

# 可视计算与交互概论

Final Project A.4

姓名: 陆昶安 学号: 2300012991

## 一、选题

我的大作业选题是提供的选题中的 A 4. Fast Poisson Disk Sampling in Arbitrary Dimensions . 该选题的目标是实现论文中的算法, 并进行相关论述与应用.

## 二、实现思路

### 1. 算法描述和分析

根据论文中的描述, 算法的流程如下:

- (1) 生成一个  $n$  维数组, 记录一个  $n$  维空间中每个网格中是否有样本, 以及(如果有样本)样本的位置. 由于我们要求两个采样点的欧式距离小于  $r$ , 我们可以让网格的边长为  $\frac{r}{\sqrt{n}}$ , 这样每个网格里至多只有一个样本.
- (2) 随机选择  $n$  维空间中的一个点作为第一个样本, 并把它加入一个列表中.
- (3) 如果这个列表非空, 从中随机选择一个样本, 在以它为中心的内外半径分别为  $r$  和  $2r$  的球环内开始生成至多  $K$  个点(一般  $K = 30$ ). 如果次生成的点与之前的采样点的距离大于  $r$ , 就把它作为样本并加入列表中, 并结束生成; 如果这  $K$  个点都不能作为新的样本, 则从列表中删除选出的样本(删除后仍视为已经存在的样本). 重复该步骤直到列表为空.

如果我们以最后生成的样本数量  $N$  作为复杂度的考量, 不难看到, 只有步骤 (3) 对复杂度有影响. 步骤 (3) 的结果只有两种结果, 其一是将一个新的样本加入列表, 其二是将一个已有的样本移除列表, 而每个样本都在列表中出现过, 因此总共向列表中加入  $N$  个样本(其中之一在步骤 (2) 中加入), 从列表中移除  $N$  个样本.

每次迭代中检查  $K$  个点中的某个点与已有样本的距离时, 只需要检查它所在的网格及其周围与它距离为  $\sqrt{n}$  个网格边长的网格, 即检查的网格数量是  $O(f(n))$  的, 而  $K$  与  $n$  都是不太大的常数, 所以这个算法是  $O(N)$  的; 但是如果要与之前所有样本点计算距离, 复杂度将会达到  $O(N^2)$ , 这就是这个算法快速的原因.

值得一提的是, 在实际应用中, 一般我们需要将蓝噪声充满整个空间, 所以样本数量  $N$  与空间的大小有关, 可以认为  $N = O((L/r)^n)$ .

### 2. 算法实现<sup>1</sup>

以二维情况为例, 我们来生成一个充满  $L \times L$  正方形的噪声.

首先, 声明一个二维布尔数组  $A$ , 大小为  $(\frac{L}{r/\sqrt{2}})^2$ , 用于记录网格的是否被占用, 声明另一个同样大小的二维  $(x,y)$  坐标数组  $B$ , 用于记录每个网格中的样本在大正方形中的坐标. 初始时  $A$  每个元素均为 `false`. 再声明两个列表  $List$  和  $Sample$ , 其中  $List$  为上面描述的列表, 应当具有快速随机插入删除的特点,  $Sample$  来记录所有样本.

在正方形内随机选取一个坐标  $(x_0, y_0)$ , 那么它属于网格  $(X_0, Y_0)(\lfloor \frac{x_0}{r/\sqrt{2}} \rfloor, \lfloor \frac{y_0}{r/\sqrt{2}} \rfloor)$ . 将  $A[X_0][Y_0]$  设为 `true`, 表示被占用, 同时将  $B[X_0][Y_0]$  设置为  $(x_0, y_0)$ , 表示这个网格被该点占据. 将  $(x_0, y_0)$  添加到  $List$  和  $Sample$  中.

接下来, 如果  $List$  非空, 从中随机取出一个点  $(x_i, y_i)$ . 下面尝试生成至多  $K = 30$  个点: 获得两个随机数  $\theta$  和  $d$ , 其中  $p_1(\theta) = \frac{1}{2\pi}, \theta \in [0, 2\pi], p_2(d) = \frac{2}{3}d, d \in (1, 2)$ , 为尝试生成的点  $(x, y)$  相对于取出的点的方向与距离, 它属于网格  $(X, Y) = (\lfloor \frac{x}{r/\sqrt{2}} \rfloor, \lfloor \frac{y}{r/\sqrt{2}} \rfloor)$ . 检查数组  $A$  中索引与  $(X, Y)$  的欧式距离小于  $\sqrt{n}$  的所有下标  $(U, V)$ , 如果  $A[U][V] == true$  且  $Distance((x, y), B[U][V]) < r$ , 说明这个生成的点与一个样本点距离过近, 应该被舍弃, 否则把它加入  $List$  和  $Sample$ , 且将  $A[X][Y]$  设为 `true`,  $B[X][Y]$  设置为  $(x, y)$  并立刻停止尝试. 如果 30 次尝试均失败, 则从  $List$  中移除点  $(x_i, y_i)$ .

任意维的蓝噪声需要任意维数组, 因此只需实现任意维的数组即可, 其他与二维情况类似.

### 3. 可视化与交互

对于上面我们生成的二维蓝噪声, 接下来我们考虑可视化. 我们利用 `lab` 的框架, 直接在一个 `ImageRGB` 的实例上画出这些样本点即可.

<sup>1</sup>源代码位于 `src` 文件夹中, 具体来说是 `src/VCX/Labs/Final` 下的文件, 其中 `task.cpp` 实现了该算法. 此外 `src/VCX/Engine/app.cpp` 也有所改动. 像其他 `lab` 一样, 本项目的编译方式为在 `src` 的上级文件夹的 `shell` 中输入命令 `xmake`, 随后输入 `xmake run` 或 `xmake run Final` 运行. 正常运行仍需要 `assests` 文件夹, 但不便在压缩包内包含, 需要从以往的 `lab` 分支上获取.

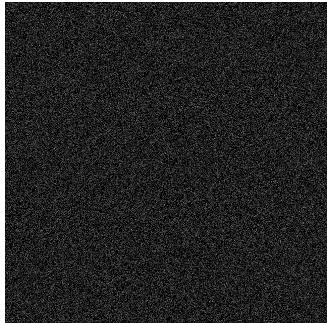
此外,我在第一项中实现了修改半径 `radius`,设置采样率 `sample rate` 模拟多次生成噪声叠加并平均,以及生成 `iter times` 次噪声并叠加平均以降低噪声的能量等交互. 此外,你还可以把一个图片文件拖动到窗口中的画布上面,利用这个蓝噪声重现 **Lab 1** 中的 dithering (我在压缩包中放置了一些图片以供选择,也可以使用其他图片).

对于高维蓝噪声,在第二项中,实现了修改半径 `radius`,维度 `dimension`,每一维的边长 `dim length` 等交互. 由于任意维情况下数组变得较为复杂,且需要较大的内存,不建议使用过大的维度和边长.

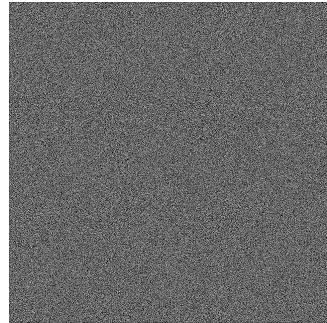
对于更高维蓝噪声,我仅实现了噪声的采样,并没有专门设置可视化的方案,取而代之的是将噪声的位置输出到文件中;在三维情况下,我用 Python 实现了可视化<sup>2</sup>,需要手动修改或在命令行中输入文件名.<sup>3</sup>

### 三、结果展示

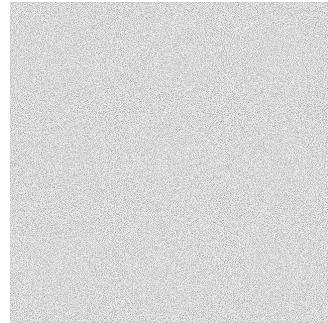
我们先来看半径分别为 `0.75`, `1`, `2` 生成的噪声:



(a)  $r = 0.75$



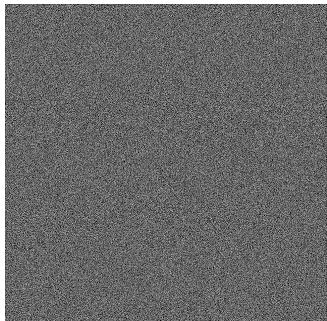
(b)  $r = 1$



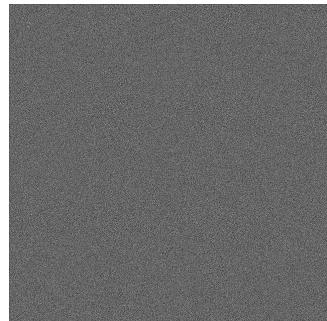
(c)  $r = 2$

可以看出,样本点的距离越小,样本数量越大,表现为图像更暗.

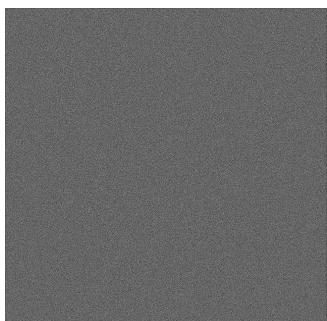
再来看生成次数分别为 `1`, `5`, `10`, `20` ( $r=1$ ) 叠加平均的噪声:



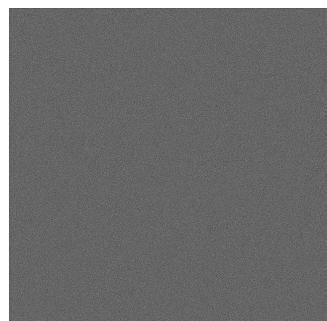
(a)  $i = 1$



(b)  $i = 5$



(c)  $i = 10$



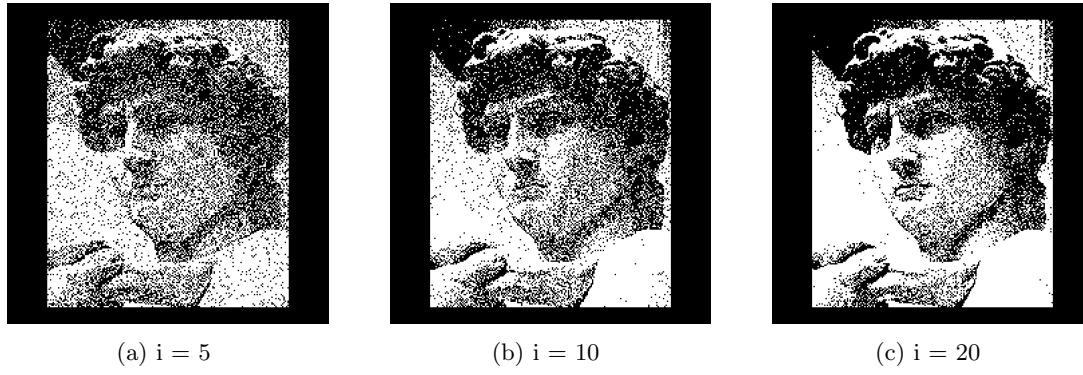
(d)  $i = 20$

<sup>2</sup>Python 源代码在 example 文件夹下,运行方式为在 example 文件夹的 shell 中输入 `python main.py` 或 `python main.py -file="Noise R=2.000, n=3, l=25.txt"` 等命令.

<sup>3</sup>输出的文件在 build 文件夹(编译后产生)下,可以用来作为 Python 程序的输入.

可以看出，生成次数越大，频率越低，能量也越低，表现为看起来更加平均。

然后我们加入 David 的头像，重现 lab 1 中的抖动：



不经叠加平均的蓝噪声具有极高的能量，会直接导致 David 的头像被噪声完全覆盖。而叠加平均的噪声更柔和，可以实现图像抖动处理。

最后我们来看三维的蓝噪声：

