

GSD - SCI 6338  
Introduction to Computational Design  
Assignment 1: Local-Global  
Togo Kida, Hanif Wicaksono

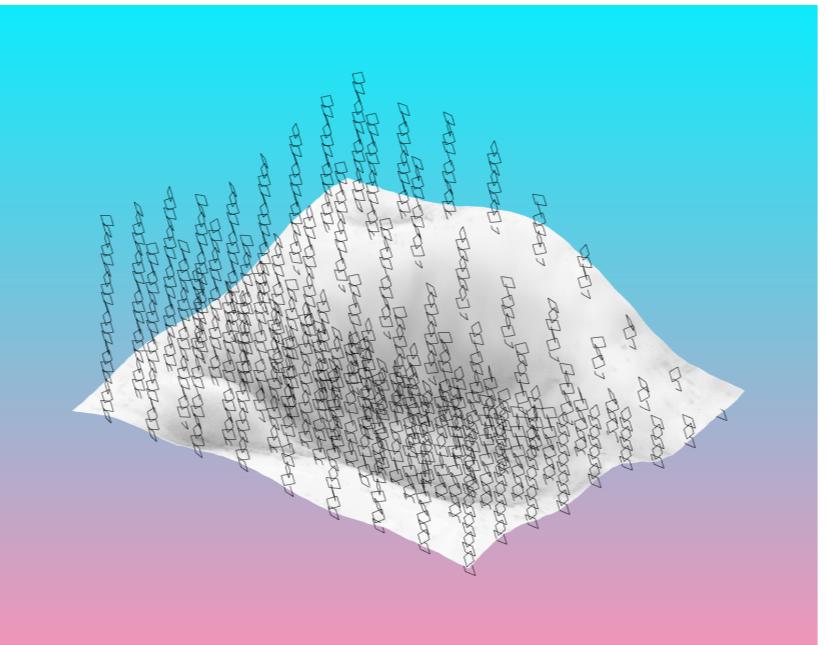
## Concept



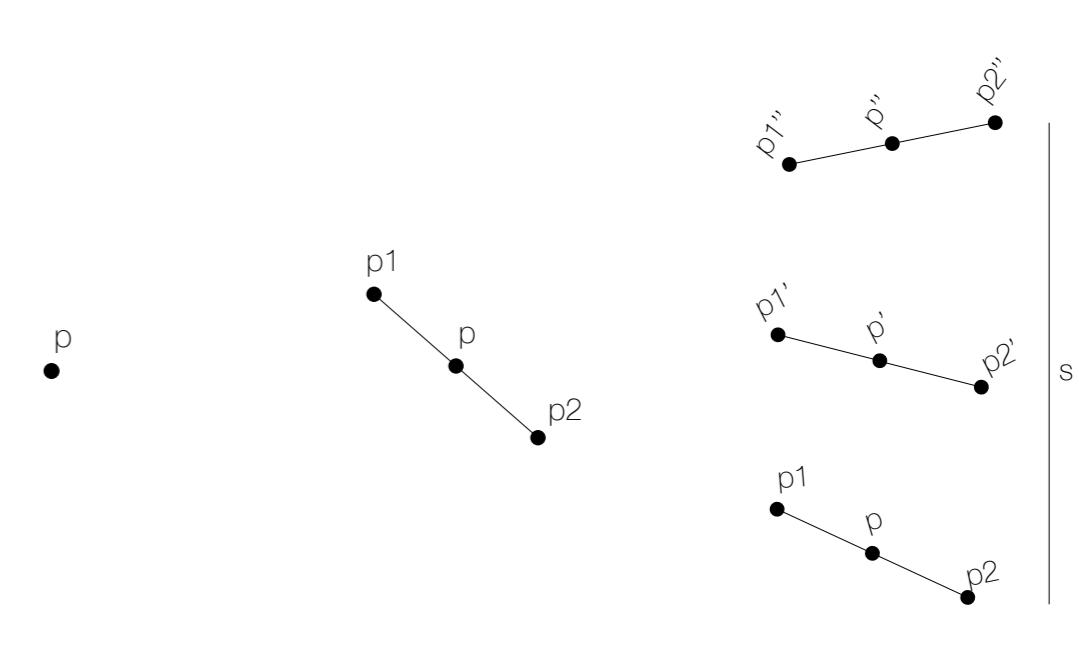
For this assignment, we are interested in landscape scenario since we get the chance to modify our objects in 3 dimensional space. What would it look like if there is actually a man-made landscape in the moon? We took a sample of a landscape in the moon, particularly a crater and tried to create an abstract representation, akin to sculpture. **Togo was responsible for creating the local component and Hanif was responsible for the global.**

## Global - Local

### Global



### Local



We take a sample of crater and using the premade proto landscape component, we set the width and height into 80, and put the elevation/slope value into 30 to set our landscape. We intend to populate it with sculptural objects that will resonate with the single focus point.

#### Global (focus point):

Create lists of points, polylines, & vectors

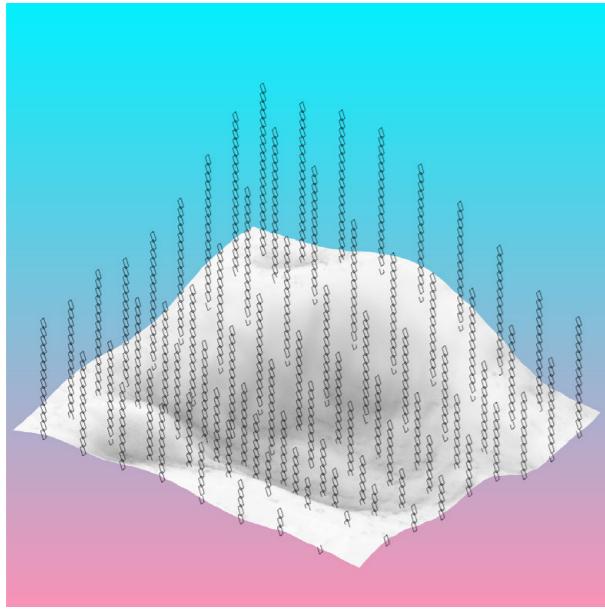
Decide where to populate the local

Number of local iterations made

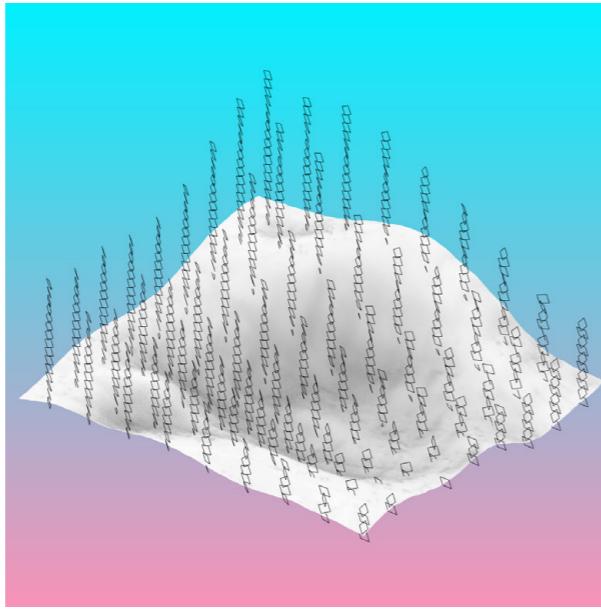
Make rotations when focus moves

This local component takes the input p as the starting point to construct the geometry. After defining p as the middle point, it then calculates p2 and p1 to define the range of the rectangle. The input s is used to determine numIterations which defines the number of times the rectangular base form is elevated. Each time the base form is elevated, the form is rotated by a certain angle which is determined by the angle between p and the focus point. Therefore this gives the structure an unique conglomerate form.

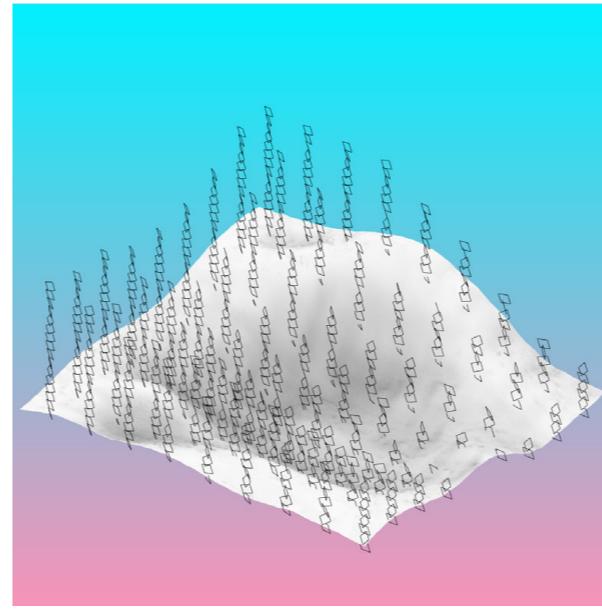
## Variations



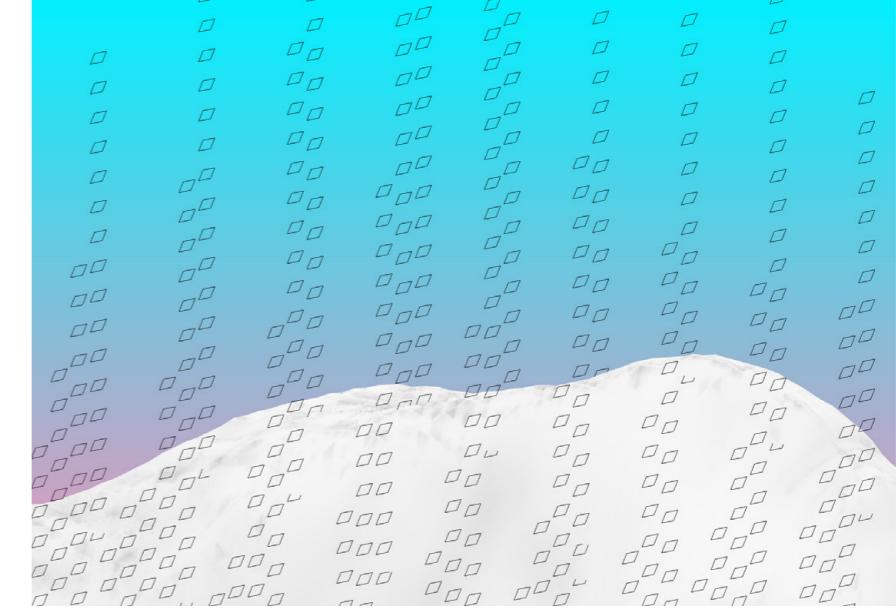
Point3d focus (x,y,z) : 0,0,0



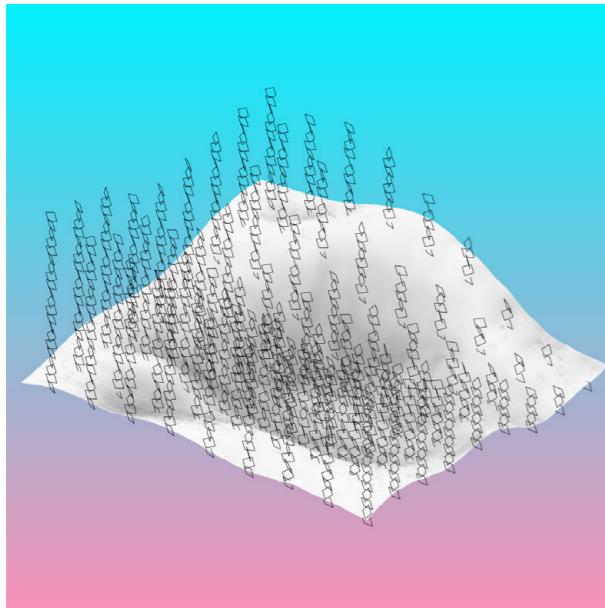
Point3d focus (x,y,z) : 20,0,0



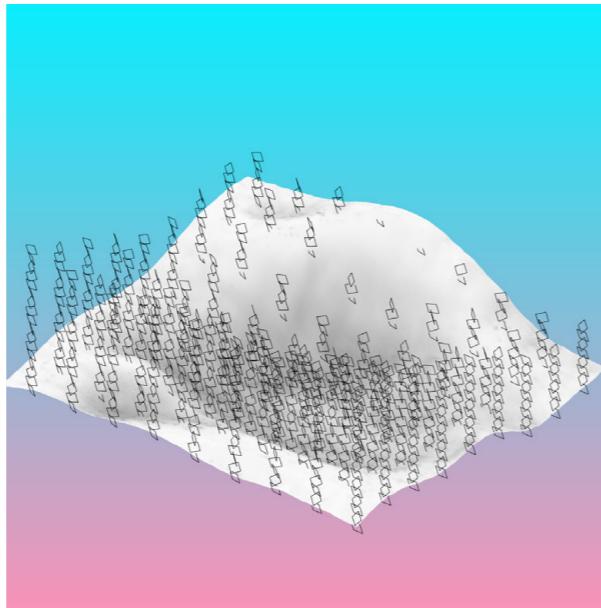
Point3d focus (x,y,z) : 40,0,0



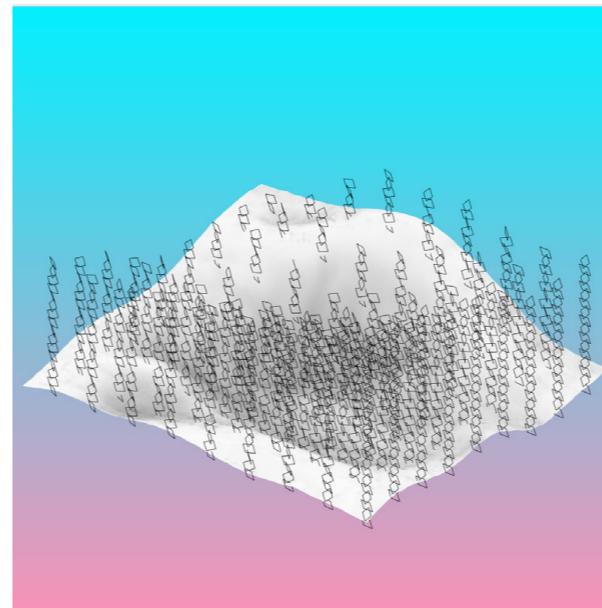
local components without focus point



Point3d focus (x,y,z) : 0,0,0



Point3d focus (x,y,z) : 20,0,0



Point3d focus (x,y,z) : 40,0,0



Local components rotated to focus point

## Code (Global)

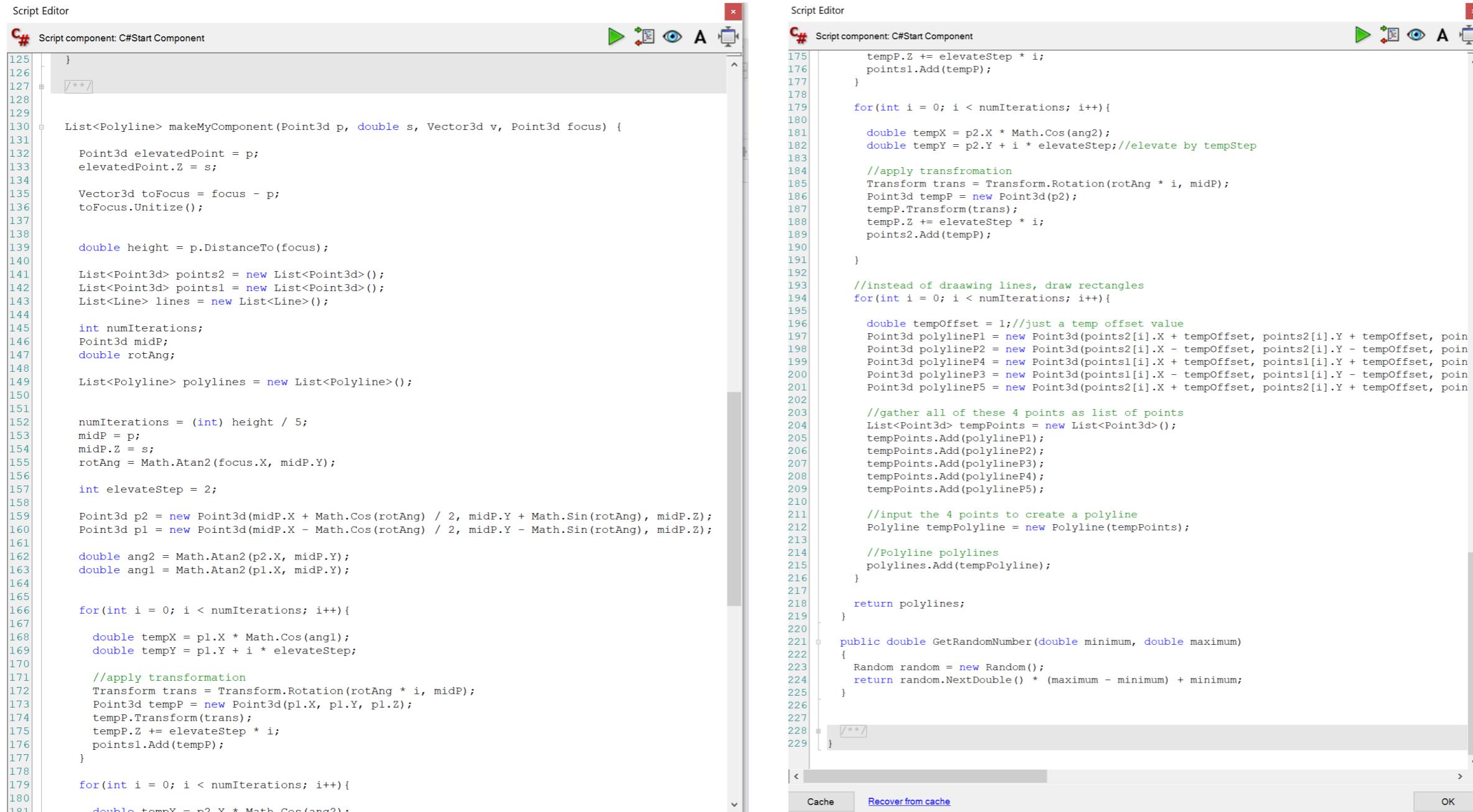
```
1  using
2
3
4
5  /// <summary>
6  /// This class will be instantiated on demand by the Script component.
7  /// </summary>
8  public class Script_Instance : GH_ScriptInstance
9  {
10     Utility functions
11
12     Members
13
14     /**
15      private void RunScript(List<double> s, List<Vector3d> v, int ry, int rx, double sizeX, double s
16
17      //START GLOBAL 0
18
19      //creater an empty list to add the lines for the output
20      List<Line> lines = new List<Line>();
21
22      //creater an empty list to add the lines for the output
23      List<Polyline> polylines = new List<Polyline>();
24
25      //create an empty list of vectors for the focus point
26      List<Vector3d> vectors = new List<Vector3d>();
27
28      //create an empty list of point to put the local components
29      List<Point3d> points = new List<Point3d>();
30
31      //END GLOBAL 0
32
33      //compute steps of grid so that we can calculate the physical locations of points from the rc
34      double dx = sizeX / (double) (rx - 1.0);
35      double dy = sizeY / (double) (ry - 1.0);
36
37      //start looping through all possible index pairs in the grid ry x rx
38      for (int j = 0; j < ry; ++j) {
39          for(int i = 0; i < rx; ++i) {
40              Point3d p = new Point3d(i * dx, j * dy, 0.0);
41              //compute the physical point location on the XY plane
42              int k = j * rx + i;
43              //compute the serial index so that we can pickup values from the lists s and v
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
```

Script Editor

C# Script component: C#Start Component

```
83     //compute the serial index so that we can pickup values from the lists s and v
84
85
86     //START GLOBAL1
87
88     //Set up focus point to vector
89     Vector3d toFocus = focus - p;
90     toFocus.Unitize();
91
92     //Set a limit to the vector length
93     toFocus *= 1.50;
94
95     //Connecting the focus point to vector list
96     vectors.Add(toFocus);
97
98     //Populate the points with connected vector
99     points.Add(p + toFocus);
100
101    //END GLOBAL1
102
103    //GLOBAL 2
104
105    //Set up a modulo for the grid to make four walls
106    //Populate based on slope value
107    if (i % 9 == 0 && j % 9 == 0 && s[k] > 0 )
108
109    //GLOBAL 2
110
111    {
112        //call the makeComponent method from below to get hold of the local wireframe geometry
113        List<Polyline> polylineComponent = makeMyComponent(p, s[k], vectors[k], focus);
114        polylines.AddRange(polylineComponent);
115
116    }
117
118    }
119
120
121
122    B = lines;
123    A = polylines;//adjust to this guy
124
```

## Code (Local)



The image shows two side-by-side Script Editor windows, both titled "Script component: C#Start Component".

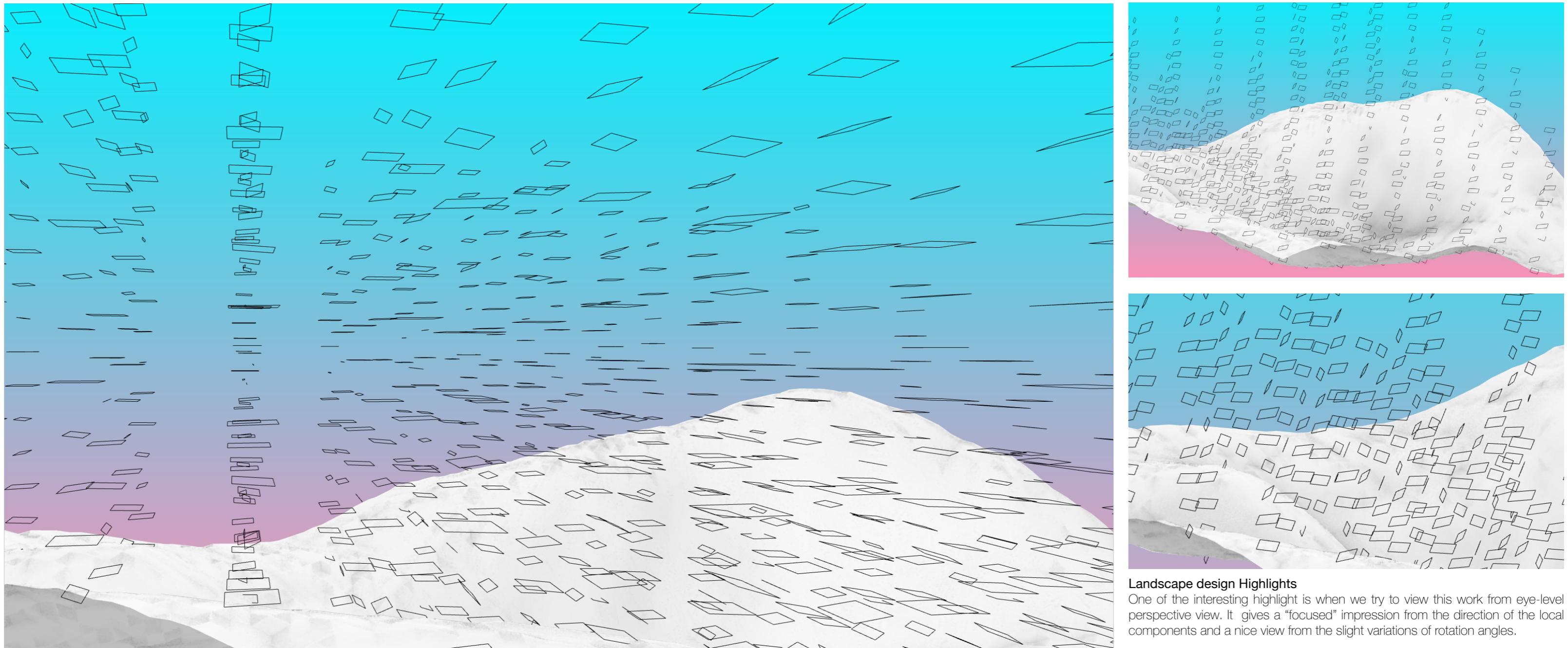
**Left Window:**

```
125 }
126 /**
127 *
128 *
129 * List<Polyline> makeMyComponent(Point3d p, double s, Vector3d v, Point3d focus) {
130     Point3d elevatedPoint = p;
131     elevatedPoint.Z = s;
132
133     Vector3d toFocus = focus - p;
134     toFocus.Unitize();
135
136     double height = p.DistanceTo(focus);
137
138     List<Point3d> points2 = new List<Point3d>();
139     List<Point3d> points1 = new List<Point3d>();
140     List<Line> lines = new List<Line>();
141
142     int numIterations;
143     Point3d midP;
144     double rotAng;
145
146     List<Polyline> polylines = new List<Polyline>();
147
148     numIterations = (int) height / 5;
149     midP = p;
150     midP.Z = s;
151     rotAng = Math.Atan2(focus.X, midP.Y);
152
153     int elevateStep = 2;
154
155     Point3d p2 = new Point3d(midP.X + Math.Cos(rotAng) / 2, midP.Y + Math.Sin(rotAng), midP.Z);
156     Point3d p1 = new Point3d(midP.X - Math.Cos(rotAng) / 2, midP.Y - Math.Sin(rotAng), midP.Z);
157
158     double ang2 = Math.Atan2(p2.X, midP.Y);
159     double ang1 = Math.Atan2(p1.X, midP.Y);
160
161
162     for(int i = 0; i < numIterations; i++){
163         double tempX = p1.X * Math.Cos(ang1);
164         double tempY = p1.Y + i * elevateStep;
165
166         //apply transformation
167         Transform trans = Transform.Rotation(rotAng * i, midP);
168         Point3d tempP = new Point3d(p1.X, p1.Y, p1.Z);
169         tempP.Transform(trans);
170         tempP.Z += elevateStep * i;
171         points1.Add(tempP);
172     }
173
174     for(int i = 0; i < numIterations; i++){
175         double tempX = p2.X * Math.Cos(ang2);
176         double tempY = p2.Y + i * elevateStep;
177
178         //apply transformation
179         Transform trans = Transform.Rotation(rotAng * i, midP);
180         Point3d tempP = new Point3d(p2.X, p2.Y, p2.Z);
181         tempP.Transform(trans);
182         tempP.Z += elevateStep * i;
183         points2.Add(tempP);
184
185         //gather all of these 4 points as list of points
186         List<Point3d> tempPoints = new List<Point3d>();
187         tempPoints.Add(polylineP1);
188         tempPoints.Add(polylineP2);
189         tempPoints.Add(polylineP3);
190         tempPoints.Add(polylineP4);
191
192         //input the 4 points to create a polyline
193         Polyline tempPolyline = new Polyline(tempPoints);
194
195         //Polyline polylines
196         polylines.Add(tempPolyline);
197
198     }
199
200     return polylines;
201 }
202
203 public double GetRandomNumber(double minimum, double maximum)
204 {
205     Random random = new Random();
206     return random.NextDouble() * (maximum - minimum) + minimum;
207 }
208
209 /**
210 */
211 }
```

**Right Window:**

```
175     tempP.Z += elevateStep * i;
176     points1.Add(tempP);
177 }
178
179 for(int i = 0; i < numIterations; i++){
180
181     double tempX = p2.X * Math.Cos(ang2);
182     double tempY = p2.Y + i * elevateStep;//elevate by tempStep
183
184     //apply transformation
185     Transform trans = Transform.Rotation(rotAng * i, midP);
186     Point3d tempP = new Point3d(p2);
187     tempP.Transform(trans);
188     tempP.Z += elevateStep * i;
189     points2.Add(tempP);
190
191 }
192
193 //instead of drawing lines, draw rectangles
194 for(int i = 0; i < numIterations; i++){
195
196     double tempOffset = 1;//just a temp offset value
197     Point3d polylineP1 = new Point3d(points2[i].X + tempOffset, points2[i].Y + tempOffset, points2[i].Z);
198     Point3d polylineP2 = new Point3d(points2[i].X - tempOffset, points2[i].Y - tempOffset, points2[i].Z);
199     Point3d polylineP3 = new Point3d(points1[i].X + tempOffset, points1[i].Y + tempOffset, points1[i].Z);
200     Point3d polylineP4 = new Point3d(points1[i].X - tempOffset, points1[i].Y - tempOffset, points1[i].Z);
201
202     //gather all of these 4 points as list of points
203     List<Point3d> tempPoints = new List<Point3d>();
204     tempPoints.Add(polylineP1);
205     tempPoints.Add(polylineP2);
206     tempPoints.Add(polylineP3);
207     tempPoints.Add(polylineP4);
208     tempPoints.Add(polylineP5);
209
210     //input the 4 points to create a polyline
211     Polyline tempPolyline = new Polyline(tempPoints);
212
213     //Polyline polylines
214     polylines.Add(tempPolyline);
215
216 }
217
218 return polylines;
219 }
220
221 public double GetRandomNumber(double minimum, double maximum)
222 {
223     Random random = new Random();
224     return random.NextDouble() * (maximum - minimum) + minimum;
225 }
226
227
228 /**
229 */
230 }
```

## Highlights



### Landscape design Highlights

One of the interesting highlights is when we try to view this work from eye-level perspective view. It gives a “focused” impression from the direction of the local components and a nice view from the slight variations of rotation angles.