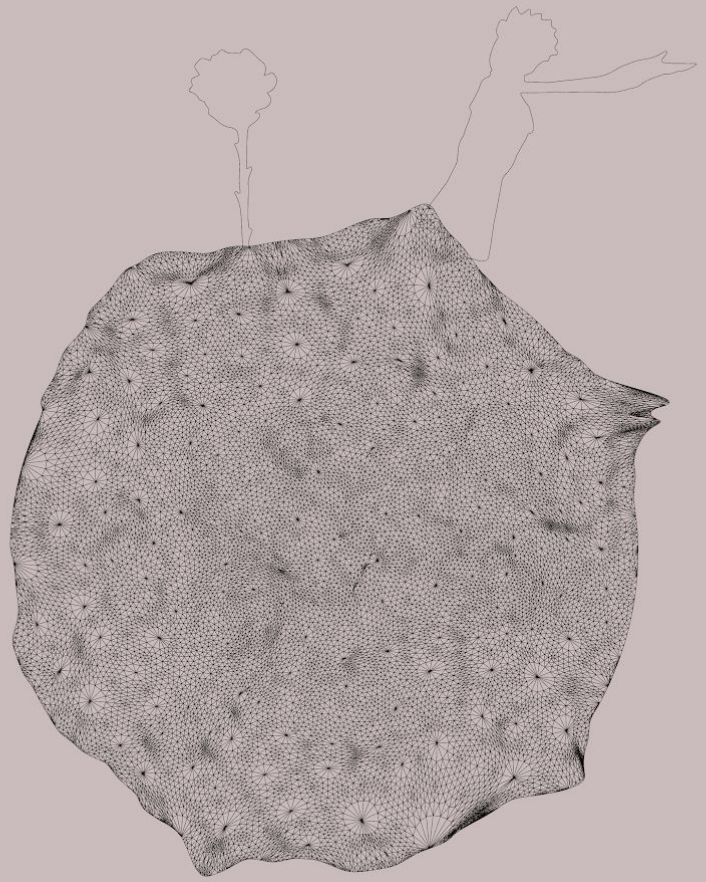
SCI 6338 INTRODUCTION TO COMPUTATIONAL DESIGN
ASSIGNMENT A

LE PETIT PRINCE

THE PLANET OF LOVE

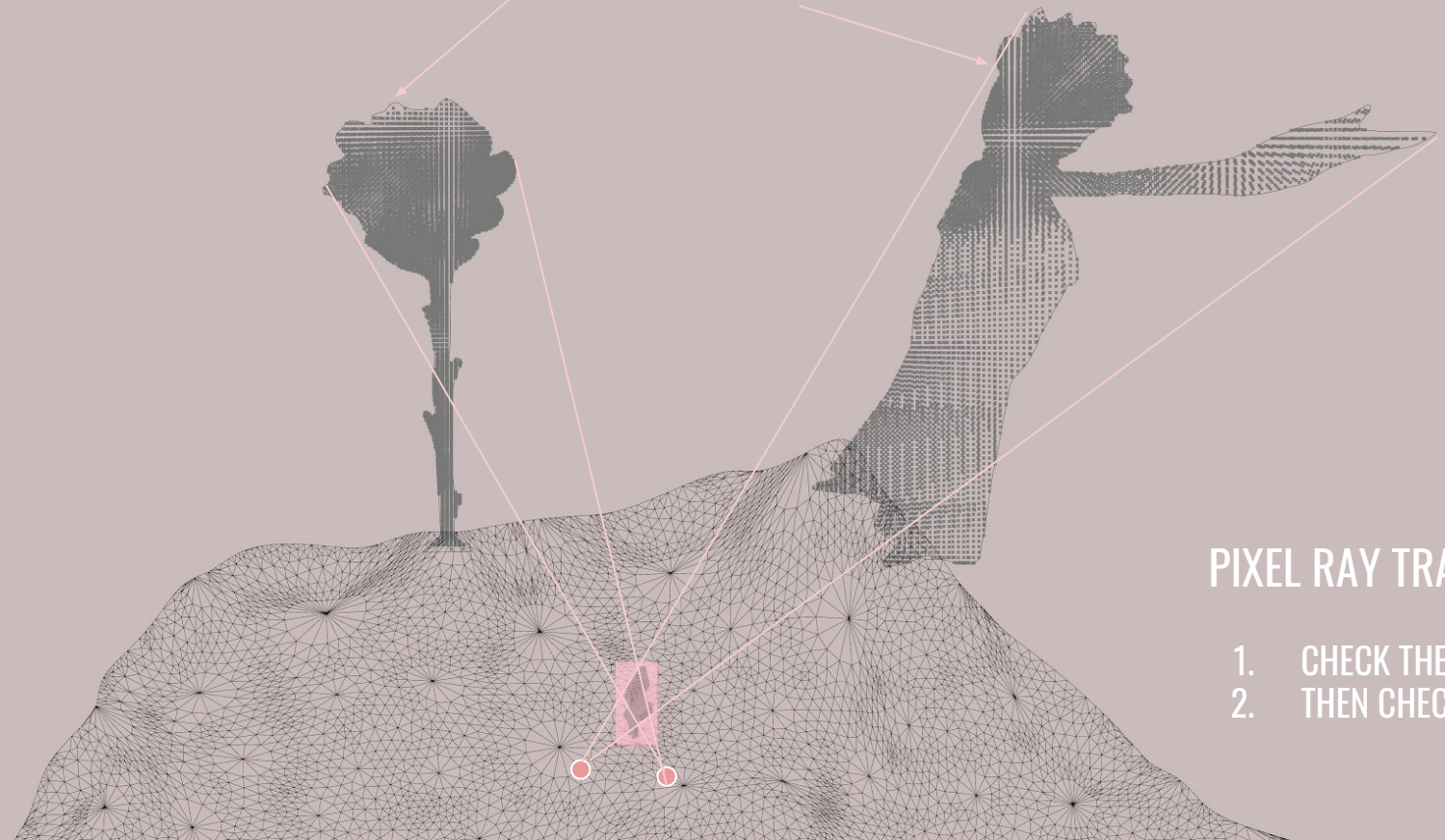
SHIYI PENG | *MDES TECH 2020*
GUANGYU DU | *MDES TECH 2020*



SHADOW CURVES

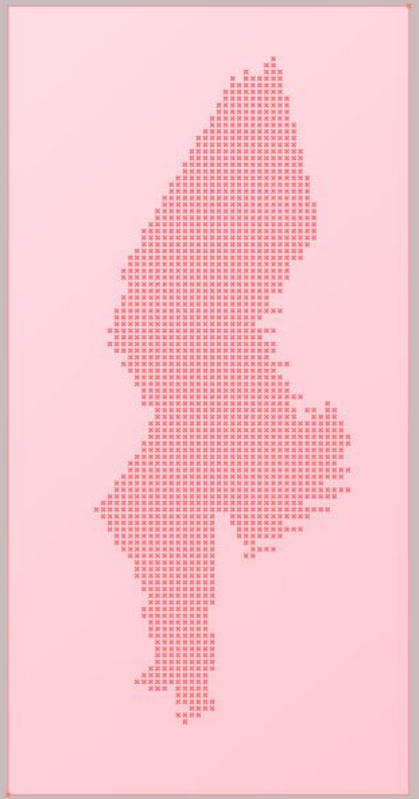
GOAL:
1. THE SHADOW OF LITTLE PRINCE AND ROSE PROJECTED TO THE WALL
2. THE OBJECT WHICH CAST SHADOW IS LIKE A SMALL STONE ON THE PLANET

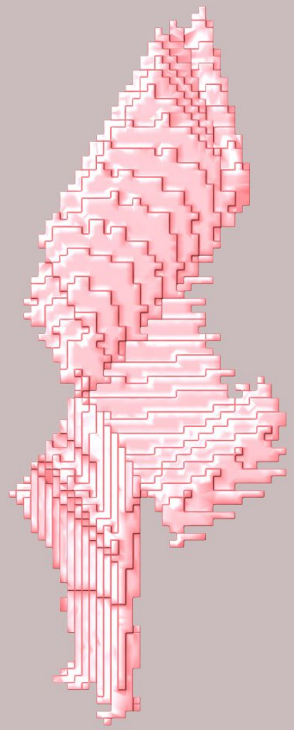
These shadow points are generated at the same time to aid design



PIXEL RAY TRACING

1. CHECK THE PRINCE
2. THEN CHECK THE ROSE

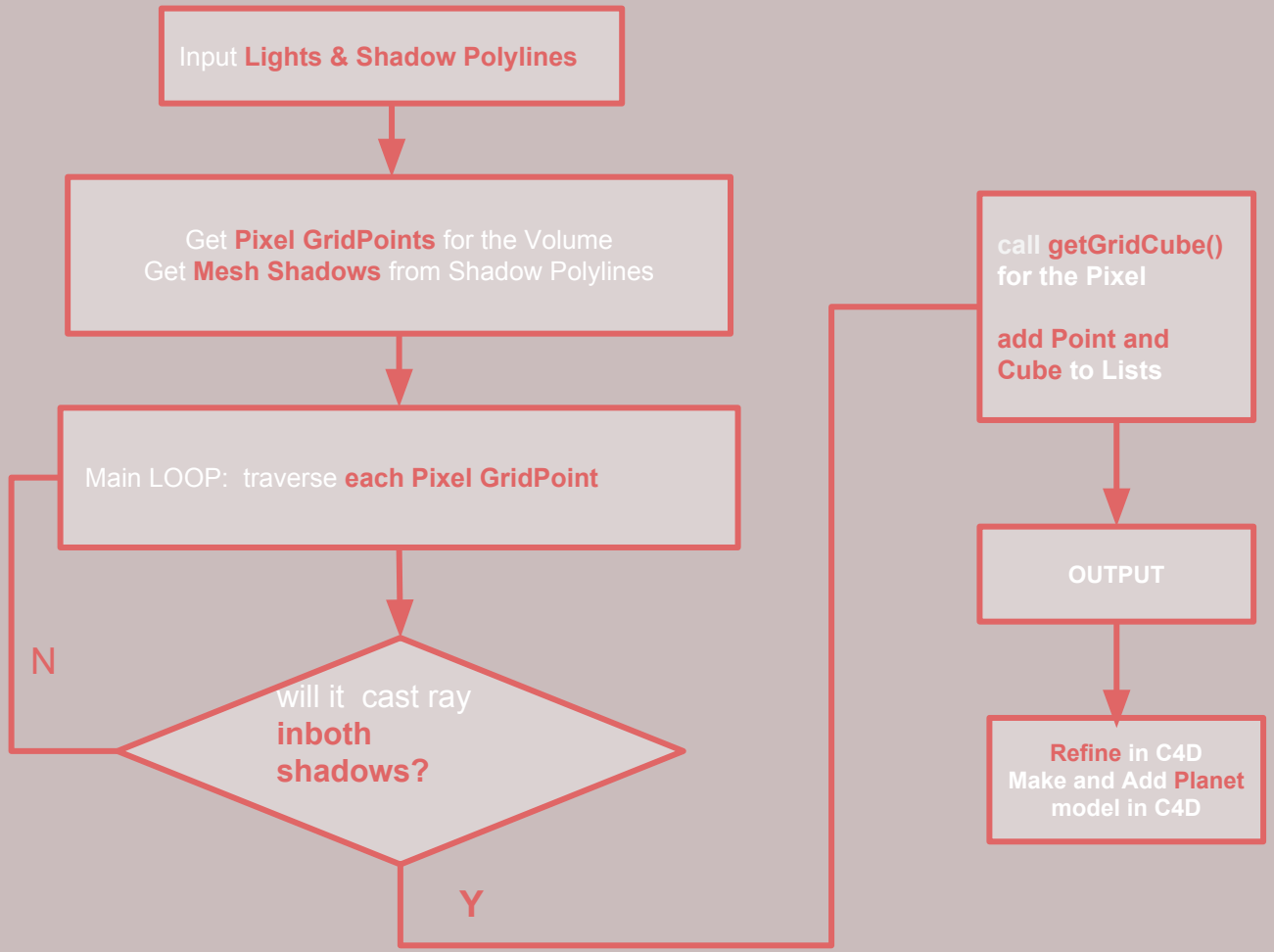




REFINEING THE MESH CUBES

THE RESULT WE GOT IS ABOUT 8000 MESH CUBES, WHICH REALLY TAXING THE COMPUTER. WE BUILD THE VOLUME MESH OF THESE CUBES USING CINEMA4D.

WORKFLOW


Input Lights & Shadow Polylines
Get Pixel GridPoints for the Volume
Get Mesh Shadows from Shadow Polylines
Main LOOP: traverse each Pixel GridPoint
will it cast ray inboth shadows?
N
Y
call getGridCube() for the Pixel
add Point and Cube to Lists
OUTPUT
Refine in C4D
Make and Add Planet model in C4D

```
private void RunScript(Point3d light, Polyline boundary, Point3d light2, Polyline boundary2, Point3d cornerA, Point3d cornerB, Mesh wall, int xCount, int yCount, int zCount, ref object A, ref object B, ref object C)
  {
// Get Pixel GridPoints
    List <Point3d> pts = getGridPoints(cornerA, cornerB, xCount, yCount, zCount);
// Calculate the edge Length of pixel Mesh Cube
    double dx = 0.5 * (cornerB.X - cornerA.X) / (xCount - 1.0);
    double dy = 0.5 * (cornerB.Y - cornerA.Y) / (yCount - 1.0);
    double dz = 0.5 * (cornerB.Z - cornerA.Z) / (zCount - 1.0);
// Get Mesh Shadows
    Mesh shadow = Rhino.Geometry.Mesh.CreateFromClosedPolyline(boundary);
    Mesh shadow2 = Rhino.Geometry.Mesh.CreateFromClosedPolyline(boundary2);
// Empty Storage Lists
    List <Point3d> volumePoints = new List <Point3d> ();
    List <Mesh> volume = new List<Mesh>();
    List <Point3d> shadowPoints = new List <Point3d> ();
// Loop The Pixel Grid
for (int i = 0; i < pts.Count; i++)
    {
      Point3d p = pts[i];
      Vector3d lightDirection = p - light;
      Ray3d ray = new Ray3d(light, lightDirection);
     // Check Whether the point will project ray to the first shadow
      double alongTheRay = Rhino.Geometry.Intersect.Intersection.MeshRay(shadow, ray);
      if (alongTheRay > 0.0)
      {
        Vector3d lightDirection2 = p - light2;
        Ray3d ray2 = new Ray3d(light2, lightDirection2);
}
 // Check Whether the point will project ray to the 2nd shadow
        double alongTheRay2 = Rhino.Geometry.Intersect.Intersection.MeshRay(shadow2, ray2);
        if (alongTheRay2 > 0.0)
        {
        // Condition is True
          double wallPoint = Rhino.Geometry.Intersect.Intersection.MeshRay(wall, ray);
          // This is for display the shadow directly in Rhino
         // Help Design Process
          shadowPoints.Add(ray.PointAt(wallPoint));
          double wallPoint2 = Rhino.Geometry.Intersect.Intersection.MeshRay(wall, ray2);
          shadowPoints.Add(ray2.PointAt(wallPoint2));
         //Add this point to result volume
          volumePoints.Add(p);
          Mesh cube = getGridCube(p, dx, dy, dz);
          volume.Add(cube);
        }
      }
    }
// Output
    A = volumePoints;
    B = volume;
    C = shadowPoints;
  }
```

```
// Generating the Pixel GridPoints
  public List <Point3d> getGridPoints(Point3d cornerA,
Point3d cornerB, int xCount, int yCount, int zCount)
  {
    List<Point3d> points = new List<Point3d>();
    double x0 = cornerA.X;
    double y0 = cornerA.Y;
    double z0 = cornerA.Z;
    double x1 = cornerB.X;
    double y1 = cornerB.Y;
    double z1 = cornerB.Z;
    double dx = (x1 - x0) / (xCount - 1.0);
    double dy = (y1 - y0) / (yCount - 1.0);
    double dz = (z1 - z0) / (zCount - 1.0);

    for (int k = 0; k < zCount; ++k)
    {
      double z = z0 + k * dz;
      for (int j = 0; j < yCount; ++j)
      {
        double y = y0 + j * dy;
        for (int i = 0; i < xCount; ++i)
        {
          double x = x0 + i * dx;
          Point3d p = new Point3d(x, y, z);
          points.Add(p);
        }
      }
    }
    return points;
  }
```

```
// Generating Pixel Mesh Cube for the Point
public Mesh getGridCube(Point3d p, double x, double y,
double z)
  {
    Point3d p1 = (new Point3d(-x, -y, -z)) + p;
    Point3d p2 = (new Point3d(x, -y, -z)) + p;
    Point3d p3 = (new Point3d(x, -y, z)) + p;
    Point3d p4 = (new Point3d(-x, -y, z)) + p;
    Point3d p5 = (new Point3d(-x, y, z)) + p;
    Point3d p6 = (new Point3d(x, y, z)) + p;
    Point3d p7 = (new Point3d(x, y, -z)) + p;
    Point3d p8 = (new Point3d(-x, y, -z)) + p;

    Mesh cube = new Mesh();
    cube.Vertices.Add(p1);
    cube.Vertices.Add(p2);
    cube.Vertices.Add(p3);
    cube.Vertices.Add(p4);
    cube.Vertices.Add(p5);
    cube.Vertices.Add(p6);
    cube.Vertices.Add(p7);
    cube.Vertices.Add(p8);
    cube.Faces.AddFace(0, 1, 2, 3);
    cube.Faces.AddFace(1, 6, 5, 2);
    cube.Faces.AddFace(6, 7, 4, 5);
    cube.Faces.AddFace(7, 0, 3, 4);
    cube.Faces.AddFace(2, 5, 4, 3);
    cube.Faces.AddFace(0, 7, 6, 1);

    return cube;
  }
```