2 - Continuous Cylinder with Base in Grasshopper

Link to Grasshopper Code

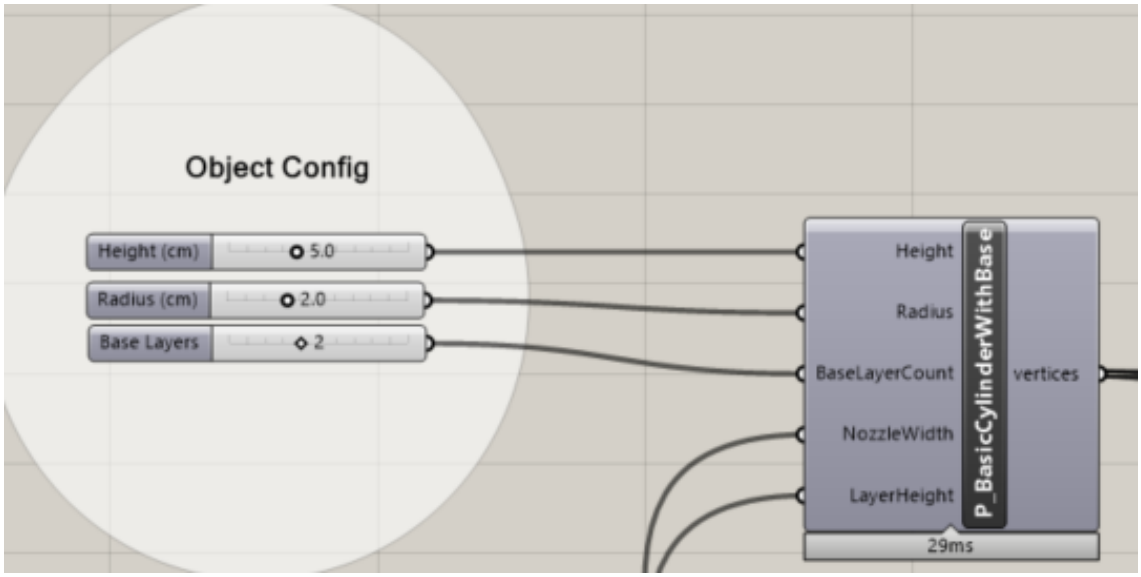
This tutorial builds on the previous example (1 - Continuous Cylindrical GCode in Grasshopper) by adding a base to the code. Refer to the previous document for cylinder and GCode creation.

Generating a base

Clay printers work really well with vertical walls but they are somewhat problematic for printing horizontal structures like bases for objects. The printed lines need to adhere together well enough that they would survive the drying and firing shrinkages. Even if there is a slight gap in between one of the lines, these shrinkages can make it open more, or it could explode in the kiln.

However, it is possible to print a base with Potterbot. We modify the previous example to add base support. Object Config now had another slider that specify the base height. Each base layer is actually composed of two layers, so you will see 4 vertical layers if *Base Layers* is 2. This is due to two reasons:

1) The height of the base is dependent on the layer height, which is much smaller than the nozzle diameter. So compared to the walls of our object, the base would have been too thin if we use only one layer. On top of this, since clay shrinks after drying and firing, the already thin base would become even thinner compared to the walls.
2) The order of the vertices we generate for the toolpath matters. If we have only one layer, the nozzle head would have to travel from the middle of the object to the sides. Since the Potterbot does not have retraction, this would result in extra material in the middle of our print.



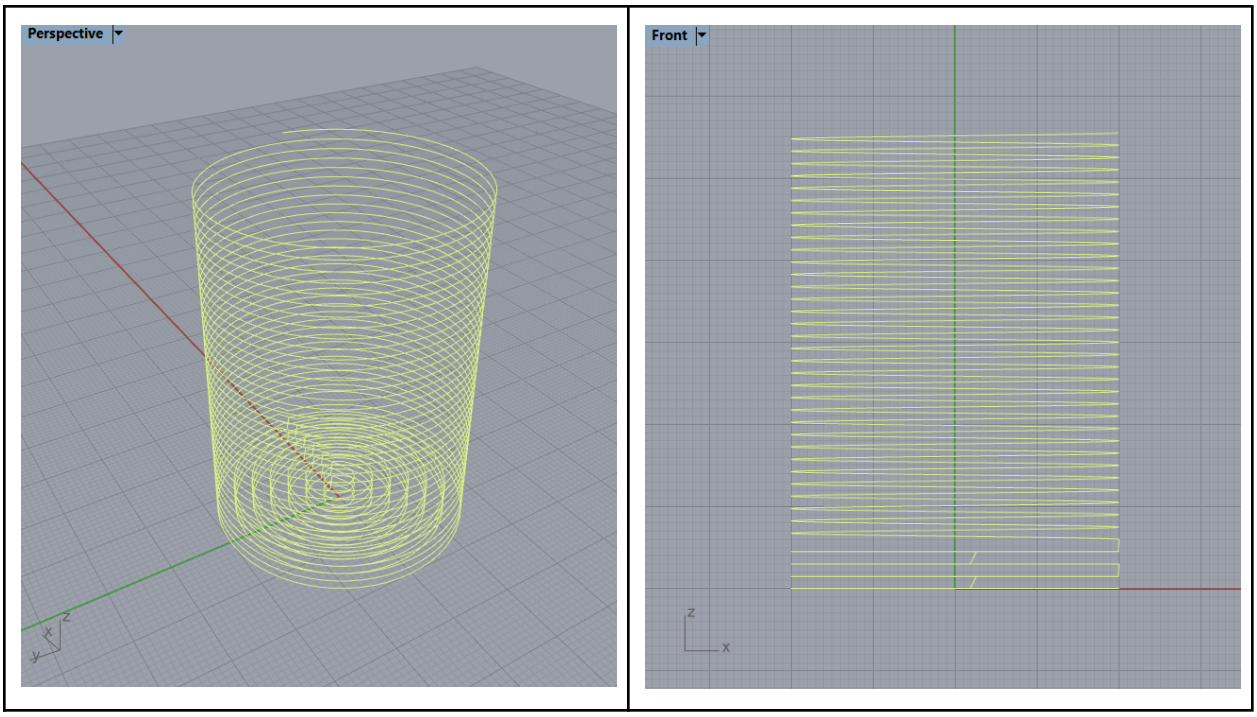
```
# P_BasicCylinderWithBase

# input
Height = Height * 10            # Convert to mm
Radius = Radius * 10            # Convert to mm
NozzleWidth
LayerHeight
BaseLayerCount

# output
vertices = []

# BASE
def makeBaseLayers(base_index):
    base_index *= 2                                 # Each base layer consists
                                                    # of two layers
    spiralCount = Radius/(NozzleWidth - 0.1)
    _r_dec = Radius / spiralCount                   # Radius decrement
    _r = Radius
    for i in range(int(2*(spiralCount+1))):
        circumference = math.pi*2*_r
        pointCount = int(circumference * 0.5)       # One vertex every 2 mm
        for j in range(pointCount):
            ang = math.pi*2 * j/pointCount          # Angle of the polar coord.

            x = math.cos(ang)*_r
            y = math.sin(ang)*_r
            z = base_index*LayerHeight
            if i >= spiralCount:                    # Second base layer
                z = (base_index+1)*LayerHeight
            vertices.append(rs.CreatePoint(x,y,z))
        if i < spiralCount:                         # First or second base layer
            _r -= _r_dec
        else:
            _r += _r_dec



# WALLS
```

```
def makeWalls(wallStartHeight):
    layerCount = int(Height / LayerHeight)
    for i in range(layerCount):
        circumference = math.pi*2*Radius           # Circumference of the layer
        pointCount = int(circumference * 0.5)      # One vertex every 2 mm
        for j in range(pointCount):
            ang = math.pi*2/pointCount * j
            inc = LayerHeight/pointCount           # Amount to raise each point
                                                   # for seamless toolpath
            x = math.cos(ang)*Radius
            y = math.sin(ang)*Radius
            z = wallStartHeight + i*LayerHeight + j*inc
            vertices.append(rs.CreatePoint(x,y,z))

''' Make the base '''
for i in range(BaseLayerCount):
    makeBaseLayers(i)

''' Make the walls '''
makeWalls(wallStartHeight = BaseLayerCount*2*LayerHeight)
```

The above code shows a basic approach to printing bases with concentric circles. Previous code that makes the walls are wrapped inside the *makeWalls* function with a parameter to adjust its starting height.

makeBaseLayers function creates the vertices for the base. It takes the index of the current base layer as a parameter to adjust the height of the current points.

We first find how many concentric circles we need per layer. Each circle needs to be touching one another, so we divide the radius by nozzle width, minus a small offset value to make sure they will touch each other. We run the function twice for the first and second base layers, with the same polar coordinates that we used for generating the walls.



After generating the vertices, we pass them to the same GCode generating block to create the file.