# 基于 VR 的水上运动项目模拟

钱韦克、梁剑川、陈仕潼

2024 年 6 月 21 日

## 1 项目摘要

本项目实现了一个包括水体的高真实感虚拟场景下的 VR 水上运动交互模拟系统，提供高沉浸感和人机交互性，并能够结合实际运动装置的用户数据实现对虚拟场景中水体的交互和运动数据可视化，模拟在水面上的行船运动方式。

## 2 项目划分与介绍

在本项目中，我们将任务划分为了三个部分。第一个是场景部分，第二个是 VR 互动和物理逻辑部分，第三个则是水体拟真部分。通过将三个部分分工完成后整合来得到最终的 VR 水上运动模拟器。在接下来，我们将对各个部分进行详细的介绍。

### 2.1 场景部分

在场景部分，我们通过多个关键步骤实现了复杂且逼真的环境生成，包括地形塑形、河流塑形、植被生成和沼泽生成。

1. 地形塑形：

我们使用了一种名为 ‘CarveTerrain‘ 的方法，通过调整地形高度、平滑度和噪声，实现自然过渡的地形。具体实现包括：

- 地形雕刻：通过曲线控制地形的内外雕刻，使用平滑度参数确保地形过渡自然，并添加噪声以增加地形的自然感。实时调试视图用于查看地形变化效果，确保地形修改的准确性和效果。

- 平滑度调整：通过调整平滑度参数，可以控制地形从一个状态到另一个状态的过渡平滑度，从而实现更加自然的地形过渡。

- 噪声添加：通过在地形中添加噪声，使得地形表面更加逼真和自然，模拟真实世界中的地形变化。

2. 河流塑形：

河流塑形是通过创建样条对象并添加控制点来实现的。样条面板提供了多种选项，包括基础设置、点管理和顶点颜色修改等。具体实现包括：

- 样条创建和管理：通过创建样条对象并添加控制点，灵活调整河流路径。样条面板提供了基础设置、点管理和顶点颜色修改等多种选项。

- 添加和删除点：通过 ‘AddPoint‘ 和 ‘RemovePoint‘ 方法，灵活添加和删除样条点，实现河流路径的灵活调整。

- 移动和旋转点：通过 ‘MovePoint‘ 和 ‘RotatePoint‘ 方法，精确调整样条点的位置和旋转角度，进一步精细化河流路径。

- 材料设置：通过 ‘SetMaterial‘ 方法快速应用预设的河流材料，使得河流的视觉效果更加真实和生动。

3. 植被生成：

我们实现了自动管理河流和湖泊周围的植被。通过检测样条下的地形并刷新植被掩码，自动管理植被的生长和分布。具体实现包括：

- 初始化植被系统：通过初始化植被系统，设置基础参数和资源，为自动生成植被做准备。

- 检测地形特征：通过检测地形高度和坡度，判断是否适合种植植被。

- 生成植被：根据地形特征，自动生成植被。在合适的位置种植植被，实现高效的植被管理。

- 刷新植被掩码：在移动样条点时自动刷新植被区域掩码，确保植被的分布始终与地形特征相匹配。

4. 沼泽生成：

通过设置特定的材料和着色器，我们创建了逼真的沼泽环境。具体实现包括：

- 材料设置：通过 ‘SetSwampMaterial‘ 方法设置沼泽材料，应用特定的材料和着色器，确保沼泽环境的逼真效果。

- 自动生成沼泽环境：通过 ‘GenerateSwampEnvironment‘ 方法自动生成沼泽环境，包括生成沼泽地形和添加沼泽植被。

- 沼泽地形生成：通过自定义的沼泽地形生成逻辑，创建真实的沼泽地形。

- 沼泽植被添加：通过自定义的植被添加逻辑，在沼泽地形中添加适合的植被，增强沼泽环境的逼真效果。

## 2.2 VR 互动和水上工具物理逻辑

本部分主要描述为了用户在 VR 环境中运动的真实感、沉浸感所做的 VR 交互以及物理模拟的工作。

### 2.2.1 VR 项目配置和交互

项目采用 Meta Quest VR 头显作为发布平台，需要使用相关的 SDK 进行开发。经过对比我们使用了 Meta XR SDK 进行开发，其具有功能完善、配置简易的特点，并且与使用的头显相配套，能够方便项目的配置以及部署。

在 VR 交互部分，为了实现用户在 VR 场景中对船的控制和交互，并且突出沉浸感、真实感，采用了手部追踪结合抓取动作的方式对交通工具中的可交互部件的控制。通过 Meta XR Interaction SDK 中的 HandGrabInteractable、HandGrabInteractor 接口获取用户是否抓取物体，以及用户抓取物体时的手部位置，以供可交互部件使用。

对于交通工具中的可交互部件，编写脚本在每一帧轮询其是否被手抓取，以及手部位置等数据。通过本帧与上一帧的手部位置差异，根据可交互部件具体的逻辑转化为位置、角度等变换数据相较于上一帧的增量，以获得当前帧的数据，提供接口交由物理部分查询，并在 LateUpdate 时应用到部件的变换中以实现视觉的反馈。通过插值实现不同位置之间的平滑过渡，避免了 VR 头显手部追踪数据突跃引起的视觉问题。

本项目实现了方向盘快艇以及双桨划船的 VR 交互。对于方向盘快艇，实现了操控转向的方向盘交互以及控制动力的控制杆交互。其中方向盘根据手部数据计算并提供其在 xz 平面上的转向角度，控制杆根据手部数据计算并提供其在 yz 平面上的转向角度。对于双桨划船，根据手部位置，通过 LookRotation 计算桨的朝向并应用，并为物理模块提供桨叶当前的位置。

通过以上的功能实现，基本实现了用户在 VR 环境中对水上运动交通工具的自然、流畅交互。用户能够实时操作控制部件，并实时得到反馈。

### 2.2.2 水上运动工具物理逻辑

为了实现水上运动的真实性和沉浸感，需要对船体进行贴近真实世界的物理模拟，从而为用户提供真实的体验。受限于项目进度以及实现难度，本项目仅实现了对于船体受力的简单模拟，包括浮力、重力、动力、阻力等。

1. 浮力

   真实的浮力模型需要考虑船浸没水中的排水体积，在项目中难以实现，且开销较大。为了在更简单的计算下实现较为逼真的浮力效果，需要对浮力模型进行简化。项目具体的实现为：在船体的四角分别设置一个控制点，共 4 个，通过四个控制点浸没在水中的深度与设定的完全浸没深度的比值来模拟船体靠近控制点的部分浸没水中的深度，从而模拟排开水的体积。使用这个比值乘以一个浮力系数 $c$，作为一个竖直向上的力作用于该点，产生的力和力矩就在一定程度上模拟了浮力产生的效果。四个控制点的浮力综合起来就实现了较好的浮力效果。通过浮力系数 $c$ 的设置，船体能够模拟在不同密度的水面上的浮沉效果。

2. 重力

   在计算重力时，并未使用 Unity 的刚体重力，或是直接将重力施加在船体的质心上。为了

与浮力相结合，模拟船体的质量分布，可以为计算浮力使用的各个控制顶点设定其所占船体重量的系数 $c_m$，总和为 1，并在实际计算时应用该系数求取作用于该控制点的实际重力：$\hat{G} = c_m mg$，作用于控制点上，能够在刚体上产生力和力矩的作用，以模拟船体质量分布不均的情况。在实际使用中，我们简单地将 4 个控制顶点的系数均设置为 0.25，也能起到不错的效果。

3. 动力

   为了使水上运动交通工具能够在水面上航行，并根据用户的控制作出反馈，需要实现船只的动力系统。本项目为方向盘快艇以及双桨划船实现了动力系统，能够根据用户的交互实时计算船只动力，并应用到场景中。由于实际船舶的动力较为复杂，我们对其进行了简化。

   – 方向盘快艇

     快艇通过水下尾部的推进装置向后喷射水流，通过反作用力向前运动。项目中简化为作用于推进装置位置，方向为快艇当前前进方向的动力。快艇通过尾部舵的转动改变水流方向以实现转动，项目在计算这个转向力时，查询方向盘提供的当前转动方向，乘以一个系数后得到该力的朝向。将一个该方向的力作用于尾部舵位置，其产生的力矩能够使得快艇转动。为了实现快艇的速度控制，为快艇设定一个最大的动力和转向力，计算动力实际大小时，根据控制杆当前角度得到当前动力的比例系数，作用于上述过程，就得到了实际的动力和转向力。

   – 双桨划船

     双桨划船通过手摇船桨推动水产生的反作用力运动。项目模拟该过程，根据用户划桨的速度为船体提供动力。在船桨末端的桨叶上设置一个控制点，当该点低于水面高度时，视为桨在水中，从而在此时用户的控制所产生的桨的运动会反作用于船体。通过两次模拟时船桨的位置差得到船桨的速度，乘以一个系数并反向后就得到了作用于船体的动力。

4. 阻力

   船体在水中运动时，会受到水的阻力，否则将一直运动下去。项目中采用线性的阻力模型，即船体受到的阻力与其速度呈线性关系。设定线速度阻力系数 $k_v$、角速度阻力系数 $k_\omega$，则船体在每个时间步受到的阻力和阻力矩可以表示为:$\mathbf{F}_{drag} = -k_v\mathbf{v}, \mathbf{M}_{drag} = -k_\omega\omega$。将阻力和阻力矩作用于船体，就模拟了阻力的效果。

综上所述，通过四种力的模拟，项目构造了一个较为逼真的水上运动船体物理模型，能够结合用户的交互，为用户提供一个具有沉浸感的环境。与此同时，对船体浮力、动力等的模拟还比较粗略，未能很好地模拟现实环境。

## 2.3 水体仿真模拟

本部分主要描述通过自定义脚本和表现方式来实现对水体的仿真模拟的方法。

### 2.3.1 水体表现方式选择

要表现一个水体，可以使用三维的空间层面上的表示方法，也可以使用二维的平面层面上的表示方法，而本项目采用的是后者。平面的水体模拟比起空间而言，能够模拟的精度更加有限，而且只能采取欧拉视角下的模拟方案。但是在海平面这种大规模的模拟需求上，二维模拟所需要的运算量更少，所以更适合本次项目。并且，像海平面这种有波浪的情况下，若要在拉格朗日视角下对流体粒子根据风力进行模拟来产生波浪的话，所需要的开销也是更加巨大的，而对于平面而言则只需要根据波函数进行高度上的修正即可。

另外，对更加之后的实现情况进行考虑的话，二维的平面实现比起三维的空间实现而言，较难检测入水中的摄像头并施加滤镜效果，所以在视觉效果的精细实现上可能需要进一步的深度考量。不过本项目中为对此功能进行实现，不过简单推测的话，可以以摄像头的高度与海平面对应位置的高度进行比较来判断是否需要施加滤镜效果。

而无论是欧拉视角还是拉格朗日视角，有一个在模拟层面上的好处是，每个单元下一时刻的状态都由这一状态决定，而不会由两个状态之间的中间状态所影响。所以这样的模拟运算非常适合通过 GPU 并行运算来大幅提升运算效率。在 unity 中，可以使用 compute shader 来进行并行的运算，所以后续所有的平面状态推进都是由 compute shader 来完成。而相关功能的调用、物理量的程序间通信和简易数据的前后处理则是由 C sharp 脚本调用 CPU 来执行。而使用 compute shader, 对被处理数据的规模有比较严格的要求，比如本项目中就要求平面尺寸的长和宽都是 8 的倍数。

经过以上分析，对于水体的模拟过程的第一步，就是通过脚本生成一个 N*N 大小的平面（这里的大小指的是网格的点数，平面的大小可以通过脚本任意指定），随后为各个 compute shader 里的 kernel 代码准备好前置数据即可。在接下来讨论复杂度的过程中，将以 N 代表网格尺寸的边长，以 M 代表网格顶点数的总和。而具体数目的选择，将在最后一部分进行展开分析。

### 2.3.2 海面波浪模拟

对于一个平面，要产生波浪起伏的效果，最简单的办法是为其使用三角函数的正弦波来修正网格顶点的高度，即：

$$\mathbf{h}(\mathbf{x}, t) = \mathbf{h}_0 + A\cos(\mathbf{k} \cdot \mathbf{x} - \omega t + \phi)$$

通过这个公式，可以得到形如图 1的波形。然而这么简单的波形肯定无法用来表现波涛汹涌的海面，所以需要更多更加复杂的波，赋予每个波不同的频率和振幅，来得到更加复杂的叠加波。这就是 Gerstner 波叠加公式。

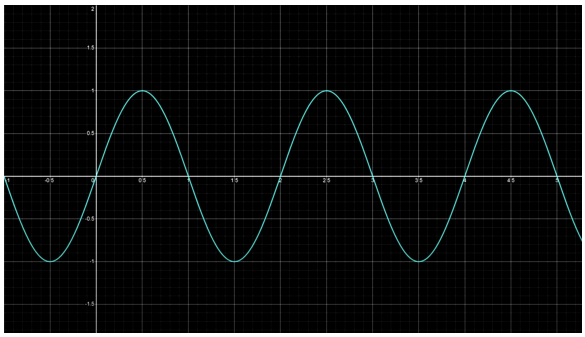

图 1: A Simple Sine Wave

$$\mathbf{h}(\mathbf{x}, t) = \mathbf{h}_0 + \sum_{i=1}^{N} A_i \cos(\mathbf{k}_i \cdot \mathbf{x} - \omega_i t + \phi_i)$$

通过波的叠加，就能得到像图 2一样的一个更加复杂、不规则的波形：

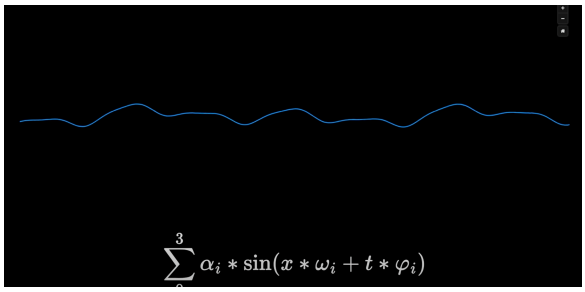


图 2: A Multiple Sine Wave

然而，如果仅仅通过这样的波叠加的话，得到的结果可能会非常夸张。在不限制振幅和频率的情况下，波与波之间的叠加所产生的值的变化幅度可能会非常大，产生如图 3所示的结果。

所以，为了避免这种情况，需要使用 Fractional Brownian Motion(FBM) 的方法来控制每个波的最大振幅和频率之间的关系。简单来讲，就是在随机控制波的数据的同时，要让频率越高的波振幅越小。这样就能让得到的水面有一系列比较逼真的波形了。

另外，从上图也不难发现，波的数量越多，水面的效果越好。但是越多的波数也会给运算带来更多的负荷。若是通过这种波叠加的方式来计算的话，总体的运算复杂度是 O(MW)，其中 W 是波的数量。所以在这种即时运算的项目中，波的数量严格受到每帧之间渲染用时的限制。那么就像图里的 32 个波就已经有比较不错的视觉效果，是否可以为了效率牺牲波的数量呢？答案是否定的。波的数量不仅仅影响了单一平面的质量，还会导致重复的问题。因为每个

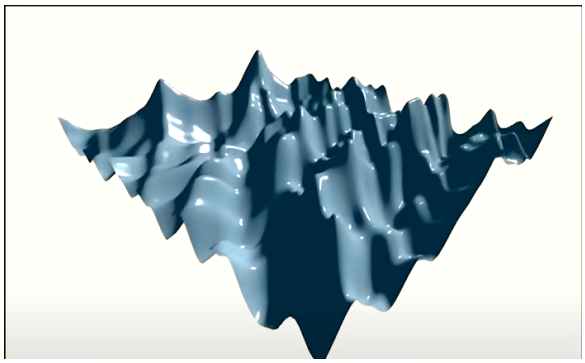

图 3: A Explosive Wave

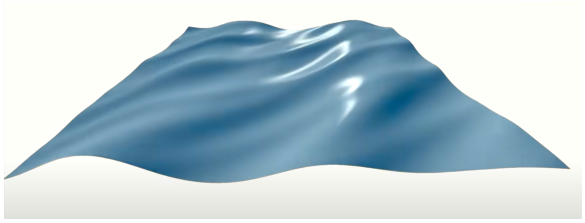

图 4: 8 Waves

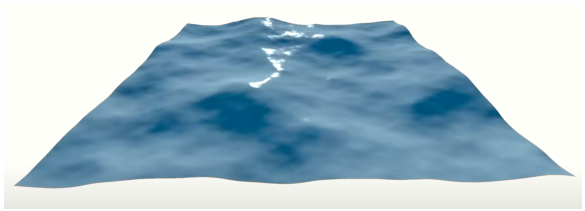

图 5: 32 Waves

正弦波都是重复性的，所以最终得到的叠加波也是重复性的。叠加使用的波越少，那么最终波里也就会更快地出现重复。这将使得海面变成如图所示的效果。

所以，为了最后的视觉效果，还是得增加波的数量。而 Gerstner 波叠加公式已经不足以支撑这样的需求，因此就需要新的方法来得到拥有更多波的海面。在这里，就需要使用到傅里叶变换了。傅里叶变换用于将空间域的波形转换到频率域，以便进行频谱分析。而其变换公式如下：

正变换：

$$\hat{f}(\mathbf{k}) = \int_{-\infty}^{\infty} f(\mathbf{x}) e^{-i\mathbf{k}\cdot\mathbf{x}}\, d\mathbf{x}$$

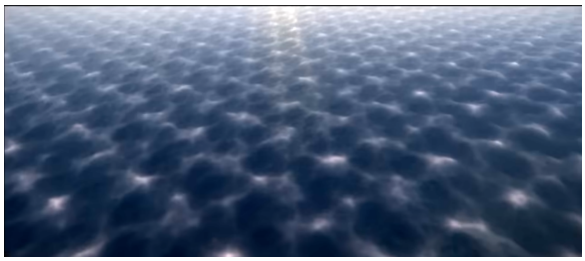

图 6: Repetition

负变换：

$$f(\mathbf{x}) = \frac{1}{(2\pi)^n} \int_{-\infty}^{\infty} \hat{f}(\mathbf{k}) e^{i\mathbf{k}\cdot\mathbf{x}}\, d\mathbf{k}$$

在波的叠加中，每个波都是一个正弦函数，也就是换算到频域空间上一个高度为其振幅的点。而我们最终需要得到的是空域（或者说时域）空间下某个坐标点在某个时间点上，经各个波影响后的高度值。所以只需要通过傅里叶逆变换，将频域空间上的所有波的信息转换到空域空间，得到的结果就是每个点的高度了。所以海面高度的计算公式如下：

$$h(\mathbf{x}, t) = \sum_{\mathbf{k}} \hat{h}(\mathbf{k}, t) e^{i\mathbf{k}\cdot\mathbf{x}}$$

这里的 $\hat{h}$ 就是海洋波浪的频谱。在有了公式之后，就该获取频谱了。现在比较主流的集中频谱有 Phillips 频谱和 JHONSWAP 频谱，这两个频谱都是基于海洋统计学而得到的海洋学数据，比起上一步里自己手动随机出的频谱而言更加真实，而本次项目中所使用的是前者。计算 Phillips 频谱的整个过程和公式如下：

$$P(\mathbf{k}) = \frac{A \exp\left(\frac{-1}{(kL)^2}\right)}{k^4} \left(\mathbf{k} \cdot \hat{\mathbf{w}}\right)^2$$

这个式子主要描述风驱动的海浪的能量分布。其中的 k 是当前频率的模长，w 是表示风向的向量，L 代表最大波浪峰值，数值上为

$$L = \frac{V^2}{g}$$

其中 V 是风速，g 是重力加速度。

接下来，就可以生成具有随机相位的初始频谱。

$$\hat{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}} \left(\mathcal{N}_1(0,1) + i\mathcal{N}_2(0,1)\right) \sqrt{P(\mathbf{k})}$$

其中的两个 $\mathcal{N}$ 代表符合标准正态分布的随机数。在项目的代码里，生成该随机数的部分参考了Box-Muller 方法 的代码。

接下来为了模拟海浪，我们需要将初始频谱和其共轭对称频谱进行组合：

$$\hat{h}_0(\mathbf{k}) = \frac{1}{\sqrt{2}} \left(\mathcal{N}_1(0,1) + i\mathcal{N}_2(0,1)\right) \sqrt{P(\mathbf{k})}$$

$$\hat{h}_0^*(-\mathbf{k}) = \frac{1}{\sqrt{2}} \left(\mathcal{N}_1(0,1) - i\mathcal{N}_2(0,1)\right) \sqrt{P(\mathbf{k})}$$

并且还要加上时间 t，来得出海浪的频谱在时间 t 的演变：

$$\hat{h}(\mathbf{k}, t) = \hat{h}_0(\mathbf{k}) e^{i\omega(\mathbf{k})t} + \hat{h}_0^*(-\mathbf{k}) e^{-i\omega(\mathbf{k})t}$$

至此，我们就得到了 Phillips 频谱的所有计算公式，也就可以通过傅里叶变换来算出每个点在任意时刻的高度值。然而此时复杂度依然为 O($M^2$)，无法满足即时运算的需求，所以就需要使用快速傅里叶变换的算法（FFT）来进行优化。然而由于篇幅原因，快速傅里叶变换的具体推导和实现方法，例如代码逻辑，数组设计，和蝶形网络等优化就不在这里介绍。

然而整个流程的具体实现部分，包括生成 Phillips 频谱的部分和将快速傅里叶应用到之前的公式的过程，在所有找到的资料中，都多少存在些缺少的错误或未详尽之处。所以我在基于找到的资料的基础上，进行了进一步的补充和完善。例如，资料说在推算公式时需要将平面的尺寸设为 1，但实际上平面的尺寸可以通过合并项来消除；或者资料给出的公式转换过于复杂，涉及非常频繁的函数变换，所以我在此基础上进行了一些优化，使最终的公式使用的都是原来的物理量，但也因此需要在每个 FFT 的过程前后进行一些额外的预处理和后处理。具体逻辑都有在代码中体现。

总而言之，在生成波浪的部分，每一帧的过程具体分为以下步骤：

1. 根据当前时间生成当前的 Phillips 频谱数据
2. 对 Z 轴执行一遍 FFT 算法
3. 对 X 轴执行一遍 FFT 算法

经过以上步骤，就可以得到每一帧状态下的所有物理量，包括高度、梯度和计算雅可比行列式和尖峰修正所需要的 D 函数等。并且时间复杂度仅仅为 O(Mlog(M))。最终在尺寸为 128 即 16384 个波的情况下，可以得到下面的结果。

### 2.3.3 海洋的物理模拟效果

在有了动画效果之后，就需要考虑实现海洋与其他物体产生互动时的物理模拟效果。在大部分精细的水体模拟中，使用的都是基于粒子的 SPH 方法等来对每个流体粒子进行模拟，再对液体表面进行重建，这样确实能够得到非常逼真和细致的流体效果，包括水花、波浪等等。但是正如之前部分讨论中所说，本项目中的海面模拟不足以支持将整个大海以粒子形式模拟所需

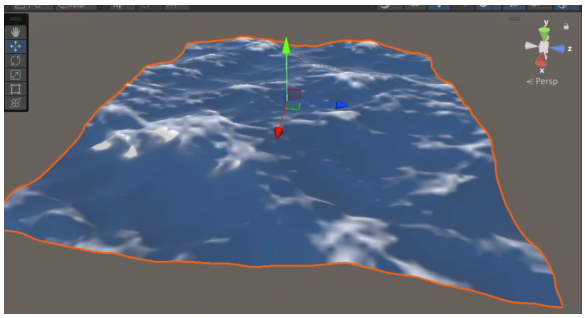

图 7: FFT waves

要的开销。本项目最初的设想是将涉及到物理互动的部分以粒子形式来计算互动效果，其余部分仍然用网格模拟，但这个功能最后没有进行实现，取而代之的仍然是对海面的表面平面进行模拟。而无论是哪个视角下的模拟，都需要使用流体力学中的 Navier-Stokes 方程来描述流体的运动，其形式为：

动量方程：

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}$$

质量守恒方程（连续性方程）

$$\nabla \cdot \mathbf{u} = 0$$

但是在海面的流体模拟的过程中，其实可以对以上方程进行优化处理。首先，可以将水视作密度 $\rho = 1$，并且无粘性 $\nu = 0$ 的流体。其次，上面这个式子的左边其实是一个物质导数，但在计算的过程中不需要这样子的表示形式，所以在进行简单的移项后，第一个方程就可以简化为：

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{f}$$

在这个式子里，左边代表速度场随时间的变化，右边则分别是平流，压强和外力的作用结果。也就是说，这个过程也就是根据上一时刻的流体信息来根据 NS 方程计算出下一时刻的流体信息。从这个式子来看，要得到下一时刻的速度场可以分为三个步骤，即分别计算这三个项。

在前两步，应该计算平流项或者外力项。这两个的顺序其实可以前后交换，是不会有影响的。在外力项的计算过程中，只需要使用简单的前向欧拉法即可，即

$$\mathbf{u} = \mathbf{u}_0 + f\Delta t$$

而对流项的计算过程则要复杂许多。如果也采用前向欧拉法计算的话：

$$\mathbf{u}_i^{n+1} = \mathbf{u}_i^n - \Delta t \mathbf{u}_i^n \frac{\mathbf{u}_{i+1}^n - \mathbf{u}_{i-1}^n}{2\Delta x}$$

虽然看似没有问题，但是由于前向欧拉法是无条件不稳定的空间离散方法：无论取多么小的 $\Delta t$，随着时间步的推进，累积误差终将发散。即使使用更稳定的时间积分方法来取代前向欧拉方法，解决了时间上的 PDE 计算，空间上的 PDE 计算还是会带来重大的麻烦。标准中心差分方法不可避免地会出现的零空间问题，具有高频震荡性质的速度场对空间的导数被错误地计算为 0 或几乎为 0，低离速度分量被分离出来，从而导致模拟效果中出现许多奇怪的高频摆动和震荡。

所以，在这里需要使用的是另一个方法，半拉格朗日法。简而言之，就是根据当前点的速度场推断出上一时刻是哪个粒子跑到了当前的位置来进行反向追踪，从而以那个粒子的数据作为当前位置的数据。

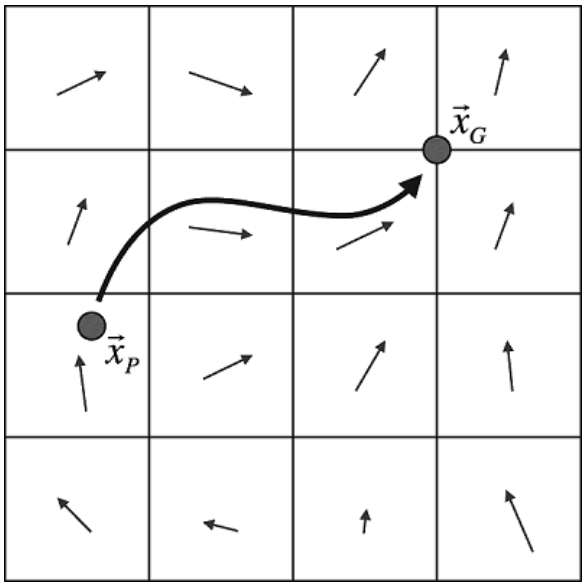


图 8: Semi-Lagrangian Advection

在得到位置之后还得进行一个插值，本项目里所采用的是双线性插值，所以在精度上比起三线性或其他插值方法还是存在一些精度上的损失。

另外，由于整个流程都是基于网格进行处理，所以进行物理模拟时所使用的网格结构也是 MAC(marker and cell) 结构。它将水平的速度存在网格的左右边界，而垂直的速度存在上下边界。这种分开存储的方式在后续的实现过程中可以发现是必须使用的，因为这样可以在数值计算时使得我们可以准确地采用中心差分法计算压力梯度和速度的散度，同时克服了中心差分法无法在某些情况下达到无偏的缺点。

另外，对于时间步长 $\Delta t$ 也有要求。越小的步长能带来越高的精度，但会增加对性能的要求。而步长过长会导致不满足 CFL 条件，产生意料之外的结果。在本项目中，根据时间进行的前推采用的也是前向欧拉法，并没有使用二阶的龙格库塔法，因此也产生了一些精度损失。

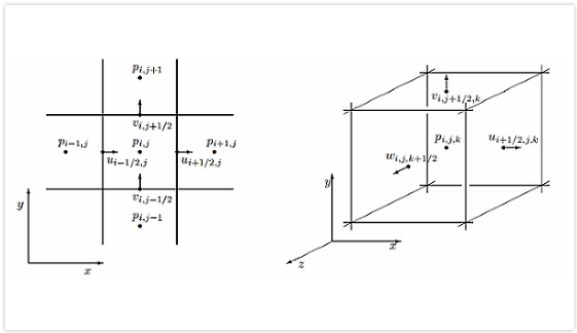


图 9: MAC grid

在计算完平流项之后，就需要根据压力场计算出最终的速度场。但是欧拉方案不会着重研究具体粒子间的作用力，因而不会正向去求解 $\nabla p$，而是利用最终结果的速度场散度为 0 这一特性，将当前的仅计算完外力和平流的速度中间量投影到一个散度为零的平面上，来间接地得到这个压力场和最终的速度场。也就是说，现在已经计算出了一个散度非 0 的速度场 u，我们需要得到散度为 0 的最终速度场 $\mathbf{u}^*$，那么可以得到：

$$\mathbf{u}^* = \mathbf{u} - \Delta t \nabla p$$

而根据速度场散度为 0，有

$$\nabla \cdot \mathbf{u}^* = 0$$

所以对左右两边同时进行算符 P 后，可以得到

$$\nabla^2 p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}$$

其中式子右边的项是完全可以求出来的，那么这就已经变成一个泊松压力方程了。通过对这个方程使用雅可比迭代法进行求解，就能得到最终的压力场和速度场了。

$$p^{k+1}(\mathbf{x}) = \frac{1}{4}\left(p^k(\mathbf{x} + \Delta x \mathbf{e}_x) + p^k(\mathbf{x} - \Delta x \mathbf{e}_x) + p^k(\mathbf{x} + \Delta y \mathbf{e}_y) + p^k(\mathbf{x} - \Delta y \mathbf{e}_y) - \frac{\rho \Delta x^2}{\Delta t}(\nabla \cdot \mathbf{u})\right)$$

通过以上分析，就完成了一个基本的通过 NS 方程求解流体的流程。但在具体实现的细节上，还有很多问题。例如怎么设置压强场？怎么构建整个坐标系？如果是二维空间，对应的值应该满足什么样的映射关系，并且也不可能满足结果的散度场为 0；而如果是三维空间，就无法仅仅对一个二维空间进行 NS 方程的求解。物体的物理交互效果又该怎么做，各个符号之间的单位关系又是什么等等。我花了大量的时间来试图理解这个方程的本质来试图实现三维空间到二维空间的抽象和构建一个可行的算法，在最终的实现代码中，我使用的是如下设定：

将整个水体以二维空间进行模拟，二维空间上的每个点实际代表的是该点位置处从底部到海面的一个“水柱”，每个水柱的横截面边长都是 dx，并且视水柱为长方体。

在这个前提下，$\mathbf{u}(\mathbf{x},\mathbf{z})$ 代表在该位置处的流体速度，但并不是真实意义下单位为 m/s 的速度，而是表示流入/流出该位置处水柱的水量对该水柱高度产生每单位变化的影响。这是一个欧拉视角，以静止的网格视角来记录流动的水体信息。

压强场 $\mathbf{p}$ 表示的是该水柱处位于海平面位置的压强，数值上根据阿基米德公式为 $\mathbf{p}=\rho\mathbf{gh}$。

由于考察的单位是水柱，所以没有在 y 坐标轴上的分析计算，因此忽略重力作为外力，并且忽略散度场为 0 的约束条件。但忽略散度场的约束条件并不意味着放弃真实模拟，而是默认 $\mathbf{u}_y$ 的值能够直接满足三维视角下的散度场约束，所以不用再进行额外计算和推演。

通过上述的设定，可以正常实现位于流体内部的无外部物体交互情况下的物理拟真压力场。在接下来，还需要对边界情况和外部物体约束进行进一步分析。

首先分析边界情况。边界情况指的是 mac 网格中最边上区域的速度值，以及在区域外部的压力场的设置情况。在本项目中，边界处的速度值设为 0，而压力的贴图则设为了 Clamp 模式，即边界外的值返回的是边界处的值。这样的设置方式能最好的模拟容器内的流体，但是对于海面的流体，由于模拟区域与外部区域是联通的，所以不能以容器的方式来进行物理模拟。但是处于时间原因，没有完成进一步的实现。

接下来是物体交互的部分。通过物体与流体的互动，也会对流体产生诸多约束条件，例如对压力场和速度场的约束条件。但是同样出于时间原因，这一部分的实现也被简化了。从使用策略的角度来看，在碰撞体的选择上，考虑到船只的特性，选择了方形碰撞体，所以只需要处理碰撞体边界和底部的情况。对于边界处，采用的是和容器壁一样的策略，即将速度约束为 0，以及在半拉格朗日方法下将倒推进碰撞体的粒子的速度也设为 0。而对于碰撞体底部，我选择将压力场强制设为对应值，并将所有排开的流体均匀分配到碰撞体周围。

而在具体的实现过程中，需要一系列的预处理步骤来完成这个策略。大致逻辑为根据方形碰撞体大小来设定约束高度，并根据约束高度和实际高度之间的区别作为是否被约束的依据。并分别在处理平流项的过程中、以及泊松解得压力场之后，根据约束情况来强制对速度场和压力场进行操作。

经过以上的步骤后，就得到了一个勉强能够基于 NS 方程产生物理拟真效果的水面。

接下来，是这个实现方案出现的问题：

第一个，最显然的问题，是在每次进行模拟的时候有大概率会出现的数值爆炸现象。

而另一个问题，是在模拟过程中，没有粘性的液体表现出了粘性。这事实上也是数值耗散所带来的影响。

对于这些问题，缩小模拟时的 dx 都能在一定程度上得到优化。但是在本项目中，没能将物理模拟的尺度和之前计算傅里叶变换波时的尺度区分开来，所以无法限制模拟时的尺度，导致最终结果中的物理模拟效果仍然存在诸多问题。

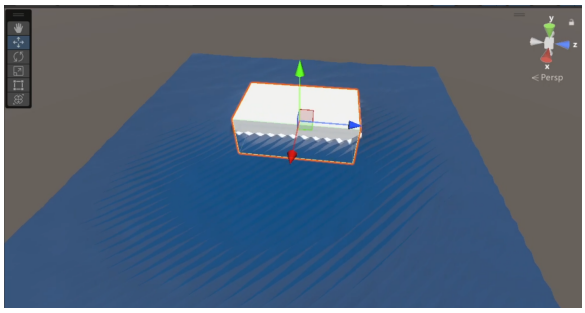

图 10: Explosive

#### 2.3.4 最终效果优化和补充

经过上面两步，分别得到在海面在风力作用下多重波浪的高度和梯度，以及水体受物理效果时的具体压力场信息。将两个信息相加，就可以得到一个简单的带有物理效果的海面。但在实现海面之外，还可以对网格顶点进行 xz 方向上的变形，将网格向波峰位置挤压，即用 sin 挤压 cos，cos 挤压 sin，来得到更加尖，更加逼真的波浪效果。但有的时候可能会挤压过头，这样就会产生波浪碎裂的情况。所以可以通过雅可比行列式，来检测是否出现负平面的情况来判断是否出现了挤压过度的情况，将雅可比行列式的值作为贴图传给着色器后，着色器就可以在对应位置显示泡沫效果。例如：

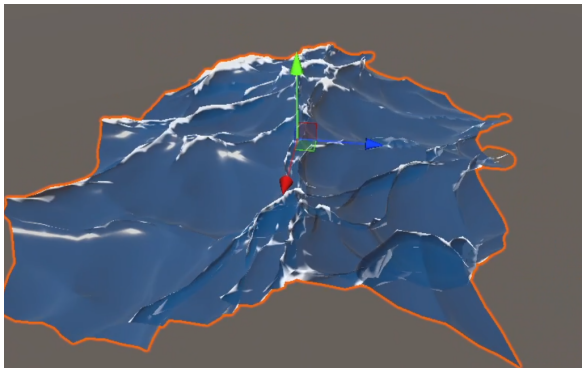

图 11: Huge waves with foam

至此，就完成了一个简单的带有（不完全的）物理模拟效果的 FFT 海面平面。但是在演示视频中，出于效果上的考虑，最终还是去除了物理拟真的效果，因为那个数值爆炸的效果对眼睛的伤害过大。

# 3 结果展示

最终的项目在演示视频里已经展示了一部分。另外的部分演示的话放在了另一个文件夹里，包括其他的交通工具，以及物理拟真水面的效果，和更加多样的 FFT 海面效果。其中进行复杂度分析的话，FFT 的海面模拟为 O(MlogM)，物理拟真部分的模拟为 O(M)，这是不考虑常数项的情况下。因此从理论上分析而言，这样的复杂度还是足以支持一个比较复杂的海面的。

# 4 成员分工

钱韦克：水体模拟部分（33%）
梁剑川：VR 互动和物理拟真（33%）
陈仕潼：场景部分（33%）

# 5 参考资料

naturemanufacture.com
https://zhuanlan.zhihu.com/p/64414956
https://yangwc.com/2019/05/01/fluidSimulation/
https://www.cs.ubc.ca/ rbridson/courses/533d-winter-2005/cs533d-slides-mar9.pdf
https://matthias-research.github.io/pages/publications/hfFluid.pdf