北京大学

硕士研究生学位论文

题目：**基于 UE5 的真实感水体渲染技术研究与实现**

姓　　名：	邱松涛
学　　号：	2001210398
院　　系：	软件与微电子学院
专　　业：	计算机技术
研究方向：	虚拟与增强现实
导师姓名：	王平　教授
	许捷　讲师

二〇二三　年　六　月

版权声明

摘 要

水体渲染不论在离线渲染还是实时渲染中都是十分重要的一个课题，一直是计算机图形学和游戏开发领域的核心难点之一。随着计算机技术及硬件的不断发展，水体渲染的技术在不断发展着。目前业界主流的水体渲染技术主要分成了几大类，比较常用的如基于波形叠加方法的正弦波和 Gerstner 波，基于统计模型的快速傅里叶变换方法，基于粒子物理的欧拉方法、拉格朗日方法等。其中基于波形的方法适用于大规模的水体，如海洋等的渲染，但是在互动和水流的物理逼真程度上不如基于粒子物理的方法。基于粒子物理的方法则是利用流体力学的原理，模拟粒子之间的黏性和碰撞，水互动方面也是基于物理实现，效果十分逼真。但缺点在于性能耗费较高。

目前计算机硬件的逐渐强大，同时 Unreal Engine 提供了强大粒子物理引擎 Niagara，所以本文将利用 Niagara 粒子引擎强大的并行特性，在其上构建一套基于流体力学模拟的水体系统，其中包括水体粒子的粘度、压力、速度解算，水体材质渲染、场景交互实现等一系列工作。本文也将探索和解决基于流体模拟的真实感水体系统的性能优化方案，以便在实时渲染项目和游戏中推广普及该方法，使得虚拟世界变得更加真实。

本文将介绍项目背景和研究意义，并分析相关技术的发展现状以及市场案例。同时会给出水体模拟的理论依据，以及在 UE 的 Niagara 中的具体实现。同时还会对整个系统进行优化，提升性能和效果表现。

关键词：**实时渲染，流体模拟，水体渲染**

Research and Implementation of Realistic Water Simulation based on Unreal Engine 5

Qiu Songtao(Computer Technology)
Directed by Xu Jie

ABSTRACT

Water rendering is a crucial topic in both offline and real-time rendering and has always been one of the core challenges in the fields of computer graphics and game development. With the continuous development of computer technology and hardware, water rendering technology has been constantly evolving. Currently, the mainstream water rendering technologies can be divided into several categories, such as sine waves and Gerstner waves based on wave superposition methods, fast Fourier transform methods based on statistical models, and Euler and Lagrange methods based on particle physics. Among them, wave-based methods are suitable for rendering large-scale bodies of water such as oceans but may not be as physically accurate as particle-based methods in terms of interactivity and water flow. Particle-based methods, on the other hand, use fluid mechanics principles to simulate the viscosity and collision between particles, and their interactivity is also based on physical implementation, resulting in highly realistic effects. However, the drawback is that they consume a relatively high amount of computational resources.

With the increasing power of computer hardware and the powerful particle physics engine Niagara provided by the Unreal Engine, this paper will leverage Niagara's parallel processing capabilities to build a fluid dynamics-based water system, including viscosity, pressure, velocity calculation of water particles, water material rendering, scene interaction implementation, and other related work. This paper will also explore and propose performance optimization solutions for physic-based water systems, so that this method can be widely applied in real-time rendering projects and games, making virtual worlds more realistic.

This paper will introduce the background and research significance of the project, analyze the current development status and market cases of related technologies. It will also provide the theoretical basis of water simulation and its implementation in UE's Niagara. Finally, the entire system will be optimized to enhance performance and visual quality.

KEY WORDS: real-time rendering, fluid simulation, water rendering.

目 录

第一章 绪论 1
1.1 研究背景与意义 1
1.2 研究现状与文献综述 2
1.2.1 研究现状 2
1.2.2 水体渲染相关案例分析 3
1.2.3 文献综述 5
1.3 本文主要工作 6
1.4 本章小结 6
第二章 流体模拟的理论基础和技术分析 8
2.1 流体模拟理论基础 8
2.1.1 Navier-Stokes 方程 8
2.1.2 基于拉格朗日视角的 SPH 方法 11
2.1.3 基于位置的动力学 PBD 17
2.1.4 基于拉格朗日视角的 PBF 方法 19
2.2 基础技术简介 22
2.2.1 Unreal Engine 22
2.2.2 HLSL 着色器语言 22
2.3 Unreal Engine 5 的 Niagara 粒子系统 22
2.3.1 Niagara 概述 23
2.3.2 Niagara 的 Simulation Stage 27
2.3.3 Niagara 的缓存和空间加速结构 28
2.4 本章小结 31
第三章 水体渲染系统的架构和实现方案 32
3.1 水体渲染系统需求分析 32
3.1.1 水体形态模拟和渲染功能需求分析 32
3.1.2 水体编辑器插件功能需求分析 32
3.2 水体渲染系统设计与实现 33
3.2.1 总体流程 33
3.2.2 水体运动形态模拟 33

3.2.3 水体材质渲染 33
3.2.4 场景交互实现 33
第四章 水体渲染系统的测试和性能优化 34
4.1 测试方案设计 34
4.1.1 算法性能测试 34
4.2 系统优化 34
4.2.1 渲染效率优化 34
4.2.2 性能优化 34
第五章 总结与展望 35
5.1 项目总结 35
5.2 未来展望 35
参考文献 36
附录 A 37
致 谢 38
北京大学学位论文原创性声明和使用授权说明 39

主要符号对照表

x, y, u, v	标量，通常为变量
$\vec{u}, \vec{v}, \vec{w}$	矢量或向量
$\mathbf{x}, \mathbf{u}, \mathbf{v}$	矢量或向量
A, L, D	超参数
∇	梯度
$\nabla \cdot$	散度
$\nabla \times$	旋度
∇^2	拉普拉斯算子

第一章 绪论

1.1 研究背景与意义

电子游戏伴随着信息产业的发展而诞生，至今已有数十年的历史。其发展至今，不仅仅成为了人们生活娱乐的方式，本身对文化的承载也让它成为了国家民族文化输出的载体。在我国，2022 年游戏用户已达到了 6.66 亿人的规模，仅上半年就创造了 1477.89 亿元的销售收入[1]。电子游戏产业的不断发展和用户需求的不断变化也要求游戏工业技术的不断迭代。图形硬件和相关的图形开发技术与游戏产业相辅相成，共同促进，不断带给玩家更为逼真和精致的游戏画面。

现在的真实感渲染技术力求做到照片般的画面，采用了 PBR（基于物理的渲染 ）技术，包括基于物理光照系统、基于物理的材质、基于物理的相机。而 PBR 所能构建的只是一个静态的世界。真实的世界不仅仅包括各种光照和材质效果，同时也包括和世界中各种物体运动的交互，所以基于物理的模拟也是让虚拟世界真正活起来的关键所在。水是生命之源，在真实世界中无处不在，发挥着重要作用，点缀着绚丽多彩的世界。所以在虚拟世界中，对于水体的渲染也是我们在还原真实世界的工作中必不可少的一部分。

水体渲染技术伴随着电子游戏技术的发展一同进步。对于电子游戏来说，一个可以使用的水体或者说流体系统要满足一下几个条件：

1. 算法复杂度低

一个实时交互项目至少要满足 30 帧每秒的帧生成速率，这也是判断项目是否满足实时要求的重要依据。由于场景里还存在着其他需要占用耗时的运算，所以对于水体模拟来说，所能占用的时间则更少。一般认为，在游戏中一个效果或者特性，帧生成时间最好不多于 3ms。所以性能永远是需要考虑的重要因素。

2. 较低的内存占用

和算法复杂度相应的就是算法运行时的内存占用情况。这一点在成熟的游戏框架下，比如 UE 中可能需要考虑的不是很多，因为游戏引擎本身已经为你做好了这方面的工作。但要仍要注意的是在实现项目的过程中不要有额外的消耗。

3. 稳定

游戏必须要在一个稳定的帧率下运行，如果出现较大的帧率波动，会造成体验大打折扣，甚至无法运行的结果。所以水体模拟方法不管在边界条件或者复杂度较高的交互场景中，都必须要在给定的时间步长内稳定运行。

4. 真实合理的视觉效果

这一点毋庸置疑是在实现水体渲染效果时的重要目标，在用算法尽可能逼近现实水体效果的同时，也要顾及渲染结果的合理性。

目前市面上大多数游戏的水体渲染还是通过波形叠加对网格体进行偏移实现水波，然后编写相应的着色器实现水体效果，例如经典的 Gerstner Wave 方法。这一方法简单高效，但是在水体交互以及水流模拟上由于只能采用贴图绘制的方法而显得不够真实。基于流体力学的流体模拟技术则提供了另外一种水体渲染的方法，它通过流体动力学中的 Navier-Stokes 方程[2]，根据流体粒子之间的粘度、压力等物理特性，计算出粒子的速度场从而实现流体的模拟。这种方法在物理上是准确的，同时对水体交互也有很好的表现效果，但是由于有大量的粒子需要参与解算，性能上耗费很高。

得益于当前 GPU 和游戏引擎技术的发展，GPU 的并行性得到了充分的利用，最新的 DirectX12 图形 API 可以实现 CPU 和 GPU 之间完全的并行，极大解放了 GPU 的算力。同时 Unreal Engine（以下简称 UE）提供了 Niagara 粒子系统，可以充分利用 GPU 强大的并行计算能力，实现各种复杂的粒子效果。这些都为本文的项目提供了坚实的技术基础。

UE 发展至今已经迭代到了 5.1 版本，在渲染方面，最新的版本提供了诸如 Lumen、Nanite 等强大的功能特性，使得虚拟世界更加真实。同时它也更新了在 4.26 版本中加入的 Niagara 粒子系统，提供了最新的空间网格加速结构 Neighbor Grid，极大加速了对粒子进行物理模拟的效率。

目前 UE5.1 版本提供了基于 PIC/FLIP 的流体模拟系统，但是由于 FLIP 算法性能耗费较高，导致无法在游戏项目中使用。同时，市面上的游戏水体渲染项目，基本以波形叠加和基于统计学模型的方法实现，在对水体流动的效果表现上不如流体模拟的方法。本项目的主要意义在于探索基于流体动力学的水体模拟和渲染技术在游戏项目中的运用，同时探索一个高性能的解决方案，让其可以较大规模地进行实装。同时本项目还会验证多种流体模拟算法在 Niagara 粒子系统中的性能表现，为以后的研发提供参考。

1.2 研究现状与文献综述

1.2.1 研究现状

水体渲染技术是计算机图形学中的一大分支，同时也是计算机图形学发展至今都在不断探索的研究方向。水体渲染技术主要涉及两个大方向，一是水体状态的模拟，例如波形的形成，水流的实现；另外就是水体材质的渲染，包括水体本身材质和水下

后处理材质。近年来的很多研究主要集中在水体状态模拟以及如何加速渲染效率上。

Gerstner Wave 理论最早于 1804 年被 Gerstner F.J.提出，提供以摆线模拟周期性水波的方法[3]。 伴随着可编程渲染管线的发展与成熟，Laeuchli 于 2002 年实现了根据 Gerstner Wave 计算水面波形的着色器。Gerstner Wave 方法采用多个不同振幅、周期和频率的正弦波叠加的方式产生高度场。然后根据高度场计算出水面法线朝向，这样就可以完成水面的渲染。这种方法十分简单快速，适用于湖泊和简单的海面渲染，直到今天依然被大规模应用到各类游戏项目中。缺点在于它并非物理正确，对于河流等有具体流向的水体，无法表现流体的细节。目前 UE 中自带的水体系统使用的就是 Gerstner Wave 方法。

Fast Fourier Transform Ocean 方法在 SIGGRAPH 2001 上由 Jerry Tessendorf 提出[4]。这种方法基于统计学模型，将三角函数所描述的海浪波形，包括振幅（波高）、频率（海洋运动速度）、相位（海洋的初相）由实数域通过欧拉公式转换到了复数域，从而可以用离散的复数域的点表示海洋波形，然后使用海洋波普和高斯噪声作为初始化的振幅。最后利用逆快速傅里叶变换就可以将波形从频域空间转换到时域空间，从而得到了随时间变化的海洋波形。这种方法因为需要实时进行计算，消耗较高，较早应用在离线渲染中。近年来由于实时渲染技术，特别是 Compute Shader 技术的发展，使得其得以在实时渲染项目中得到使用。它可以生成极为逼真的海浪波形，并具有十分优秀的动态效果。

基于流体动力学的不可压缩流体模拟通常由著名的 Navier-Stokes 方程来描述。求解该方程主要有两种方法，一种是基于网格的欧拉视角，另一种则是基于粒子的拉格朗日视角。对于水体模拟来说，一般使用基于粒子的拉格朗日视角来进行解算，例如光滑粒子动力学（Smoothed Particle Hydrodynamics，SPH）方法。它最早由 Lucy、Gingold 等人提出并应用于天体物理学领域[5]，后来被拓展到流体力学领域解决流体模拟问题[6]。它具有实现简单直观，材料边界划分准确，适用性广等优点。但是其效率不够高，同时存在数值计算不稳定的问题。针对它的缺点和不足，后续又有多种解决方案提出，其中在实时渲染中应用较多的则是基于位置的流体模拟方法（Postion Based Fluid，PBF）[8]方法，它来源于基于位置动力学方法（Position Based Dynamics，PBD）[9]。它通过约束实现粒子物理量的更新，可以有效解决模拟过程中的不稳定问题，同时允许较大的时间步长，进而可以提高实时模拟的效率。

目前游戏项目中使用最为广泛的还是 Gerstner Wave 方法，在一些大规模的水模拟中则会用到 FFT 方法。此外还有一些诸如波动粒子方法等[11]。而基于流体力学的水体模拟技术还没有在游戏中广泛使用。

1.2.2 水体渲染相关案例分析

对于工业化的游戏项目来说，优秀高效的实时水体方案是重中之重，通常他们会根据项目需求自行研发水体方案。但对于中小团队和个人开发者而言，使用市场已公开的解决方案并进行个性化定制更符合需求。

1. Fluid Flux[12]

图 1.1　Fluid Flux 示例关卡

该插件总体采用的是波形叠加的技术形成波形，实现重点在于大面积、实时的浅水区渲染，实时的水体交互，可烘焙成静态模型水面的工具等。它是目前市场上在水体渲染方面的标杆产品，集合了众多工具，方便开发者在任何地方都可以找到应用场景。

2. Waterline Pro[13]

图 1.2　Waterline Pro

该插件是基于快速傅里叶变换实现的海洋模拟。主要功能包括：基于快速傅里叶变换的海洋波形实现；浅水模拟；水下和吃水线实现；浮力模拟；基于物理的水体材质等。基于以上功能，Waterline Pro 提供了一套完整的实时海洋模拟方案，同时可以

完美结合 UE 中的大气光照等功能，实现十分逼真的水体渲染效果。

3. UIWS(Unified Interactive Water System)[14]

图 1.3　Unified Interactive Water System

不同于上面两款水体模拟项目，UIWS 在波形生成方面比较简单一些，它着重于水体交互方面，提供了和场景、人物等碰撞交互功能。同时也提供了基于样条的瀑布和河流的实现。

以上三款都是在基于 UE 的水体渲染方向鼎鼎大名的水体模拟项目，它们均采取了比较成熟的基于高度图的波形生成方法。而基于流体力学模拟的水体渲染在 UE 中还并未开始流行，但是随着计算机硬件的发展，为该方法提供了基础支撑，同时它相比其他方法在物理上的准确性更高，可以得到精美的水体渲染效果。

1.2.3 文献综述

1. 《Fluid Simulation for Computer Graphics 2rd》[2]

本书是流体模拟的入门书籍。总共分为三个部分。第一部分介绍了流体模拟领域的基本方程，也就是 Navier-Stokes 方程的推导过程，以及欧拉视角和拉格朗日视角的求解方法。同时也介绍了数值模拟方法的原理和实现。第二部分则介绍了具体流体类型，如火、水等的模拟方法。第三部分介绍了涡方法、流体和固体的交互等算法。

本书深入浅出，由表及里地讲述了流体模拟的基础框架结构，十分适合入门流体模拟领域使用。同时其中也加入了很多优化方法和解决方案，例如水体表面张力的实

现、海洋模型等。为读者研究流体模拟和理解相关概念提供了十分有利的参考。

2. Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids[7]

该论文是 SPH 的入门文章，详细讲述了 SPH 方法的前世今生。前两章主要介绍了 SPH 的原理以及离散化形式的求解方法。第三章介绍了近邻搜索等空间加速结构。

第四章主要介绍压力求解，以及各种不同的求解方法，例如预测矫正的 SPH，Divergence-Free SPH 等。第五、六、七三章则分别介绍了边界条件、粘度求解、表面张力求解的方法。最后则是对于一些特殊流体和弹性物体的模拟。

该论文介绍的基础 SPH 方法原理简单，方便在 UE 中使用 Niagara 进行部署，搭建了流体模拟从理论到实践的桥梁。

3. Position Based Fluids[8]

该文章详细讲述了 PBF 中对不可压缩项、表面张力和粘度项的解决方法。同时给出了对比实验结果。PBF 方法是基于 PBD 的流体模拟方法，它和 SPH 方法不同之处在于使用约束来实现对压力、粘度、密度等物理量的更新。弥补了 SPH 方法在交互反馈上的弱势。这也是本项目着重实验和使用的方法之一。

4. 《虚拟现实引擎特效制作》[10]

游戏特效中对于粒子效果的模拟是不可或缺的一部分，流体的模拟也可以用粒子实现。所以对于 UE 中粒子系统的学习和研究也是十分有必要的一环。

本书主要介绍了 UE 中 Cascade 和 Niagara 两种特效系统的基础使用方法。提供了包括力场特效、云层、角色粒子等特效案例，方便读者深入研究和理解 Niagara 粒子系统。

1.3 本文主要工作

本文的主要工作有：

1. 对实现基于流体模拟的水体渲染所需要的相关技术进行研究学习并分析游戏项目中的需求。主要包括利用 UE 进行游戏开发以及 Niagara 粒子系统相关技术。
2. 基于 Niagara 实现各种流体模拟算法，进行比较和测试，找到了适合实时交互的流体模拟、材质渲染、场景交互的方法。通过对各种算法的测试和改造，提升渲染和模拟的效率，使得可以在当前硬件水平流畅运行。
3. 利用 UE 的插件机制，部署水体渲染插件，并根据用户需求提供相关可调节参数。同时搭建示例关卡，展示水体模拟的不同使用场景。
4. 在项目中暴露的性能问题和渲染问题进行修复和调优。完成 UE 蓝图和特效系统的协作。

1.4 本章小结

本章主要讨论了水体渲染技术的背景，以及在游戏中的应用现状。不论是写实画风或者风格化的游戏项目，抑或其他实时渲染项目，水体渲染都是不可或缺的一环。基于流体力学的模拟方法在离线渲染中因其优异的视觉效果从而得到了广泛的应用，

但在实时渲染中由于性能原因并未普及。但随着计算机软硬件技术的发展，在游戏中部署基于流体力学的水体渲染的时机越来越成熟，所以目前对于此方面的探索和开发也是十分有必要的。随后本章对现有的水体渲染技术方法，以及目前较为流行的水体渲染方案进行了探讨和论述，最后对主要参考文献进行了介绍。下一章将具体介绍流体模拟的理论基础和实现的技术基础。

第二章 流体模拟的理论基础和技术分析

2.1 流体模拟理论基础

前面提到，水体模拟是流体模拟中的一种，常用的水体模拟方法如波形叠加的高度场，基于统计模型的方法在这里不进行过多讨论。我们主要介绍的是基于 Navier-Stokes 方程的流体模拟方法，并讲述在液体模拟上使用的相关解法。这些解法也是本文在项目中会使用的方法。

2.1.1 Navier-Stokes 方程

基于流体力学的水体模拟方法的主要依据就是大名鼎鼎的不可压缩 Navier-Stokes 方程，其形式如下：

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho}\nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \tag{2.1.1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2.1.2}$$

2.1.1.1 符号标记

首先解释一下 N-S 方程中的符号标记。

$\vec{u}$表示速度矢量，通常为 3 维向量$\vec{u} = (u, v, w)$。

ρ表示流体密度，对于水来说，大约为$1000kg/m^3$。

p表示压力，代表了流体对任意物体的单位面积的压力。

$\vec{g}$表示重力加速度，通常为$9.8m/s^2$。这一力通常表示流体所受外力的合力，也就是说在计算流体受外力的作用时，会将合力算入这一项。

希腊字符ν表示流体的粘度，用来衡量流体的粘滞性。

式(2.1.2)表示了流体的不可压缩性。

2.1.1.2 动量方程

式(2.1.1)本质上是牛顿第二定律的形式，也称为动量方程，是以$\vec{F} = m\vec{a}$的形式来表示的。假如以粒子系统来模拟流体，每个粒子的质量为m，体积为V，速度为$\vec{u}$。由牛顿第二定律可以得知：

$$m\vec{a} = m\frac{D\vec{u}}{Dt} = \vec{F} \tag{2.1.3}$$

其中D表示物质导数，表示对流体质点的物理属性求导数。接下来对流体粒子做一下受力分析。首先质点会受到重力的作用，即$m\vec{g}$。其他流体粒子会对当前的流体粒子有挤压力，我们用压力的负梯度（总是从压力大的方向指向压力小的方向）来表示当前流体粒子的压力不平衡性，即$-\nabla p$。那么流体粒子受到的压力的合力就是对$-\nabla p$在整个流体粒子的体积上的积分，即$-V\nabla p$。

除此之外还有流体的粘滞力，可以理解为其他流体粒子拖慢或加快当前粒子速度的力，进一步可以理解为让当前流体粒子的速度和周围粒子的平均速度的差距最小化。而在数学中这一差距可以用拉普拉斯算子来衡量，即$\nabla\cdot\nabla\vec{u}$，表示当前粒子速度矢量和周围区域平均速度矢量之差。同样的，需要将此项在粒子的体积上进行积分，即$V\mu\nabla\cdot\nabla\vec{u}$，其中μ是一个表示粘度系数的物理量。

综上所述有：

$$m\frac{D\vec{u}}{Dt}=\vec{F}=m\vec{g}-V\nabla p+V\mu\nabla\cdot\nabla\vec{u} \tag{2.1.4}$$

由于我们以粒子视角看待这个问题的时候，随着粒子数量趋于无穷大，每个粒子的体积可能趋于 0，这时体积会为 0。为了解决这个问题，我们引入流体的密度$\rho=m/V$，那么(2.1.4)式两边同时除以V，再同时除以密度ρ就有：

$$\frac{D\vec{u}}{Dt}+\frac{1}{\rho}\nabla p=\vec{g}+\frac{\mu}{\rho}\nabla\cdot\nabla\vec{u} \tag{2.1.5}$$

进一步简化，令$\nu=\mu/\rho$，表示运动粘度，则上式变为：

$$\frac{D\vec{u}}{Dt}+\frac{1}{\rho}\nabla p=\vec{g}+\nu\nabla\cdot\nabla\vec{u} \tag{2.1.6}$$

到这里我们即将得到 N-S 方程的动量形式，但还差一步就是弄清楚物质导数$D\vec{u}/Dt$。为此要引入两种描述方法，拉格朗日描述和欧拉描述。

2.1.1.3 拉格朗日描述和欧拉描述

当我们研究流体或可形变的固体时，通常采用两种方法来描述要模拟的物体：拉格朗日描述（Lagrangian Viewpoint）和欧拉描述（Eulerian Viewpoint）。

拉格朗日描述是十分朴素的，它把待模拟物体看成类似由一个个粒子的形式组成的。对于流体而言，就是由很多个粒子组成了这个流体，每个粒子有相应的位置$\vec{x}$和速度$\vec{u}$。

欧拉视角则采用了另外一种完全不同的方法，它关注的是空间中固定的点，并考察在这个点上流体属性如何随着时间变化。流体流经空间中某个固定位置时可能会导致这个点的物理性质发生变化，比如一个温度较高的粒子流经这个点，后面紧跟着一个温度较低的粒子，那么这个固定位置的流体温度属性就发生了变化。

可以看到，拉格朗日视角和欧拉视角展示了完全不同的两种看待物体的角度，把

他们联系在一起的关键就是物质导数。首先把物体看成粒子，即从拉格朗日视角出发，每个粒子有自身的物理属性，记为q（表示密度、速度、温度等）。那么方程$q(\vec{x},t)$可以表示给定物理量q在时间点t，位置$\vec{x}$处的值。要知道q随时间的变化率，只需要用链式求导法则对时间求导数：

$$\frac{d}{dt}q(\vec{x},t)=\frac{\partial q}{\partial t}+\nabla q\cdot\frac{d\vec{x}}{dt}=\frac{\partial q}{\partial t}+\nabla q\cdot\vec{u}=\frac{Dq}{Dt} \tag{2.1.7}$$

式(2.1.7)就是物质导数，把它带入式(2.1.6)就得到了流体的动量方程(2.1.1)。从推导过程可以看出物质导数针对的是粒子而不是空间中的固定点。在给定的速度场下，研究流体的物理属性在速度场下的变化的问题称为对流。一个最简单的对流方程就是令物质导数为 0，即：

$$\frac{Dq}{Dt}=\frac{\partial q}{\partial t}+\vec{u}\cdot\nabla q=0 \tag{2.1.8}$$

式(2.1.8)表示在拉格朗日视角下，流体粒子的物理量保持不变。这在我们后面使用特定方法求解时会使用到。

2.1.1.4 不可压缩属性

在 N-S 方程中还有一项是不可压缩项。通常情况下流体的体积变化很小，可以近似看成不可压缩的，对于水来说也是如此。我们任取流体的一部分设体积为Ω，那么这部分闭合的曲面为$\partial\Omega$，我们可以用流体速度$\vec{u}$在曲面法线上的分量的积分来衡量流体体积的变化率：

$$\frac{d}{dt}Volume(\Omega)=\iint_{\partial\Omega}\vec{u}\cdot\vec{n} \tag{2.1.9}$$

对于不可压缩流体，体积保持不变，因而上式为 0，由高斯散度定理，把式(2.1.9)转换成体积积分：

$$\iint_{\partial\Omega}\vec{u}\cdot\vec{n}=\iiint_{\Omega}\nabla\cdot\vec{u}=0 \tag{2.1.10}$$

由于上式成立与否和Ω无关，所以有：

$$\nabla\cdot\vec{u}=0 \tag{2.1.11}$$

由此可得 N-S 方程中的不可压缩条件。满足不可压缩条件的流体在任何情况下不会膨胀或坍缩，模拟不可压缩流体的关键也在于要保持流体速度场的无散度状态。

2.1.1.5 连续方程

2.1.1.4 节中我们推导的得出了 N-S 方程的动量形式。由于物质守恒，还可以得到其连续方程形式。

继续考虑区域Ω中质量为m，密度为ρ的一块流体，有：

$$m = \iiint_{\Omega} \rho \tag{2.1.12}$$

则流体的动量为：

$$P = \iiint_{\Omega} \rho \vec{u} \tag{2.1.13}$$

同上，流体的变化量可以用流经表面$\partial\Omega$的流量来表示：

$$\frac{\partial m}{\partial t} = -\iint_{\partial\Omega} \vec{u} \cdot \vec{n} \tag{2.1.14}$$

进一步由高斯散度定理得到最终此区域流体质量关于时间的导数：

$$\frac{\partial m}{\partial t} = -\iiint_{\Omega} \nabla \cdot (\rho \vec{u}) \tag{2.1.15}$$

同样的，上式关于任意的区域Ω成立，即：

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \tag{2.1.16}$$

这就是 N-S 方程的连续方程形式。后面在 SPH 算法的推导过程中我们还会使用这种形式。

综上就是 Navier-Stokes 方程的推导过程，也是本文水体模拟项目的基础。在此我们不会继续深入 N-S 方程的整体求解方法，而是直接进入和项目相关的 SPH、PBF 等算法的求解过程。

2.1.2 基于拉格朗日视角的 SPH 方法

SPH 最初提出于天文领域，后来被广泛应用于流体力学领域，是拉格朗日视角模拟流体的经典算法。它是一种基于光滑粒子的物理模型，将模拟对象分散成一个个粒子，然后在一定范围内的粒子之间建立关系。它本质上是一种核密度估计（Kernel Density Estimation）方法。首先来看下 KDE。

2.1.2.1 KDE

KDE 最初是由 Lucy 等人在天文学中计算气体的物理量提出的[16]。首先引入 Dirac Delta 函数$\delta(x)$：

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases}, s.t. \int_{-\infty}^{+\infty} \delta(x) dx = 1 \tag{2.2.1}$$

假设空间中某位置$\mathbf{r}$的粒子物理量为$A(\mathbf{r})$，那么在整个空间上$A(\mathbf{r})$可以近似表示成：

$$A(\mathbf{r}) \approx \int A(\mathbf{r}') W(\mathbf{r} - \mathbf{r}', h) d\mathbf{r}' \tag{2.2.2}$$

这里的W就是一个光滑核函数，它的支撑集为h，并且满足：

1. $\int_{\Omega} W(\mathbf{r},h)d\mathbf{r}=1$，这是 Dirac Delta 函数的性质。

2. $\lim_{h\leftarrow 0^{+}} W(\mathbf{r},h)=\delta(\mathbf{r})$，意味着它是 Dirac Delta 函数的一个近似。

3. $W(\mathbf{r},h)\geq 0$，这是显然的。

2.1.2.2 SPH 和 N-S 方程

2.1.1 节中推导的 N-S 方程的连续方程形式如下：

$$\frac{D\vec{u}}{Dt}=-\frac{1}{\rho}\nabla p+\mu\nabla^{2}\vec{u}+\vec{g} \tag{2.2.1}$$

$$\frac{\partial\rho}{\partial t}+\nabla\cdot(\rho\vec{u})=0 \tag{2.2.2}$$

记粒子的体积为V_j，每个粒子的物理量为A_j，由 KDE 可知，物理量可以表示为：

$$A_S(\mathbf{r})=\sum_j A_j\,V_j W\left(\mathbf{r}-\mathbf{r}_j,h\right) \tag{2.2.3}$$

由$V=\frac{m}{\rho}$可得：

$$A_S(\mathbf{r})=\sum_j A_j\frac{m_j}{\rho_j}W\left(\mathbf{r}-\mathbf{r}_j,h\right) \tag{2.2.4}$$

这就是 SPH 的标准形式。进一步可得其一阶导数和二阶导数：

$$\nabla A_S(\mathbf{r})=\sum_j A_j\frac{m_j}{\rho_j}\nabla W(\mathbf{r}-\mathbf{r}',h) \tag{2.2.5}$$

$$\nabla^{2}A_S(\mathbf{r})=\sum_j A_j\frac{m_j}{\rho_j}\nabla^{2}W(\mathbf{r}-\mathbf{r}',h) \tag{2.2.6}$$

该方程描述了流体的密度随时间的变换速率，由于模拟的是不可压流体，所以密度守恒。满足无散度的条件。每个粒子的加速度为：

$$\vec{a}=-\frac{1}{\rho}\nabla p+\mu\nabla^{2}\vec{u}+\vec{g} \tag{2.2.7}$$

根据(2.2.9)这个方程和不同的核函数，我们就可以对方程中的力进行逐个求解，最后得到总的加速度进而求解粒子的速度核位置。下面我们就针对其中的各个项来进行求解。

2.1.2.3 密度求解

首先是密度，想根据 Müller 等人的工作[6]，除了压力项和粘滞力项，其余的物理量都可以用W_{poly6}和函数：

$$W_{poly6}(r,h)=\frac{315}{64\pi h^9}\begin{cases}(h^2-r^2)^3, & 0\le r\le h\\ 0, & Otherwise\end{cases} \tag{2.2.8}$$

梯度形式和拉普拉斯形式为：

$$\nabla W_{poly6}(r,h)=-\frac{945}{32\pi h^9}r(h^2-r^2)^2 \tag{2.2.9}$$

$$\nabla^2 W_{poly6}(r,h)=-\frac{945}{32\pi h^9}(h^2-r^2)(3h^2-7r^2) \tag{2.2.10}$$

这个核函数只包含了平方项，不需要开平方，因而比较节省性能。但是如果用它计算压力，由于在式(2.2.7)中压力需要求梯度，而该核函数的梯度在r趋于 0 的时候会出现排斥力消失，引起粒子聚集现象：

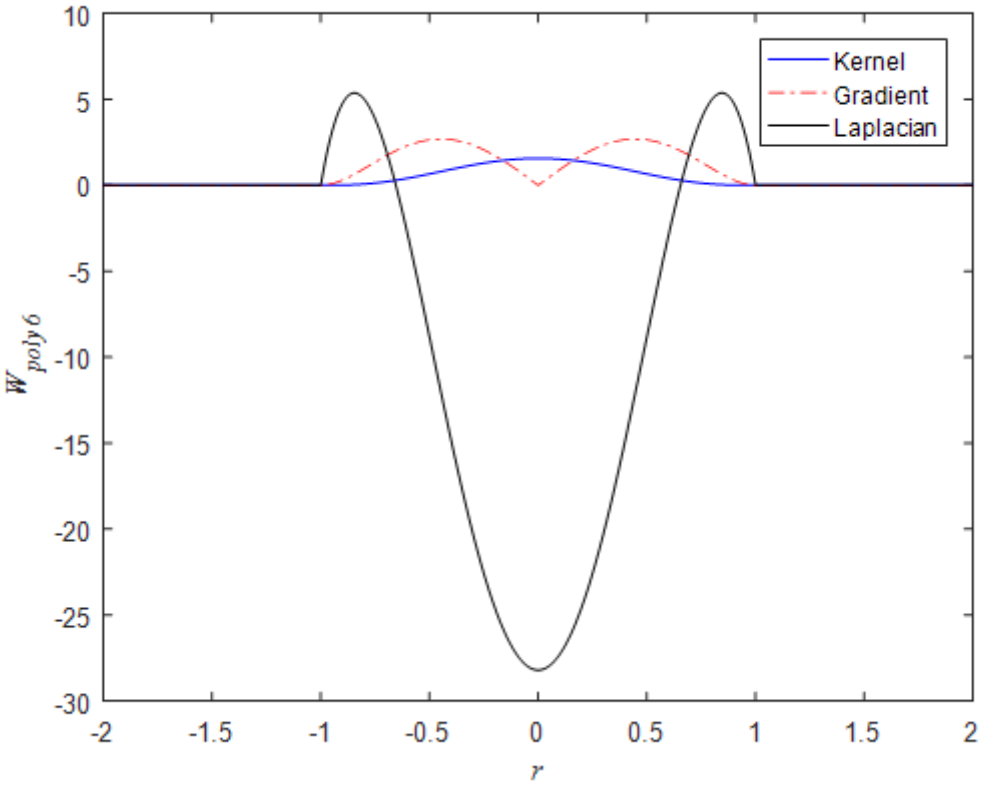

图 2.1 Poly6 核函数图像[17]

根据 SPH 公式(2.2.4)，替换密度，就得到了密度求解公式：

$$\rho_i=\sum_{i\neq j} m_j\, W_{poly6}(\mathbf{r}_i-\mathbf{r}_j,h) \tag{2.2.10}$$

将W_{poly6}带入得：

$$\rho_i=m\frac{315}{64\pi h^9}\sum_{i\neq j}\left(h^2-|\mathbf{r}_i-\mathbf{r}_j|^2\right)^3 \tag{2.2.11}$$

这样就得到了密度的求解形式。

2.1.2.4 压力求解

首先对于压强，需要一个恒定密度ρ_0防止液体密度数值耗散。在计算流体力学（Computational Fluid Dynamics，CFD）中，压力$p = K\left(\left(\frac{\rho}{\rho_0}\right)^{\lambda} - 1\right)$。其原理来自下图[18]：

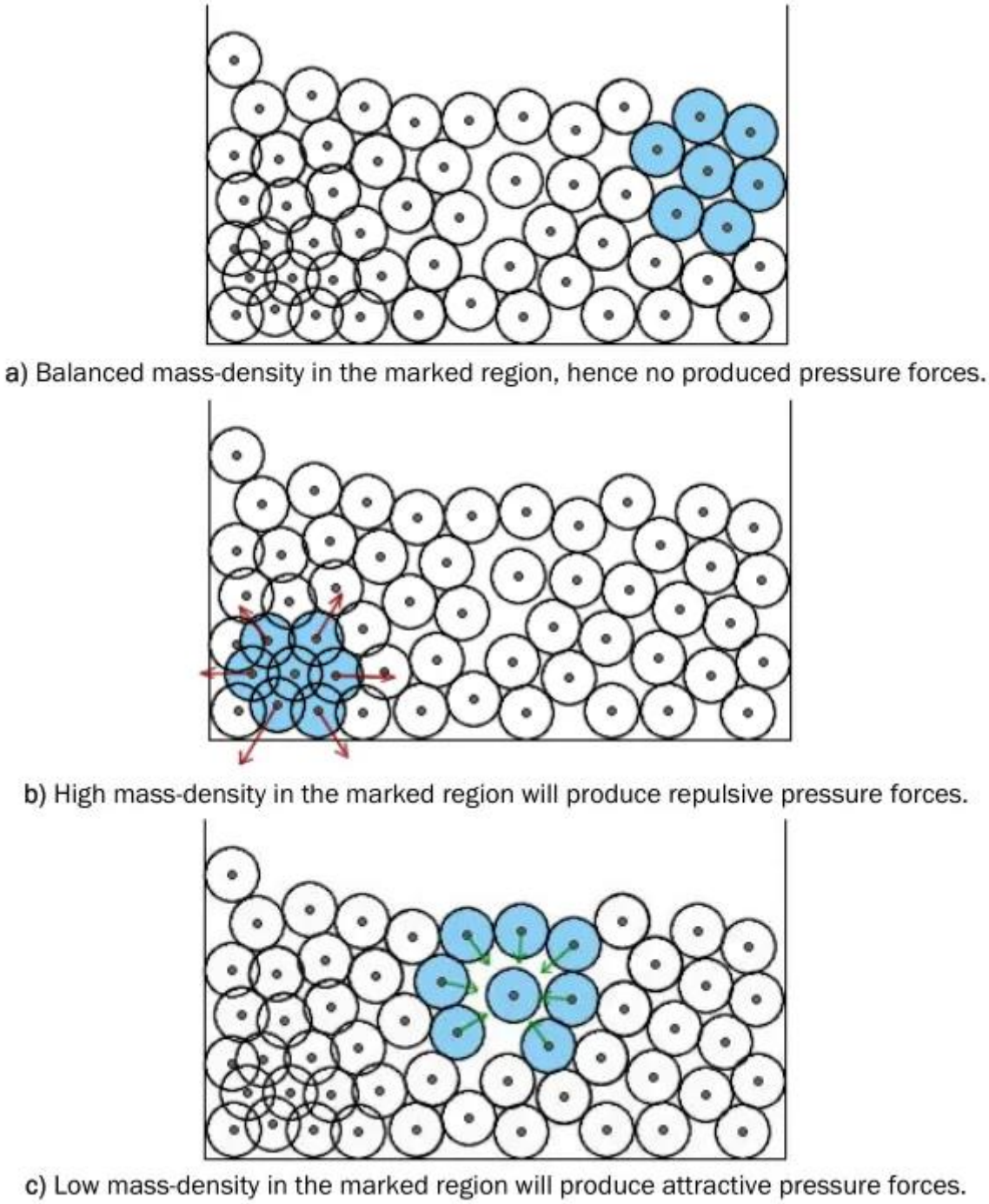

图 2.2 不同密度下的压力

不同的K和λ会产生不同的效果，通常我们取$p = k(\rho - \rho_0)$。

对于压力，因为W_{poly6}函数的梯度在中心变为 0，为了使粒子在接近时具有较大的压力，必须使用一种梯度在 0 时取值较大的函数，这里使用W_{spiky}核函数计算，W_{spiky}形式如下：

$$W_{spiky}(r,h)=\frac{15}{\pi h^6}\begin{cases}(h-r)^3, & 0\le r\le h\\ 0, & Otherwise\end{cases} \tag{2.2.12}$$

同样的梯度形式和拉普拉斯形式：

$$\nabla W_{spiky}(r,h)=-\frac{45}{\pi h^6}\frac{1}{r}(h-r)^2 \tag{2.2.13}$$

$$\nabla^2 W_{spiky}(r,h)=-\frac{90}{\pi h^6}\frac{1}{r}(h-r)(h-2\mathrm{r}) \tag{2.2.14}$$

其函数图像如下：

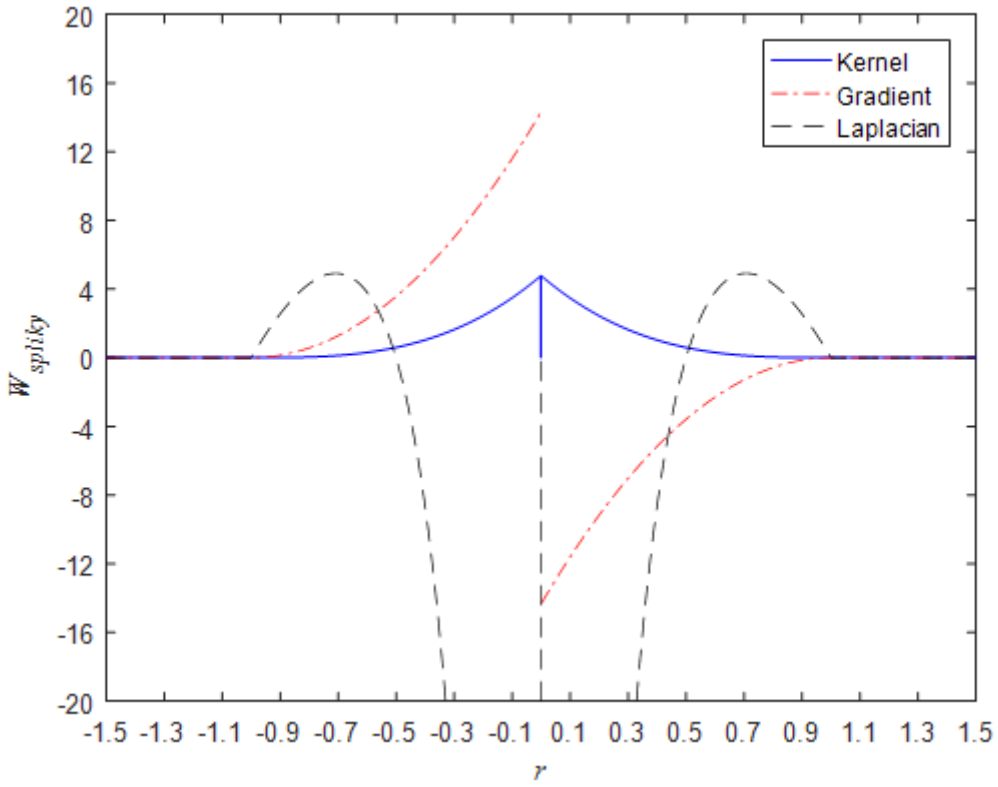

图 2.3 W_{spiky}核函数图像[17]

根据式(2.2.4)，提出压力项，并带入W_{spiky}核函数：

$$-\nabla p(\mathbf{r}_i)=-\sum_{i\neq j} m_j\frac{p_j}{\rho_j}\nabla W_{spiky}(\mathbf{r}_i-\mathbf{r}_j,h) \tag{2.2.15}$$

这是一个非对称形式，违反了牛顿第三定律，即力的作用是相互的，因此我们采用以下的对称形式：

$$-\nabla p(\mathbf{r}_i)=-\sum_{i\neq j} m_j\frac{p_i+p_j}{2\rho_j}\nabla W_{spiky}(\mathbf{r}_i-\mathbf{r}_j,h) \tag{2.2.16}$$

带入核函数，除以密度就可以得到加速度：

$$\mathbf{F}_i^{Pressure} = -\frac{\nabla p(\mathbf{r}_i)}{\rho_i} = -m\frac{45}{\pi h^6}\sum_{i\neq j}\left(\frac{p_i+p_j}{2\rho_i\rho_j}\left(h-|\mathbf{r}_i-\mathbf{r}_j|\right)^2\right) \tag{2.2.17}$$

2.1.2.5 粘滞力求解

同样的W_{poly6}函数的拉普拉斯量在 0 出很快变为负值，如果将其用于粘性力则会造成粒子靠近时的加速效果，与粘性力造成粒子减速不符，所以用$W_{viscosity}$计算粘滞力，其函数图像如下：

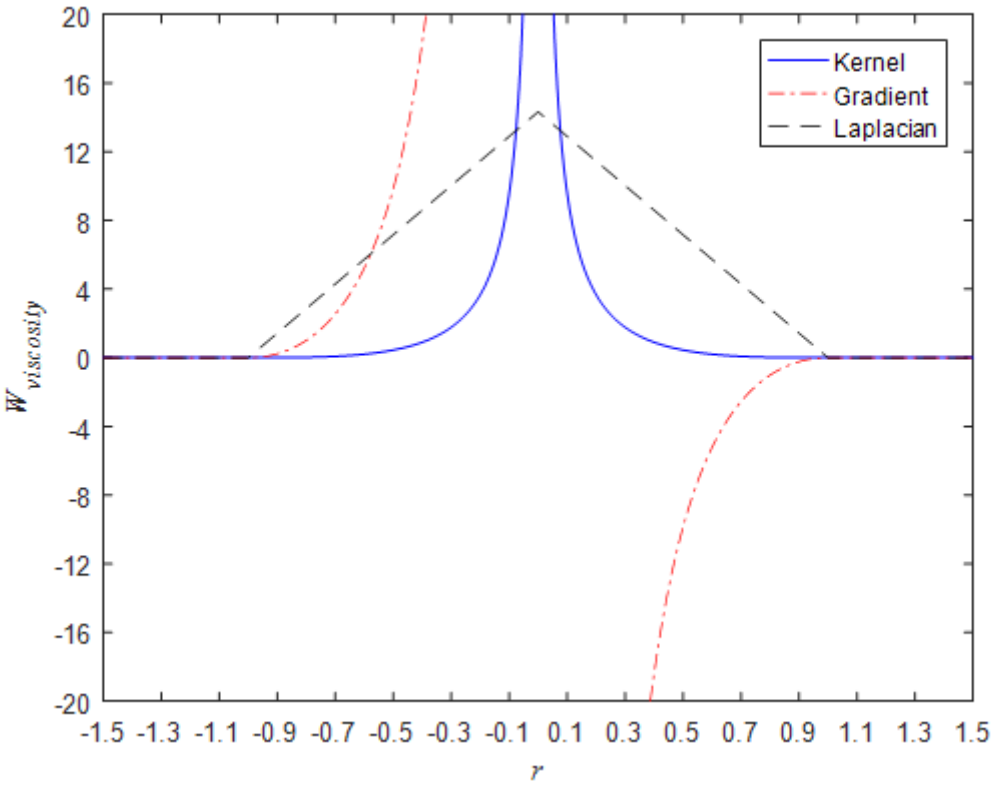

图 2.4 Viscosity 核函数的图像[17]

$$W_{viscosity}(r,h) = \frac{15}{2\pi h^3}\begin{cases} -\frac{r^3}{2h^3}+\frac{r^2}{h^2}+\frac{h}{2r}-1, & 0\le r\le h \\ 0, & Otherwise \end{cases} \tag{2.2.18}$$

梯度和拉普拉斯形式：

$$\nabla W_{viscosity}(r,h) = \frac{15}{2\pi h^3}r\left(-\frac{3r}{2h^3}+\frac{2}{h^2}-\frac{h}{2r^3}\right) \tag{2.2.19}$$

$$\nabla^2 W_{viscosity}(r,h) = \frac{45}{\pi h^6}(h-r) \tag{2.2.20}$$

带入 SPH 公式中的粘度项有：

$$\mu\nabla^2\vec{u}(\mathbf{r}_i) = \mu\sum_{i\neq j} m_j \frac{\overrightarrow{u_j}}{\rho_j}\nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{2.2.21}$$

带入核函数：

$$\mu\nabla^2\vec{u}(\mathbf{r}_i) = \mu\sum_{i\neq j} m_j \frac{\overrightarrow{u_j} - \overrightarrow{u_i}}{\rho_j}\nabla^2 W_{viscosity}(\mathbf{r}_i - \mathbf{r}_j, h) \tag{2.2.22}$$

带入拉普拉斯形式即可得到粘滞力产生的力：

$$\mathbf{F}_i^{Viscosity} = \frac{\mu\nabla^2\vec{u}(\mathbf{r}_i)}{\rho_i} = m\mu\frac{45}{\pi h^6}\sum_{i\neq j}\left(\frac{\overrightarrow{u_j} - \overrightarrow{u_i}}{\rho_i\rho_j}\left(h - |\mathbf{r}_i - \mathbf{r}_j|\right)\right) \tag{2.2.23}$$

2.1.2.6 综合求解

外力在 N-S 方程的 SPH 形式中表示为$\vec{g}$，直接带入即可。最后带入上面的(2.2.17)和(2.2.23)两式到(2.2.7)中即可得到最后的加速度：

$$\mathbf{a}_i = \vec{g} + \frac{45}{\pi h^6}\sum_{i\neq j}\left(\frac{p_i + p_j}{2\rho_i\rho_j}\left(h - |\mathbf{r}_i - \mathbf{r}_j|\right)^2\right) + \mu\frac{45}{\pi h^6}\sum_{i\neq j}\left(\frac{\overrightarrow{u_j} - \overrightarrow{u_i}}{\rho_i\rho_j}\left(h - |\mathbf{r}_i - \mathbf{r}_j|\right)\right) \tag{2.2.24}$$

然后就可以用这个加速度乘以质量计算出力去更新粒子的物理量。

2.1.3 基于位置的动力学 PBD

不同于基于力的方法先求解力再根据力求解，PBD（Position Based Dynamic）使用的是基于约束的方法，可以理解为当位置不满足约束条件时，就会修正自身位置尽可能满足约束。

基于力的模拟在发生碰撞时，首先计算碰撞导致穿透的力，然后根据这个力更新物体的速度和位置。这个过程需要三步：计算力、计算速度、计算位置。同时还需要给物体设定一个刚度系数，来反映碰撞后力的大小，这个系数也比较难设置。

PBD 在处理碰撞时，首先检测是否发生碰撞，然后直接根据约束修正物体的位置，然后再更新速度。这个过程也是三步：检测碰撞、修正位置、更新速度。这三步相比基于力的模拟的步骤而言反应更加迅速，十分适合游戏和其他实时交互类项目。

2.1.3.1 PBD 简介

PBD 算法中，物体由N个顶点和M个约束组成。顶点$i \in [1, \dots, N]$质量为m_i，位置$\mathbf{x}_i$和速度$\mathbf{v}_i$。约束$j \in [1, \dots, M]$满足：

1. 约束的基数为n_j，即第j个约束影响的顶点数为n_j。
2. 约束为实数域的函数：$C_j: \mathbb{R}^{3n_j} \in \mathbb{R}$。

3. 约束的索引为$\{i_1, \dots, i_{n_j}\}, i_k \in [1, \dots, N]$。
4. 每个约束都有刚度系数$k_j \in [0,1]$，可以理解为约束的强度。
5. 约束分为等式约束$C_j\left(x_{i_1}, \dots, x_{i_{n_j}}\right) = 0$和不等式约束$C_j\left(x_{i_1}, \dots, x_{i_{n_j}}\right) \geq 0$。

PBD 算法描述如下：

Position Based Dynamic

1. **forall** vertices i
2. initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
3. **endfor**
4. **loop**
5. **forall** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
6. damp Velocities$(\mathbf{v}_i, \dots, \mathbf{v}_N)$
7. **forall** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
8. **forall** vertices i **do** generateCollsionConstrains$(\mathbf{x}_i \rightarrow \mathbf{p}_i)$
9. **loop** iteration **times**
10. $projectionConstraints(C_1, \dots, C_{M+M_{coll}}, \mathbf{p}_1, \dots, \mathbf{p}_N)$
11. **endloop**
12. **forall** vertices i
13. $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
14. $\mathbf{x}_i \leftarrow \mathbf{p}_i$
15. **endfor**
16. $velocityUpdate(\mathbf{v}_1, \dots, \mathbf{v}_N)$
17. **endloop**

算法中所有外力记为$\mathbf{f}_{ext}(\mathbf{x_i})$，内部约束和碰撞约束记为$M + M_{coll}$，顶点预测位置为$\mathbf{p}_i$。$\Delta t$为时间步长。

首先对参数进行初始化，包括粒子的速度、位置和质量倒数，之所以采用质量倒数是为了避免过多的除法操作以及处理静态物体。先用外力更新一次粒子的速度。然后添加阻尼，这里其实就是对速度直接进行衰减操作。计算完成后更新粒子位置$\mathbf{p}_i$。然后生成碰撞约束，比如水和其他物体的碰撞，水自身的不可压缩约束等。有了约束后对约束进行迭代求解，也就是投影，得到矫正后的粒子位置，再通它更新粒子的速度和位置信息。

2.1.3.2 约束投影

算法中提到了我们要对约束进行投影迭代求解，那么如何投影呢？对于高维空间函数来说，一般使用其梯度方向。

设约束C的基为n，刚度系数为k，用$\mathbf{p} = [\mathbf{p}_1^T, \dots, \mathbf{p}_N^T]$表示受约束影响的点组成的

矩阵。则其等式约束可以表示为：

$$C(\mathbf{p}) = 0 \tag{2.3.1}$$

在每个时间步中，我们要找到一个位置变化量$\Delta\mathbf{p}$使得(2.3.1)仍成立，即：

$$C(\mathbf{p} + \Delta\mathbf{p}) = 0 \tag{2.3.2}$$

把(2.3.2)进行泰勒展开：

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \cdot \Delta\mathbf{p} = 0 \tag{2.3.3}$$

根据式(2.3.3)可以看到，PBD 把$\Delta\mathbf{p}$的方向限制在了约束的梯度方向，而梯度方向正好是待模拟物体表面的法线方向，这样保证了系统的动量守恒。因此，令：

$$\Delta\mathbf{p} = \lambda \nabla_{\mathbf{p}} C(\mathbf{p}) \tag{2.3.4}$$

其中λ是拉格朗日乘子，用以保证动量守恒。由公式(2.3.3)和(2.3.4)可得：

$$\Delta\mathbf{p} = -\frac{C(\mathbf{p})}{\left|\nabla_{\mathbf{p}} C(\mathbf{p})\right|^2} \nabla_{\mathbf{p}} C(\mathbf{p}) \tag{2.3.5}$$

对于具体的粒子i，就得到了约束投影后的位移向量：

$$\Delta\mathbf{p}_i = -s \nabla_{\mathbf{p}_i} C(\mathbf{p}_1, \dots, \mathbf{p}_n) \quad \mathbf{s.t.} \quad s = \frac{C(\mathbf{p}_1, \dots, \mathbf{p}_n)}{\sum_j \left|\nabla_{\mathbf{p}_j} C(\mathbf{p}_1, \dots, \mathbf{p}_n)\right|^2} \tag{2.3.6}$$

2.1.4 基于拉格朗日视角的 PBF 方法

上节主要介绍了 PBD 算法的基本原理，下面让我们进入 PBF（Position Based Fluid）的求解过程。

2.1.4.1 不可压缩约束

在 SPH 一节中我们提到了密度的 SPH 形式为：

$$\rho_i = \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{2.4.1}$$

由于我们模拟的是不可压流体，所以要让流体的动态密度尽可能接近静态密度，所以要对密度施加一个常量约束，在 PBF 中定义这个约束为：

$$C_i(\mathbf{p}_1, \dots, \mathbf{p}_n) = \frac{\rho_i}{\rho_o} - 1 \tag{2.4.2}$$

将式(2.4.1)带入(2.4.2)得：

$$C_i(\mathbf{p}_1, \dots, \mathbf{p}_n) = \frac{\sum_j W(\mathbf{p}_i - \mathbf{p}_j, h)}{\rho_o} - 1 \tag{2.4.3}$$

PBF 中对于密度的计算也用到了光滑核函数，这里计算密度使用的是W_{poly6}，而梯度计算则使用了W_{spiky}的梯度。同时这里我们也统一粒子质量相同，所以省去了。

梯度形式如下：

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_o} \sum_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{2.4.4}$$

这里分成两种情况，当等式为自身时和其他粒子时：

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_o} \begin{cases} \sum_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h), & k = i \\ \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h), & k = j \end{cases} \tag{2.4.5}$$

根据之前在 PBD 中的论述，求解位移向量关键在于求解拉格朗日乘子λ，将上式带入可得：

$$\lambda_i = \frac{C_i(\mathbf{p}_1, \dots, \mathbf{p}_n)}{\sum_k \left| \nabla_{\mathbf{p}_k} C_i \right|^2} \tag{2.4.6}$$

这样就可以求解出关于不可压缩约束的位移向量。

2.1.4.2 混合约束

对于不能违反的约束，称之为硬约。自然我们希望所有的约束都是硬约束，但是由于数值误差等因素，我们需要约束不那么严格。在 PBF 中当邻居粒子在光滑核的边界上，即$|p_i - p_j| = h$时，这是W_{spiky}的梯度为 0，由式(2.4.5)可知：

$$\nabla_{\mathbf{p}_j} C_i = -\frac{1}{\rho_o} \nabla_{\mathbf{p}_j} W(\mathbf{r}, h) = 0 \tag{2.4.7}$$

$$\nabla_{\mathbf{p}_i} C_i = \frac{1}{\rho_o} \nabla_{\mathbf{p}_i} W(\mathbf{r}, h) = 0 \tag{2.4.8}$$

带入到λ的求解中会得到分母$\sum_k \left| \nabla_{\mathbf{p}_k} C_i \right|^2 = 0$的情况，导致除零错误。所以 PBF 在这里加入了一个松弛参数 ϵ，由用户指定以避免除零错误。

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla C^T \nabla C \lambda + \epsilon\lambda = 0 \tag{2.4.9}$$

这时λ_i为：

$$\lambda_i = \frac{C_i(\mathbf{p}_1, \dots, \mathbf{p}_n)}{\sum_k \left| \nabla_{\mathbf{p}_k} C_i \right|^2 + \epsilon} \tag{2.4.10}$$

带入(2.3.4)就可以得到了位移向量：

$$\Delta\mathbf{p}_i = \lambda_i \nabla_{\mathbf{p}_i} C_i + \sum_j \lambda_j \nabla_{\mathbf{p}_j} C_i$$

$$= \frac{1}{\rho_o} \sum_j \lambda_i \nabla_{\mathbf{p}_i} W(\mathbf{r}, h) + \left(-\frac{1}{\rho_o} \sum_j \lambda_j \nabla_{\mathbf{p}_j} W(\mathbf{r}, h) \right)$$

$$= \frac{1}{\rho_o}\sum_j \lambda_i \nabla_{\mathbf{p}_i} W(\mathbf{r}, h) + \frac{1}{\rho_o}\sum_j \lambda_j \nabla_{\mathbf{p}_i} W(\mathbf{r}, h)$$

$$= \frac{1}{\rho_o}\sum_j (\lambda_i + \lambda_j) \nabla_{\mathbf{p}_i} W(\mathbf{r}, h) \tag{2.4.11}$$

2.1.4.3 拉伸不稳定性

在 SPH 模拟流体时，通常需要很多个粒子插值才能使动态密度趋于静态密度。当和半径内粒子数不足时会导致流体的动态密度小于静态密度，从而导致压强变为负值，进一步导致粒子之间的压力变成了吸引力，出现粒子团聚的现象。所以在 PBF 中引入了人工的排斥力，当粒子距离过近时将其分开，在式(2.4.11)的基础上加入排斥项s_{corr}：

$$\Delta \mathbf{p}_i = \frac{1}{\rho_o}\sum_j (\lambda_i + \lambda_j + s_{corr}) \nabla_{\mathbf{p}_i} W(\mathbf{r}, h) \tag{2.4.12}$$

其中：

$$s_{corr} = -k\left(\frac{W(\mathbf{r}, h)}{W(\Delta q, h)}\right)^n \tag{2.4.13}$$

其中Δq表示粒子i附近的固定距离点，通常为$|\Delta q| = 0.1h, \ldots, 0.3h$。$k$可以看作表面张力参数，通常取$k = 0.1, n = 4$。

2.1.4.4 涡控制和人工粘性

PBF 在最后更新速度时会引入额外的阻尼，导致整个系统能量耗散，使得一些涡流消失，PBF 通过涡控制法向系统中重新注入能量：

$$f_i^{vorticity} = \epsilon(N \times \omega_i) \tag{2.4.14}$$

其中$N = \frac{\eta}{|\eta|}, \eta = \nabla|\omega_i|$。$\omega_i$为粒子旋度：

$$\omega_i = \nabla \times \mathbf{v} = \sum_j (\mathbf{v}_j - \mathbf{v}_i) \times \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{2.4.15}$$

这样，$f_i^{vorticity}$作为外力在粒子的旋度方向加速粒子的运动，让本来消失的涡流保持存在。

最后，PBF 还加入了人工粘性用来保持数值稳定和消除非物理振荡，这一步通常在最后更新速度的时候添加：

$$\mathbf{v}_i^{new} = \mathbf{v}_i + c\sum_j v_{ij} \cdot W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{2.4.16}$$

这里通常取$c = 0.01$。

综上所述，我们完成了 PBF 算法的整个推导过程。具体的算法步骤和 PBD 类似。从上面的推导可以看出，PBF 算法相比于 SPH 算法具有更强的鲁棒性，不易出现粒子聚集等状况，在对流体进行模拟的时候也会通过人工注入的方式防止能量耗散。所以本文也会将重点放在 PBF 算法的实现上。

2.2 基础技术简介

2.2.1 Unreal Engine

Unreal Engine（简称 UE）是由 Epic Games 开发的一款集图形渲染、游戏开发、动画制作等于一身的商业引擎。自 1995 年至今，UE 经历了 20 余年的迭代和沉淀，从众多商业化引擎中脱颖而出，广泛应用于游戏、室内设计、影视、仿真等行业。

UE 发展至今，包含了所有现代引擎的绝大部分功能，主要有：资产管理、场景管理、渲染引擎、物理动画引擎、粒子引擎、调试工具等。此外，为了方便制作人快速搭建游戏原型，UE 中还提供了基于节点的编程系统蓝图，同时为了方便美术家使用，提供了材质节点编辑器，让美术家在无需写代码的情况下就可以随心所欲实现想要的效果。

自 2008 年发布以来的 UE4 版本不停迭代，推出了基于物理的渲染管线、Niagara GPU 粒子、更加真实的物理模拟系统等。尤其是其中的 Niagara 粒子系统，充分利用了 GPU 硬件强大的性能以实现各种惊艳的视觉效果。2021 年 UE5.0 版本发布，这次带来的则是举世瞩目的 Nanite 海量三角形渲染技术和全新的全局光照系统 Lumen，同时也带来了其他系统诸如 Chaos 物理破坏系统和 Niagara 系统的更新。

最新的 UE5.1 版本默认支持了 Niagara GPU 粒子和 Simulation Stage 技术，也是本次项目使用的版本。

2.2.2 HLSL 着色器语言

HLSL（High Level Shader Language）是微软推出用于 Direct 系列图形 API 的着色器语言。通过 HLSL，图形程序员可以像编写 C++代码一样为 GPU 编写通用着色器程序。在我们的水体渲染项目中多处用到了 Niagara 中的 Custom HLSL 节点，这也是 Niagara 提供给我们方便直接通过 HLSL 编写和控制粒子逻辑的方法。

2.3 Unreal Engine 5 的 Niagara 粒子系统

2.3.1 Niagara 概述

Niagara 粒子系统是 UE 最新最强大的特效系统，它提供了一套完整灵活的接口，封装了 GPU 并行编程的底层逻辑和数据结构，方便美术人员在不必知道底层逻辑的情况下可以编写各种精美的视觉特效。同时得益于 GPU 的强大并行性，它还可以模拟海量的粒子运动，对其进行物理解算，可以模拟诸如布料、群集运动、流体、弹性体等真实世界中的各类物体。

Niagara 的核心组件包括：系统、发射器、模块、参数[15]。

2.3.1.1 系统

一个 Niagara 系统是一个容器，里面可以放入各种要添加到容器中的效果，也就是发射器。同时可以在系统层级对系统内所有的发射器进行一种全局的控制，主要包括参数传递和生命周期管理等。

图 2. Niagara 系统

2.3.1.2 发射器

发射器是 Niagara 的主体单元，用来在 Niagara 中生成粒子，并控制粒子的生成、粒子在生命周期中的外观及行为。它采用类似瀑布流的方式组成堆栈，每个堆栈中可能有几个组，组中可以放入各个任务的实现模块。

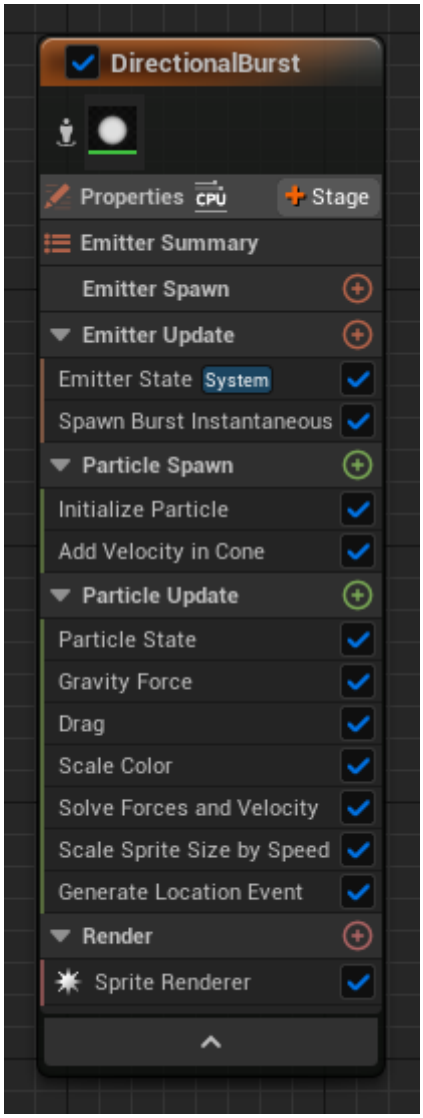

图 2. Unreal 引擎官方 Directional Burst 发射器示例

其中比较重要的有一下几个堆栈：

1. 发射器生成（Emitter Spawn）

这个单元定义了在 CPU 上首次创建发射器运行的功能。一般用于一些值的初始化。

2. 发射器更新（Emitter Update）

这个单元定义了 CPU 上每一帧发射器模块的行为。通常用来定义粒子的产生方式。

3. 粒子生成（Particle Spawn）

当每个粒子生成时会调用这个单元对自身的属性进行初始化。例如粒子的颜色，生成位置，大小，以及其他特征。

4. 粒子更新（Particle Update）

每一帧上的每个粒子都会调用粒子更新模块。可以在此处定义粒子生命周期内的会逐渐变化的所有行为。例如粒子的颜色随时间推移的变化。或者粒子受到各种力的影响，例如重力、引力等。

5. 事件处理器（Event Handler）

在事件处理器中，可以定义一个监听状态，然后在其他的发射器中触发状态。常用的比如 Generate Location Event 会记录当前粒子的位置状态，其他发射器可以读取该发射器的粒子位置用于一些诸如跟随、索引等操作。

6. 渲染（Render）

渲染模块定义了粒子的显示方式。可以在这里为粒子设置一个或多个渲染器。最新的 UE5.1 中渲染模块包括 Sprite Renderer、Ribbon Renderer、Component Renderer、Mesh Renderer、Light Renderer、Geometry Cache Renderer。它们可以渲染出各种需要的粒子显示内容。

2.3.1.3 模块

模块是 Niagara 效果实现的基础模块。可以将各类模块添加到组中构成堆栈。类似堆栈的结构，模块也是自上而下形成瀑布流的方式来运行。

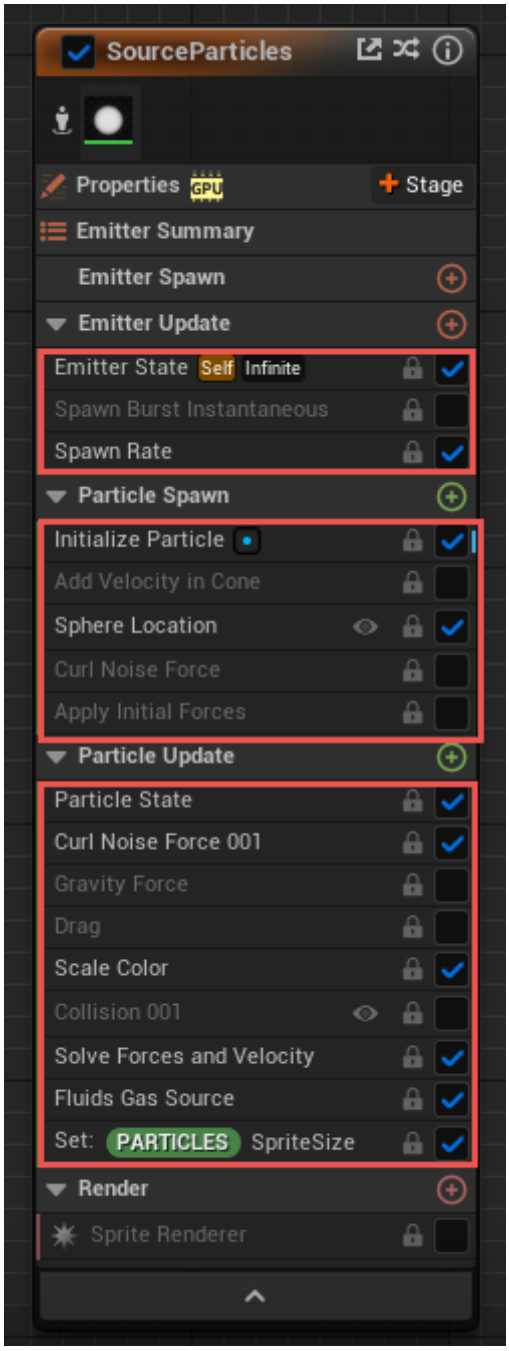

图 2. Niagara 中的模块

模块可以看作一种运算器，接收一些变量，然后计算并输出，同时数据也可以在模块间传递。模块本身时使用高级着色器语言 HLSL 进行构建的，UE 内部提供了一套图形化编程的接口来方便技术美术人员或者其他不了解编程的人员使用。在模块中除了 UE 提供的原生节点，还可以在 Custom HLSL 节点中内嵌 HLSL 以实现更高的自由度。

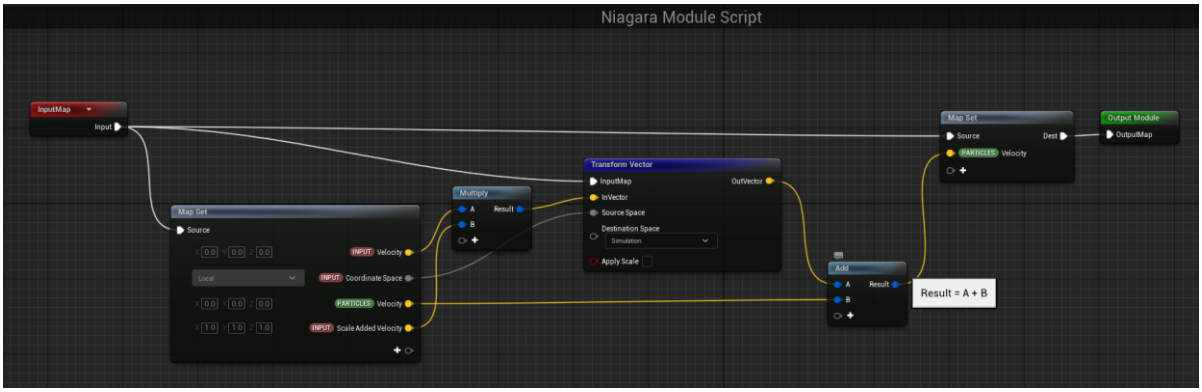

图 2. Add Velocity 模块

在运行时，模块脚本首先检索输入的速度，然后变换到正确的坐标空间最后更新到粒子的速度上。所有的模块都按照这种方式进行构建，只是复杂度不同。

2.3.1.4 参数

Niagara 中的参数是参与运算的单元，是数据的抽象化表现。参数主要分为四种类型：图元、枚举、结构体、数据接口。其中最常用的是图元类型，它表示具有各种精度和宽度的数值数据。数据接口则比较特殊，通常表示 UE 外部传入的数据源，比如 Render Target 等。

2.3.2 Niagara 的 Simulation Stage

Simulation Stage 是 Unreal Engine 4 中推出的试验性功能，它极大加强了粒子系统的灵活性，是整个粒子系统执行顺序和逻辑的核心。

在上一小节中我们提到，粒子核心逻辑在 Particle Update 中，在这个堆栈里，粒子每帧按照从上到下的顺序执行 Update 中的模块，更新自身的速度、大小、颜色等属性。这个过程是分批并行的，每一批粒子中的所有粒子同时从第一个模块开始向下执行。由于系统内部调度和算法的复杂度不同，我们无法得知哪个粒子先完成了当前模块的计算进入下一个模块。这也意味着，并行运算的粒子之间在同一帧是无法交叉影响的，每个粒子只能读取 Emitter、System 或者自己的属性，并对其进行修改，而无法得知其他粒子的属性。虽然在 Niagara 中可以通过 Attribute Reader 获取其他粒子的属性，并根据得到的数据进行运算，但由于 Particle Update 是每帧执行的，我们无法控制每个粒子的先后执行顺序，所以通过 Attribute Reader 获取到的是粒子在当前帧的初始值，在同一帧内如果粒子 A 要读取粒子 B 经过运算后的属性是做不到的。这也就是 Particle Update 的局限性所在了，由此 UE 引入了 Simulation Stage。

根据上面的论述，我们需要在同一帧下，所有的粒子执行完模块 A 的计算后更新

自身属性，然后我们利用更新的属性来计算模块 B，当所有的模块执行完成后，我们才更新当前帧的所有粒子属性。这就是 Simulation Stage 的作用。

我们可以向一个 Emitter 中添加多个 Simulation Stage，每个 Simulation Stage 类似一个同步节点，所有粒子执行完一个 Simulation Stage 前必须等待上一个 Simulation Stage 结束。所以事实上 Simulation Stage 会存在一些性能上的损失，所以只有需要运行的逻辑对当前帧其他粒子的属性有依赖时才会使用，例如水体模拟中对压力的解算就需要其他粒子在当前帧的状态，所以需要用 Simulation Stage。

之前说到 Simulation Stage 可以读取当前帧的粒子状态，这个状态的存储或者缓存是通过 Niagara 中加入的 Grid 结构实现的，这个结构可以在任意位置读写，而且可以在 Simulation Stage 内进行迭代更新。

在 UE5 中默认开启了 Simulation Stage，可以在发射器中直接添加，如下图所示。添加完成后可以选择当前 Stage 的迭代源和迭代次数。

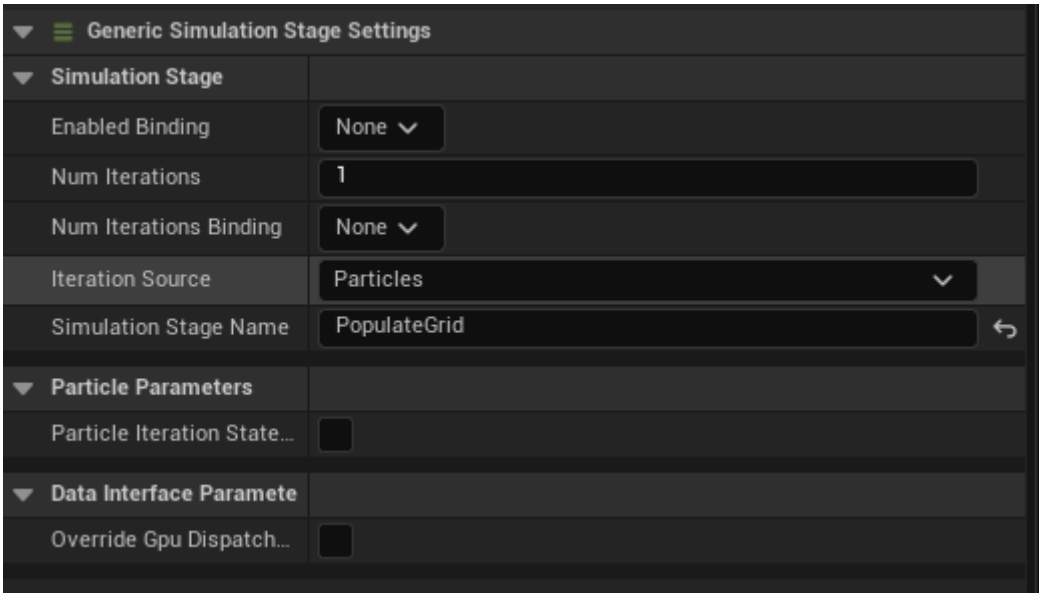

图 2. Simulation Stage 的选项菜单

根据图中的选项，我们可以选择迭代源是粒子还是我们定义的数据缓存，比如 Neighbor Grid 等。同时还可以选择在当前阶段的迭代次数，在计算较为复杂的情况下要慎重使用，否则会造成帧生成时间大幅上升。

2.3.3 Niagara 的缓存和空间加速结构

在 CPU 中可以通过多维数组来存取数据，这个多维数组还可以动态的改变大小和添加数据。但是在 GPU 中由于对并行性的要求，GPU 中的数据都是类似二维数组的集合，可以理解成很多张图片，并通过采样器读取。但是涉及到写操作则会在并行的情况下产生冲突，所以 GPU 并不能跨帧实现资源的写操作。在当前帧下，一个资

源只能有读或写的属性，在当前帧结束并对资源进行保存后才能在下一帧改变状态进行操作。可以看到这种方式其实是比较慢的，因为涉及到了资源的保存操作。而 UAV（Unordered Access View）的出现就是为了优化这一情况。虽然我们在同一个计算阶段随意读写会有逻辑问题，但是在下一个阶段可以直接使用上一阶段的结果，并在每个计算阶段都可以反复修改，而无需等待跨帧时资源的存取操作。

UAV 这一技术在 UE 中很早就获得了支持，并被广泛应用于 Computer Shader 和 UE 自身的 Global Shader 体系内，并且 Niagara 中 Simulation Stage 的底层也由 UAV 技术实现。

Niagara 中主要提供了 Grid2D Collection、Grid3D Collection 和 Neighbor Grid 三种缓存结构。Grid2D Collection 顾名思义是一个二维的 UAV，相当于一个图片，比图片强大的地方在于它可以存储人任意格式的数据结构类型。Grid3D Collection 和 Grid2D Collection 类似，只不过多出了一个维度，把 Grid2D Collection 看成纹理，那么 Grid3D Collection 就相当于 3D 纹理。

首先来看下 Grid3D Collection。在创建 Grid3D Collection 变量时可以指定网格包含的属性以及绑定的 Render Target 参数。

图 2. 创建 Grid3D Collection

随后可以用 Grid3D Set Resolution 来设置网格大小和分辨率：

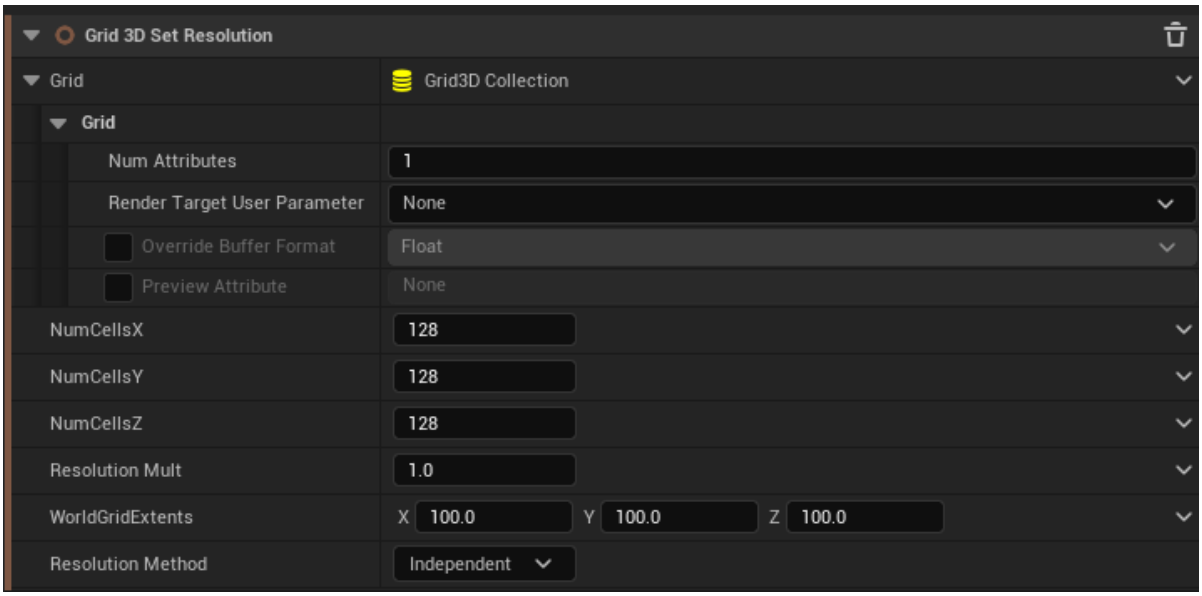

图 2. 设置 Grid3D 的网格大小及分辨率

经过这些设置后就可以在网格上保存需要的属性并在 Simulation Stage 中对网格进行迭代更新数据。

本文中主要用到的是 Neighbor Grid，它是 Grid3D Collection 的一个特化版本，它利用 Grid3D 的空间划分来进行搜索加速。所有的粒子可以保存一个网格索引，这样可以快速判断邻近粒子，这一特性在 SPH 的光滑核函数计算的时候起到了非常大的作用。新建 Neighbor Grid 时只有一个属性 Max Neighbor Per Cell，表示每个网格内部最多包含多少个粒子索引。新建完成后，可以通过 Neighbor Grid Set Resolution 来设置网格大小和分辨率：

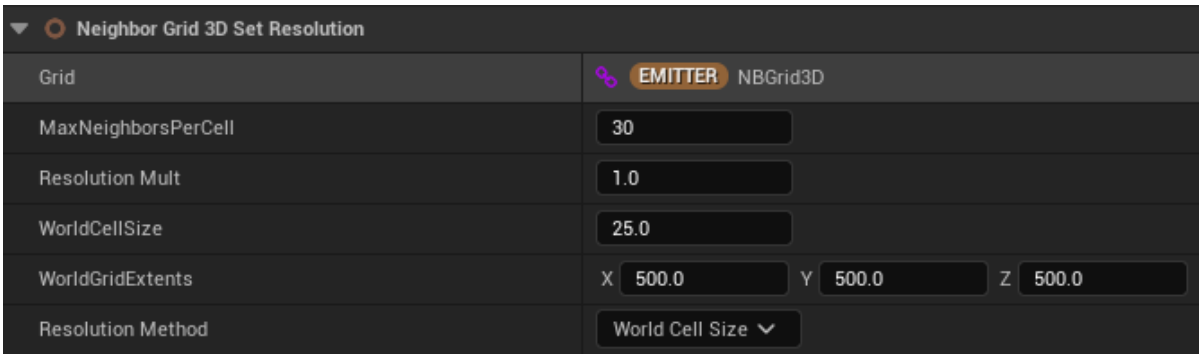

图 2. 设置 Neighbor Grid 分辨率和大小

除了图中通过每个网格大小设置网格分辨率的方法，还可以使用 NumCellsMaxAxis 方法。

通常在使用的时候，可以通过 Populate Neighbor Grid 模块来将所有的粒子划分到对应的网格中，并通过 Execution Index 取得粒子的 ID 后，就可以计算粒子所归属的网格索引。用这种方法，就可以在 SPH 和 PBF 中快速的定位光滑核范围内的粒子，

并得到它们的属性进行计算。

2.4 本章小结

本章首先介绍了在流体模拟中著名的 Navier-Stokes 方程，主要包括它的推导过程，并简单介绍了拉格朗日视角和欧拉视角两种求解思路。在实时水体模拟领域应用较多的是基于拉格朗日视角的粒子方法，而基于欧拉视角的网格方法在模拟烟雾等领域运用较多。随后则详细推导了 SPH 和 PBF 两种方法。在 SPH 中我们利用核函数对原函数进行插值求解流体的密度、压力、粘滞力等并利用它们更新粒子速度和位置。在 PBF 中讲述了约束投影的求解方法，以及如何对粒子位移进行更新。最后对两种方法进行了简单的比较。

随后则介绍了本次项目主要使用的工具，UE 和 Niagara 粒子特效引擎。介绍了它的基础使用方法，随后重点讲述了它在水体模拟中要用到的两项特性，即 Simulation Stage 和 Neighbor Grid。Simulation Stage 为我们提供了在同一帧对粒子属性进行读写和更新的方法，在 SPH 和 PBF 的临近粒子属性获取中有着重要作用。同时 Neighbor Grid 提供了空间哈希结构，让我们可以在很短的时间内搜索到邻近粒子并对其属性进行读取操作，大大加快了模拟速度。

第三章 水体渲染系统的架构和实现方案

3.1 水体渲染系统需求分析

一个完整鹅水体渲染系统主要功能应该包括：水体形态的模拟和渲染，可根据使用场景调节的水体发生方式及参数，以及用于展示用例的示例场景，让用户直观地看到效果并了解如何使用和调节。

3.1.1 水体形态模拟和渲染功能需求分析

水体模拟和渲染功能需求如下：

表 3.1 水体形态模拟和渲染功能需求

需求	**描述**
模拟水体运动形态特征	程序运行时能够模拟真实水体的状态，包括水面波纹，水流涡流等
模拟障碍物	程序运行时，水体碰到障碍物应该会做出符合物理和视觉效果的反馈
水体外力模拟	在场景外力，例如风力和推力下，水体要能做出对应的反馈
人物和水体交互模拟	表现角色和水体的交互
水体材质渲染	能够对水体进行渲染，表现出水体的颜色、通透感、体积感、撞击泡沫、水底焦散等
水下后处理特效	角色进入水下要有相应的后处理效果，展示水下的光照和环境

表 3.1 水体形态模拟和渲染功能需求

需求	**描述**
水体模拟效率	基于流体力学的水体模拟在实时运行时效率是至关重要的一环，Niagara 发射器整体耗时要控制在 10ms 内
水体渲染效率	水体渲染包括材质和后处理特效，整体耗时控制在 5ms 内
可靠性	确保在大粒子数下，模拟程序不发生崩溃和各种软硬件错误
可移植性	能够在移动和 PC 平台通用

3.1.2 水体编辑器插件功能需求分析

（TODO）

3.2 水体渲染系统设计与实现

3.2.1 总体流程

整个水体模拟渲染系统流程如下：
水体系统初始化，参数初始化，粒子运动模拟阶段，光栅化到 Grid2D，高斯模糊 Grid2D 生成的 Render Target，渲染水体材质，渲染后处理材质。

在水体系统中，首先要初始化水体模拟的参数，在世界位置中生成水体，然后在每帧运行时，根据不同的算法更新水体状态，对障碍物和外力作用做出反馈。然后在模拟出的水体基础上进行材质渲染，完成绘制。

3.2.2 水体运动形态模拟

在水体运动形态模拟上，我们分别使用 SPH 和 PBF 两种方法进行模拟实现。

3.2.3 水体材质渲染

3.2.4 场景交互实现

第四章 水体渲染系统的测试和性能优化

4.1 测试方案设计

4.1.1 算法性能测试

4.2 系统优化

4.2.1 渲染效率优化

4.2.2 性能优化

第五章 总结与展望

5.1 项目总结

5.2 未来展望

参考文献

[1] 2022 年 1-6 月中国游戏产业报告,2022

[2] Bridson, Robert. Fluid simulation for computer graphics. CRC Press, 2015

[3] Gerstner, F.J. "Theorie der Wellen", Abhandlunger der Königlichen Böhmischen Geselschaft der Wissenschaften, Prague. Reprinted in: Annalen der Physik 32(8), pp. 412–445, 1809

[4] Jerry Tessendorf. Simulation Ocean Water, SIGGRAPH 2001

[5] Lucy L B. A numerical approach to the testing of the fission hypothesis. The Astrophysical Journal. 1977, 8(12): 1013-1024.

[6] Matthias Müller, David Charypar and Markus Gross. Particle-Based Fluid Simulation for Interactive Applications. Eurographics/SIGGRAPH Symposium on Computer Animation (2003)

[7] Koschier D, Bender J, Solenthaler B, and Teschner M. Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids. In: Eurographics Proceedings. Tutorials. 2019

[8] Miles Macklin, Matthias Muller. Position Based Fluids. ACM Transcations on Graphics(TOG) 32.4 (2013):104

[9] Matthias Müller Bruno Heidelberger Marcus Hennix John Ratcliff. Position Based Dynamics. 3rd Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2006)

[10] 刘跃军, 龙姝羽. 虚拟现实引擎特效制作.

[11] 浅墨. 真实感水体渲染技术总结. [EB/OL]. https://zhuanlan.zhihu.com/p/95917609. 2021

[12] Epic Game Marketplace. Fluid Flux. [EB/OL]. https://www.unrealengine.com/marketplace/zh-CN/product/fluid-flux

[13] Epic Game Marketplace. Waterline Pro. [EB/OL]. https://www.unrealengine.com/marketplace/zh-CN/product/waterline

[14] Epic Game Marketplace. UIWS-Unified Interactive Water System. [EB/OL]. https://www.unrealengine.com/marketplace/zh-CN/product/uiws-unified-interactive-water-system?sessionInvalidated=true

[15] Epic Document. Niagara 概览. [EB/OL]. https://docs.unrealengine.com/4.27/zh-CN/RenderingAndGraphics/Niagara/Overview/

[16] Lucy, Leon B. "A numerical approach to the testing of the fission hypothesis." The astronomical journal 82 (1977): 1013-1024.

[17] 费昀. 不可压平滑粒子流体动力学算法 GPU 并行加速及其应用研究. Diss. 2013.

[18] Kelager, Micky. "Lagrangian fluid dynamics using smoothed particle hydrodynamics." University of Copenhagen: Department of Computer Science (2006).

附录 A

{附录正文…}

致 谢

{致谢正文…}

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　　月　　日

学位论文使用授权说明

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

● 按照学校要求提交学位论文的印刷本和电子版本；

● 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；

● 学校可以采用影印、缩印、数字化或其它复制手段保存论文；

● 因某种特殊原因需要延迟发布学位论文电子版，授权学校□一年/□两年/□三年以后，在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　导师签名：

日期：　　年　　月　　日