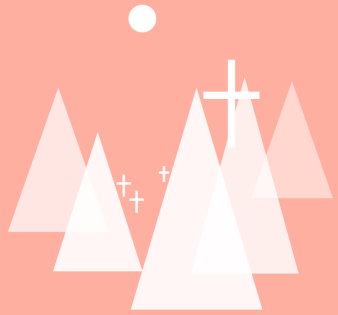
SCI 6338 INTRODUCTION TO COMPUTATIONAL DESIGN
ASSIGNMENT A



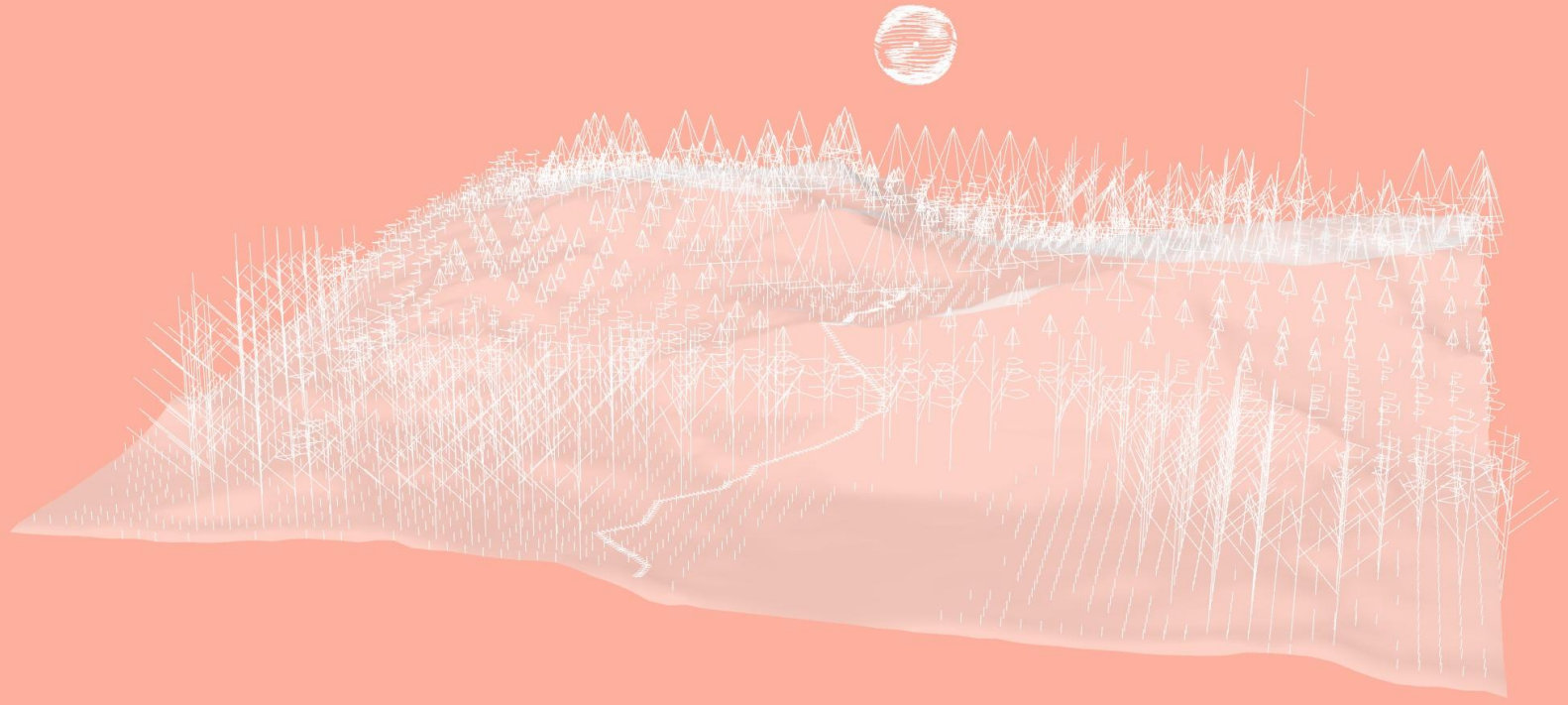
TO THE MOON

CEMETERY OF THE FOREST

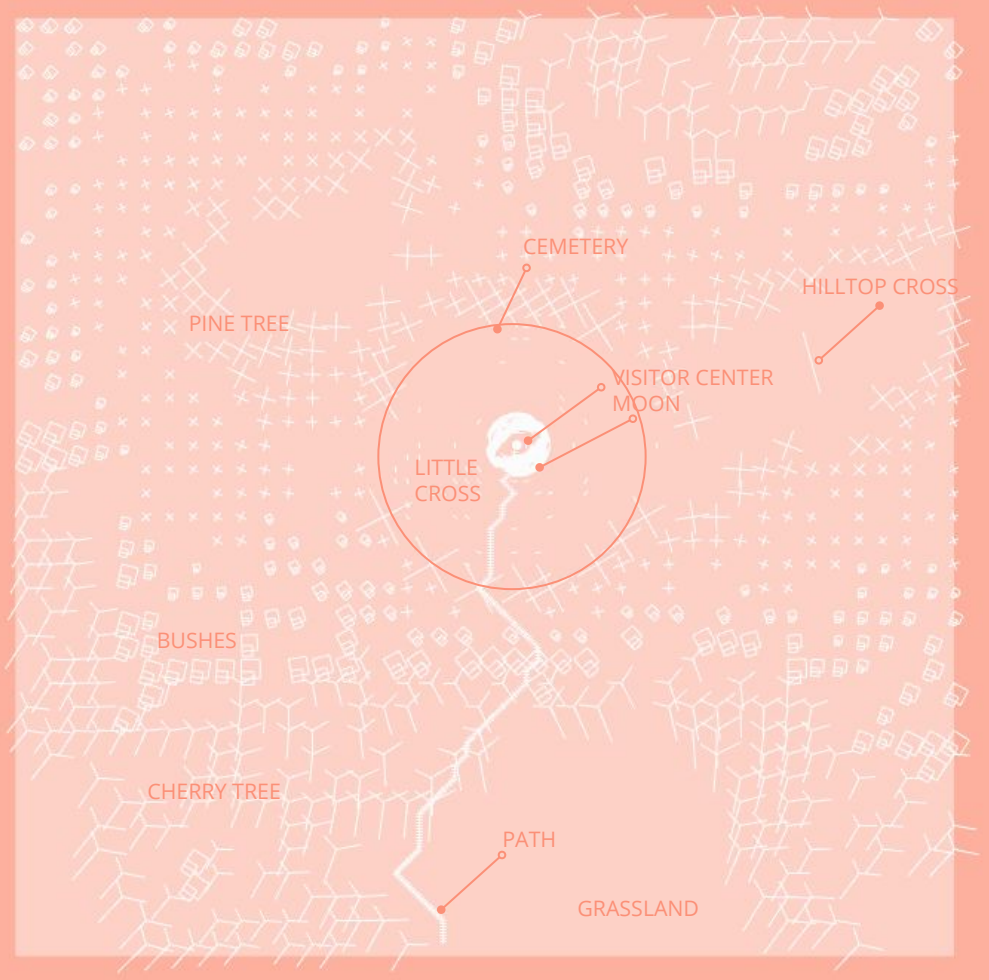
SHIYI PENG | LOCAL | *MDES TECH 2020*
GUANGYU DU | GLOBAL | *MDES TECH 2020*

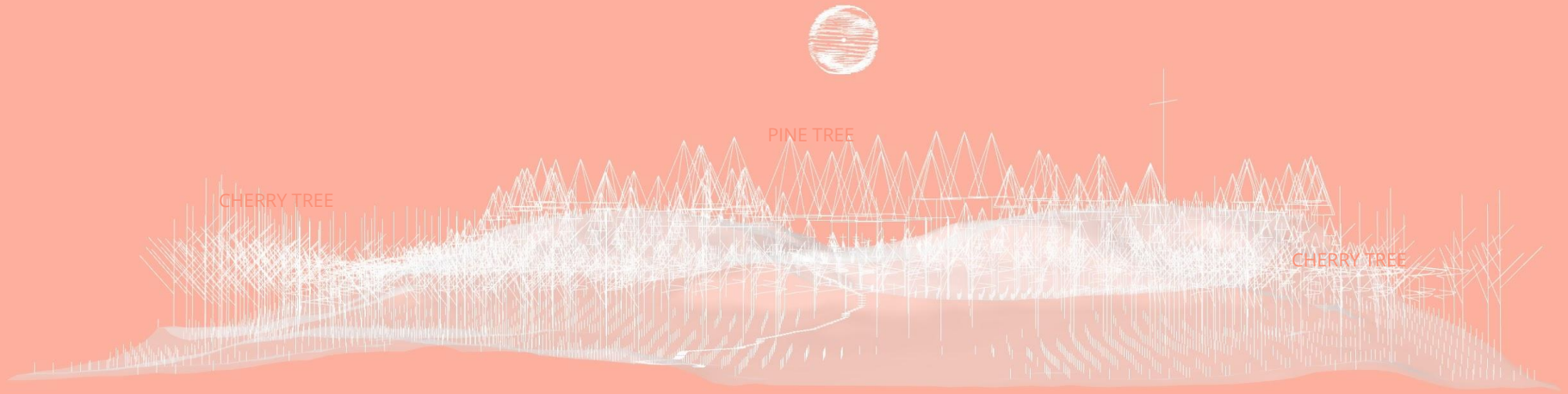
BEFORE WE START
PLEASE CHANGE THE VIEWPORT BACKGROUND COLOR IN RHINO TO

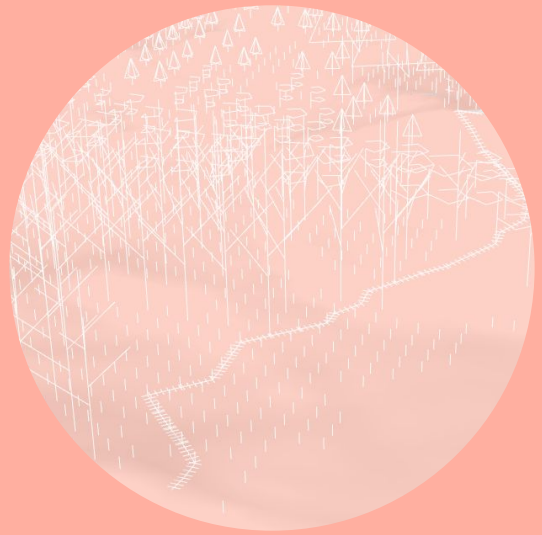
R255 G175 B160 (#ffafa0)

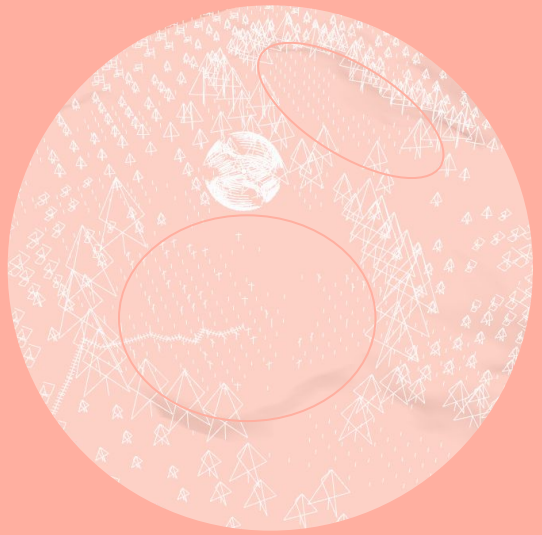


TO THE MOON

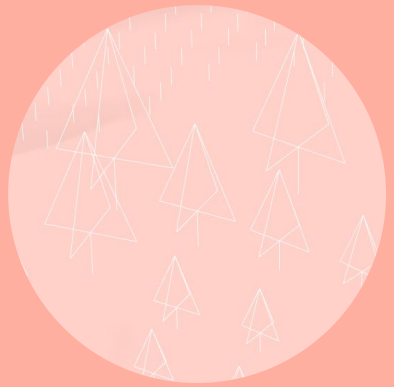

CEMETERY
HILLTOP CROSS
PINE TREE
VISITOR CENTER
MOON
LITTLE
CROSS
BUSHES
CHERRY TREE
PATH
GRASSLAND


PINE TREE
CHERRY TREE
CHERRY TREE

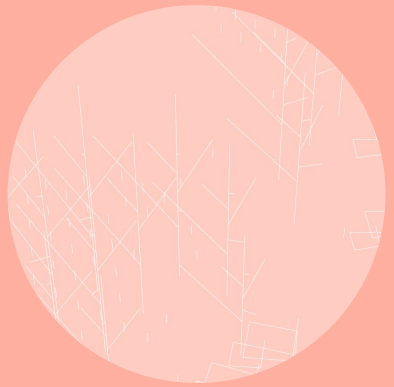




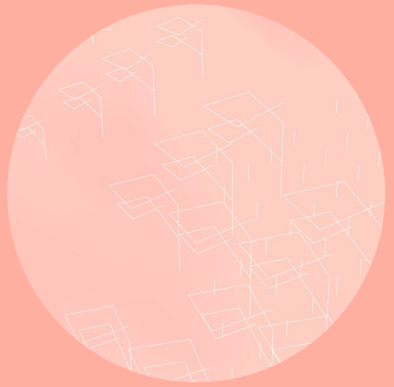
Grass



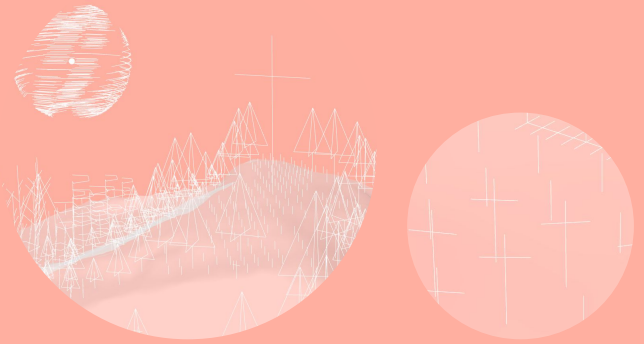
Pine



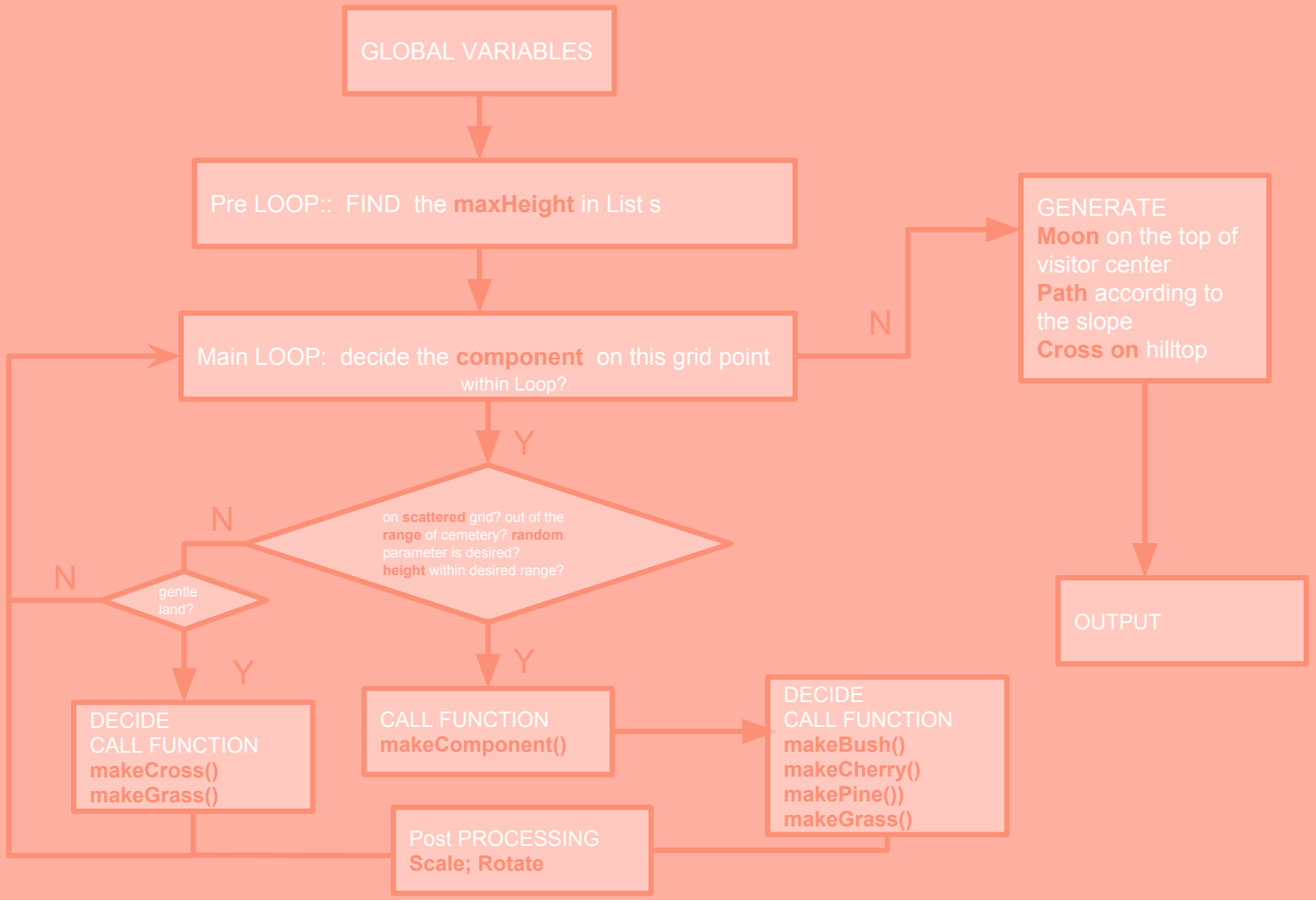
Cherry



Bush

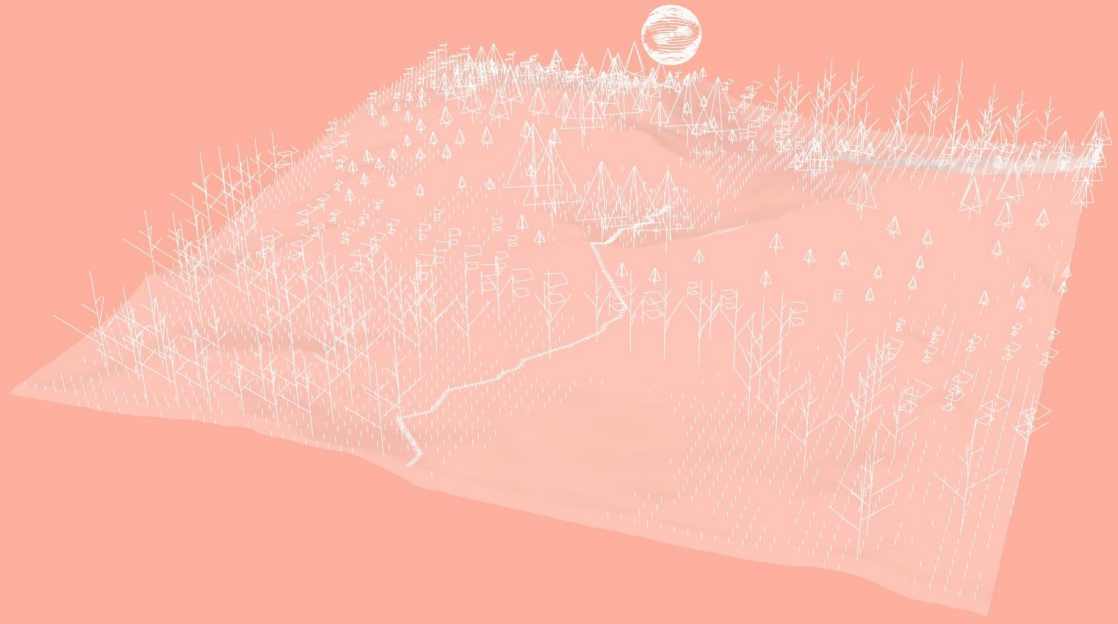


Cross


GLOBAL VARIABLES
Pre LOOP:: FIND the maxHeight in List s
Main LOOP: decide the component on this grid point
within Loop?
N
GENERATE
Moon on the top of visitor center
Path according to the slope
Cross on hilltop
OUTPUT
Y
on scattered grid? out of the range of cemetery? random parameter is desired?
height within desired range?
N
gentle land?
N
Y
Y
DECIDE
CALL FUNCTION
makeCross()
makeGrass()
CALL FUNCTION
makeComponent()
DECIDE
CALL FUNCTION
makeBush()
makeCherry()
makePine())
makeGrass()
Post PROCESSING
Scale; Rotate

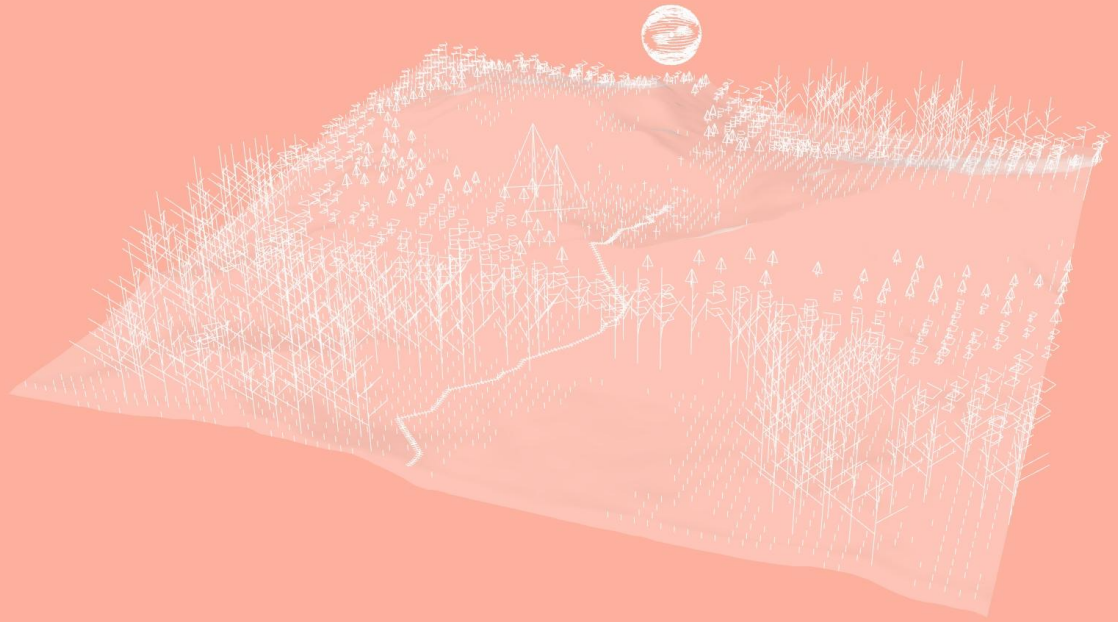
VARIATIONS

LESS DENSITY
ScatterX = 3, ScatterY = 3



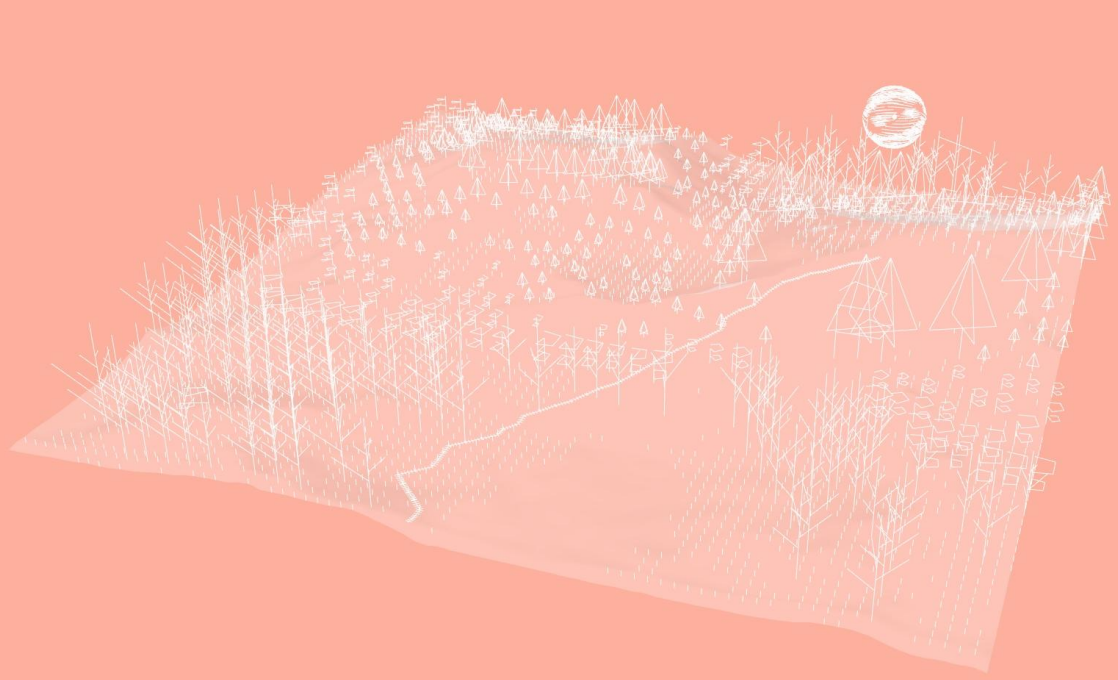
VARIATIONS

LESS PLANTS ON HILLTOP
rangeZ = 3.0



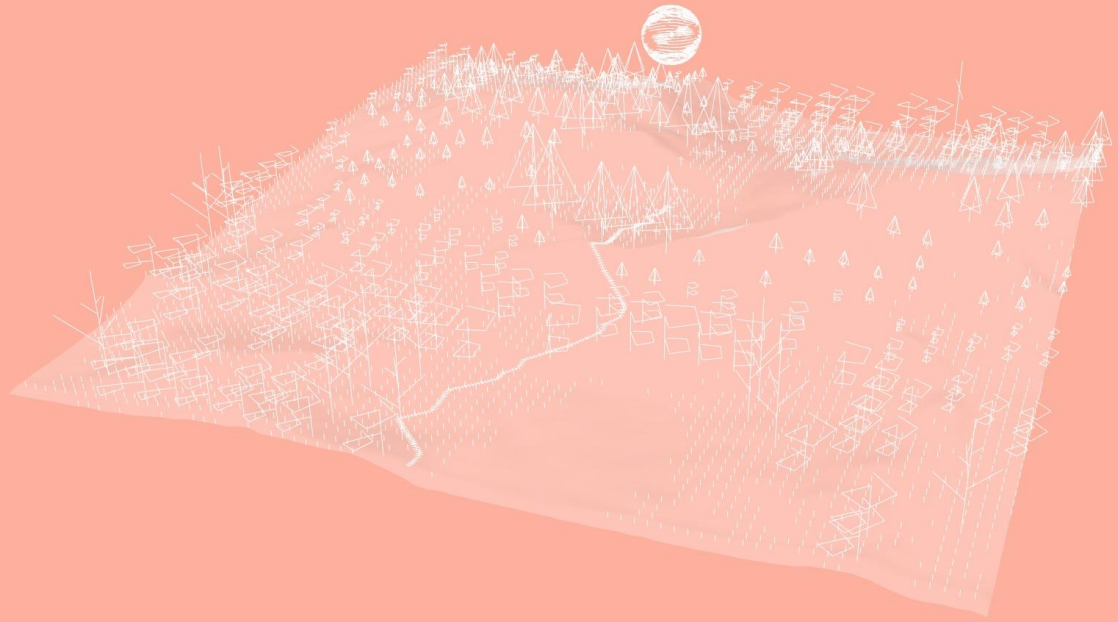
VARIATIONS

DIFFERENT PATH
move focus



VARIATIONS

MORE BUSHES
bushBoundary = 0.25



GLOBAL

Under the moon here is the visitor center, situated in a vast field (**rangeXY**), the edge of which grow the forest of trees. Some lower land and gentle basin (**rangeZ , angleZ**) is planted with grass, while mountains are planted with pine trees, cherry trees and bushes. The plants have a transition from lage scale to small scale and large scale (**elevation s[k], distanceXY**). Along the path, you will meet a forest of cherry trees and then a field of bushes, seeing the pine tree forest in the moonlight at a distance. At last the path will lead you into the pine tree forest, where you find the view is open again. Several crosses scattered in the lawn, and there is a huge cross on the hilltop. The overall density of plants is controlled by **scatterX** and **scatterY**, while elevation **s[k]** also has an effect on the random deletion (**randomZ**) of the plants.

LOCAL

There are 5 different plants and objects in the cemetery. In the lower terrain, on the steep slopes and the top of the hill grows the grassland (**makeGrass()**). On the highest spot of the entire terrain sits the big cross statue (**makeCross()**), which is facing towards the visitor center (**focus**)so that it serves as a guide sign for tour in the cemetery. In the gentle basin at the center of the terrain is where the graveyard is situated. Scattered among the grass are graves represented by little crosses. At the middle to high altitude (**pineBoundary, maxHeight**) are the pine trees (**makePine()**) with pointy tops. At the lower to middle altitude are the cherry trees (**makeCherry()**). The branches of cherry tree are scaled down, rotated, and elevated as they’re being generated. The bushes are grown near certain range of visitor center and in between the altitudes of pines and cherry trees (**makeBush(), bushBoundary**), with rotating lines that shape a square cluster of leaves. The local generation methods (**makeComponent()**), coupled with the scaling, scattering and randomizing controls in global, the different objects populate the cemetery in a more natural way.

```
55  private void RunScript(List<double> s, List<Vector3d> v, int ry, int rx, double sizeX, double sizeY, Point3d focus, ref object A)
56  {
57    //creater an empty list to add the lines for the output
58    List<Line> lines = new List<Line>();
59    //compute steps of grid so that we can calculate the physical locations of points from the row/column indices
60    double dx = sizeX / (double) (rx - 1.0);
61    double dy = sizeY / (double) (ry - 1.0);
62    //TODO GLOBAL1 [optional]: here you
63    //Global Variables
64    //General Density
65    int scatterX = 2;
66    int scatterY = 2;
67    //The Range of Cemetery
68    double rangeXY = 20.0;
69    //Hilltop will not have plants
70    double rangeZ = 4.0;
71    //Control how many plants are randomly excluded
72    double rangeRandom = 0.7;
73    //Decide whether a land is smooth and gentle
74    double angleZ = 0.2;
75    //Scale Factor for elevation, used in generate random numbers
76    double randomZ = 0.2;
77    //Control the range of the very high pine trees surrounded the cemetery
78    double rangeBelt = 0.2;
79    //The Step depth and width of path
80    double step = 0.1;
81    double stepL = 0.1;
82    //General Random Generator
83    int seed = 3;
84    Random random = new Random(seed);
85    //Useful Unit Vector and default plane
86    Vector3d unitZ = new Vector3d(0, 0, 1);
87    Plane oriPlane = new Plane(new Point3d(0, 0, 0), unitZ);
88    //END GLOBAL1
```

```
    //Store the variables of points
    double minDist = 10000.0;
    int minI = 0;
    int minJ = 0;
    int maxHeightI = 0;
    int maxHeightJ = 0;
    double maxHeight = 0.0;
    double max = 0.0;
    //looping the elevation list, find the maxHeight
    //Compute the highest elevation
    for (int n = 0; n < s.Count; n++)
    {
      if(s[n] > maxHeight){maxHeight = s[n];}
    }
    // Main Loop; Make components
    //looping through all possible index pairs in the grid ry x rx
    for (int j = 0; j < ry; ++j) {
      for(int i = 0; i < rx; ++i) {
        //compute the physical point location on the XY plane
        Point3d p = new Point3d(i * dx, j * dy, 0.0);
        //compute the serial index --> pickup values from the lists s and v
        int k = j * rx + i;
        // find distance and the closet point to focus
        double distanceXY = Math.Pow(p.X - focus.X, 2) + Math.Pow(p.Y - focus.Y, 2);
        // find the nearest point to visitor center
        if (distanceXY < minDist)
        {
          minDist = distanceXY;
          minI = i;
          minJ = j;
        }
        // store the location of the highest point in the landscape
        if (s[k] > max)
        {
          max = s[k];
          maxHeightI = i;
          maxHeightJ = j;
        }
```

```
127         //TODO GLOBAL2 : here you need to decide whether to call the makeComponent method or not according to your criteria and logic
128         double randomP = random.Next((int) (s[k] * randomZ), (int) (s[k] + 1));
129         double ranCross = random.Next(k, k + 4);
130         if ( i % scatterX == 0
131           && j % scatterY == 0
132           && distanceXY >= rangeXY
133           && randomP >= rangeRandom
134           && s[k] <= rangeZ)
135         {
136           //END GLOBAL2
137           List<Line> component = makeComponent(p, s[k], v[k], focus, maxHeight); //call the makeCompoennt method from below to get hold of the local wireframe geometry
138
139           //TODO GLOBAL3: here you can post-process the list of lines that the makeComponent returned before adding them to the master list
140           // Scale
141           Point3d anchor = new Point3d(p.X, p.Y, s[k]);
142           Transform scale3D;
143           if(distanceXY <= rangeXY * (1 + rangeBelt))
144           {
145             scale3D = Transform.Scale(anchor, 10.0 / (s[k] + 0.2));
146           }
147           else
148           {
149             scale3D = Transform.Scale(anchor, 1.0 * ( Math.Pow(Math.Abs(s[k] - 2.8), 2) + 0.5 + 0.05 * Math.Abs(distanceXY * 0.01 - 2)));
150           }
151           // Rotate
152           Transform rotate2D = Transform.Rotation(Math.PI * Vector3d.VectorAngle(p - focus, unitZ), anchor);
153           for (int m = 0, len = component.Count; m < len; m++)
154           {
155             Line temp = component[m];
156             temp.Transform(scale3D);
157             temp.Transform(rotate2D);
158             component[m] = temp;
159           }
160           //END GLOBAL3
```

```
161             lines.AddRange(component);
162         }
163         else if (Vector3d.VectorAngle(v[k], unitZ) < angleZ)
164         {
165             if(distanceXY < rangeXY && ranCross < (k + 0.5))
166                 {
167                 //Randomly generate little crosses among grass
168                 List<Line> component = makeCross(p, s[k], focus);
169 
170                 lines.AddRange(component);
171                 }
172             else if(s[k] > 0.8)
173                 {
174                 //Generate grass
175                 List<Line> component = makeGrass(p, s[k]);
176 
177                 lines.AddRange(component);
178                 }
179         }
180       }
181     }
182 
183     //TODO GLOBAL4: here you can add extra logic to add lines that may connect between grid poitns or add other graphical elements to your design
184 
185     // GENERATE PATH to visitor center
186     //Start
187     int startJ = minJ;
188     int startI = minI;
189     Point3d pStart = new Point3d(minI * dx, minJ * dy, s[minJ * rx + minI]);
190     //Next
191     int nextJ = minJ + 1;
192     int nextI = minI;
193     Point3d pNext = new Point3d(nextI * dx, nextJ * dy, s[nextJ * rx + nextI]);
194     //Path and Steps
195     List<Line> pathLine = new List<Line>();
196     List<Line> pathStep = new List<Line>();
```

```
    //Traverse until reach the border
    while((nextJ != 0 && nextJ != (ry - 1)) && (nextI != 0 && nextI != (rx - 1)))
    {
      //traverse the 3 adjacent points(southwest, south, southeast)
      double minH = 10000.0;
      int minIndex = 0;
      for(int i = -1; i < 2; i++)
      {
        double h = Math.Abs(s[nextJ * rx + (nextI + i)] - s[startJ * rx + startI]);
        if( minH > h)
        {
          minH = h;
          minIndex = i;
        }
      }
      startJ = nextJ;
      startI = nextI + minIndex;
      pNext.X = startI * dx;
      pNext.Z = s[startJ * rx + startI];
      // Add Path Line Segment
      Line path = new Line(pStart, pNext);
      pathLine.Add(path);
      double len = 0.0;
      // Calculate the step direction
      Vector3d toEnd = (new Vector3d(pNext) - new Vector3d(pStart));
      Vector3d pNormal = Vector3d.CrossProduct(toEnd, new Vector3d(0, 0, -1.0));
      pNormal.Unitize();
      // Add Steps, step : len
      while(len < path.Length)
      {
        Point3d p = path.PointAtLength(len);
        len = len + step;
        pathStep.Add(new Line(p - pNormal * stepL, p + pNormal * stepL));
      }
      //Update Start & Next
      nextJ += 1;
      nextI = startI;
```

```
234         pStart = pNext;
235         pNext.Y = nextJ * dy;
236       }
237       //Add them to the main list
238       lines.AddRange(pathLine);
239       lines.AddRange(pathStep);
240
241       //GENERATE THE LARGE CROSS at the highest point on the mountain
242       Point3d highest = new Point3d(maxHeightI * dx, maxHeightJ * dy, 0);
243       List<Line> cross = makeCross(highest, maxHeight, focus);
244       Transform scaleCross = Transform.Scale(highest + unitZ * maxHeight, 10.0);
245       for (int m = 0, len = cross.Count; m < len; m++)
246       {
247         Line temp = cross[m];
248         temp.Transform(scaleCross);
249         cross[m] = temp;
250       }
251       lines.AddRange(cross);
252
253       //GENERATE THE MOON :p
254       int moonRadius = 10;
255       double moonSegment = 0.2;
256       List < Line> moon = new List<Line>();
257       Random random1 = new Random(100);
258       Random random2 = new Random(101);
259       Random random3 = new Random(102);
260       for (int i = 0; i < 1000; i++)
261       {
262         double moonX = random1.Next(-(moonRadius), moonRadius);
263         double moonY = random2.Next(-(moonRadius), moonRadius);
264         double moonZ = random3.Next(-(moonRadius), moonRadius);
265         Vector3d vMoon = new Vector3d(moonX, moonY, moonZ);
266         vMoon.Unitize();
267         Vector3d moonNormal = Vector3d.CrossProduct(vMoon, -unitZ);
268         moonNormal.Unitize();
269         moon.Add(new Line(focus + vMoon + moonNormal * moonSegment, focus + vMoon - moonNormal * moonSegment));
270       }
271       lines.AddRange(moon);
272       //END GLOBAL4
```

```
  //OUTPUT
  A = lines;
}

// <Custom additional code>
//Function -> make grass//
List<Line> makeGrass(Point3d p, double s) {
  List<Line> grass = new List<Line>();

  Point3d elevatedPoint = p;
  elevatedPoint.Z = s;

  Point3d end = elevatedPoint;
  end.Z = s + 0.2;

  grass.Add(new Line(elevatedPoint, end));

  return grass;
}

//Function -> make bushes//
List<Line> makeBush(Point3d elevatedPoint, double s) {
  List<Line> bush = new List<Line>();

  Point3d top = elevatedPoint;
  top.Z += 0.5;

  Vector3d u = new Vector3d(1.0, 1.0, 0.2);
  u.Unitize();
  Point3d branch = elevatedPoint;
  branch.Z = s + 0.25;
  Point3d b1 = branch + 0.2 * u;
  bush.Add(new Line(elevatedPoint, top));
  bush.Add(new Line(branch, b1));

  Point3d start1 = top;
  Point3d start2 = b1;
  Vector3d v = new Vector3d(1.0, 1.0, 0.15);
  v.Unitize();
```

```
    for (int i = 0; i < 4; ++i) {
      Point3d end1 = start1 + 0.3 * v;
      Point3d end2 = start2 + 0.2 * v;
      bush.Add(new Line(start1, end1));
      bush.Add(new Line(start2, end2));
      start1 = end1;
      start2 = end2;
      v.Rotate(Math.PI * 0.5, top - elevatedPoint);
    }
    return bush;
  }

  //Function -> make cherry tree//
  List<Line> makeCherry(Point3d elevatedPoint, double s) {
    List<Line> cherry = new List<Line>();

    //make trunk//
    Vector3d up = new Vector3d(0.0, 0.0, 1.0);
    cherry.Add(new Line(elevatedPoint, elevatedPoint + up));

    Point3d node = new Point3d(elevatedPoint.X, elevatedPoint.Y, s + 0.3);
    Vector3d v = new Vector3d(1.0, 1.0, 1.5);
    v.Unitize();

    //make branches//
    for (int i = 0; i < 5; ++i) {
      cherry.Add(new Line(node, node + 0.5 * v));
      node.Z += 0.1;
      v.Rotate(Math.PI * 0.67, up);
      v *= 0.7;
    }

    return cherry;
  }
```

```
347   //Function -> make pine tree//
348   List<Line> makePine(Point3d elevatedPoint, double s) {
349     List<Line> pine = new List<Line>();
350
351     Vector3d up = new Vector3d(0.0, 0.0, 1.0);
352     Vector3d node = new Vector3d(0.0, 0.0, 0.3);
353     Vector3d r = new Vector3d(0.3, 0.0, 0.0);
354     Vector3d f = new Vector3d(0.0, 0.3, 0.0);
355     Vector3d l = -r;
356     Vector3d b = -f;
357
358     pine.Add(new Line(elevatedPoint, elevatedPoint + node));
359     pine.Add(new Line(elevatedPoint + node + r, elevatedPoint + node + l));
360     pine.Add(new Line(elevatedPoint + node + r, elevatedPoint + up));
361     pine.Add(new Line(elevatedPoint + node + l, elevatedPoint + up));
362     pine.Add(new Line(elevatedPoint + node + f, elevatedPoint + node + b));
363     pine.Add(new Line(elevatedPoint + node + f, elevatedPoint + up));
364     pine.Add(new Line(elevatedPoint + node + b, elevatedPoint + up));
365
366     return pine;
367   }
368
369   //Function -> make cross//
370   //let it face visitor center//
371   List<Line> makeCross(Point3d p, double s, Point3d focus) {
372     List<Line> cross = new List<Line>();
373
374     Point3d elevatedPoint = p;
375     elevatedPoint.Z = s;
376     Point3d focusXY = focus;
377     focusXY.Z = 0;
378     Vector3d toFocus = focusXY - p;
379     toFocus.Unitize();
380     Point3d x = elevatedPoint;
381     x.Z += 0.3;
382
383     Vector3d normal = new Vector3d(0.0, 0.0, 1.0);
```

```
    Vector3d arm = toFocus;
    arm.Rotate(Math.PI * 0.5, normal);

    cross.Add(new Line(elevatedPoint, elevatedPoint + 0.4 * normal));
    cross.Add(new Line(x - 0.1 * arm, x + 0.1 * arm));

    return cross;
  }

  //Function -> decide where to put different types of plants//
  List<Line> makeComponent(Point3d p, double s, Vector3d v, Point3d focus, double maxHeight) {
    List<Line> lines = new List<Line>();
    Point3d elevatedPoint = p;
    elevatedPoint.Z = s;

    Vector3d toFocus = focus - p;

    Vector3d normal = new Vector3d(0.0, 0.0, 1.0);

    double bushBoundary = 0.35;
    double pineBoundary = 0.55;

    if (Vector3d.VectorAngle(v, normal) > 60.0) {
      //if too steep, grow grass//
      lines.AddRange(makeGrass(p, s));
    }
    else {
      if (toFocus.Length > 3 && toFocus.Length < 4.8) {
        //grow bushes around visitor center//
        lines.AddRange(makeCherry(elevatedPoint, s));
      }
```

```
      else {
        if (s > bushBoundary * maxHeight && s <= pineBoundary * maxHeight) {
          //grow bushes//
          lines.AddRange(makeBush(elevatedPoint, s));
        }
        else if (s > 0.01 * maxHeight && s <= bushBoundary * maxHeight) {
          //grow cherry trees//
          lines.AddRange(makeCherry(elevatedPoint, s));
        }
        else if (s > pineBoundary * maxHeight && s < 1.0 * maxHeight) {
          //grow pine trees//
          lines.AddRange(makePine(elevatedPoint, s));
        }
        else {
          //if too hight or too low, grow grass//
          lines.AddRange(makeGrass(p, s));
        }
      }
    }

    return lines;
  }

  // </Custom additional code>
}
```