Coding Example: Blue Noise Sampling using DART method

In this lesson, we will try to do the blue noise sampling using the DART method discussed in the previous lesson. We will look at both solutions, i.e., Pythonic and NumPy approach and see which one is more efficient.

WE'LL COVER THE FOLLOWING

Let's consider the unit surface and a minimum radius `r` to be enforced between each point.

Knowing that the densest packing of circles in the plane is the hexagonal lattice of the bee's honeycomb, we know this density is $d = \frac{1}{6}\pi\sqrt{3}$ (in fact I learned it while writing this course). Considering circles with radius `r`, we can pack at most $\frac{d}{\pi r^2} = \frac{\sqrt{3}}{6r^2} = \frac{1}{2r^2\sqrt{3}}$.

We know the theoretical upper limit for the number of discs we can pack onto the surface, but we'll likely not reach this upper limit because of random placements.

Furthermore, because a lot of points will be rejected after a few have been accepted, we need to set a limit on the number of successive failed trials before we stop the whole process.

Python Implementation

```
# -----------------------------------------------------------------------------
# From Numpy to Python
# Copyright (2017) Nicolas P. Rougier - BSD license
# More information at https://github.com/rougier/numpy-book
# -----------------------------------------------------------------------------
```

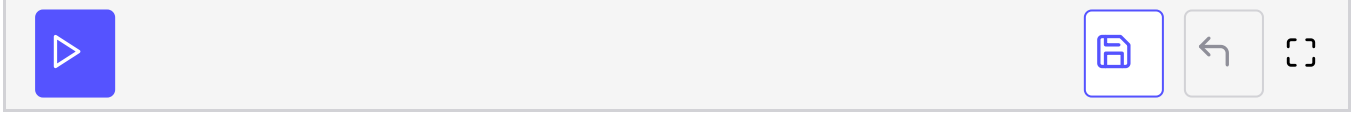
```
import math
import random
import matplotlib.pyplot as plt


def DART_sampling_python(width=1.0, height=1.0, radius=0.025, k=100):
    def squared_distance(p0, p1):
        dx, dy = p0[0]-p1[0], p0[1]-p1[1]
        return dx*dx+dy*dy

    points = []
    i = 0
    last_success = 0
    while True:
        x = random.uniform(0, width)
        y = random.uniform(0, height)
        accept = True
        for p in points:
            if squared_distance(p, (x, y)) < radius*radius:
                accept = False
                break
        if accept is True:
            points.append((x, y))
            if i-last_success > k:
                break
            last_success = i
        i += 1
    return points

if __name__ == '__main__':

    plt.figure()
    plt.subplot(1, 1, 1, aspect=1)

    points = DART_sampling_python()
    X = [x for (x, y) in points]
    Y = [y for (x, y) in points]
    plt.scatter(X, Y, s=10)
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.savefig("output/output1.png")
    plt.show()
```

NumPy Implementation

Now we will do the vectorization of the DART method. The idea is to pre-compute enough uniform random samples as well as paired distances and to test for their sequential inclusion.

```
# ----------------------------------------------------------------------------
# From Numpy to Python
# Copyright (2017) Nicolas P. Rougier - BSD license
# More information at https://github.com/rougier/numpy-book
```

```
# -----------------------------------------------------------------------------
import numpy as np
import matplotlib.pyplot as plt

from scipy.spatial.distance import cdist


def DART_sampling_numpy(width=1.0, height=1.0, radius=0.025, k=100):

    # Theoretical limit
    n = int((width+radius)*(height+radius) / (2*(radius/2)*(radius/2)*np.sqrt(3))) + 1
    # 5 times the theoretical limit
    n = 5*n

    # Compute n random points
    P = np.zeros((n, 2))
    P[:, 0] = np.random.uniform(0, width, n)
    P[:, 1] = np.random.uniform(0, height, n)

    # Computes respective distances at once
    D = cdist(P, P)

    # Cancel null distances on the diagonal
    D[range(n), range(n)] = 1e10

    points, indices = [P[0]], [0]
    i = 1
    last_success = 0
    while i < n and i - last_success < k:
        if D[i, indices].min() > radius:
            indices.append(i)
            points.append(P[i])
            last_success = i
        i += 1
    return points


if __name__ == '__main__':

    plt.figure()
    plt.subplot(1, 1, 1, aspect=1)

    points = DART_sampling_numpy()
    X = [x for (x, y) in points]
    Y = [y for (x, y) in points]
    plt.scatter(X, Y, s=10)
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.savefig("output/output2.png")
    plt.show()
```



In the next lesson, we will use another Bridson method to do the sampling!