```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d
from scipy.spatial import distance
from shapely.geometry import Point
from shapely.geometry import LineString
import math
```

Base functions:

```
# Four random points to start off with
points = np.array([[0, 0.8], [-0.15, 0.0], [0.15, -0.35], [0.2, 0.7]])

# eucl. distance
def dist(a, b):
  return distance.euclidean(a, b)

# smalled distance of a point in "points" to a given point
def get_smallest_dist(point):
  global points
  dists = [dist(p, point) for p in points]
  return min(dists)

# closest point on unit circle to a given point
def closest_point_on_circle(p):
  return [p[0] / dist(p, [0,0]),
          p[1] / dist(p, [0,0])]

# true if point is inside unit circle
def in_circle(p):
  return dist([0,0], p) <= 1.0

# true if edge intersects unti circle
def intersects_circle(p1, p2):
  return in_circle(p1) != in_circle(p2)

# intersection point of an edge with the unit circle boundary
def intersection_point(p1, p2):
  p = Point(0,0)
  c = p.buffer(1).boundary
  l = LineString([p1, p2])
  i = c.intersection(l)
  return [i.x, i.y]

# returns largest empty circle in a unit circle with samples
def get_largest_empty():
  global points
  vor = Voronoi(points)

  verts = vor.vertices
  verts_ = list(filter(lambda v:in_circle(v), vor.vertices))

  for (i1, i2) in vor.ridge_vertices:
    if intersects_circle(verts[i1], verts[i2]):
      verts_.append(intersection_point(verts[i1], verts[i2]))
```

```
  return max(verts_, key=lambda v:get_smallest_dist(v))

def add_new_sample():
  global points
  le = get_largest_empty()
  points = np.append(points, [le], axis=0)
  return le
```

Adding Samples and displaying the final graph

```
# Add samples until we have 1024
samples = 1024 - 4
for i in range(samples):
  add_new_sample()

fig,ax = plt.subplots(1,1)
vor = Voronoi(points)
fig = voronoi_plot_2d(vor, ax)
ax.add_patch(plt.Circle((0, 0), 1, color='grey'))

plt.show()
```

Projecting points from unit circle to unit hemisphere

```
# transform points into 3d

points_3d = [[x,y,math.sqrt(1-(x*x + y*y))] for [x,y] in points]
```

Output vectors in correct HLSL format. (This code is copied into the shader)

```
# Output points:

print("static const uint sampledirs_len = "+str(len(points_3d))+ ";")
```

```
print("static const float3 sampledirs[sampledirs_len] = {")
for [x,y,z] in points_3d:
  print("\tfloat3("+str(x)+"f, "+str(y)+"f, "+str(z)+"f),")
print("};")
```

```
    static const uint sampledirs_len = 1024;
    static const float3 sampledirs[sampledirs_len] = {
            float3(0.0f, 0.8f, 0.5999999999999999f),
            float3(-0.15f, 0.0f, 0.9886859966642595f),
            float3(0.15f, -0.35f, 0.9246621004453465f),
            float3(0.2f, 0.7f, 0.6855654600401044f),
            float3(0.3960526315789472f, 0.1644736842105261f, 0.9033773963419756f),
            float3(-0.07500000000000012f, 0.4f, 0.9134412953222555f),
            float3(0.15936352509179913f, -0.03840269277845787f, 0.9864727568755657f),
            float3(0.5072787154544864f, -0.2047969142562647 8f, 0.8370940979113151f),
            float3(0.22152255639097737f, 0.4042293233082706f, 0.8874268483586374f),
            float3(0.7283599624060153f, 0.5897791353383455f, 0.3487869502769013f),
            float3(0.898444858448487f, 0.11441042863127807f, 0.42391873059335966f),
            float3(0.5089335810678658f, 0.8594225457465681f, 0.0487800976211223f),
            float3(0.663079897311403f, 0.2983100278446798f, 0.6865392756928081f),
            float3(0.44843414339419996f, 0.5694095730397422f, 0.6889699247202368f),
            float3(0.2131329221685515f, 0.9762658443371027f, 0.03846243191294740 3f),
            float3(0.9218625168423844f, 0.38694805368937607f, 0.020992946125504112f),
            float3(0.6422491515718868f, 0.029083277845406097f, 0.7659439863690807f),
            float3(0.8826876618218602f, -0.26555720200984584f, 0.38773942813203716f),
            float3(0.5541641651078596f, -0.8323123769760341f, 0.012577171494933561f),
            float3(-0.6651531866398313f, -0.745131302834141f, 0.048482778793851544f),
            float3(-0.9341231831105874f, 0.35405434683323483f, 0.04538059347160964f),
            float3(-0.06644456922891623f, -0.9418697153805832f, 0.3293426156282891f),
            float3(-0.5799124918868 54f, 0.8134161627161527f, 0.045339254357664595f),
            float3(-0.6991870536452531f, -0.17095815308392892f, 0.6941979356847742f),
            float3(-0.30955558055539056f, -0.44033335476176344f, 0.8427822252701264f),
            float3(-0.49896953255413223f, 0.27246303735389965f, 0.8226744792799873f),
            float3(0.21229618061906125f, -0.7082928307271661f, 0.6732396286873322f),
            float3(-0.882616814434686f, -0.46992611972767323f, 0.012529998996795768f),
            float3(-0.9995226188704875f, 0.009717316126570917f, 0.029327600201489324f),
            float3(0.6505675436601258f, -0.5096002233270589f, 0.5630892322900891f),
            float3(-0.29362737401950845f, 0.648023882628658f, 0.7027432054237785f),
            float3(-0.33991820646863646f, -0.7647412021150891f, 0.5473815001429203f),
            float3(-0.270002644678789f, 0.9620207204185903f, 0.04018339646864259f),
            float3(-0.6773503264414749f, 0.5223041928449537f, 0.5180683983859746f),
            float3(-0.07511848163082752f, -0.6259000624242043f, 0.7762772221151772f),
            float3(-0.4021042135406566f, -0.15772388895384576f, 0.9019065230422302f),
            float3(-0.5968946575872437f, -0.43947398022048856f, 0.6712521049895999f),
            float3(-0.7337443554434331f, 0.11155807289643732f, 0.6702044592708511f),
            float3(0.37961035198190296f, -0.49463988930745867f, 0.7818103098412561f),
            float3(0.201959519693582 78f, -0.9790646359580517f, 0.025392735604205494f),
            float3(-0.1053117138705861f, -0.26526718331764526f, 0.9584063670366357f),
            float3(0.076158855468645 22f, 0.19896229950281472f, 0.9770434136261593f),
            float3(-0.25398424439809075f, 0.226528295824642f, 0.9403068301295671f),
            float3(-0.9389739132238335f, -0.22959358153811735f, 0.25615381629329786f),
            float3(-0.5142475865014592f, 0.04302381377380404f, 0.8565619482709825f),
            float3(-0.45509321578911255f, 0.4923901642993881f, 0.7419178465598101f),
            float3(0.8720558506581066f, -0.48785618518956064f, 0.038922177550274446f),
            float3(0.9960723718230658f, -0.07994870819857097f, 0.03805304387498792f),
            float3(0.019642857142857073f, 0.5892857142857143f, 0.8076858950743419f),
            float3(0.2965596062257203f, -0.19846478743469925f, 0.934164936241853f),
            float3(-0.49456713958359905f, -0.622775784375967f, 0.6062620446960174f),
            float3(-0.7283396521584022f, 0.3200604387784298f, 0.6058734741039062f),
            float3(0.4643580719872781f, 0.36149377649686876f, 0.8085133459221868f),
            float3(-0.2730944967834793f, -0.9612462517835254f, 0.03774969216697549f),
            float3(-0.0655309313888664f, 0.9967806717784059f, 0.0461951231224426f),
            float3(0.683165321562728f, -0.30819404325819744f, 0.6620434843076841f),
```

Display points in 3D view

```
fig = plt.figure(figsize=(8, 6), dpi=80)
ax = fig.add_subplot(projection='3d')

ax.set_zlim(0, 1)
ax.set_xlim(-1, 1)
ax.set_ylim(-1, 1)

for [x,y,z] in points_3d:
    ax.scatter(x, y, z, marker='^')
    ax.plot([0, 1*x], [0, 1*y], zs=[0, 1*z])

plt.show()
```

Display Voronoi diagram of 3D view

```
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull
from scipy.spatial import Delaunay
import random

fig = plt.figure(figsize=(8, 6), dpi=80)
ax = Axes3D(fig)

ax.set_zlim(0, 1)
ax.set_xlim(-1, 1)
ax.set_ylim(-1, 1)

vor = Voronoi(points)
verts = vor.vertices
```

```
cols = ["#"+''.join([random.choice('0123456789ABCDEF') for j in range(6)])
             for i in range(100)]

#print(vor.regions[2])

for pol in vor.regions:
  col = random.choice(cols)

  if len(pol) > 2:
    # This line gets rid of the "point at infinity", because those have an index of -1
    #if any([i<0 for i in pol]):
    #  continue

    polygon=[]
    lg = 0
    for i in reversed(pol):
      if i >= 0:
        lg = i
        break
    ng = 0

    for j in range(len(pol)):
      i = pol[j]

      if i >= 0:
        lg = i
        ng = 0
      else:
        if ng > 1:
          q = j + 1
          while True:
            if pol[q] >= 0:
              lg = q
              break
            q += 1
        ng += 1

      if i < 0:
        polygon.append(closest_point_on_circle(verts[lg]))
        continue
      elif not in_circle(verts[i]):
        polygon.append(closest_point_on_circle(verts[i]))
      else:
        polygon.append(verts[i])
    polygon = np.array(polygon)

    hull = Delaunay(polygon)
    for s in hull.simplices:
      try:
        pol3d = np.array([[x,y,math.sqrt(1-(x*x + y*y))] for [x,y] in polygon[s]])
      except ValueError:
        pol3d = np.array([[x,y,0] for [x,y] in polygon[s]])
      tri = Poly3DCollection(pol3d)
      tri.set_color(col)
      #tri.set_alpha(0.5)
      ax.add_collection3d(tri)

#for [x,y,z] in points_3d:
```

```
    #ax.scatter(x, y, z, marker='^')
    #ax.plot([0, 1.5*x], [0, 1.5*y], zs=[0, 1.5*z])


plt.show()
```

Colab paid products - Cancel contracts here