程序设计大作业

题目 柏林噪声的研究与应用----纹理与地图的生成,

学生姓名 刘浩翔,杨家宁,赵文铎,李思豆,范祥荣

指导教师 李妍

2024 年 12 月 20 日

摘要

随着游戏科学技术的不断发展,追求游戏图形学的高效仿真生成成为了一个重要的研究领域,本研究旨在利用柏林噪声算法(Perlin noise)来探索如何利用该算法来进行材质纹理的生成与游戏地形的生成以及其应用,我们构建了一个基于 Python 的程序来实现一维与二维的柏林噪声,并将其投入应用之中,该程序能够通过输入特定的参数与颜色,进而生成不同的柏林噪声图案,从而在游戏图形学上形成了高效随机仿真效果。

我们首先对柏林噪声函数的算法进行了研究，包括随机梯度向量生成，线性插值与非线性插值，随后，我们用 Python 构建了一个程序，并基于程序来实现柏林噪声函数，通过具体的实验，我们发现该函数在生成游戏纹理上具有突出表现，并实现了在游戏中对于地形的生成。此外，我们还探讨了在柏林噪声函数中不同参数对于图像生成的影响，并提出了优化策略。

本研究的主要贡献包括：1）分析并实现了一个有效的柏林函数图像生成器，用于实现图像的生成，2）利用该生成器实现了仿真材质纹理的随机生成，为图形学材质生成提供了丰富的资源，3）利用该生成器实现了仿真地形图的生成，并基于此地形图在游戏 Minecraft 中生成了一个地形 4）分析并探究了柏林噪声函数中不同参数对于图像生成的影响。为游戏图形学提供了新的视角。

关键词：柏林噪声，游戏材质纹理，游戏地图生成

目录

第一章 柏林噪声的简介与分析

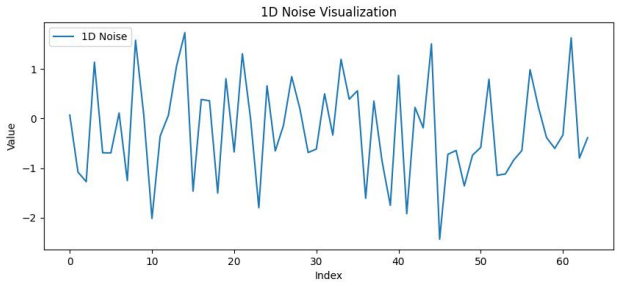
1.1. 简介

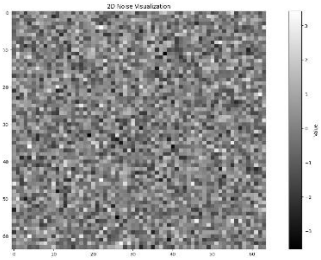
Perlin 噪声（Perlin noise，又称为柏林噪声）指由 Ken Perlin 发明的自然噪声生成算法，具有在函数上的连续性，并可在多次调用时给出一致的数值。在电子游戏领域中可以透过使用 Perlin 噪声生成具连续性的地形；或是在艺术领域中使用 Perlin 噪声生成图样。

1.2. 算法分析

1.2.1. 噪声

噪声的本质就是随机数，由随机数的排列顺序与维度分为不同的种类，如一维噪声，在固定随机数生成范围（如 0-1）的情况下，一个长度为 64 的噪声便是一个长度为 64 的，含有 64 个数的数组，而二维的 64*64 噪声便是一个 64*64 的矩阵，而噪声中随机数的实现可以通过调用 Python 中的 random 库来实现，下图便是一个可视化的噪声图像。

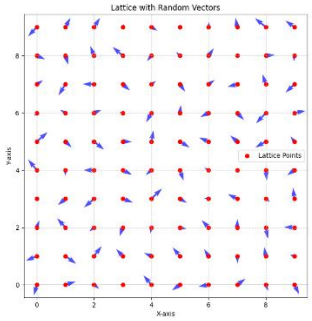




这本质上是一种白噪声,即完全的随机,并不能够很好的利用在图形学之中。

1.2.2. 图像区块化与梯度向量的随机赋值

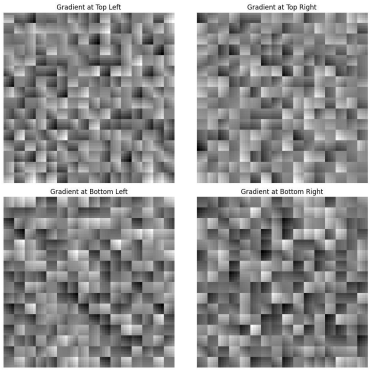
为了将噪声进行有机的连续化,我们需要对图像进行插值平滑,在第一步,我们将图像进行区块化的分割,分成一个个小的区块,比如在下图中,我们选取了 1*1 为一个区块(区块大小应当大于像素大小)接下来,采用基于 seed 生成的可重复的伪随机对区块上的每一个端点进行赋值,赋值的内容是一个长度小于 1,方向任意的矢量(或称梯度向量)。



图表 1,图像晶格化与梯度向量的随机赋值

1.2.3. 梯度向量与偏移向量的点乘计算

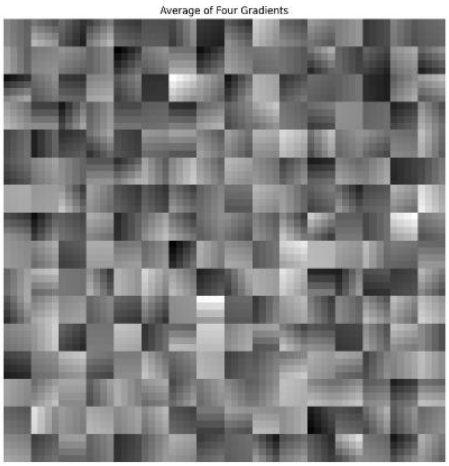
随后,对于每个单元的每个像素,计算梯度向量与偏移向量并把值返回给像素(当前像素位置指向区块顶点的向量)的点积,因为区块有四个角,所以我们得到了四幅图:



这四幅图显然并不具有很强的连续性,且在晶格边缘处出现了很强的不连续,其原因在于目前的运算只在晶格内部进行,且因为具有四个角,运算具有角度的倾向。

1.2.4. 线性插值与非线性插值

一个容易想到的思路是把这四个图像直接相加取平均,希望能够把四个角都考虑进去,于是我们得到以下图像。



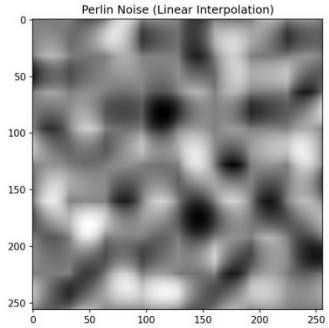
不幸的是,这并不能够有效的对图像进行平滑处理,我们还应当对插值的算法进行更深层次的研究。

1） 双线性插值

我们在计算四个角的元素灰度加和时,取消直接的取平均数的算法,而是依据像素点所在的位置与其与四个点的相对距离进行线性插值,即取 xf = (x - x1) / (x2 - x1)和 yf = (y - y1) / (y2 - y1),那么该点的值即为

$$
\begin{aligned}
f(P) = {} & f(Q11) * (1 - xf) * (1 - yf) + \\
& f(Q21) * xf * (1 - yf) + \\
& f(Q12) * (1 - xf) * yf + \\
& f(Q22) * xf * yf
\end{aligned}
$$

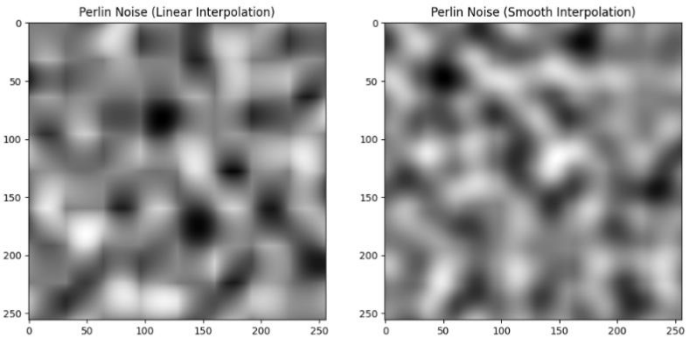
那么,只有双线性插值就够了吗?让我们把这个算法加入我们的运算,看看会得到什么。



令人不满的是,区块边界的不连续性只是减弱,依旧存在,所以接下来我们引入非线性插值。

2)非线性插值

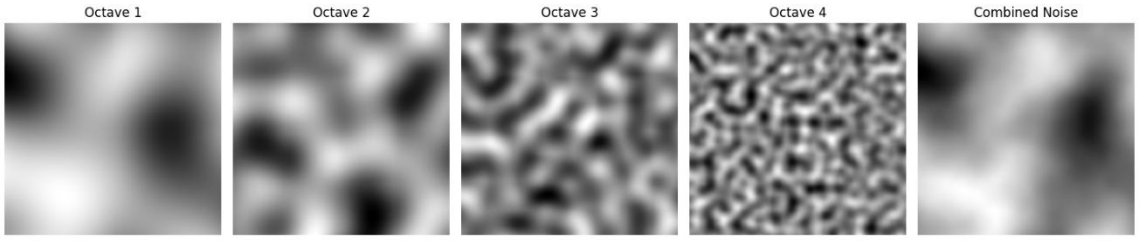
如果我们把图像的灰度比作函数曲线的话,图像之间在区块边界处变化的不连续性就像函数的导数出现了间断一样,为此我们需要一个非线性插值函数,满足在区块边界处的导数为 0,这样就能够满足条件,而 $3x^2 - 2x^3$ 就是一个满足 f(0)=0,f(1)=1,f'(0)=0,f'(1)=0 的非线性插值函数,把这个非线性插值函数加到原有的线性插值权重函数上,即可得到一个全新的,平滑的算法,如图:



图表 2:未采用非线性插值与采用非线性插值的区别

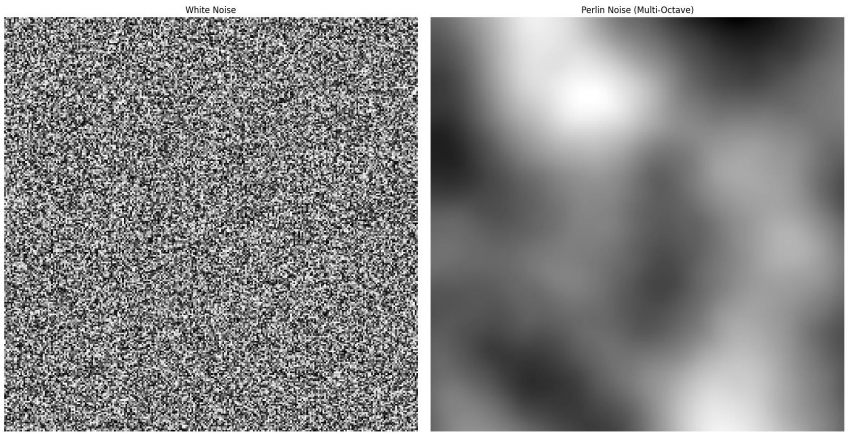
1.2.5. Octave 的叠加

上述过程即描述了生成柏林噪声的一个 Octave(八度),不同的区块分割大小即可有不同的 Octave,如下图所示,我们可以把不同的 Octave 进行叠加,即可得到最终的我们想要的柏林噪声函数灰度图像。



图表 3 不同的 Octave 与最终叠加的效果

为了更加直观的展示柏林噪声对于噪声平滑化的处理,我们在下图给出了白噪声和柏林噪声的对比图：



图表 4:白噪声和柏林噪声的对比图

上述即为对二维柏林噪声的解释与示范,结合对算法的研究与分析以及对文献的参考,我们小组自己建立了一个基于 python 的二维灰度图像生成器,可以生成如图所示的柏林噪声灰度图像,具体的代码我们放在了附录中。

1.2.6. 相关参数的说明

显然,随着区块大小选择的不同与各 Octave 叠加的次数与频率不同,最后所形成的噪声图像也不一样,于是我们规定了 4 个参数,来对噪声图像的生成进行制约。

1) Scale:通过对坐标进行乘以 Scale 的倍率来加以计算,使得对函数的粗糙度与平滑度进行调整,意味着高的 Scale 值能够获得高频率,高细节的噪声,而低的 Scale 值能获得低频率,平滑的噪声。
2) Octaves:通过控制噪声的叠加层数来对噪声进行干预,Octave 小, 噪声层数少，细节简单，看起来比较平滑;Octave 大, 噪声层数多，细节丰富，看起来更复杂。
3) Persistence: 决定每一层 Octave 相对上一层的衰减程度,persistence 越高,每一层的衰减幅度越慢, 高 persistence 值下高频细节的贡献较大，噪声看起来更粗糙、尖锐。低 persistence 值下低频细节的贡献较大，噪声看起来更平滑、柔和。
4) Lacunarity: 决定了每一层噪声的频率相对于上一层的增加比例。

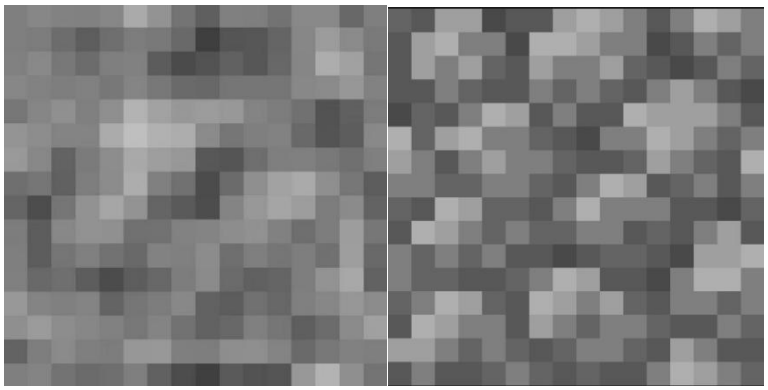
我们通过调控这些相关参数,就能生成不同的柏林噪声图像,接下来,让我们实践起来,生成实际的可以应用的图像纹理。

第二章 利用柏林噪声函数生成游戏材质纹理

2.1. 实例展示

在游戏 Minecraft 中,有很多基于像素的纹理,我们利用我们编程得到的柏林噪声图像,通过对图形颜色,与相关参数的调控,可以得到一系列的图像,下面进行展示:

例 1 对石头表面纹理的随机生成模仿

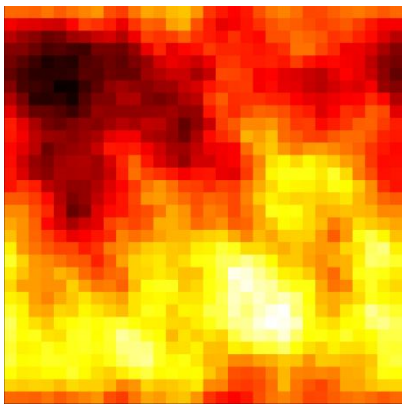


图表 5:通过灰色的颜色配置与较为粗糙的生成方式生成石头表面纹理

(左:使用柏林噪声生成;右:Minecraft 游戏中圆石材质)

例 2 对像素火焰的随机生成模仿

在尝试对参数进行不同调控时,一位组员发现利用 cmap='hot'与较为平滑的渐变函数能够生成类似火焰的图像纹理。



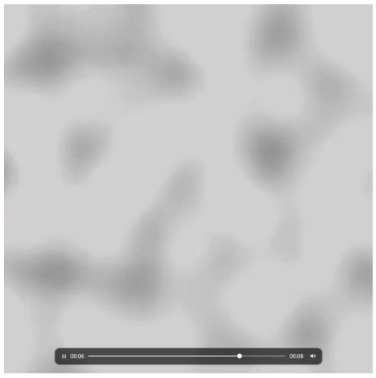
图表 6:使用 hot 颜色生成的像素火焰纹理

这个时候,看起来还是没有那么令人满意,毕竟这终究是生成了一张单纯的图像,如果要是能够生成一个视频就好了!

所以!我们在生成图像的基础上,借助 AI 的帮助,我们了解到了,可以通过把各个图片进行链接来生成视频我们连续生成了 150 张柏林噪声图片,并把这 150 张图片组合起来从而生成了一个视频,由于论文格式限制,我们把这个视频放在了附录二文件中。

例 3 动态云的随机生成模拟

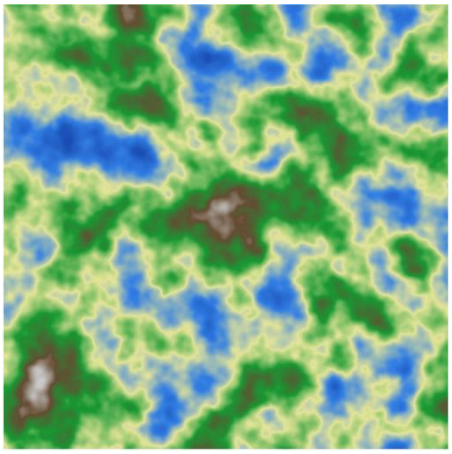
我们发现,我们可以利用柏林噪声算法生成云朵流动的效果,这里我们设计将噪声沿某一方向进行匀速运动,且不同 Octave 的流动速度不一样以造成云朵的层次感,具体代码和生成效果视频详见附录。



图表 7 利用柏林噪声生成动态云的视频的截图

例 4:用柏林噪声生成游戏的地形图

柏林噪声除了能够用来生成游戏的各种纹理,还能够用来生成,比如我们用我们生成的灰度图,我们把灰度低的地方(即图像偏白地方)看作一个地图中较高的位置,而灰度高的地方(偏黑地方)看作一个地图中较为矮的位置,并在特定高度放置水,沙滩,高山,绿地等因素,就可以生成一个地图,我们利用柏林噪声生成随机且连续的地形分布,且用 cmap 指标设定不同高度对应的覆盖颜色,附加较高的图像分辨率,就可以做到游戏中地图的模拟随机生成。

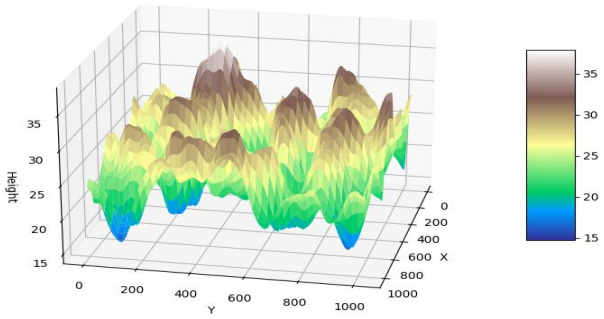


图表 8:利用柏林噪声实现地形图的生成

第三章 用柏林噪声图像生成三维实际地形模型

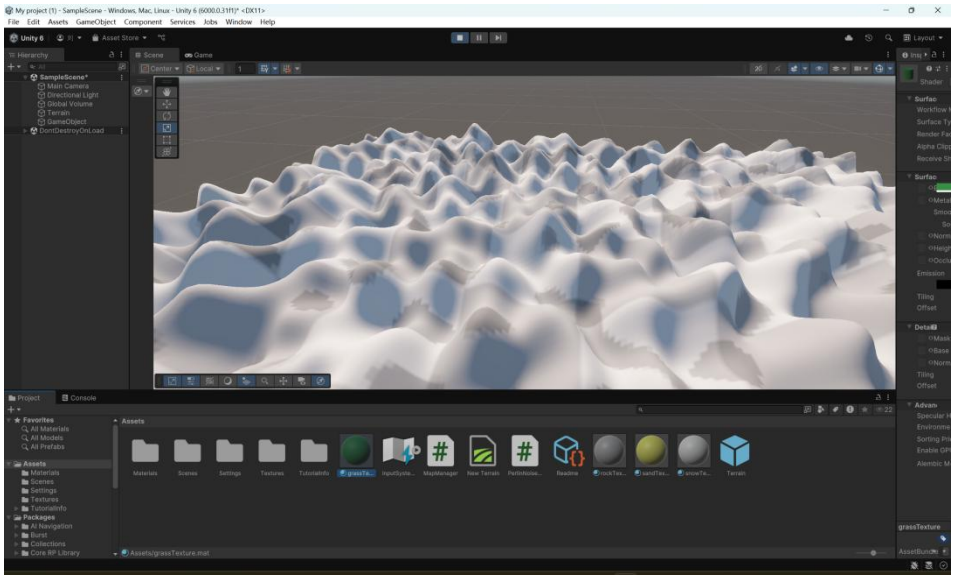
3.1.三维地理模型

一个直接的想法便是把图像用三维空间进行表示,即展现 xyz 三维坐标,把 xy 作为平面坐标,而 z 轴表示高度,把不同高度的地图进行不同颜色的渲染如下图所示：就可以随机的生成山峦，平地，河流等地形。



图表 9 用柏林噪声图像生成三维实际地形模型

除此之外,我们使用了 Unity,通过在 Unity 中用 C#内置柏林噪声算法,我们利用 AI 生成了 C#地形生成脚本,使得脚本能够在 Unity 中实现仿真地形的生成,进而切身实地的能够参与进入游戏的使用中。



上述的代码文件放在了附录中。

参考文献

[1]Perlin, K. (1985). An image synthesizer. ACM Siggraph Computer Graphics, 19(3), 287–296.

[2]Perlin, K. (2002). Improving noise. In Proceedings of the 29th annual conference on Computer graphics and interactive techniques (pp. 681-682).

[3]Wu, Z., & Liu, J. (2022). Perlin noise and its improvements: A literature review. Proceedings of the 2nd International Conference on Software Engineering and Machine Learning .

[4]Conner Addison. (2019-06-03). Random Noise Generation in Python with Numpy & Vectorization.

[5]Unity 柏林噪声生成 2d 随机地图_柏林噪声算法生成 2d 地图. CSDN 博客. https://blog.csdn.net/xyyskl99/article/details/106139761

[6]Bilibili.　分形柏林噪声. Retrieved from https://www.bilibili.com/video/BV1vx4y1i7gn/

[7]知乎 游戏开发技术杂谈 4：柏林噪声 1 王子饼干

小组分工：

第11组分工情况	
2412089刘浩翔	问题的提出,理论的分析,工作的布置,进行算法的书写与分析
2411081杨家宁	案例的制作,算法的书写与分析
2410961赵文铎	案例的制作,算法的书写与分析
2412366李思豆	论文的制作与排版,代码的书写,对结果的总结
2411771范祥荣	PPT演示的制作,代码的书写,对结果的总结