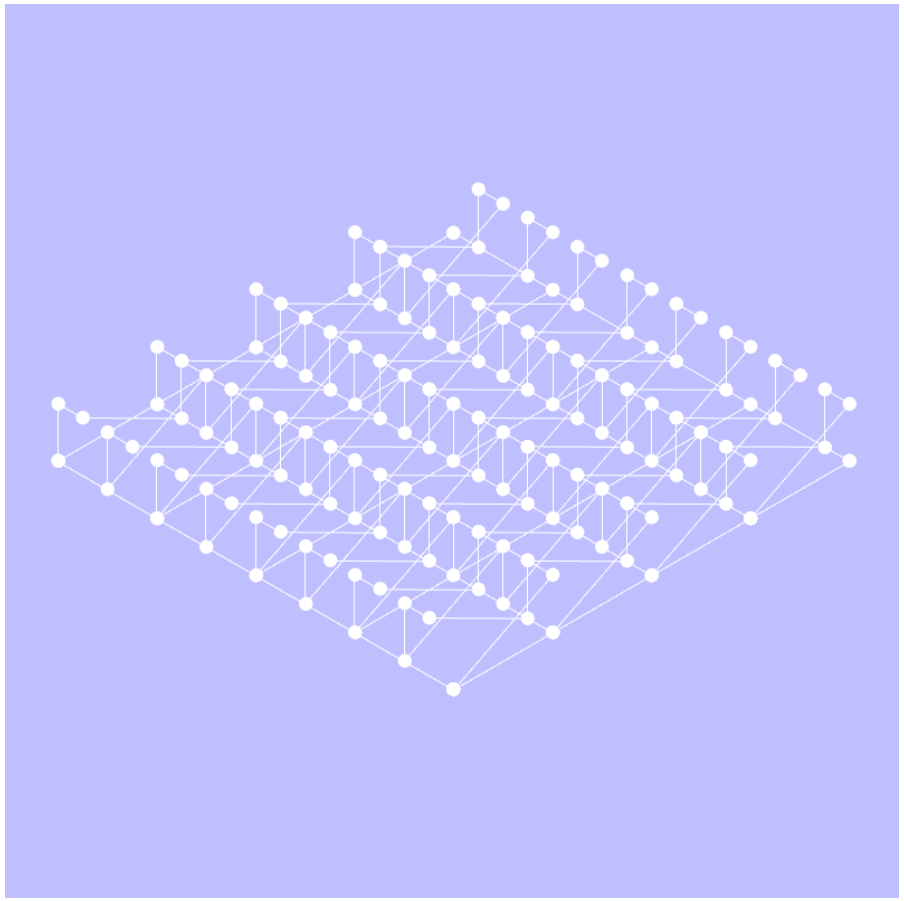


Gabled Urbanism

Design Goal

// This project is inspired by the urban fabric registered by **a continuous landscape of gable roofs** in historical European towns. The design will address this endless repetitive pattern of gable-ness through a code that makes the tool path “weave” around the mill board not only planarly but along the Z-axis as well to carve out the form.

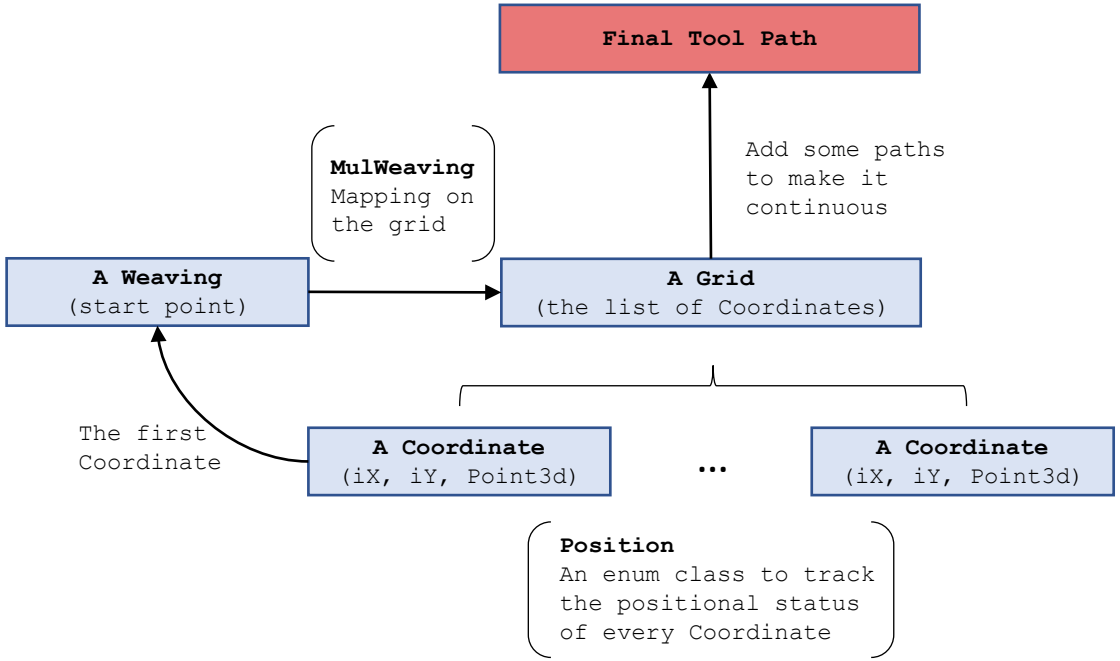
// Image:
https://www.britannica.com/technology/gable

The Code | Data Structure

```
98  //An enum for documenting the positional status of each Coordinate.
99  public enum Position...
105
106 //A Coordinate class that documents a point coordinate and its X,Y indices.
107 class Coordinate...
137
138 //A grid class that is defined by int res, double size, and a list of Coordinates.
139 class Grid...
257
258 //A Weaving class of the polyline moving along z-axis.
259 public class Weaving...
303
304 //A method to draw multiple weaving pattern.
305 public Polyline MulWeaving(Weaving w, int n)...
```

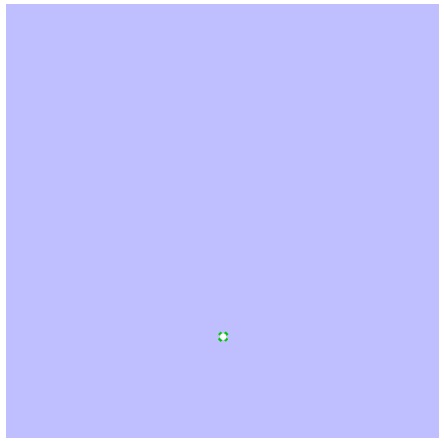
1
2
3

// The code is basically comprised of three major classes as components: the Coordinate class, the Grid class, and the Weaving class.



The Code | Data Structure
Coordinate Class

```
98    //An enum for documenting the positional status of each Coordinate.
99    public enum Position {
100     Inner,
101     OuterX,
102     OuterY,
103     Vertex
104   }
105
106   //A Coordinate class that documents a point coordinate and its X,Y indices.
107   class Coordinate {
108     public int iX, iY;
109     public Point3d origin;
110     public Position pos {get;set;}
111
112     public void ChangePosStatus(Position p) {
113       this.pos = p;
114     }
115
116     public Coordinate() {
117       iX = 0;
118       iY = 0;
119       origin = new Point3d();
120     }
121     public Coordinate(int x, int y, Point3d pt) {
122       this.iX = x;
123       this.iY = y;
124       this.origin = pt;
125     }
126     public void ChangeIndex (int x, int y) {
127       this.iX = x;
128       this.iY = y;
129     }
130     public Point3d DrawPt() {
131       return this.origin;
132     }
133     public void Translate (Vector3d v) {
134       this.origin = this.origin + v;
135     }
136   }
137
```



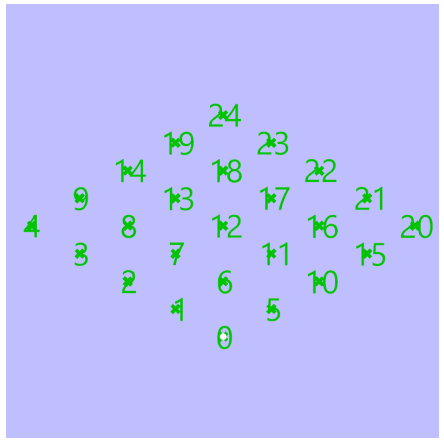
// A **Coordinate** is comprised of its geometric coordinate **Point 3d origin**, its X/Y indices in the Grid **int iX and iY**, and its Positional status **enum pos.**

// The starting point of everything is **a simple Coordinate(0, 0, Point3d())** with Position Inner.

// The Coordinate class has methods like ChangePosStatus, ChangeIndex, **DrawPt**, and Translate.

The Code | Data Structure
Grid Class

```
//A grid class that is defined by int res, double size, and a list of Coordinates.
class Grid {
  public double size;
  public int res;
  public List<Coordinate> cLst = new List<Coordinate>();

  public Grid() {
    size = 0.0;
    res = 0;
  }

  public Grid(int res, double size, Point3d start) {
    this.size = size;
    this.res = res;
    double interval = size / (res - 1);
    List<Coordinate> cLst = new List<Coordinate>();
    for (int i = 0; i < res; i++) {
      for (int j = 0; j < res; j++) {
        Coordinate c = new Coordinate(i, j, new Point3d(i * interval + start.X, j * interval + start.Y, start.Z));
        if (c.iX == res - 1 && c.iY == res - 1) {
          c.ChangePosStatus(Position.Vertex);
        } else if(c.iX == res - 1 && c.iY < res - 1) {
          c.ChangePosStatus(Position.OuterX);
        } else if (c.iY == res - 1 && c.iX < res - 1) {
          c.ChangePosStatus(Position.OuterY);
        } else {
          c.ChangePosStatus(Position.Inner);
        }
        cLst.Add(c);
      }
    }
    this.cLst = cLst;
  }

  public void ChangeRes(int r) {
    this.res = r;
    foreach (Coordinate c in this.cLst) {
      if (c.iX == r - 1 && c.iY == r - 1) {
        c.ChangePosStatus(Position.Vertex);
      } else if(c.iX == r - 1 && c.iY < r - 1) {
        c.ChangePosStatus(Position.OuterX);
      } else if (c.iY == r - 1 && c.iX < r - 1) {
        c.ChangePosStatus(Position.OuterY);
      } else {
        c.ChangePosStatus(Position.Inner);
      }
    }
  }

  public void ChangeSize(double s) {
    this.size = s;
  }

  public Coordinate GetCoordinate(int x, int y) {
    List<Coordinate> cLst = this.cLst;
    foreach (Coordinate c in cLst) {
      if (c.iX == x && c.iY == y) {
        return c;
      }
    }
    return new Coordinate();
  }

  public List<Point3d> DrawGridPts() {
    List<Coordinate> newLst = this.cLst;
    List<Point3d> result = new List<Point3d>();
    foreach (Coordinate item in newLst) {
      result.Add(item.DrawPt());
    }
    return result;
  }

  public void TranslateGrid(Vector3d v) {
    foreach (Coordinate c in this.cLst) {
      c.origin = c.origin + v;
    }
  }
}
```



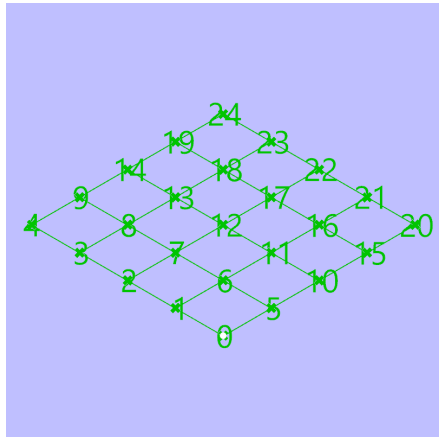
// A **Grid** is comprised of **a list of Coordinates cLst**, its resolution **int res**, and its size **double size**.

// The constructor of Grid class **reassigns the Position class** to each Coordinate element in its cLst, for instance, the 4/9/14/19 will be assigned to OuterY, the 20/21/22/23 will be assigned to OuterX, the 24 will be Vertex, and the rest will be Inner.

// The Grid class has methods like ChangeRes, ChangeSize, **DrawGridPts**, and TranslateGrid.

The Code | Data Structure
Grid Class

```
217     public Polyline DrawGridLne() {
218       List<Point3d> pts = this.DrawGridPts();
219       Polyline result = new Polyline();
220       int k = 0;
221
222       Point3d temp_pt = this.GetCoordinate(0, 0).origin;
223       for(int j = 0; j < res; j++){
224         for(int i = 0; i < res; i++){
225           if(j % 2 == 0){
226             temp_pt = pts[k];
227             result.Add(temp_pt);
228           }else{
229             temp_pt = pts[k + res - 1 - 2 * i];
230             result.Add(temp_pt);
231
232           }
233           k++;
234         }
235       }
236       result.Add(this.GetCoordinate(0, res - 1).origin);
237       result.Add(this.GetCoordinate(0, 0).origin);
238
239       int l = 0;
240       temp_pt = this.GetCoordinate(0, 0).origin;
241       for(int j = 0; j < res; j++){
242         for(int i = 0; i < res; i++){
243           if(j % 2 == 0){
244             temp_pt = pts[i * res + j];
245             result.Add(temp_pt);
246           }else{
247             temp_pt = pts[l + (res - j) * (res - 1) - i * (res + 1)];
248             result.Add(temp_pt);
249
250           }
251           l++;
252         }
253       }
254
255       return result;
256     }
257   }
```

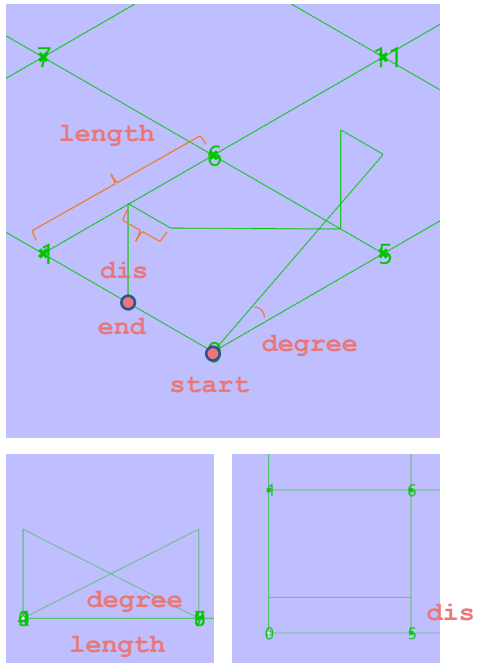


// The most important method in Grid class is **DrawGridLne**. This method allows a continuous tool path going through the whole grid with minimal length of overlapped path.

// It basically goes as s-shape in one direction to draw the path along the Y-axis and then change its direction to draw it along the X-axis.

The Code | Data Structure
Weaving Class

```
259   //A Weaving class of the polyline moving along z-axis.
260   public class Weaving {
261     public Point3d start = new Point3d();
262     public Point3d end = new Point3d();
263     public double dis;
264     public double length;
265     public double degree;
266
267     public Weaving() {
268     }
269
270     public Weaving(Point3d start, double dis, double length, double degree) {
271       this.start = start;
272       this.end = new Point3d(start.X, start.Y + dis * 2.0, start.Z);
273       this.dis = dis;
274       this.length = length;
275       this.degree = degree;
276     }
277
278     public void ChangeStart() {
279       Point3d newStart = this.end;
280       this.start = newStart;
281       this.end = new Point3d(this.start.X, this.start.Y + dis * 2.0, this.start.Z);
282     }
283
284     public Polyline DrawOne () {
285       Polyline poly = new Polyline();
286       double dz = Math.Tan(this.length) * Math.PI * this.degree;
287       poly.Add(this.start);
288       Point3d pt2 = new Point3d(start.X + this.length, start.Y, start.Z + dz);
289       Point3d pt3 = new Point3d(pt2.X, pt2.Y + this.dis, pt2.Z);
290       Point3d pt4 = new Point3d(pt3.X, pt3.Y, pt3.Z - dz);
291       Point3d pt5 = new Point3d(pt4.X - this.length, pt4.Y, pt4.Z + dz);
292       Point3d pt6 = new Point3d(pt5.X, pt5.Y + this.dis, pt5.Z);
293
294       poly.Add(pt2);
295       poly.Add(pt3);
296       poly.Add(pt4);
297       poly.Add(pt5);
298       poly.Add(pt6);
299       poly.Add(this.end);
300
301       return poly;
302     }
303   }
```



Section Plan

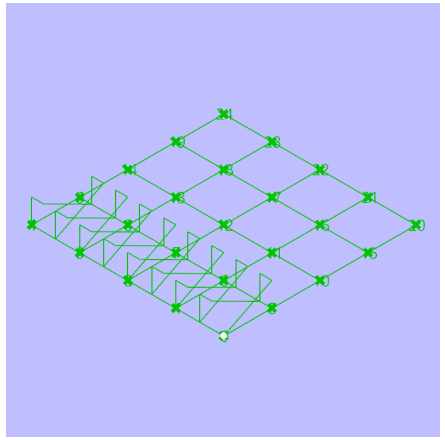
// A Weaving class generates a polyline that forms **a shape of X in sectional view** through its DrawOne function.

// The constructor takes a start point, the degree of the inclining angle, the distance of the turn, and the length of the whole span.

// The **ChangeStart** method will be used to use the current "end" point as a "start" for next Weaving segment.

The Code | Data Structure
Weaving Class

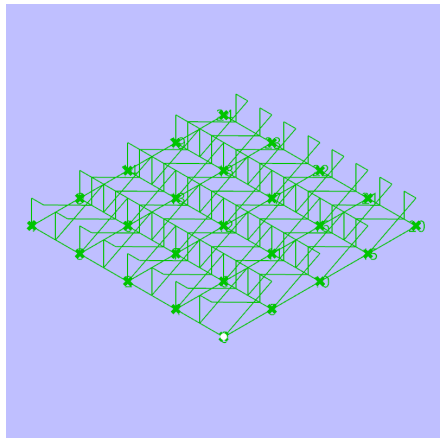
```
309    //A method to draw multiple weaving pattern.
310    public Polyline MulWeaving(Weaving w, int n){
311      int i = 0;
312      Polyline poly = new Polyline();
313      while (i < n){
314        Polyline result = w.DrawOne();
315        poly.AddRange(result);
316        w.ChangeStart();
317        i++;
318      }
319      return poly;
320    }
```



// The **MulWeaving** method is used to generate multiple Weaving segment into a polyline by using **ChangeStart** through a **while loop.**

The Code | Tool Path Logic
Main Function

```
55  private void RunScript(Point3d start, double size, double degree, int res, int density, ref object _polyline, ref object _timecost, ref object _length, ref object _text, ref object A, ref object B)
56  {
57
58    Grid g = new Grid(res, size, start);
59    double length = size / (res - 1);
60    int nunmberOfTurns = density;
61    double dis = length / (density * 2);
62
63    List<string> text = new List<string>();
64    text.Add("start milling");
65
66
67    int n = 0;
68    Polyline result = new Polyline();
69    while (n < res - 1){
70      Point3d pt = g.GetCoordinate(n, 0).origin;
71      Weaving w = new Weaving(pt, dis, length, degree);
72      Polyline onePoly = MulWeaving(w, density * (res - 1));
73      result.AddRange(onePoly);
74
75      result.Add(pt);
76      n++;
77    }
78
79    A = g.DrawGridPts();
80    B = g.DrawGridLne();
81    result.AddRange(g.DrawGridLne());
82
83    result.MergeColinearSegments(0.01, false);
84
85    double time = result.Length + result.Count * 3;
86
87    text.Add("end milling");
88    _polyline = result;
89    _text = text;
90    _length = result.Length;
91    _timecost = time;
92
```



// **Step 1**: Input parameters(**res**, **size**, **start point**) to create a **Grid**.

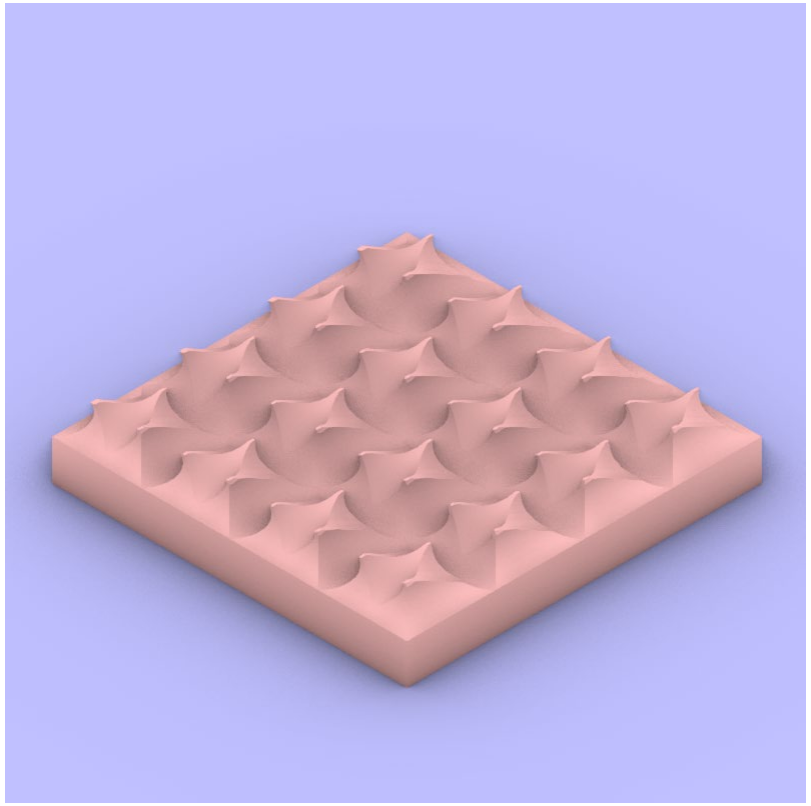
// **Step 2**: Use the **density** parameter to set the **dis** variable for **Weaving**.

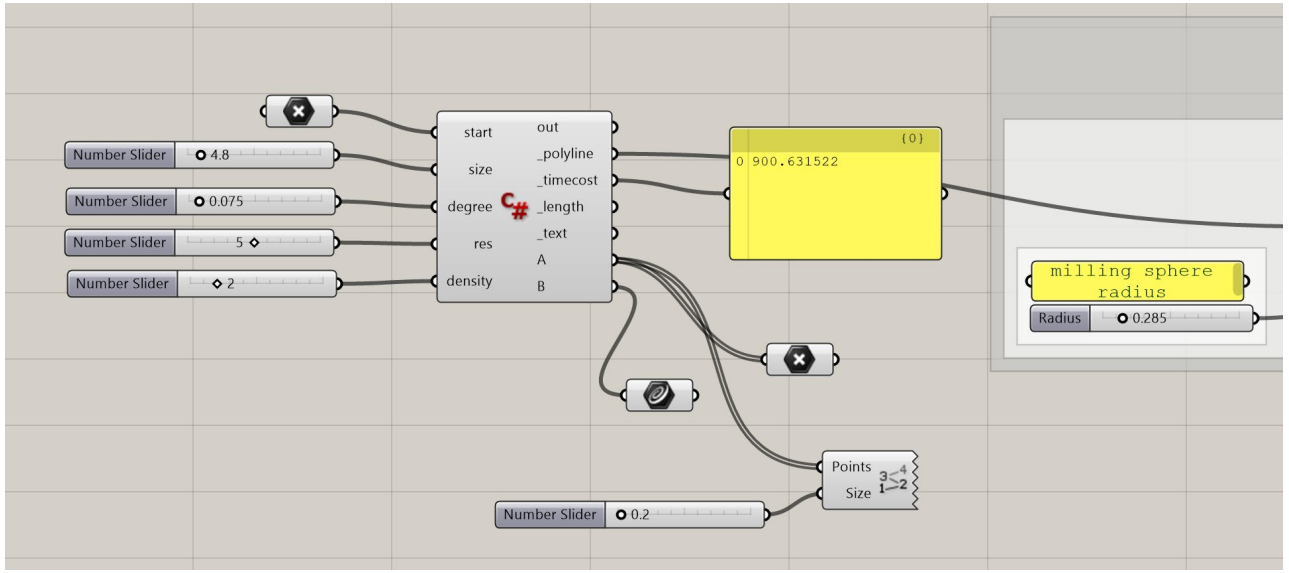
//**Step 3**: Use a while loop to draw multiple Weavings with the parameter **degree** on every column of the Grid, and make sure the **tool path always return to the bottom** to start drawing.

//**Step 4**: **Concatenate** two lists of polylines—the grid and the weaving pattern—and make sure that are **one continuous path**.

//**Step 5**: Calculate the **time** and the **length**.

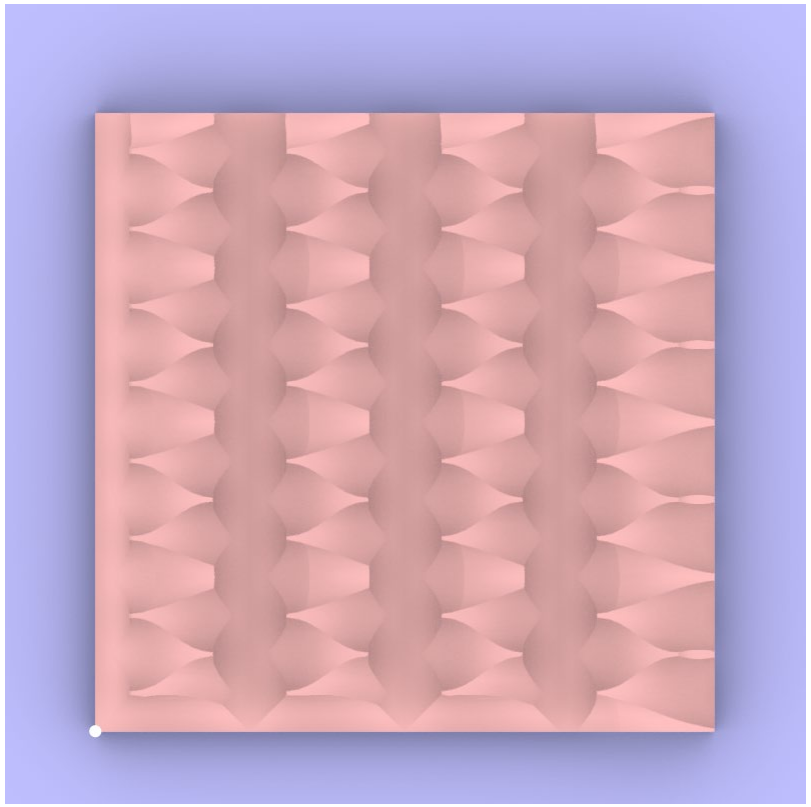
The Design | Iteration 1
A steep gabled urbanism



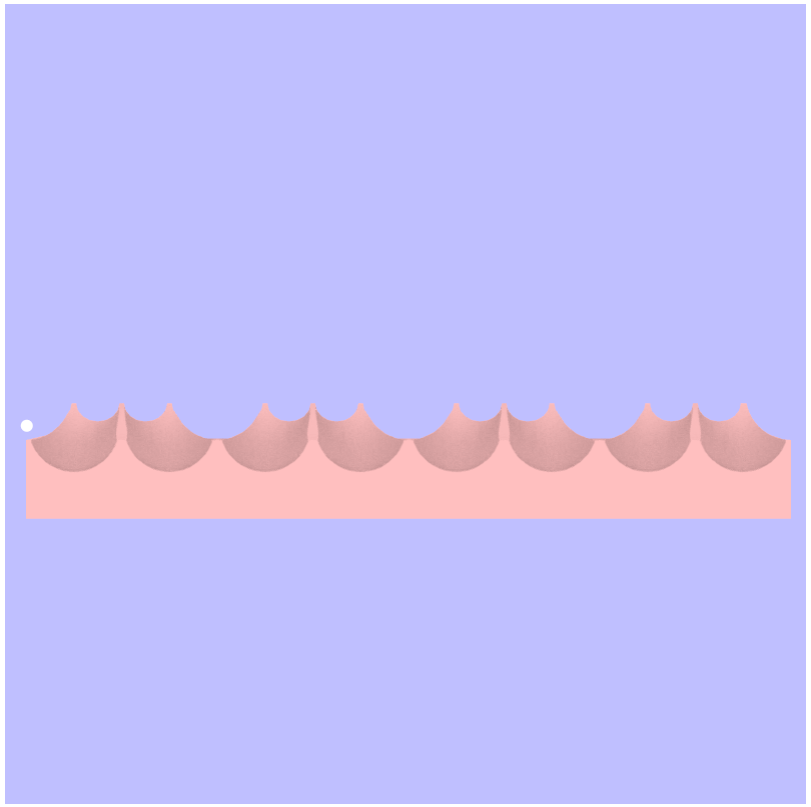


// Configuration:
Size: 4.8; **degree**: 0.075 (*Pi); **res**: 5; **density**: 2

The Design | Iteration 1
A steep gabled urbanism



Plan

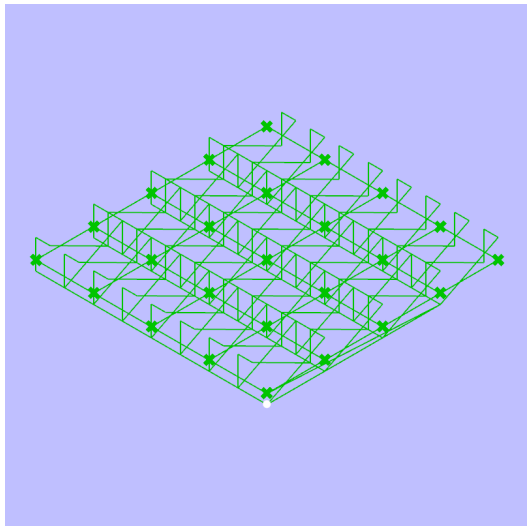


Section

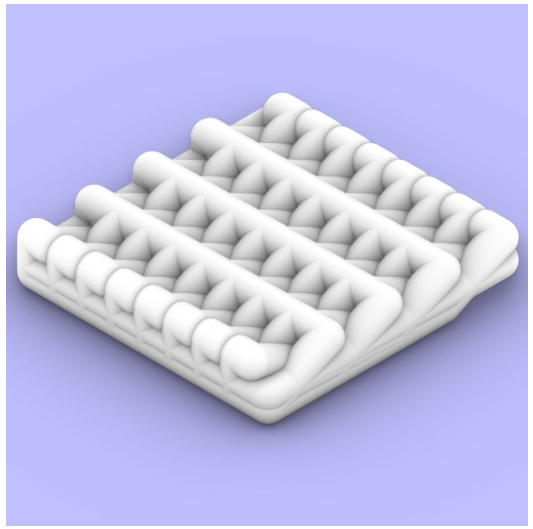
```
g.TranslateGrid(new Vector3d(0.0, 0.0, 0.2));

// Slightly adjust the grid to make sure there is not undercuts on Z-axis.
```

VIS 2225 | Assignment
Yubo Zhao

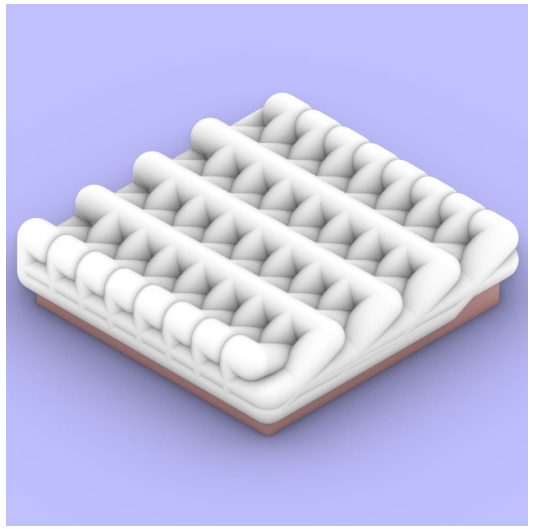
The Design | Iteration 1
A steep gabled urbanism



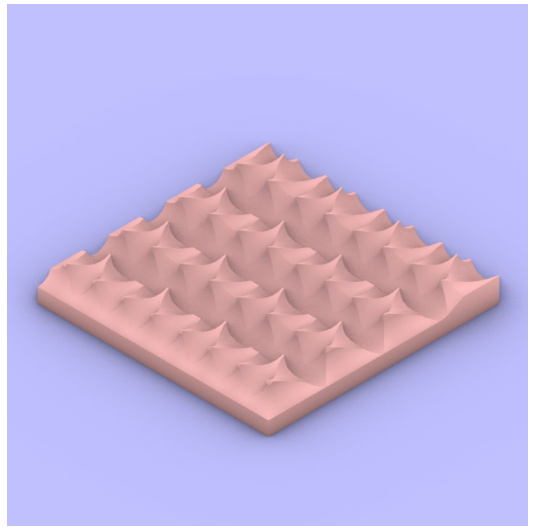
Tool Path



Nozzle

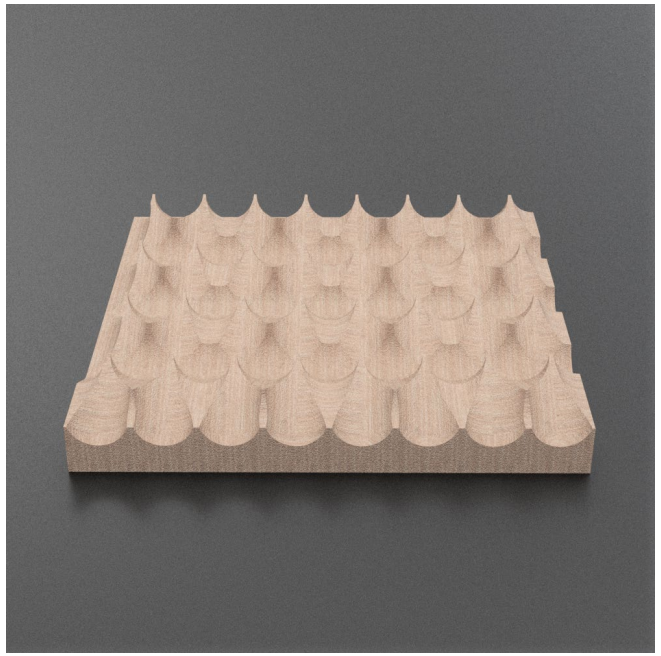


Boolean Out



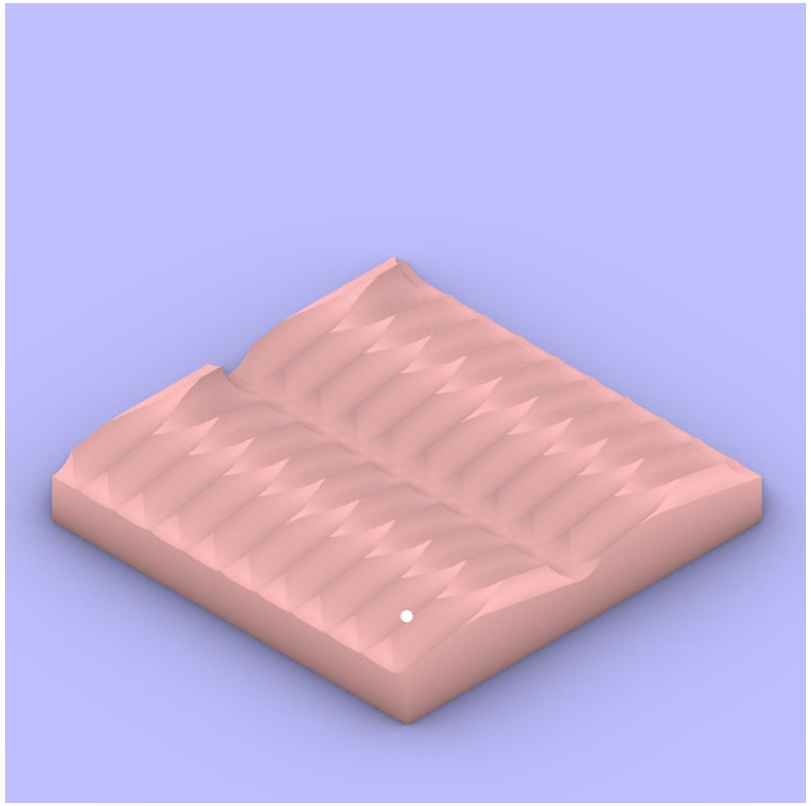
Milled Block

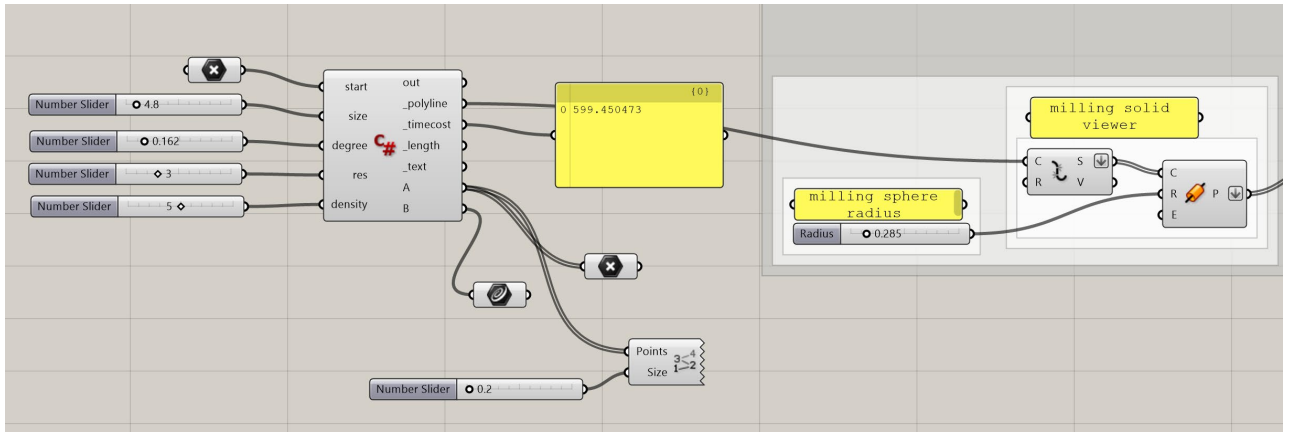
The Design | Iteration 1
A steep gabled urbanism



// This iteration creates a 4*4 city block with steep and pointy gables. At the street level a continuous set of building-scale facades is registered through the steep gables.

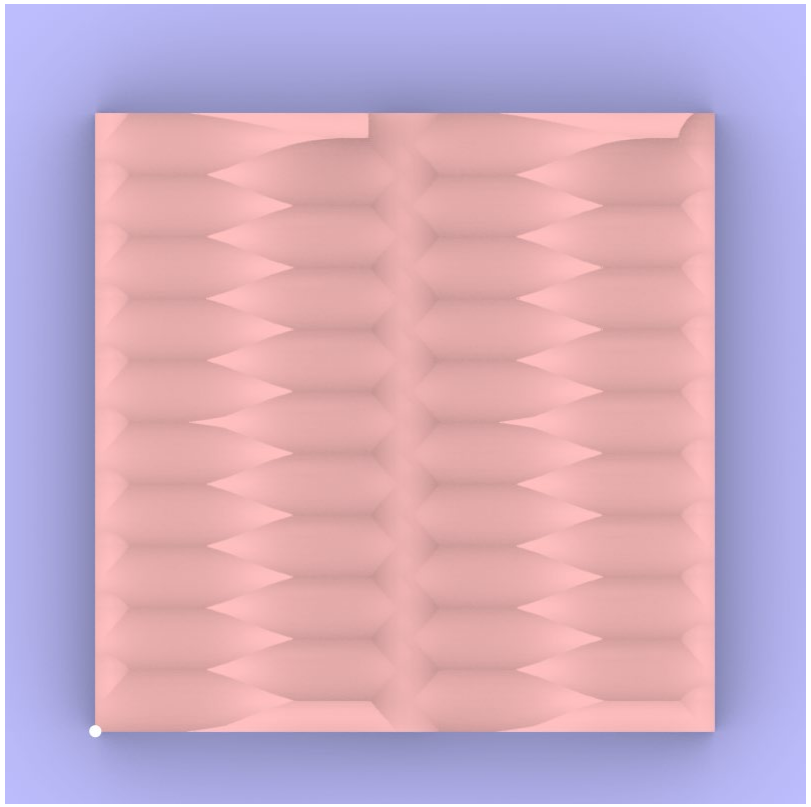
The Design | Iteration 2
A gradual gabled urbanism



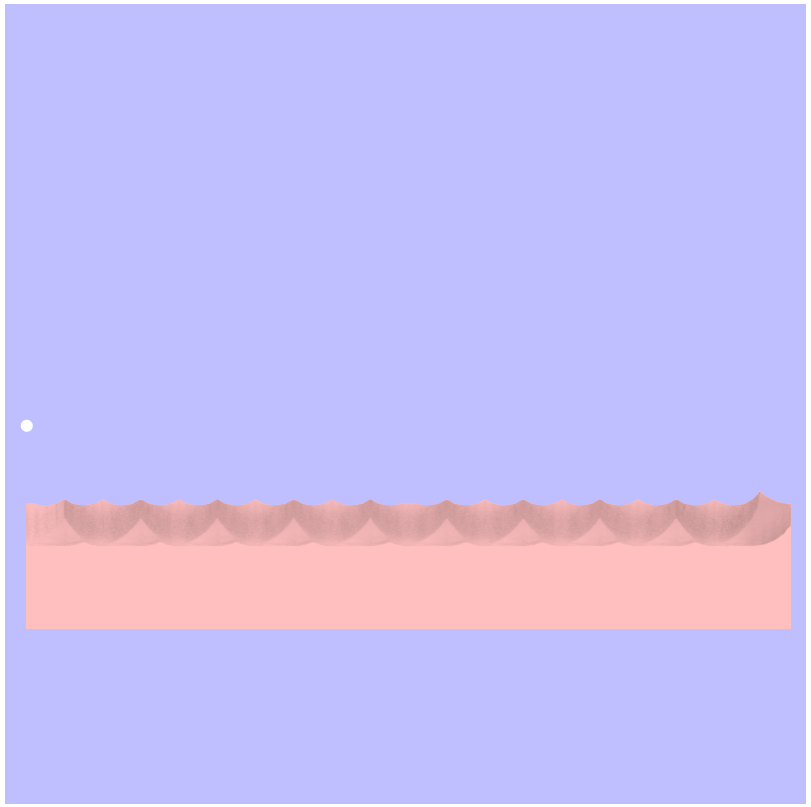



```
// Configuration:
Size: 4.8; degree: 0.162 (*Pi); res: 3; density: 5

// Lower resolution but higher density of gables
```

The Design | Iteration 2
A gradual gabled urbanism



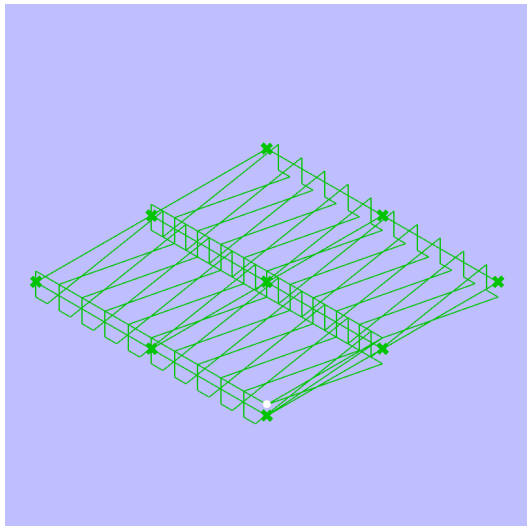
Plan



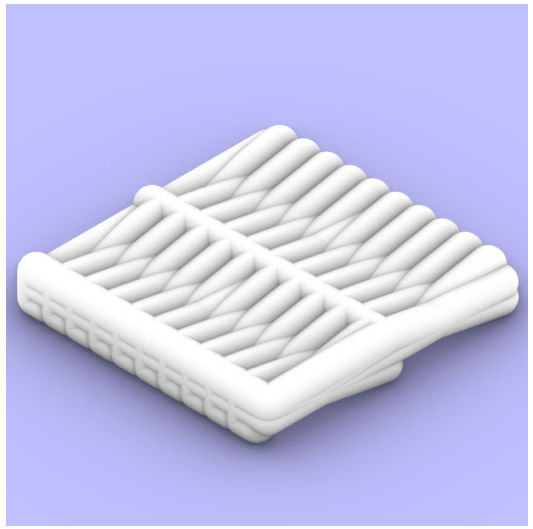
Section

```
g.TranslateGrid(new Vector3d(0.0, 0.0, -0.2));

// Slightly adjust the grid to make sure there is not undercuts on Z-axis.
```

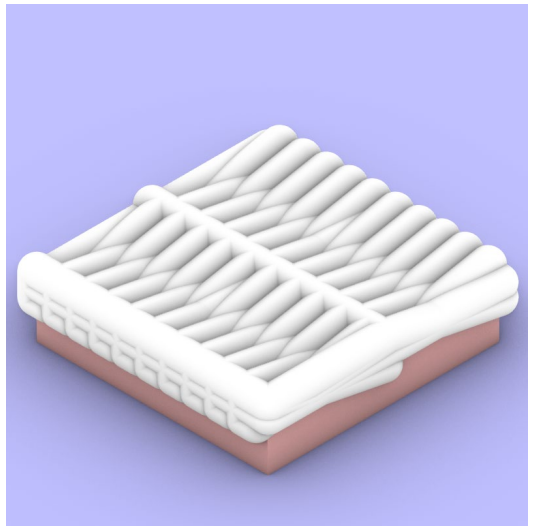
The Design | Iteration 2
A gradual gabled urbanism



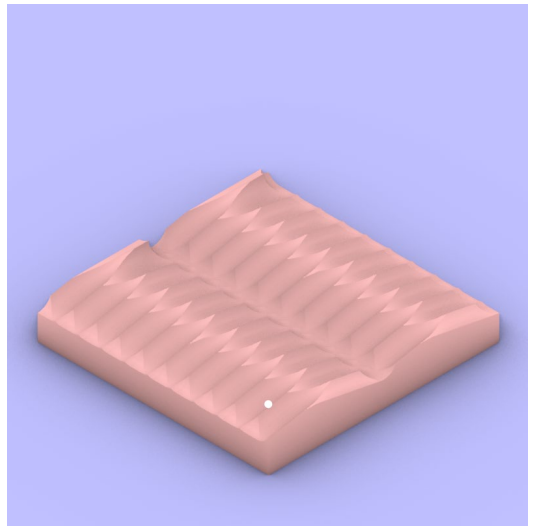
Tool Path



Nozzle



Boolean Out



Milled Block

The Design | Iteration 2
A gradual gabled urbanism

// This iteration creates a 2*2 city block with gradual and flat gables. At the street level the gables have not become architecture yet but remains in a semi-landscape and semi-building formal gesture.