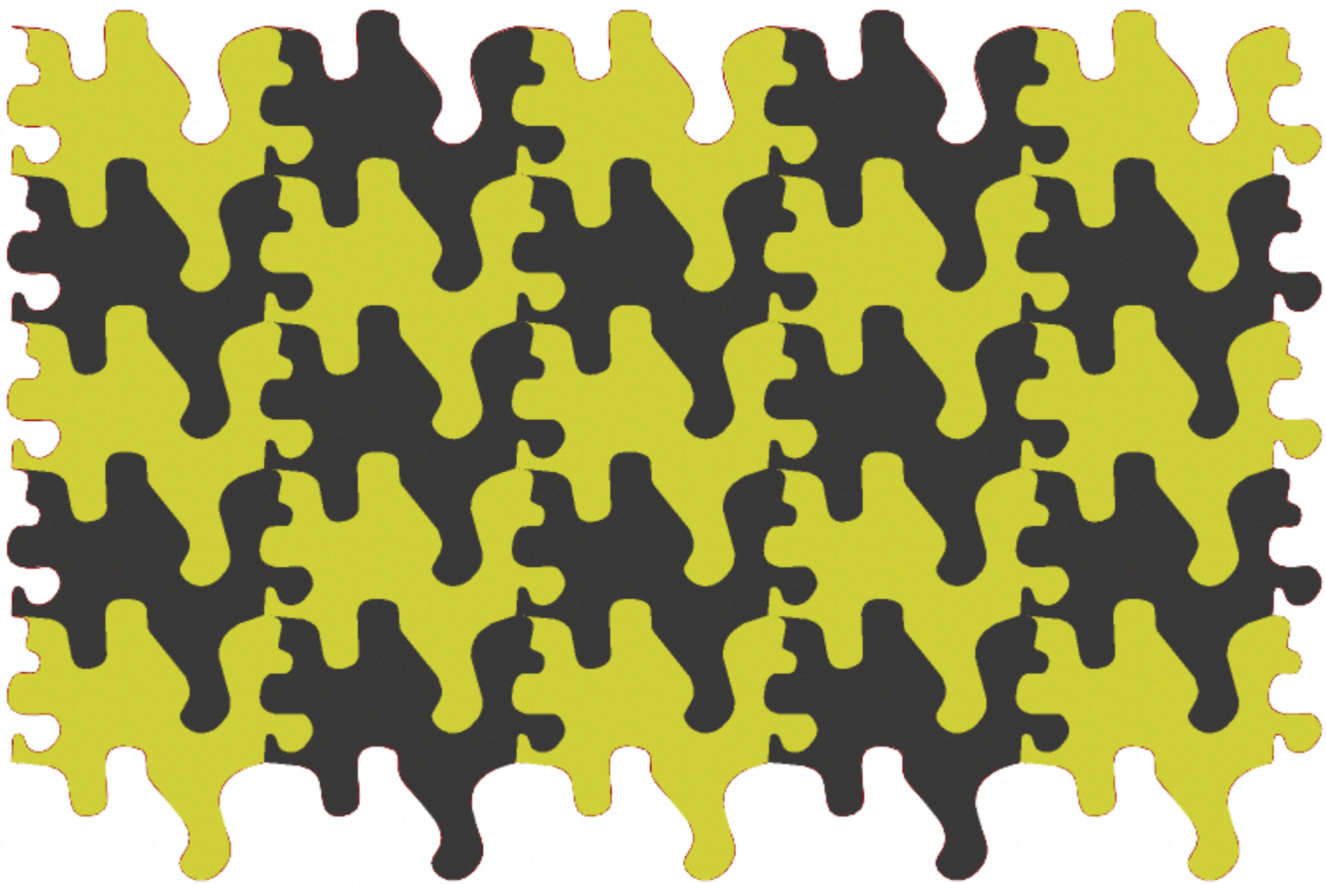
Add Some Color



Play with Coding & Color Patterns



Play with Different Input Curves



questions?

Thank you!

CS 491 and 591
Professor: Leah Buechley
https://handandmachine.org/classes/computational_fabrication