

educative

From Python to Numpy

0% COMPLETED

Search Course

Coding Example: Blue Noise Sampling using Bridson method

Conclusion

Custom Vectorization

Typed list

Coding Example: Modifying the list

Memory-aware Array: Glumpy

Memory-aware Array: Array Subclass in GPUData class

Conclusion

Beyond NumPy

EXPLORE

TRACKS

MY COURSES

EDPRESSO

REFER A FRIEND

Coding Example: Blue Noise Sampling using Bridson method

In this lesson, we will try to do the blue noise sampling using the Bridson method. First we will discuss the step-by-step approach and then implement it in code.

We'll cover the following

Bridson method

Step 0:

Step 1:

Step 2:

Implementation

Random vs. Regular vs. Bridson Sampling

Further Readings

Bridson method

If the vectorization of the previous method poses no real difficulty, the speed improvement is not so good and the quality remains low and dependent on the k parameter. The higher, the better since it basically governs how hard to try to insert a new sample. But, when there is already a large number of accepted samples, only chance allows us to find a position to insert a new sample. We could increase the k value but this would make the method even slower without any guarantee in quality. It's time to think out-of-the-box and luckily enough, Robert Bridson did that for us and proposed a simple yet efficient method:

Step 0:

Initialize an n -dimensional background grid for storing samples and accelerating spatial searches. We pick the cell size to be bounded by $\frac{T}{\sqrt{n}}$, so that each grid cell will contain at most one sample, and thus the grid can be implemented as a simple n -dimensional array of integers: the default `-1` indicates no sample, a non-negative integer gives the index of the sample located in a cell.

Step 1:

Select the initial sample, `x_0`, randomly chosen uniformly from the domain. Insert it into the background grid, and initialize the "active list" (an array of sample indices) with this index (zero).

Step 2:

While the active list is not empty, choose a random index from it (say i). Generate up to k points chosen uniformly from the spherical annulus between radius r and $2r$ around x_i . For each point in turn, check if it is within distance r of existing samples (using the background grid to only test nearby samples). If a point is adequately far from existing samples, emit it as the next sample and add it to the active list. If after k attempts no such point is found, instead remove i from the active list.

Implementation

The implementation poses no real problem. Note that not only is this method fast, but it also offers a better quality (more samples) than the DART method even with a high k parameter. Here's the complete Bridson implementation:

```
1 # -----
2 # From Numpy to Python
3 # Copyright (2017) Nicolas P. Rougier - BSD license
4 # More information at https://github.com/rougier/numpy-book
5 # -----
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9
10 def Bridson_sampling(width=1.0, height=1.0, radius=0.025, k=30):
11     # References: Fast Poisson Disk Sampling in Arbitrary Dimensions
12     # Robert Bridson, SIGGRAPH, 2007
13     def squared_distance(p0, p1):
14         return (p0[0]-p1[0])**2 + (p0[1]-p1[1])**2
15
16     def random_point_around(p, k=1):
17         # WARNING: This is not uniform around p but we can live with it
18         R = np.random.uniform(radius, 2*radius, k)
19         T = np.random.uniform(0, 2*np.pi, k)
20         P = np.empty((k, 2))
21         P[:, 0] = p[0]+R*np.sin(T)
22         P[:, 1] = p[1]+R*np.cos(T)
23         return P
24
25     def in_limits(p):
26         return 0 <= p[0] < width and 0 <= p[1] < height
27
28     def neighborhood(shape, index, n=2):
29         row0, col = index
30         row0, row1 = max(row-n, 0), min(row+n+1, shape[0])
31         col0, col1 = max(col-n, 0), min(col+n+1, shape[1])
32         I = np.dstack(np.mgrid[row0:row1, col0:col1])
33         I = I.reshape(I.size//2, 2).tolist()
34         I.remove([row, col])
35         return I
36
37     def in_neighborhood(p):
38         i, j = int(p[0]/cellsize), int(p[1]/cellsize)
39         if M[i, j]:
40             return True
41         for (i, j) in N[(i, j)]:
42             if M[i, j] and squared_distance(p, P[i, j]) < squared_radius:
43                 return True
44         return False
45
46     def add_point(p):
47         points.append(p)
48         i, j = int(p[0]/cellsize), int(p[1]/cellsize)
49         P[i, j], M[i, j] = p, True
50
51     # Here '2' corresponds to the number of dimension
52     cellsize = radius/np.sqrt(2)
53     rows = int(np.ceil(width/cellsize))
54     cols = int(np.ceil(height/cellsize))
55
56     # Squared radius because we'll compare squared distance
57     squared_radius = radius*radius
58
59     # Positions cells
60     P = np.zeros((rows, cols, 2), dtype=np.float32)
61     M = np.zeros((rows, cols), dtype=bool)
62
63     # Cache generation for neighborhood
64     N = {}
65     for i in range(rows):
66         for j in range(cols):
67             N[(i, j)] = neighborhood(M.shape, (i, j), 2)
68
69     points = []
70     add_point((np.random.uniform(width), np.random.uniform(height)))
71     while len(points):
72         i = np.random.randint(len(points))
73         p = points[i]
74         del points[i]
75         Q = random_point_around(p, k)
76         for q in Q:
77             if in_limits(q) and not in_neighborhood(q):
78                 add_point(q)
79     return P[M]
80
81
82 if __name__ == '__main__':
83     plt.figure()
84     plt.subplot(1, 1, 1, aspect=1)
85
86     points = Bridson_sampling()
87     X = [x for (x, y) in points]
88     Y = [y for (x, y) in points]
89     plt.scatter(X, Y, s=10)
90     plt.xlim(0, 1)
91     plt.ylim(0, 1)
92     plt.savefig("output/BridsonSampling.png")
93     plt.show()
```

RUN

SAVE

RESET

Random vs. Regular vs. Bridson Sampling

The image below shows the pattern of three samplings, i.e., Random Sampling, Regular Sampling and Bridson Sampling.



Comparison of uniform, grid-jittered and Bridson sampling.

Here's the code to plot all three samplings:

```
1 # -----
2 # From Numpy to Python
3 # Copyright (2017) Nicolas P. Rougier - BSD license
4 # More information at https://github.com/rougier/numpy-book
5 # -----
6 import numpy as np
7
8 def Bridson_sampling(width=1.0, height=1.0, radius=0.025, k=30):
9     # References: Fast Poisson Disk Sampling in Arbitrary Dimensions
10     # Robert Bridson, SIGGRAPH, 2007
11     def squared_distance(p0, p1):
12         return (p0[0]-p1[0])**2 + (p0[1]-p1[1])**2
13
14     def random_point_around(p, k=1):
15         # WARNING: This is not uniform around p but we can live with it
16         R = np.random.uniform(radius, 2*radius, k)
17         T = np.random.uniform(0, 2*np.pi, k)
18         P = np.empty((k, 2))
19         P[:, 0] = p[0]+R*np.sin(T)
20         P[:, 1] = p[1]+R*np.cos(T)
21         return P
22
23     def in_limits(p):
24         return 0 <= p[0] < width and 0 <= p[1] < height
25
26     def neighborhood(shape, index, n=2):
27         row, col = index
28         row0, row1 = max(row-n, 0), min(row+n+1, shape[0])
29         col0, col1 = max(col-n, 0), min(col+n+1, shape[1])
30         I = np.dstack(np.mgrid[row0:row1, col0:col1])
31         I = I.reshape(I.size//2, 2).tolist()
32         I.remove([row, col])
33         return I
34
35     def in_neighborhood(p):
36         i, j = int(p[0]/cellsize), int(p[1]/cellsize)
37         if M[i, j]:
38             return True
39         for (i, j) in N[(i, j)]:
40             if M[i, j] and squared_distance(p, P[i, j]) < squared_radius:
41                 return True
42         return False
43
44     def add_point(p):
45         points.append(p)
46         i, j = int(p[0]/cellsize), int(p[1]/cellsize)
47         P[i, j], M[i, j] = p, True
48
49     # Here '2' corresponds to the number of dimension
50     cellsize = radius/np.sqrt(2)
51     rows = int(np.ceil(width/cellsize))
52     cols = int(np.ceil(height/cellsize))
53
54     # Squared radius because we'll compare squared distance
55     squared_radius = radius*radius
56
57     # Positions cells
58     P = np.zeros((rows, cols, 2), dtype=np.float32)
59     M = np.zeros((rows, cols), dtype=bool)
60
61     # Cache generation for neighborhood
62     N = {}
63     for i in range(rows):
64         for j in range(cols):
65             N[(i, j)] = neighborhood(M.shape, (i, j), 2)
66
67     points = []
68     add_point((np.random.uniform(width), np.random.uniform(height)))
69     while len(points):
70         i = np.random.randint(len(points))
71         p = points[i]
72         del points[i]
73         Q = random_point_around(p, k)
74         for q in Q:
75             if in_limits(q) and not in_neighborhood(q):
76                 add_point(q)
77     return P[M]
78
79
80 def draw_voronoi(ax, X, Y):
81     from voronoi import voronoi
82     from matplotlib.path import Path
83     from matplotlib.patches import PathPatch
84     cells, triangles, circles = voronoi(X, Y)
85     for i, cell in enumerate(cells):
86         codes = [Path.MOVETO] \
87                 + [Path.LINE2] * (len(cell)-2) \
88                 + [Path.CLOSEPOLY]
89         path = Path(cell, codes)
90         patch = PathPatch(path,
91                           facecolor="none", edgecolor="0.5", linewidth=0.5)
92         ax.add_patch(patch)
93
94
95
96 if __name__ == '__main__':
97     import matplotlib.pyplot as plt
98
99     # Benchmark
100     # from tools import print_timeit
101     # print_timeit("poisson_disk_sample()", globals())
102
103     fig = plt.figure(figsize=(18, 6))
104
105     ax = plt.subplot(1, 3, 1, aspect=1)
106     n = 1000
107     X = np.random.uniform(0, 1, n)
108     Y = np.random.uniform(0, 1, n)
109     ax.scatter(X, Y, s=10, facecolor='w', edgecolor='0.5')
110     ax.set_xlim(0, 1), ax.set_ylim(0, 1)
111     ax.set_xticks([]), ax.set_yticks([])
112     ax.set_title("Random sampling", fontsize=18)
113     draw_voronoi(ax, X, Y)
114
115     ax = plt.subplot(1, 3, 2, aspect=1)
116     n = 32
117     X, Y = np.meshgrid(np.linspace(0, 1, n), np.linspace(0, 1, n))
118     X += 0.45*np.random.uniform(-1/n, 1/n, (n, n))
119     Y += 0.45*np.random.uniform(-1/n, 1/n, (n, n))
120     ax.scatter(X, Y, s=10, facecolor='w', edgecolor='0.5')
121     ax.set_xlim(0, 1), ax.set_ylim(0, 1)
122     ax.set_xticks([]), ax.set_yticks([])
123     ax.set_title("Regular grid + jittering", fontsize=18)
124     draw_voronoi(ax, X.ravel(), Y.ravel())
125
126     ax = plt.subplot(1, 3, 3, aspect=1)
127     P = Bridson_sampling(width=1.0, height=1.0, radius=0.025, k=30)
128     plt.scatter(P[:, 0], P[:, 1], s=10, facecolor='w', edgecolor='0.5')
129     ax.set_xlim(0, 1), ax.set_ylim(0, 1)
130     ax.set_xticks([]), ax.set_yticks([])
131     ax.set_title("Bridson sampling", fontsize=18)
132     draw_voronoi(ax, P[:, 0], P[:, 1])
133
134     plt.tight_layout(pad=2.5)
135     plt.savefig("output/sampling.png")
136     plt.show()
```

RUN

SAVE

RESET

Further Readings

- Visualizing Algorithms, Mike Bostock, 2014.
- Stippling and Blue Noise, Jose Esteve, 2012.
- Poisson Disk Sampling, Herman Tulleken, 2009.
- Fast Poisson Disk Sampling in Arbitrary Dimensions, Robert Bridson, SIGGRAPH, 2007.

☒ Mark as Completed

← Back

Next →

Coding Example: Blue Noise Sampling...

Conclusion

Stuck? Get help on [DISCUSS](#)

Send feedback

Recommend