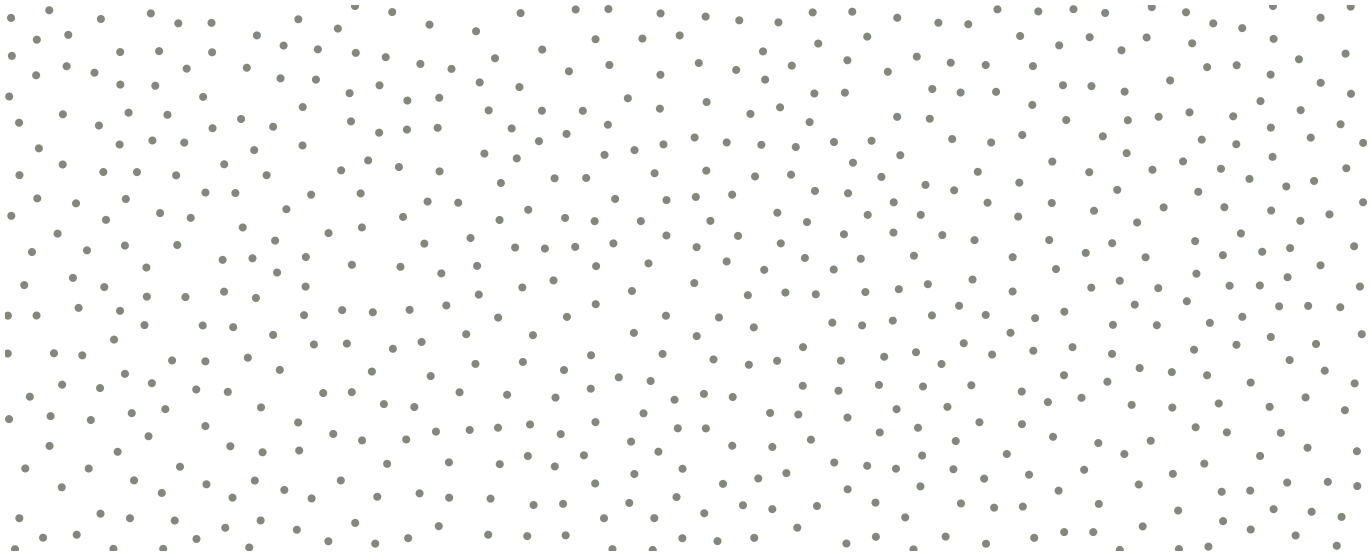
泊松分布Poisson-Disc采样点的生成

2018.12 / D3.js

泊松分布采样生成的点尽量紧密堆积，并且各点之间的距离均不小于指定的最小距离，从而可以产生更自然的采样图案。下面使用Robert Bridson提出的一种高效算法，时间复杂度为O(n)：

1. 规定采样点之间的最小距离为r。
2. 在画布尺寸范围内随机生成一个活跃采样点，在这个采样点周围的环形区域中再随机生成k个候选采样点，这个环形区域以该活跃采样点为圆心，半径从r延伸到2r。
3. 在这k个随机候选采样点中，剔除掉与已选定的采样点距离小于r的点，剩下的作为新的活跃采样点。
4. 如果这k个采样点都被剔除了，没有剩下任何可用的点，则将此环形区域圆心处的所选活跃采样点标记为非活跃，不再用于生成候选项。
5. 当所有采样点均为非活跃状态时，算法迭代结束。
6. 在对候选采样点剔除筛选时，使用了对角线长度为r的单元网格来加速距离检查。每个单元网格最多只能包含一个采样点，只需检查候选采样点周边固定数量的相邻单元网格即可。



```
const width = document.querySelector('svg#d3').parentNode.clientWid
const height = Math.round(width * 0.4)
const radius = Math.round(width / 50)
```

```
const sample = poissonDiscSampler(width, height, radius)

const svg = d3.select('svg#d3')
    .attr('width', width)
    .attr('height', height)
    .attr('viewBox', `0 0 ${width} ${height}`)
    .attr('fill', '#83887c')

const timer = d3.timer(function() {
    const s = sample()
    if (!s) return timer.stop()
    svg.append("circle")
        .attr("cx", s[0])
        .attr("cy", s[1])
        .attr("r", 0)
        .transition()
        .attr("r", 2)
})

function poissonDiscSampler(width, height, radius) {
    const k = 30,
        cellSize = radius * Math.SQRT1_2,
        gridWidthLength = Math.ceil(width / cellSize),
        gridHeightLength = Math.ceil(height / cellSize),
        grid = new Array(gridWidthLength * gridHeightLength),
        queue = []
    let queueLength = 0,
        sampleLength = 0

    return function() {
        if (!sampleLength) return sample(Math.random() * width, Mat

        while (queueLength) {
            const i = Math.random() * queueLength | 0,
                s = queue[i]

            for (let j = 0; j < k; ++j) {
                const a = 2 * Math.PI * Math.random(),
                    r = radius * (Math.random() + 1),
                    x = s[0] + r * Math.cos(a),
                    y = s[1] + r * Math.sin(a)

                if (0 <= x && x < width && 0 <= y && y < height && 
            }

            queue[i] = queue[--queueLength]
            queue.length = queueLength
```

```
            }
        }

        function far(x, y) {
            let i = x / cellSize | 0,
                j = y / cellSize | 0
            const i0 = Math.max(i - 2, 0),
                j0 = Math.max(j - 2, 0),
                i1 = Math.min(i + 3, gridWidthLength),
                j1 = Math.min(j + 3, gridHeightLength)

            for (j = j0; j < j1; ++j) {
                for (i = i0; i < i1; ++i) {
                    const s = grid[j * gridWidthLength + i]
                    if (s) {
                        const dx = s[0] - x,
                            dy = s[1] - y
                        if (dx * dx + dy * dy < radius * radius) return
                    }
                }
            }

            return true
        }

        function sample(x, y) {
            const s = [x, y]
            queue.push(s)
            grid[gridWidthLength * (y / cellSize | 0) + (x / cellSize |
            ++sampleLength
            ++queueLength
            return s
        }
    }
```

参考：https://bl.ocks.org/mbostock/19168c663618b7f07158

« JavaScript四则运算 / 绘制D3树状图tree和集群图cluster »