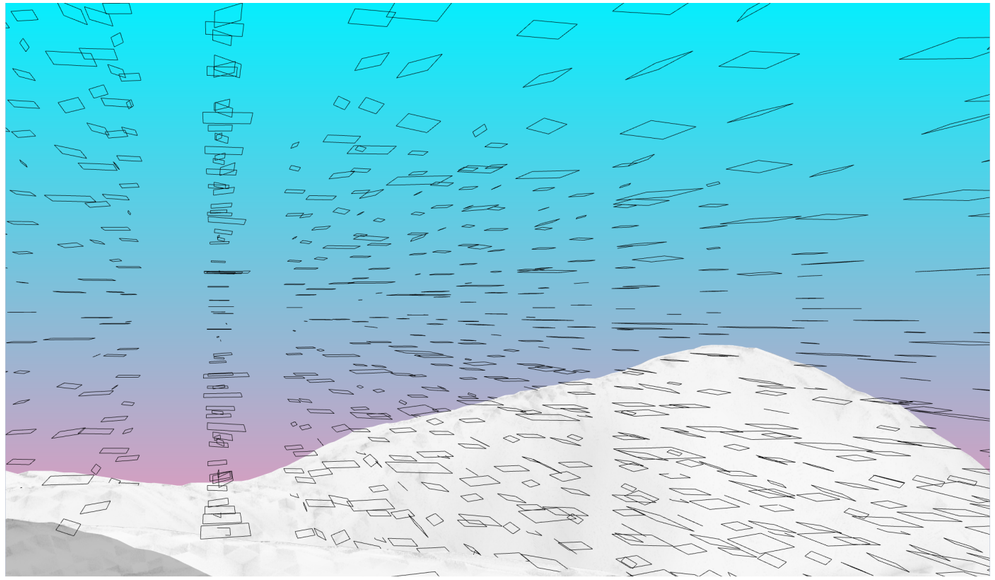
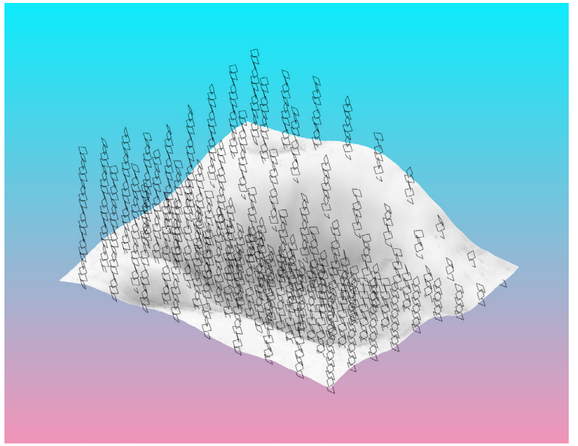
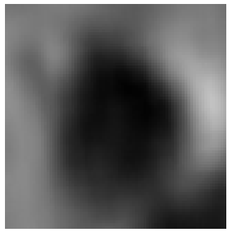




GSD - SCI 6338
Introduction to Computational Design
Assignment 1: Local-Global
Togo Kida, Hanif Wicaksono

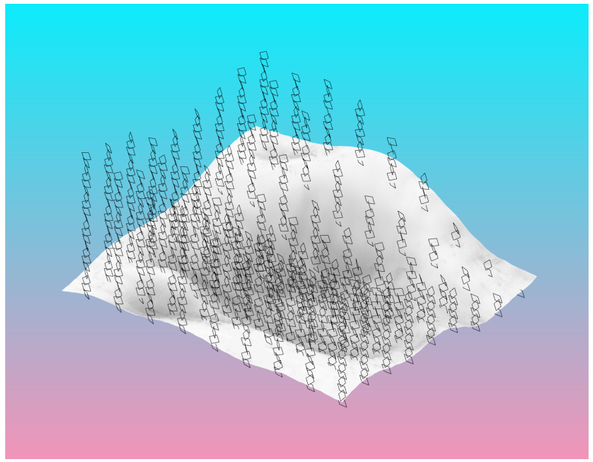
Concept



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

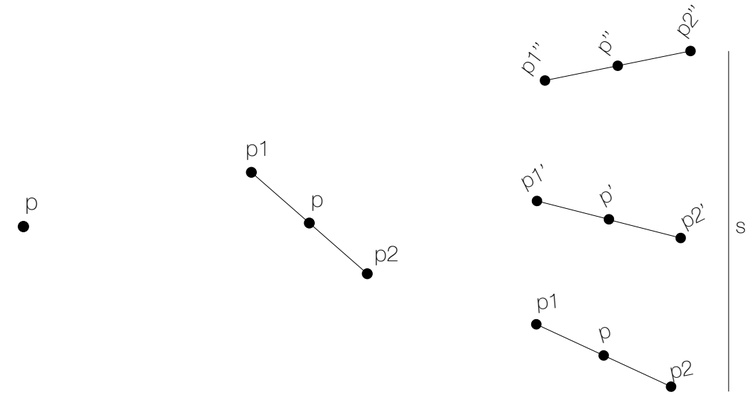
Global



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 resonates with the single focus point.

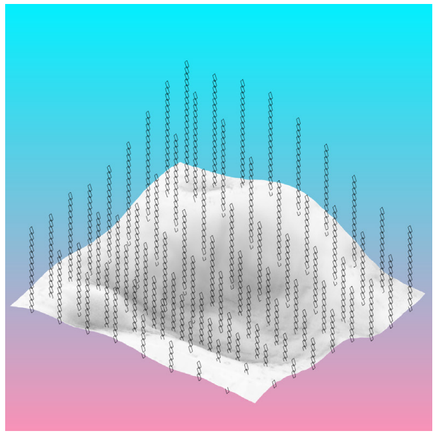
Global (focus point):
Create lists of points, polylines, & vectors
Decide where to populate the local
Number of local iterations made
Make rotations when focus moves

Local

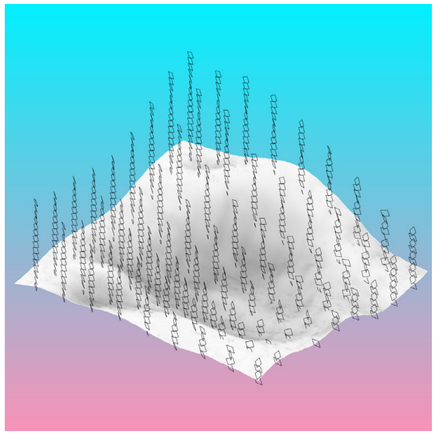


This local component takes the input p as the starting point to construct the geometry. After defining p as the middle point, it then calculates p2 and p1 to define the range of the rectangle. The input s is used to determine numIterations which defines the number of times the rectangular base form is 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.

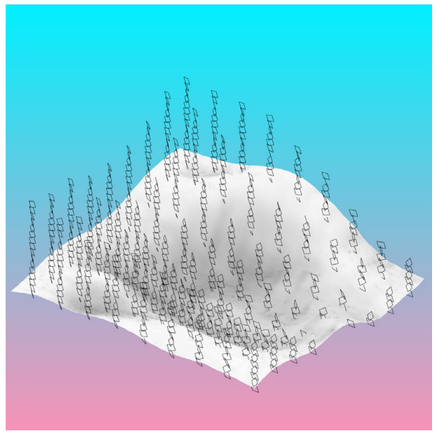
Variations



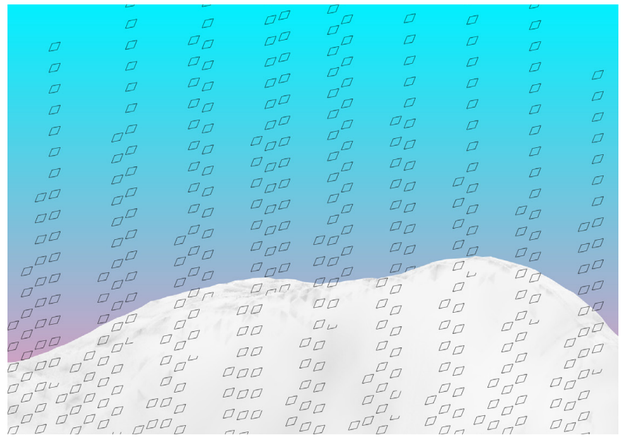
Point3d focus (x,y,z) : 0,0,0



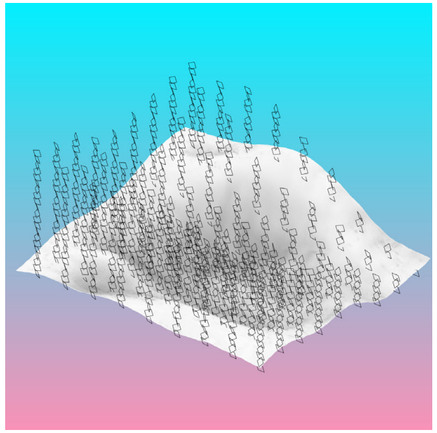
Point3d focus (x,y,z) : 20,0,0



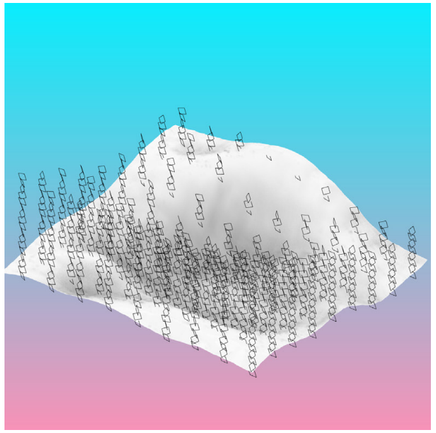
Point3d focus (x,y,z) : 40,0,0



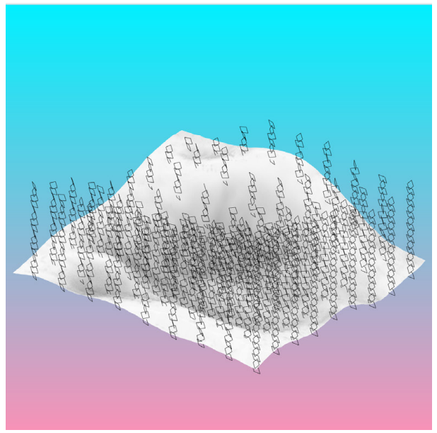
local components without focus point



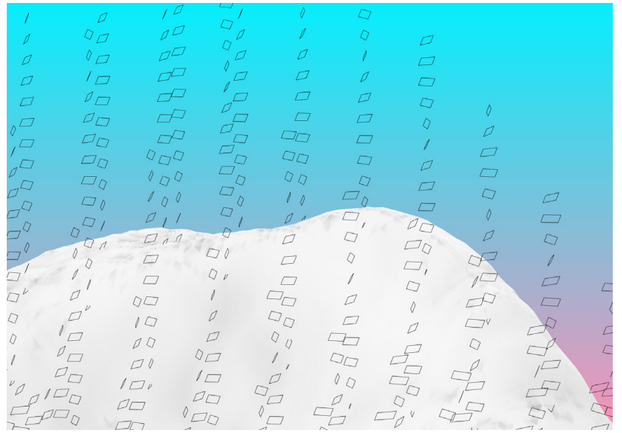
Point3d focus (x,y,z) : 0,0,0



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



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



Local components rotated to focus point

Code (Global)

```
using


/// <summary>
/// This class will be instantiated on demand by the Script component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
Utility functions

Members

  /**/
  private void RunScript(List<double> s, List<Vector3d> v, int ry, int rx, double sizeX, double s
  {
    //START GLOBAL 0

    //creater an empty list to add the lines for the output
    List<Line> lines = new List<Line>();

    //creater an empty list to add the lines for the output
    List<Polyline> polylines = new List<Polyline>();

    //create an empty list of vectors for the focus point
    List<Vector3d> vectors = new List<Vector3d>();

    //create an empty list of point to put the local components
    List<Point3d> points = new List<Point3d>();

    //END GLOBAL 0

    //compute steps of grid so that we can calculate the physical locations of points from the rc
    double dx = sizeX / (double) (rx - 1.0);
    double dy = sizeY / (double) (ry - 1.0);

    //start 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) {
        Point3d p = new Point3d(i * dx, j * dy, 0.0);
        //compute the physical point location on the XY plane
        int k = j * rx + i;
        //compute the serial index so that we can pickup values from the lists s and v
```

Script Editor

Script component: C#Start Component

```
        //compute the serial index so that we can pickup values from the lists s and v


        //START GLOBAL1

        //Set up focus point to vector
        Vector3d toFocus = focus - p;
        toFocus.Unitize();

        //Set a limit to the vector length
        toFocus *= 1.50;

        //Connecting the focus point to vector list
        vectors.Add(toFocus);

        //Populate the points with connected vector
        points.Add(p + toFocus);

        //END GLOBAL1

        //GLOBAL 2

        //Set up a modulo for the grid to make four walls
        //Populate based on slope value
        if (i % 9 == 0 && j % 9 == 0 && s[k] > 0 )

          //GLOBAL 2

        {
          //call the makeComponent method from below to get hold of the local wireframe geometry
          List<Polyline> polylineComponent = makeMyComponent(p, s[k], vectors[k], focus);
          polylines.AddRange(polylineComponent);

        }

      }
    }


    B = lines;
    A = polylines;//adjust to this guy
```

Code (Local)

Script Editor

Script component: C#Start Component

```
  }
  /**/


List<Polyline> makeMyComponent(Point3d p, double s, Vector3d v, Point3d focus) {

  Point3d elevatedPoint = p;
  elevatedPoint.Z = s;

  Vector3d toFocus = focus - p;
  toFocus.Unitize();


  double height = p.DistanceTo(focus);

  List<Point3d> points2 = new List<Point3d>();
  List<Point3d> points1 = new List<Point3d>();
  List<Line> lines = new List<Line>();

  int numIterations;
  Point3d midP;
  double rotAng;

  List<Polyline> polylines = new List<Polyline>();


  numIterations = (int) height / 5;
  midP = p;
  midP.Z = s;
  rotAng = Math.Atan2(focus.X, midP.Y);

  int elevateStep = 2;

  Point3d p2 = new Point3d(midP.X + Math.Cos(rotAng) / 2, midP.Y + Math.Sin(rotAng), midP.Z);
  Point3d p1 = new Point3d(midP.X - Math.Cos(rotAng) / 2, midP.Y - Math.Sin(rotAng), midP.Z);

  double ang2 = Math.Atan2(p2.X, midP.Y);
  double ang1 = Math.Atan2(p1.X, midP.Y);


  for(int i = 0; i < numIterations; i++){

    double tempX = p1.X * Math.Cos(ang1);
    double tempY = p1.Y + i * elevateStep;

    //apply transformation
    Transform trans = Transform.Rotation(rotAng * i, midP);
    Point3d tempP = new Point3d(p1.X, p1.Y, p1.Z);
    tempP.Transform(trans);
    tempP.Z += elevateStep * i;
    points1.Add(tempP);
  }

  for(int i = 0; i < numIterations; i++){
```

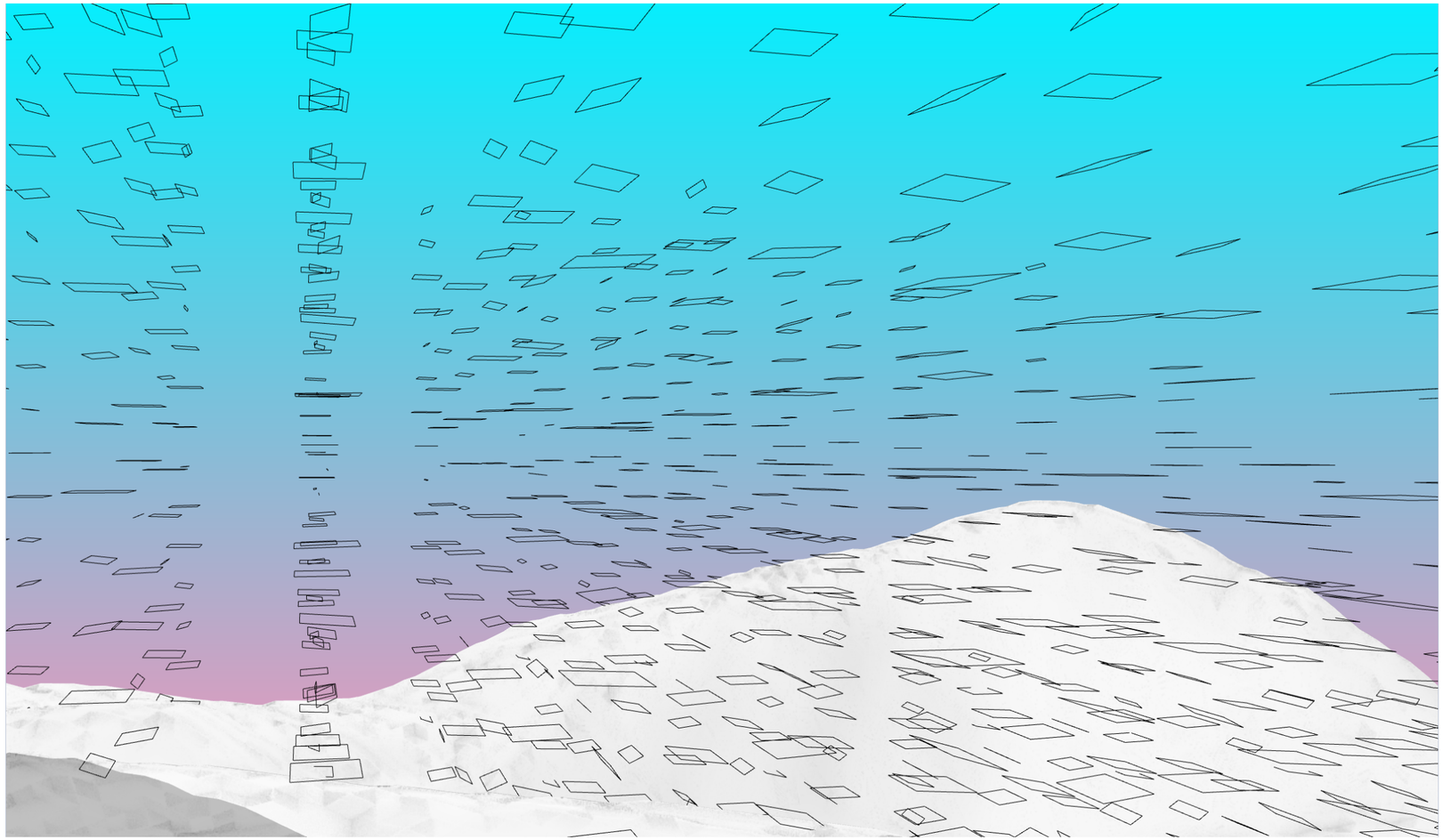
Script Editor

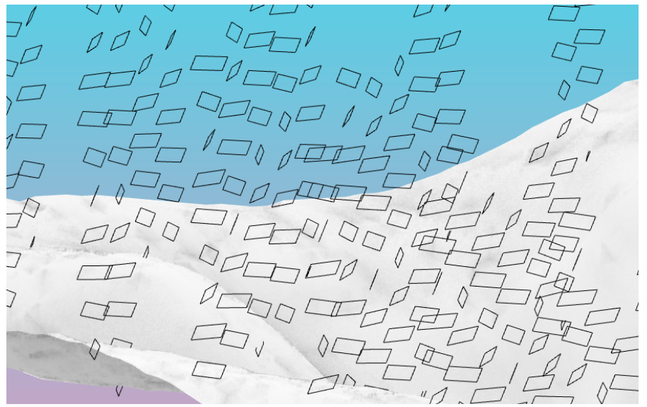
Script component: C#Start Component

```
      tempP.Z += elevateStep * i;
      points1.Add(tempP);
    }

    for(int i = 0; i < numIterations; i++){

      double tempX = p2.X * Math.Cos(ang2);
      double tempY = p2.Y + i * elevateStep;//elevate by tempStep

      //apply transfromation
      Transform trans = Transform.Rotation(rotAng * i, midP);
      Point3d tempP = new Point3d(p2);
      tempP.Transform(trans);
      tempP.Z += elevateStep * i;
      points2.Add(tempP);

    }

    //instead of draawing lines, draw rectangles
    for(int i = 0; i < numIterations; i++){

      double tempOffset = 1;//just a temp offset value
      Point3d polylineP1 = new Point3d(points2[i].X + tempOffset, points2[i].Y + tempOffset, poin
      Point3d polylineP2 = new Point3d(points2[i].X - tempOffset, points2[i].Y - tempOffset, poin
      Point3d polylineP4 = new Point3d(points1[i].X + tempOffset, points1[i].Y + tempOffset, poin
      Point3d polylineP3 = new Point3d(points1[i].X - tempOffset, points1[i].Y - tempOffset, poin
      Point3d polylineP5 = new Point3d(points2[i].X + tempOffset, points2[i].Y + tempOffset, poin

      //gather all of these 4 points as list of points
      List<Point3d> tempPoints = new List<Point3d>();
      tempPoints.Add(polylineP1);
      tempPoints.Add(polylineP2);
      tempPoints.Add(polylineP3);
      tempPoints.Add(polylineP4);
      tempPoints.Add(polylineP5);

      //input the 4 points to create a polyline
      Polyline tempPolyline = new Polyline(tempPoints);

      //Polyline polylines
      polylines.Add(tempPolyline);
    }

    return polylines;
  }

  public double GetRandomNumber(double minimum, double maximum)
  {
    Random random = new Random();
    return random.NextDouble() * (maximum - minimum) + minimum;
  }


  /**/
}
```

Cache | Recover from cache | OK

Highlights





Landscape design Highlights

One of the interesting highlight 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.