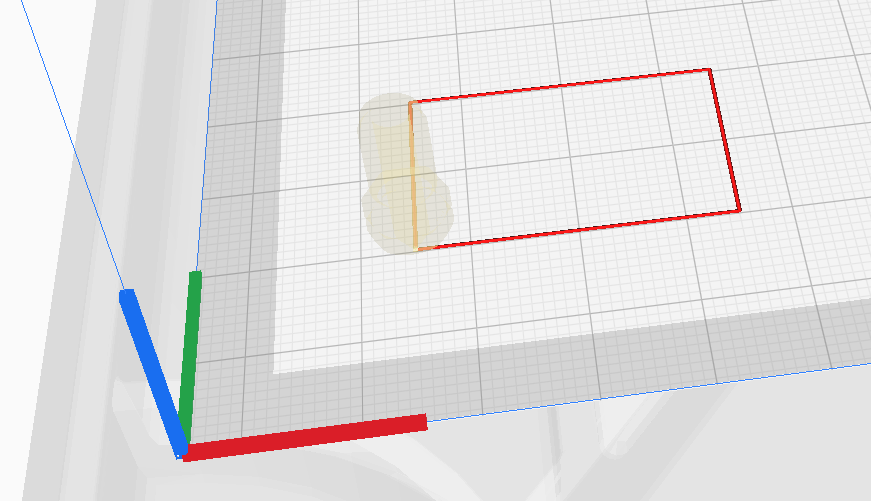
# Session 5: Make Abacus

*It turns out that abacus can be made entirely with squares and rectangles. No need for other shapes for now!*

## Make Rectangle functions

### Rectangle Boundary



l

b

l

b

Use what you learned with **structures** to make a rectangle.

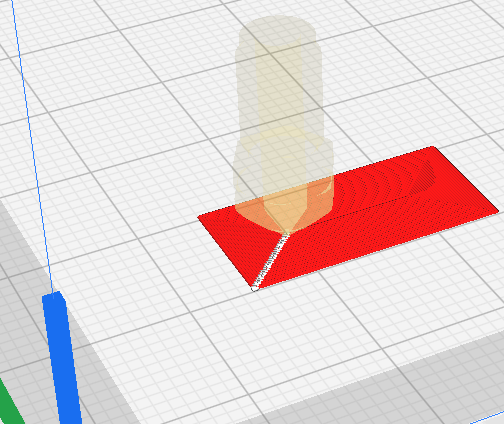
Combine all essential properties of the rectangle in one structure:

1. Length (l)
2. Breadth (b)
3. Corner coordinates c[]
4. Distance to move between any 2 points (horizontally anti-clockwise it is: l,b,l,b)

The corner coordinates are the starting point for the rectangle.

Assignment: Make a function to make a rectangle out of a given rectangle structure variable.

### Filled rectangle

Use x axis as the longer axis and y axis as the shorter one for convention.

Then make the outer rectangle (boundary) first.

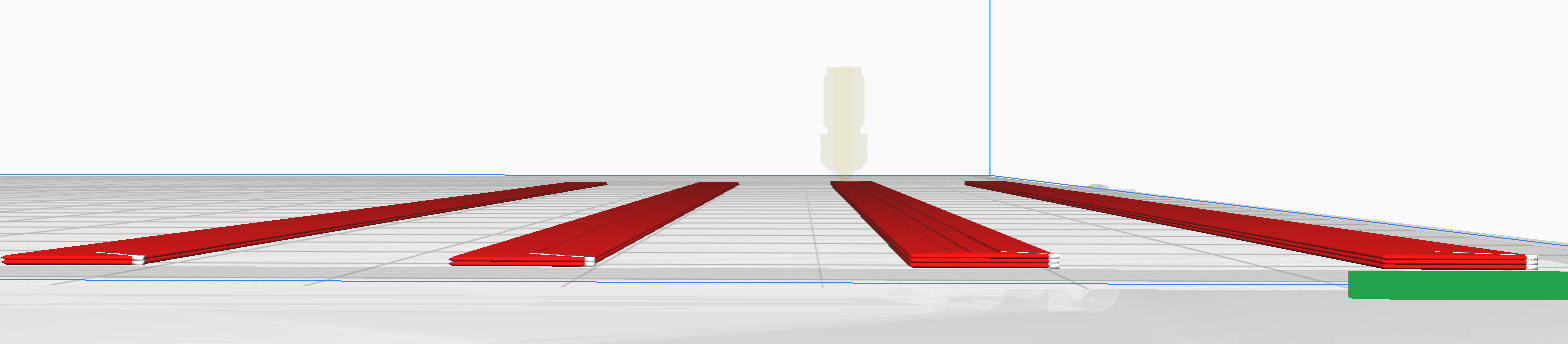
Then make another rectangle with start point (c[]) on the inside and with length and breadth reduced by the printing width. This makes an inner rectangle in contact with the outer one. Keep doing this until the breadth of the next rectangle becomes 0.

Assignment: Make a function that calls the rectangle function repeatedly to make a filled rectangle.

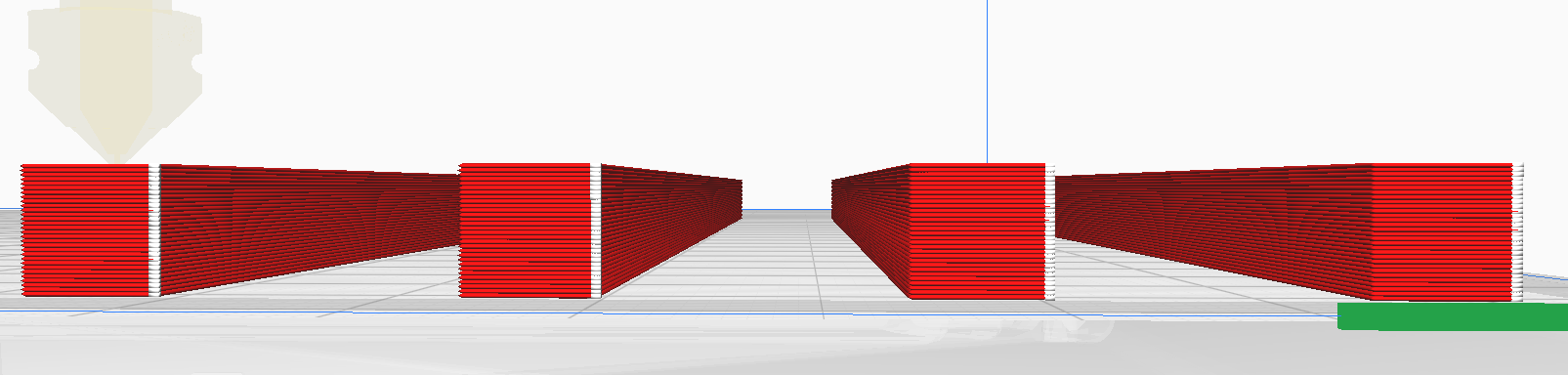
## Rods/Poles/Sticks

If we just stack rectangles on top of each other, we make a cuboid. If the breadth is equal to the height, we have a rod with a square cross section.

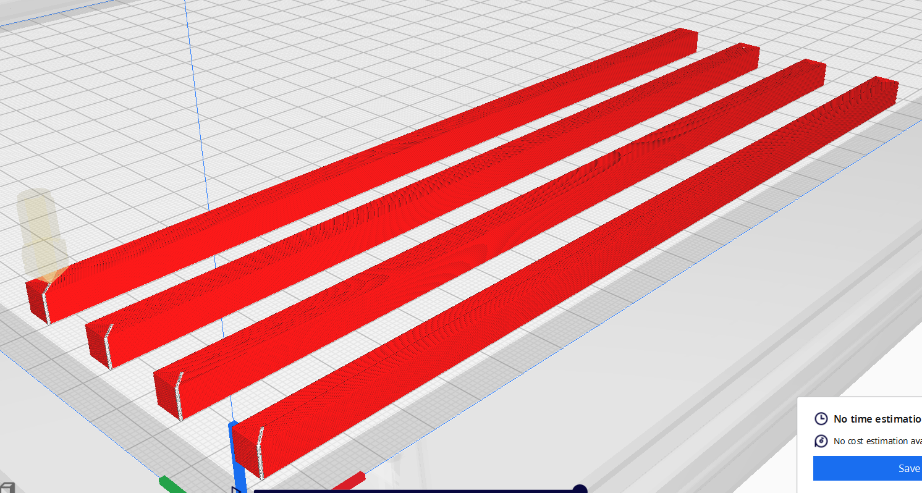
To make 1 stick, you can just stack one above the other as needed. However, when making multiple sticks, you need to ensure that the same layer (same z value) is finished for all the sticks at the same time. So you have to move the tool between the sticks as you finish layer 2 on all the sticks starting from the first stick to the last. Then go to layer 3 for all sticks as shown here.



All sticks completed.



This can be the first and simplest print to try on the machine. You have 4 sticks. We make the pegs and the base and we have an abacus set!

Sticks iso and top views



Note that there are a lot of variables to keep track of. Instead of inputting everything manually, make a text file containing all the info and get the function to take the values from there. Less confusion and easier to change values for a different desired result.

The text file for this execution was:

n length breadth xcoord ycoord ydist height p-width

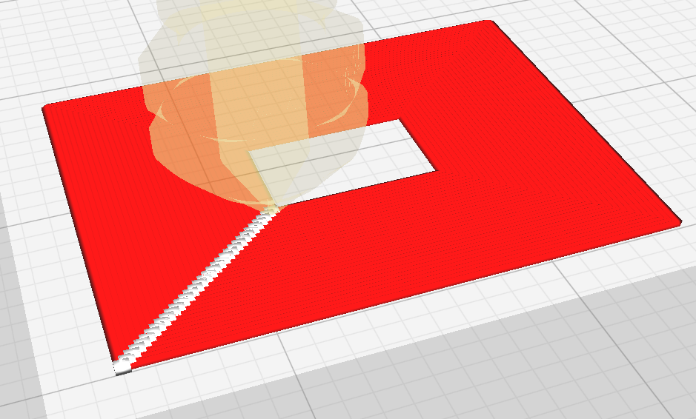
4 200 6 10 10 20 6 0.2

This takes in the values for the number of sticks first since the dynamic array needs to be allocated with “n” instances. We take that first and then declare the rect structure variable arrays where the first instance takes in these values and the rest of the instances are off-set by “ydist” amount.

Assignment: Make a function to call the filler function sequentially to make n number of sticks.

## Base

### Make Holes

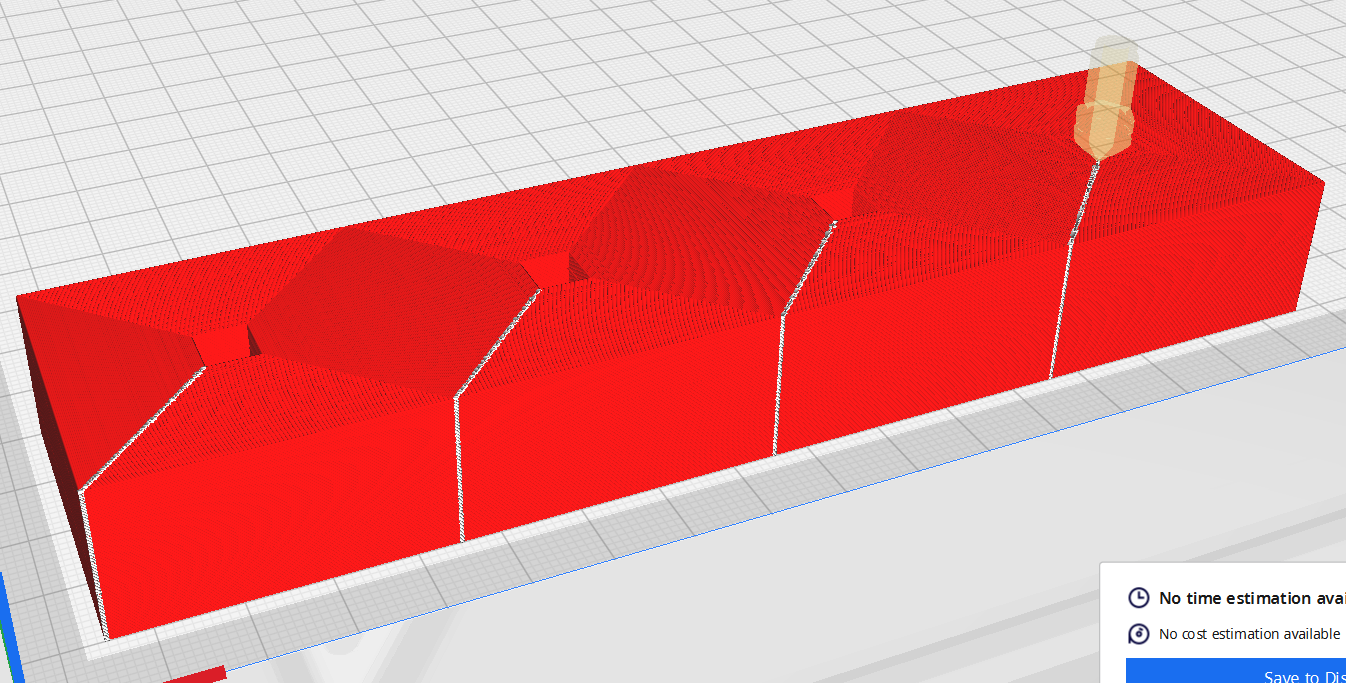
Make a new function here that fills a rectangle area between a minimum length and maximum length. The ratio of length to breadth has to be the same for both inner (id) and outer (od) boundaries.

Basically, this function makes holes:

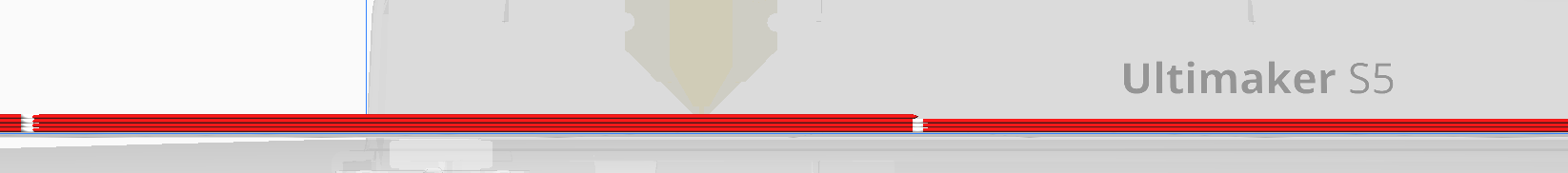
Assignment: Make a function that calls the rectangle function repeatedly between an outer and inner boundary to make a hole.

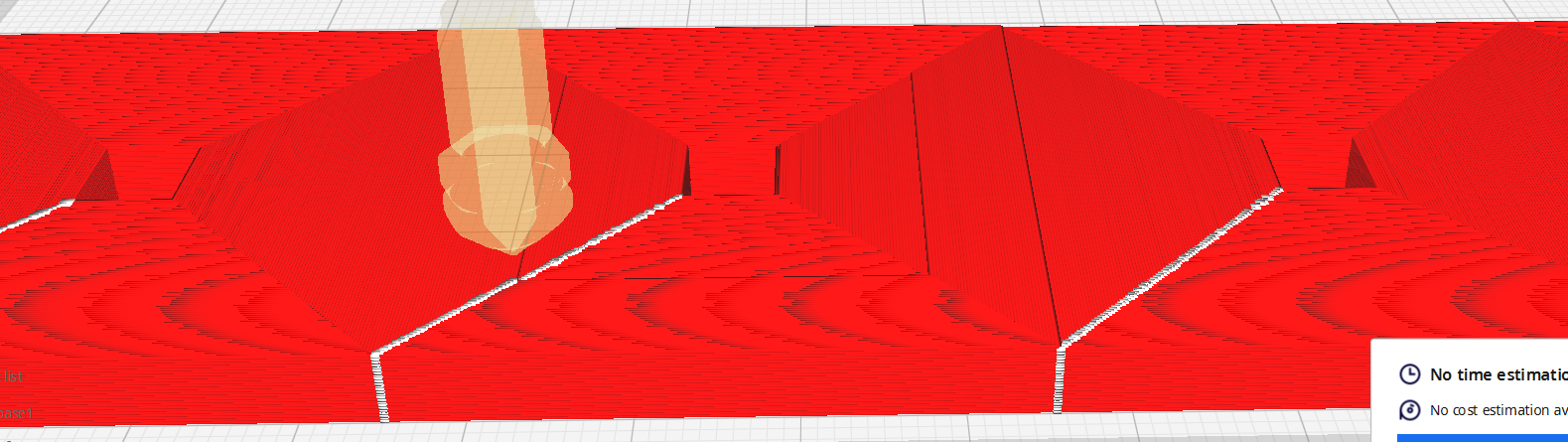
### Hole sequences

Then you combine these holes consecutively and make a base.



Similar to the series of sticks, we need to make sure we print 1 layer at a time:





Also, similar to the sticks, it is best to put all variables in an information .txt file and then read from there. The values for this execution are:

num id.l id.b od.l od.c[0] od.c[1] w h

4 6.8 6.8 40 10 10 0.2 20

Num is the number of pegs we will have. 4 sticks, 4 holes.

Note that we don’t need “ydist” or equivalent variable since the blocks need to be in contact with each other. So the distance between them is fixed at **od.l + w** so it covers all of the length and then adds another w to ensure there is no overlap while still keeping contact.

We keep od.l = 40mm because we want to give enough space between the pegs to insert and remove the beads easily. We still try to keep everything as small as possible to minimize material use and print time.

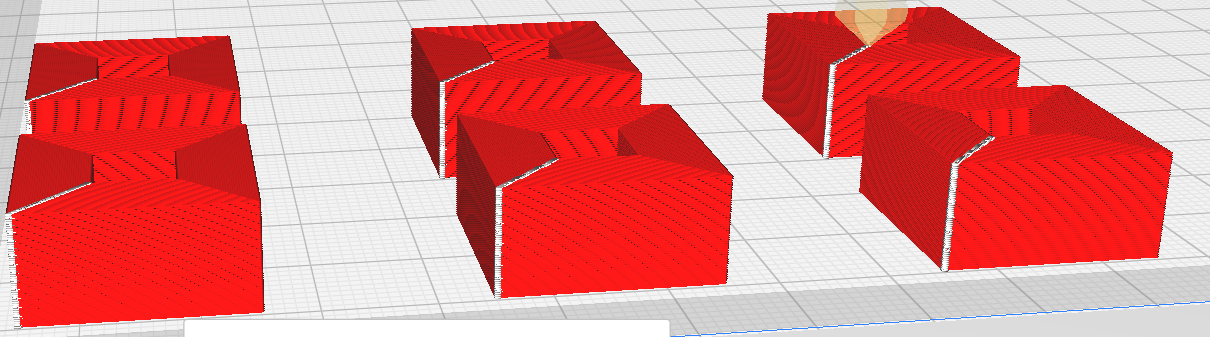
We choose a height of 20mm so that the sticks are stable and don’t bend or fall off. The id is 6.8 since we want to give 0.4mm (or 2 layers) all around clearance. This way, the stick will still fit even if it is a little distorted. But the fit will be relatively tight. If it is a little loose, you can add glue to the hole and/or coat the sticks with glue to execute.

Assignment: Write a function to call the hole function sequentially to make the base with 4 pegs

## Beads

It would be very boring to have beads that are exactly straight. So, we make beads that have a curve.

### Function to make array of Beads

First we make a function to make an array of the beads.

Note that this would require dynamic declaration of a 2D rect array with n rows and m columns.

Just like the base, here we just make holes repeatedly in different locations. This time, instead of a fixed distance, the x and y distances can be user given.

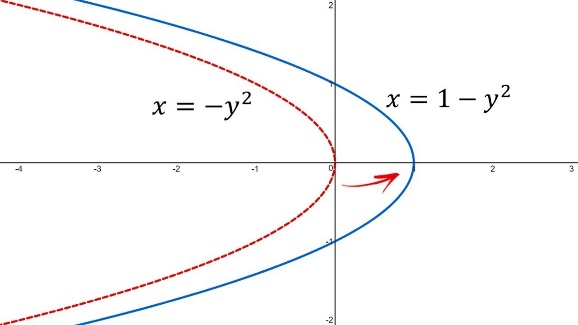
Assignment: Write a function to call the hole function sequentially to make a nxm 2d array of pegs

### Equation for curving

Here, we are taking the outer and inner curve surface as a parabola.

The equation for a horizontal parabola is: for passing through the origin.

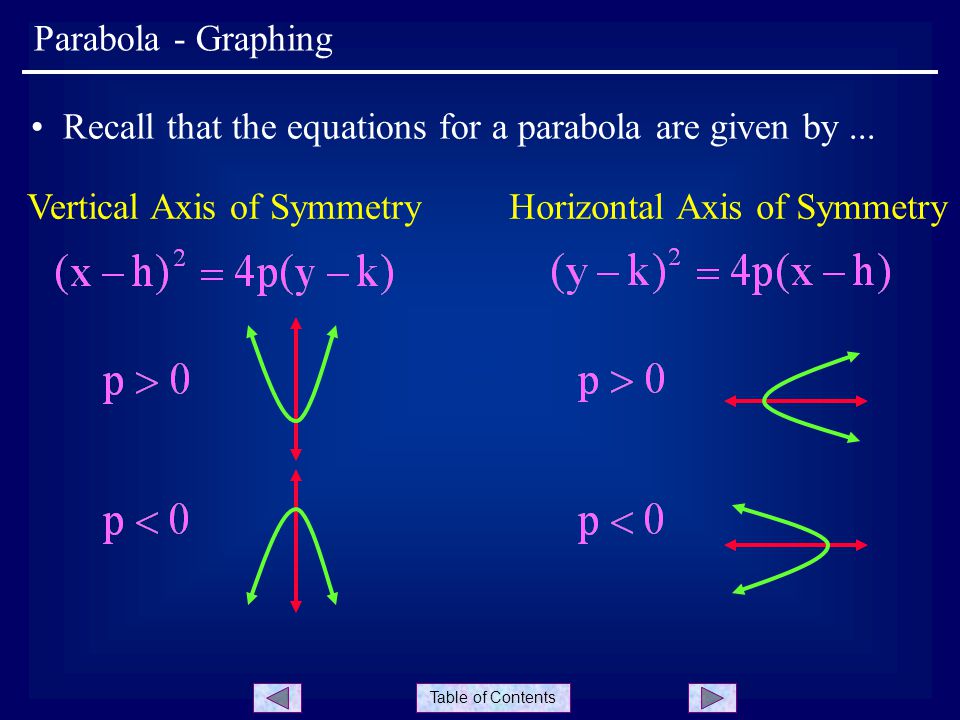
Here y is the height and x will be the length or breadth of the bead.



If the coefficient of x is positive, we get a concave shape. If it is negative, we get a positive shape. We want the bead to bulge outward and we want to see the equation between a max and a min dimension.

Also, if x or y have a value subtracted from them, the graph moves along the x or y axis.

The generalized equations are:



In our case, horizontal symmetry with p<0, we want the middle of the parabola to be at the midpoint of the height of the bead H. So, k = H/2.

We also want the length to be lmin when h=0 or h=H and length = lmax when h = H/2. So, the x axis offset would be lmax. Solving for the 2 end points h=0 and h=H we get a term for the difference of lmax and lmin. We use this as ldif.

x

y

lmax

lmin

ldif

H/2

H

So, the equation is:

x

y

lmax

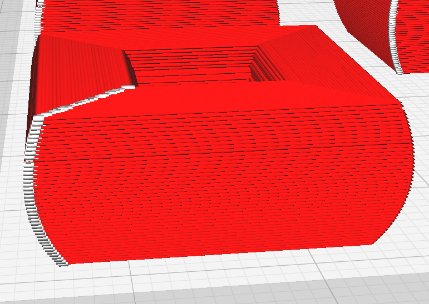
lmin

ldif

H/2

H

For the inside (the hole in the bead) we want it to be concave. So, the curve faces the other way and p>0 and everything else is the same. And the equation becomes:

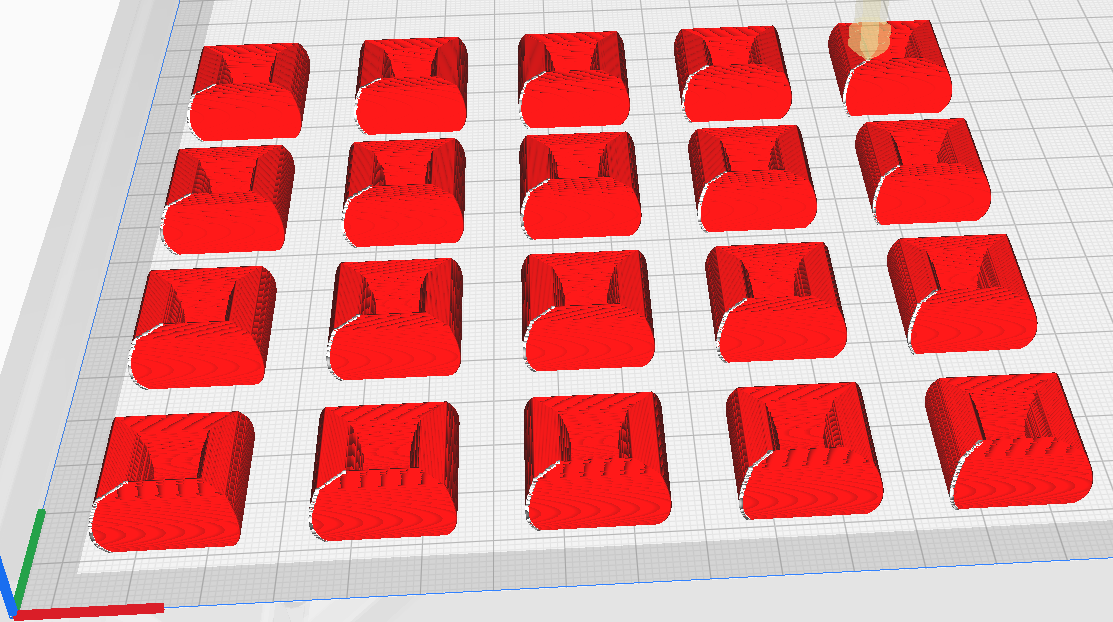


* Using these equations, we get beads that look like this.
* The ldif is the curve or the amount of indent we want.
* Note that the length and breadth values need to be changed at every layer.
* Also, the start point of the hole (origin c[0], c[1]) also changes here based on the height. So for each instance, change the height.

Here we use H and h values in the equation. However, in the loop we can use the current iteration variable (i) as h and replace H by the total number of layers.

Assignment: Incorporate the curving of the beads in the beads function and take data from a text file

### Curved beads array



The text file for this set of beads is:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rows(n) | Col(m) | l\_in | b\_in | l\_out | c[0] | c[1] | xdist | ydist | w | h | curve |
| 4 | 5 | 7 | 7 | 20 | 10 | 10 | 30 | 35 | 0.2 | 10 | 3 |

Where curve is the ldif.

Additional things you can try:

1. Use a different curve for the boundary surface instead of a parabola.
2. Think of and make another item using only rectangles or using the other shapes learned in session 4.