学号	姓名	论文规范性（10）	问题分析与调研（30）	方案创新性（20）	实验结果分析与讨论（40）	结课论文总成绩（100）
21301021	肖斌	8	24	18	38	88

缺乏相关工作的引用

北京交通大学

计算机图形学期末课程论文

基于分型柏林噪声改进的地形生成算法

An Improved Terrain Generation Algorithm Based on Fractal Perlin Noise

学　　院：软件学院

专　　业：软件工程

学生姓名：肖斌

学　　号：21301021

指导教师：吴雨婷

北京交通大学

2024 年 6 月

摘要

程序化生成技术是图形学领域中的一个关键技术，其中就包含自然地形的生成，如何快速、高效地生成符合真实世界的地形一直都是一个值得研究的问题。本次结课实验选取经典的分型柏林噪声地形生成算法，并根据网上资料尝试对其进行探索与改进，通过基于梯度的技巧改善地形生成效果，在保留生成性能的前提下使生成结果更加符合真实世界的规律。

关键词：计算机图形学；地形生成；柏林噪声；

目　　录

1 引言

地形生成算法在图形学领域是一个常见的课题，它不仅是虚拟地形制作、渲染的核心，也是如 Minecraft、Terraria 等众多游戏玩法的核心。然而，性能与生成效果之间的权衡一直是一个问题。

调查地形生成相关工作，找到了分形柏林噪声这一基础算法以及在其上迭代模拟流水侵蚀的改进方法，这两者各有利弊：

（1）分形柏林噪声是极为经典的生成算法，其性能高效、各个块之间的生成不相互依赖，但是其生成结果并不符合自然地形的流水侵蚀，缺乏真实感

（2）在分形柏林噪声的结果上多次迭代模拟流水侵蚀的算法，虽然从物理与视觉上都更加符合自然世界，但是各个块之间的生成过程却变得不再独立，使得性能有极大损失。

本次实验尝试根据 B 站视频【强大的山地生成算法（非柏林噪声和侵蚀算法）】[1]中所讲述的思路来进行探索与尝试，将二者进行结合，来达到性能与生成效果之间的“两全”。

2 方法描述

首先是普通的柏林噪声，将输入坐标(x,y)分割为整数部分(x_{dec},y_{dec})和小数部分(x_{frac},y_{frac})。然后根据$(x_{dec},y_{dec}),(x_{dec}-1,y_{dec}),(x_{dec},y_{dec}-1),(x_{dec},-1\ y_{dec}-1)$生成输入坐标周围四个整数格点的梯度值，之后根据$(x_{frac},y_{frac})$在四个点之间使用$6x^5-15x^4+10x^3$映射后进行插值得到该点的输出，最后将噪声图转换为地形高度图即可生成地形。通过普通的柏林噪声得到的地形已经有山峰和山谷的雏形，但是缺乏细节与变化性。

然后是分形柏林噪声，其原理为依次生成多张频率增加、振幅衰减的噪声图，并叠加，这种方法为地形增添了更多的细节与变化，但是依旧不够真实。

考虑梯度大的地方会有更加严重的水流侵蚀，高处的泥土会被带到低处，可以通过多次迭代模拟这个过程，但是这会破坏柏林噪声本身优秀的不同区块之间无依赖，可并行化的性质。

于是可以按照这个思路转变一下，不通过迭代的方式来模拟流水，而直接根据梯度大小来决定不同位置上的不同噪声图对其的贡献度。

令 $fade(x)=6x^5-15x^4+10x^3$ 则 $fade'(x)=30x^4-60x^3+30x^2$，再令$lerp(t,a,b)=a+t\times(b-a)$，设四个点的梯度为 $g_{00},g_{01},g_{10},g_{11}$，则有最终的输出为：

$$
\begin{aligned}
n &= \mathrm{lerp}(\mathrm{fade}(y),\mathrm{lerp}(\mathrm{fade}(x),p_{00},p_{01}),\mathrm{lerp}(\mathrm{fade}(x),p_{10},p_{11}))\\
&= \mathrm{lerp}(\mathrm{fade}(x),p_{00},p_{01})+\\
&\quad \mathrm{fade}(y)\cdot(\mathrm{lerp}\,[\mathrm{fade}(x),p_{10},p_{11})-\mathrm{lerp}(\mathrm{fade}(x),p_{00},p_{01})]\\
&= p_{00}+\mathrm{fade}(x)\cdot(p_{01}-p_{00})+\\
&\quad \mathrm{fade}(y)\cdot\{[p_{10}+\mathrm{fade}(x)\cdot(p_{11}-p_{10})]-[p_{00}+\mathrm{fade}(x)\cdot(p_{01}-p_{00})]\}\\
&= p_{00}+(p_{01}-p_{00})\cdot\mathrm{fade}(x)+(p_{10}-p_{00})\cdot\mathrm{fade}(y)+\\
&\quad (p_{11}-p_{10}-p_{01}+p_{00})\cdot\mathrm{fade}(x)\cdot\mathrm{fade}(y)
\end{aligned}
$$

于是可以简单地得到梯度的计算公式：

$$
\begin{aligned}
\frac{\partial n}{\partial x} &= [(p_{01}-p_{00})+(p_{11}-p_{10}-p_{01}+p_{00})\cdot\mathrm{fade}(y)]\cdot\mathrm{fade}'(x)\\
\frac{\partial n}{\partial y} &= [(p_{10}-p_{00})+(p_{11}-p_{10}-p_{01}+p_{00})\cdot\mathrm{fade}(x)]\cdot\mathrm{fade}'(y)
\end{aligned}
$$

依此，在叠加噪声图时使用$1/(1+\mathrm{m})$作为当前层的系数，其中m为各层中当前点梯度的累加值的平方。如此即考虑了梯度来使得生成结果具有水流侵蚀的效果，又保证了各区块生成独立，性能损失不大。

3 实验设置

本次实验为了不引入更多的复杂性，选择使用 Python 语言来实现，通过 manim[2]图形动画库进行结果可视化与展示，未来将其实现到我们小组的 BJTU-Game-Engine[3]中。

Python 版本：3.12.4

Manim-community 图形动画库版本：0.18.1

4 实验过程

首先根据柏林噪声[4]的原始实现[5]，重新实现 Python 版本，用其生成一张噪声图，效果如图 4-1 所示：

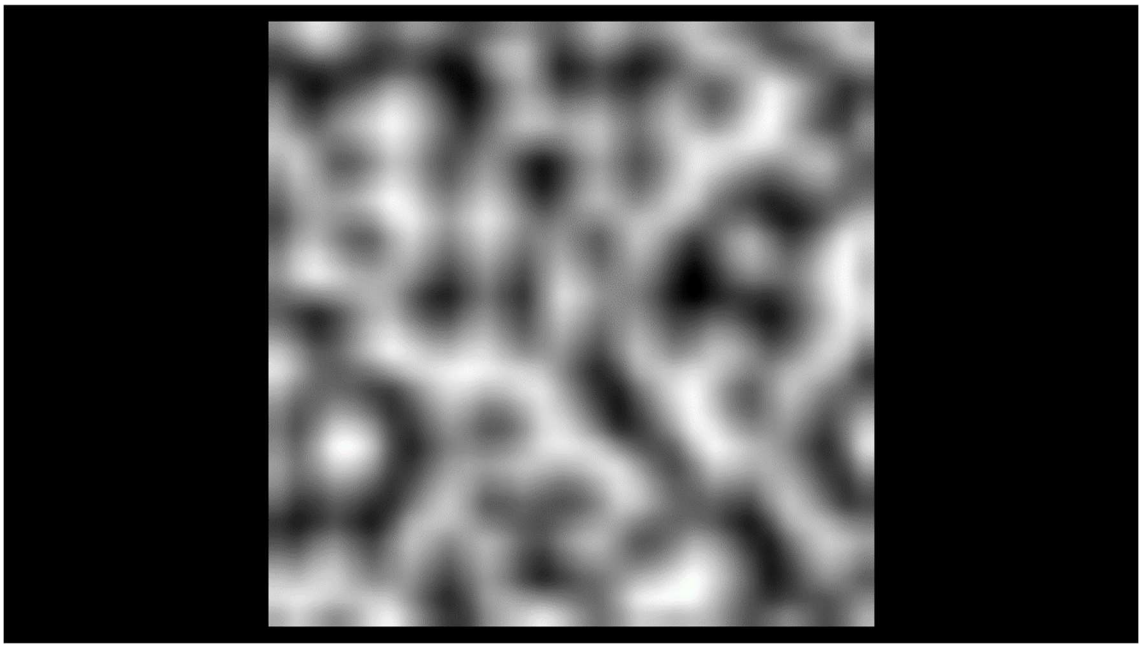

图 4-1 柏林噪声生成的噪声图

然后通过叠加不同尺度的柏林噪声图，为噪声图增加细节，实现分形柏林噪声。使用 8 个倍频生成的效果如图 4-2 所示：

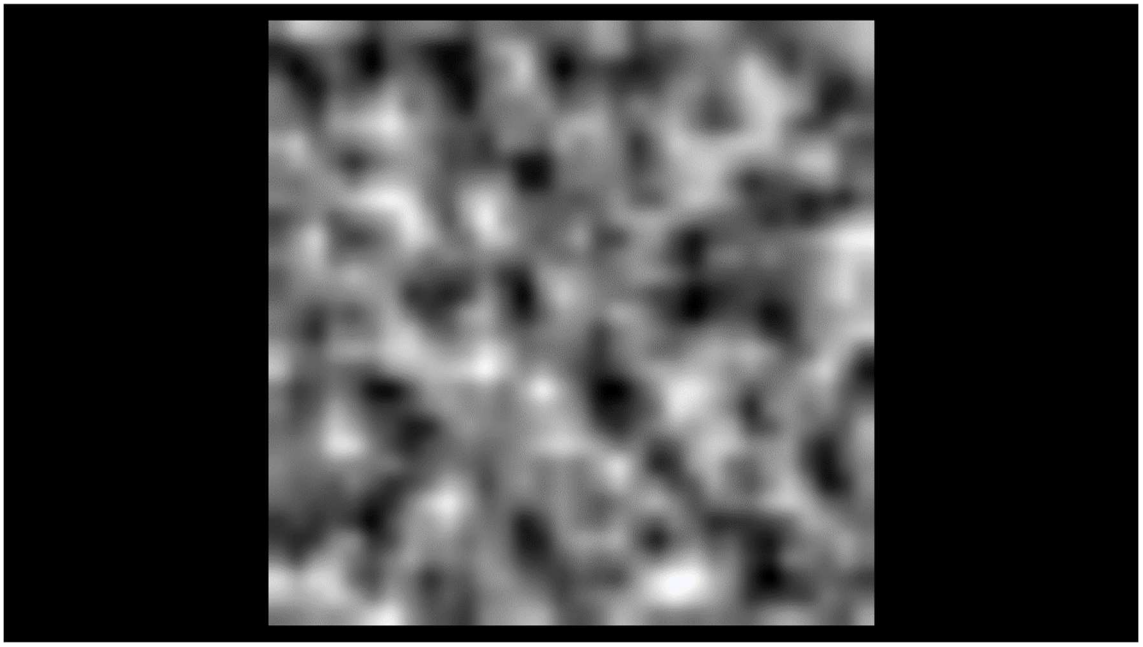

图 4-2 分形柏林噪声生成的噪声图

最后加入梯度系数以调整各层权重、模拟侵蚀效果，效果如图 4-3 所示：

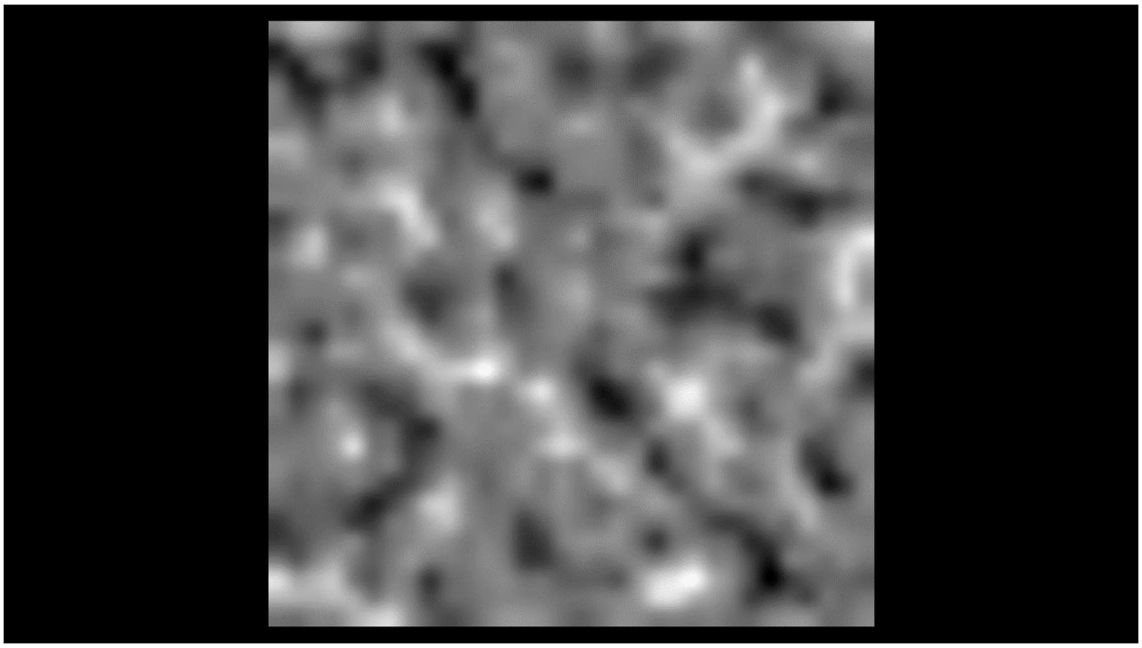

图 4-3 加入梯度系数后生成的噪声图

有了噪声图，现在就可以渲染地形了。

噪声图的输出是$[-1,1]$内的浮点数，将其乘以$depth$即可映射到$[-depth, depth]$区间。再根据高度进行颜色映射，设定多个关键点，关键点之间进行插值：

[(BLUE_E, -1.0), (BLUE_E, -0.9), (BLUE_C, -0.8), (GOLD_E, -0.7), (GRAY_BROWN, -0.1), (GRAY_BROWN, 0.1), (DARK_GRAY, 0.25), (GRAY, 0.6), (WHITE, 0.7), (WHITE, 1.0)]]

所使用的颜色集为 manim 预定义的颜色，如图 4-4 所示

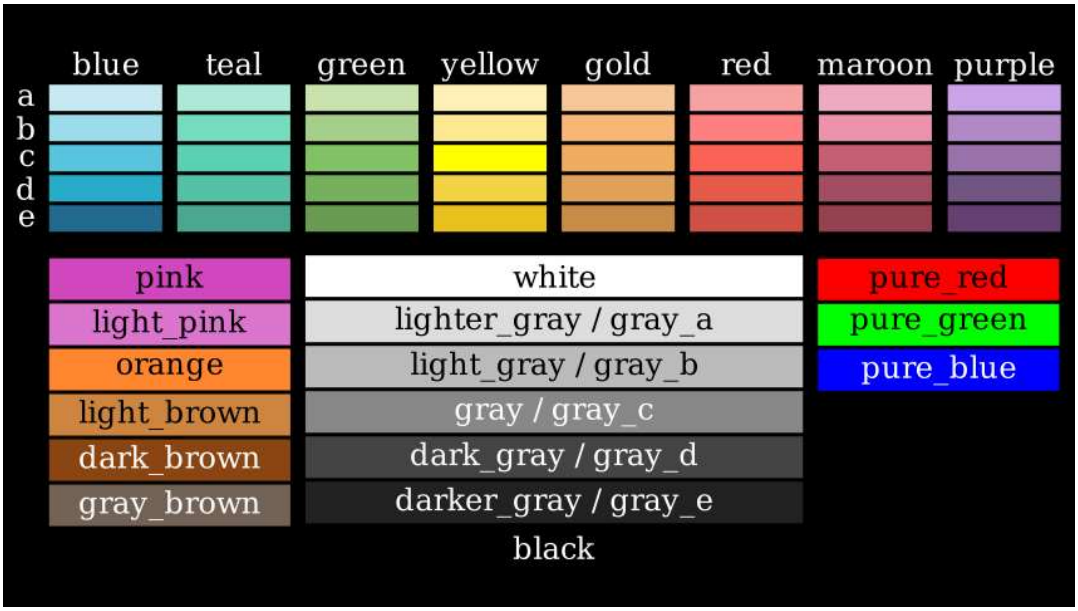

图 4-4 manim 预定义颜色

最终得到通过柏林噪声生成的地形如图 4-5、图 4-6 所示。

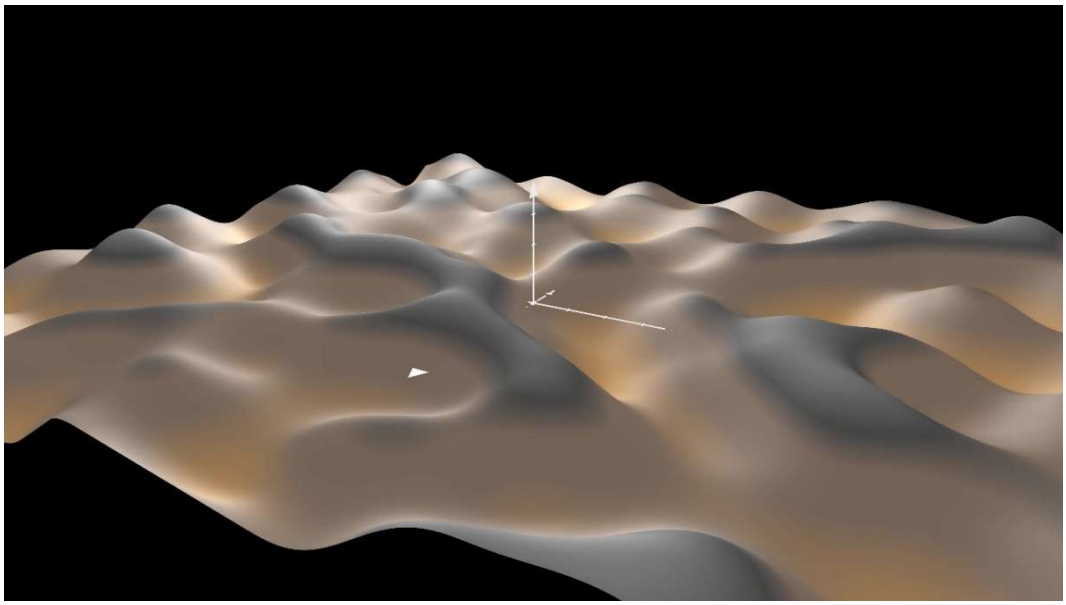

图 4-5 柏林噪声生成的地形

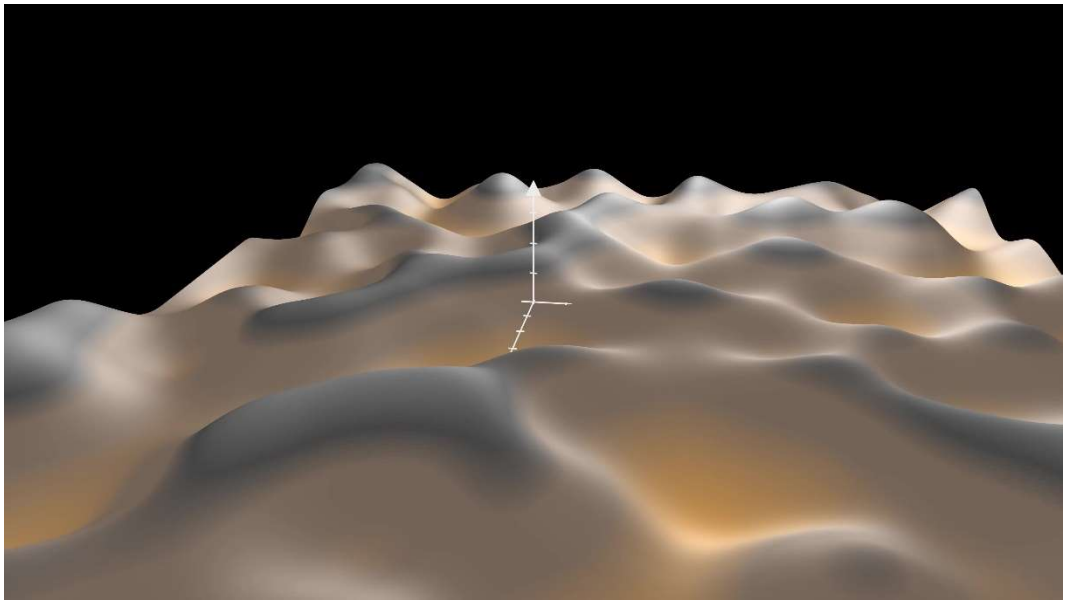

图 4-6 柏林噪声生成的地形

通过分形柏林噪声生成的地形如图 4-7、图 4-8、图 4-9、图 4-10 所示，其中图 4-8 和图 4-10 为引入梯度系数后的柏林噪声生成的结果。

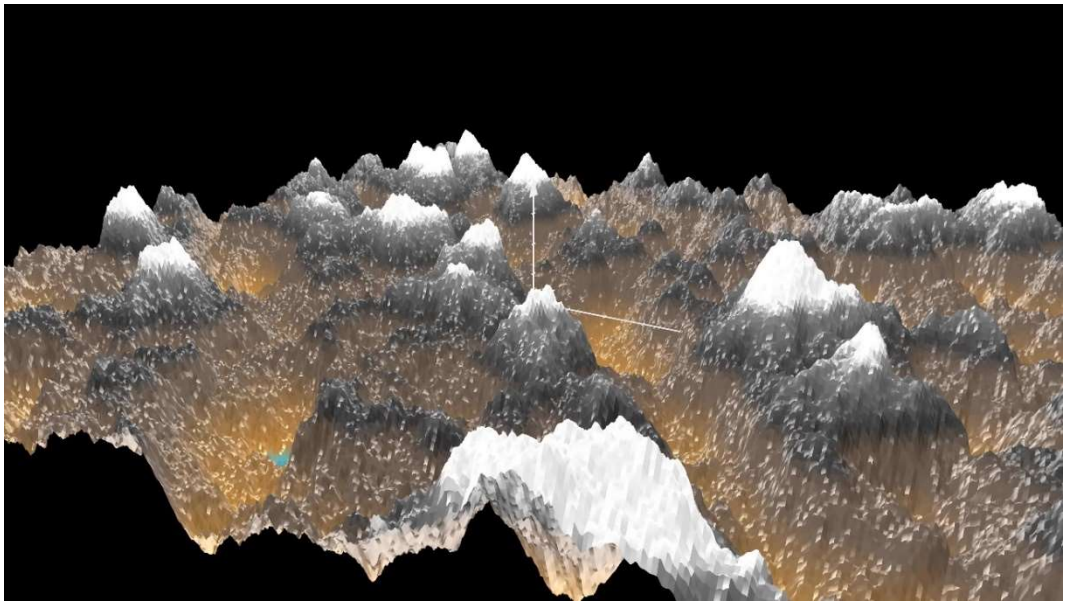

图 4-7 普通分形柏林噪声生成的地形

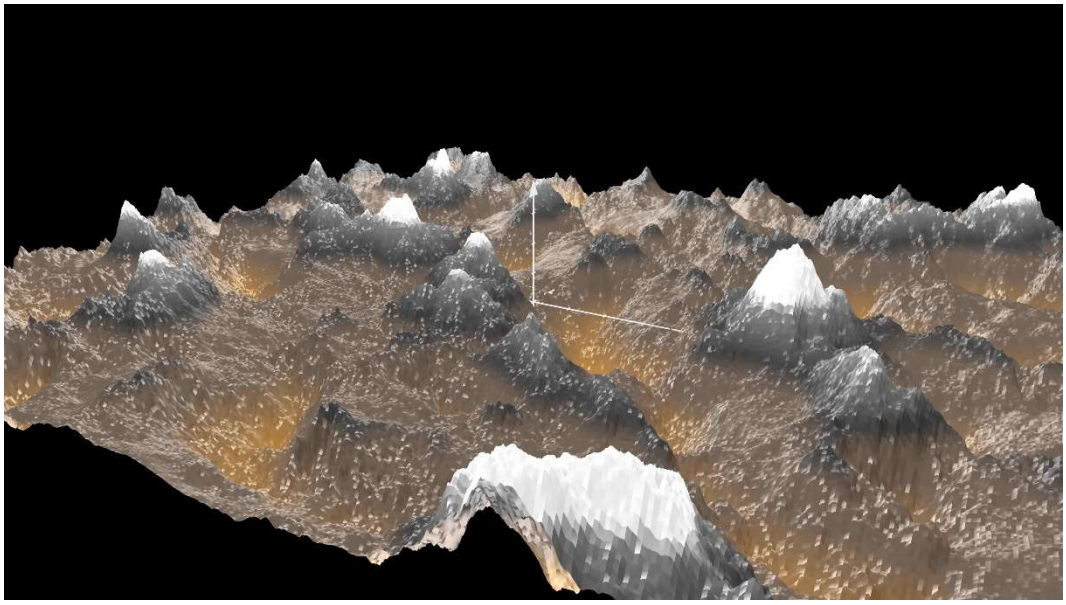

图 4-8 加入梯度系数的柏林噪声生成的地形

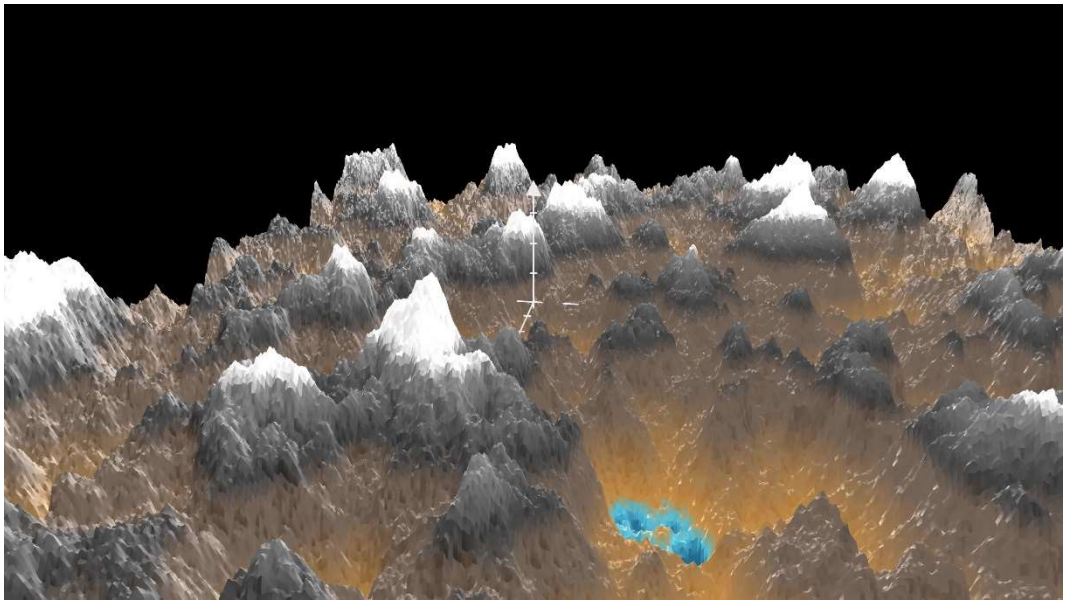

图 4-9 普通分形柏林噪声生成的地形

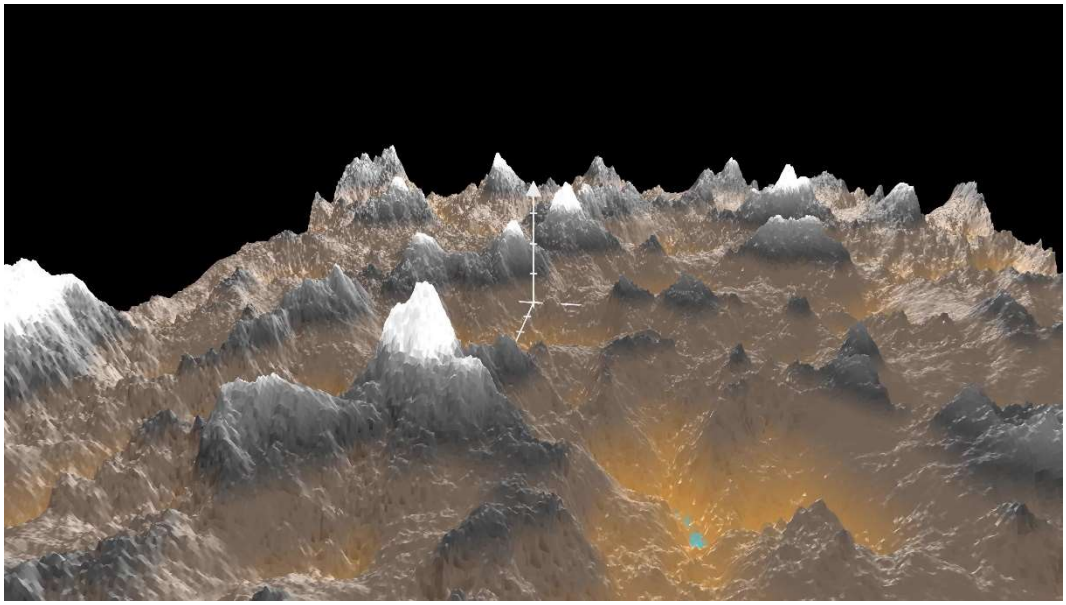

图 4-10 加入梯度系数的柏林噪声生成的地形

5 实验结果与分析

通过对比三种方法的噪声可以明显看出：

（1）分形柏林噪声通过叠加倍频噪声图为地形引入了更多细节与变化性，使得结果更加符合自然。

这种方式的思想类似傅里叶变换，通过叠加不同频率的相同信号以形成复杂信号。区别在于在这里频率与振幅是成反比的，也就是说频率越高的噪声图对地形影响越小，这是因为我们希望在保留粗糙尺度的地形信息的情况下，为地形图添加细节，如此越是低频的信号对地形影响越大（产生地形的轮廓），越是高频的信号对地形影响越小（对地形细节进行雕琢）。

（2）加入梯度系数的分形柏林噪声与普通分形柏林噪声相比，可以看到与普通的分形柏林噪声相比，且山峰与山谷之间的过渡形式更加自然。

控制梯度系数的映射函数$1/(1+m)$在噪声图之间的混合显得尤为重要，m前添加一个系数k，即$1/(1+km)$，如此通过调整k可以调整该函数衰减的速度。当值较大、衰减速度快时，地形整体尖峰的效果会更明显，而值较小、衰减速度慢时，会更加圆润。这个映射函数也可以进行调整更换为其他函数，它相当于规定了梯度值对于流水侵蚀成都的影响关系。

本次实验中实现的地形生成算法并没有利用柏林噪声算法的性质进行并行加速，后续可以考虑通过 compute shader 使用 GPU 进行并行化。

本次实验代码已在 GitHub 开源[6]。

6 总结

通过这次实验中，手动实现了从无到有的地形生成，感受了三种算法的改进过程，更加体会到了图形学的魅力。

通过一个学期以来对图形学的学习与实践，愈发对一个看法深有感触——“图形学关注的不是往往产生结果的过程多么符合物理、符合自然，而是如何通过奇妙的手段达成符合物理、符合自然的结果”。网络上常能见到大家形如“看起来对就是对的”的调侃，但是其实确实不无道理。我们确实有无数物理规律、数学公式，然而很多时候他们并不足以建模我们所处的世界，即便如果真的完完全全按照规律建模了我们所处的世界，虽然我们得到的图形结果必然是真实的，但是这也必然带来极大的算力需求。从光栅化到光线追踪，其思想变得更加符合物理规律（虽然引入了一些并不符合物理规律的假设，但是这是不可避免的），但是其对算力的需求也有极大的增长。本次实验中的地形生成算法也是相同，如果按照无比精准的纯物理建模来进行迭代模拟流水侵蚀，虽然结果必然具有很高的真实度，但是在性能上也会有代价。万物皆如此，有舍才有得。因此我们会选用一种，过程没那么符合实际物理的方法来逼近最终符合物理、符合真实的结果。而这个尝试逼近的过程中的探索、尝试、思考，我认为是图形学最有魅力的地方所在。

最后我回想起学期初刚开课时老师所讲到的“CV 和 CG 两者的过程正好相反”的思想。图形学承担的是从数据到图形，从计算机世界到现实世界；而计算机视觉承担的是从图像到数据，从现实世界到计算机世界。如今计算机视觉领域卷积神经网络随处可见，然而它并不能被解释，这何尝也不是一种“看起来是对的”。世界是一个巨大的黑箱，我们人类穷其一生思考、探索，我们构建出了无数理论体系用于建模这个世界，然而这个世界却从未真正彻底被我们建模。设想如果我们真正地将世界建模为一个个命题与规律，那么 CV 必然可以正确地理解世界，CG 也必然可以正确地模拟世界，通用人工智能也当然会来到现世。然而我们或许永远都无法真正地将世界完整建模，但是我们会永无止境地向其前行，向真理逼近。

此即人类智慧的赞歌。

参考

[1] 强大的山地生成算法(非柏林噪声和侵蚀算法). https://www.bilibili.com/video/BV1CZ421i7qu/.

[2] ManimCommunity/manim: A community-maintained Python framework for creating mathematical animations. (github.com). https://github.com/ManimCommunity/manim/.

[3] YXHXianYu/BJTU-Game-Engine: Boundless Jovial Thriving Utopia Game Engine (github.com).https://github.com/YXHXianYu/BJTU-Game-Engine.

[4] Ken Perlin.Improving Noise[EB/OL]. https://mrl.cs.nyu.edu/~perlin/paper445.pdf.

[5] Improved Noise reference implementation. https://mrl.cs.nyu.edu/~perlin/noise/.

[6] AzurIce/cg-report (github.com). AzurIce/cg-report (github.com)