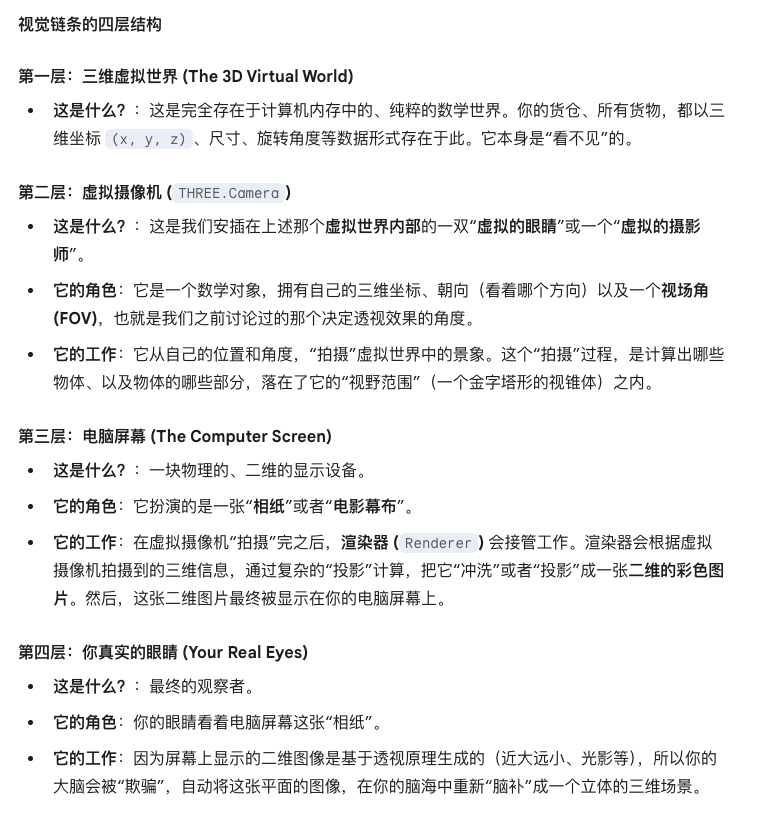
技术选型：unity vs three.js

前者：更成熟， “顶级”三维引擎

后者：更轻量 需要手动实现渲染引擎逻辑。

vue + sheet.js +three.js

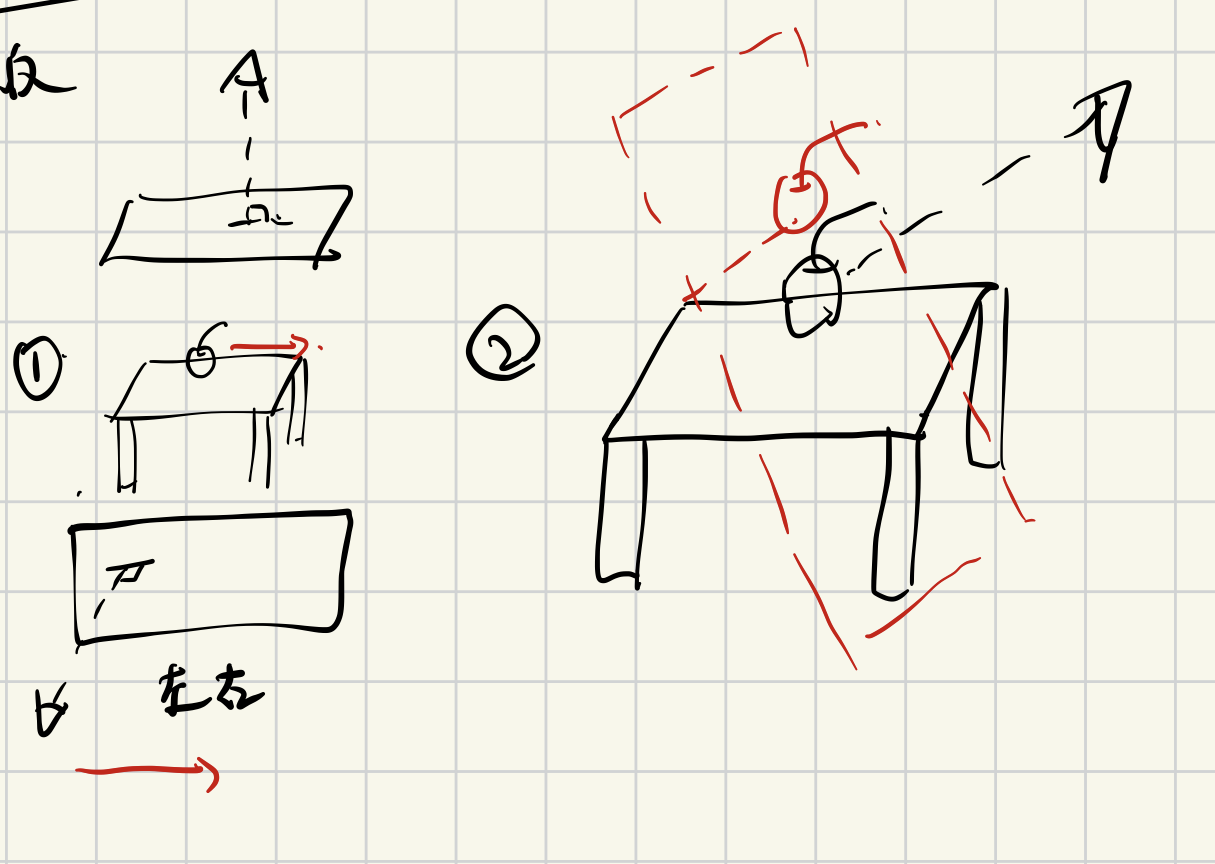


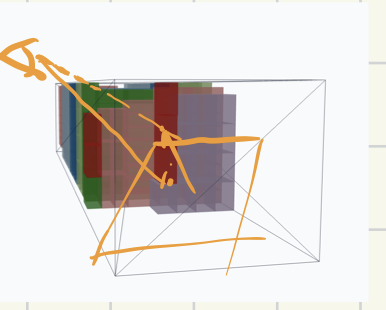
先前版本的问题：货物“乱飞”

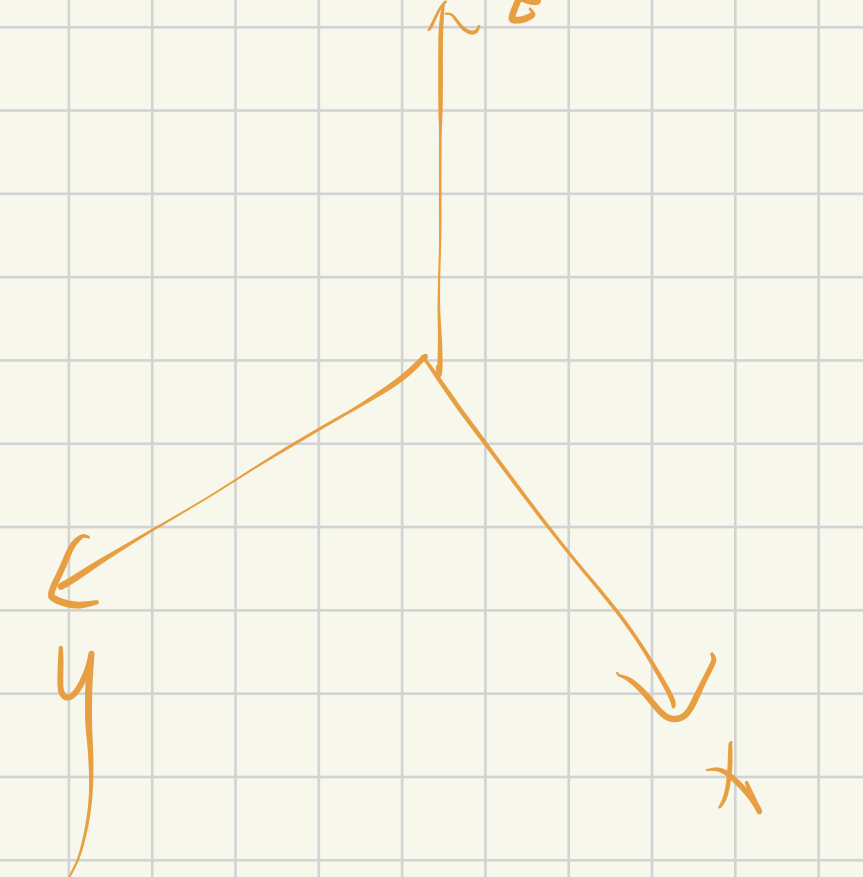
鼠标是在一个二维平面（屏幕）上移动，但要控制一个三维空间里的物体。从你眼睛（相机）的位置，穿过你的鼠标指针，可以画出一条无限长的直线。计算机并不知道，你希望物体在这条直线上的哪个“深度”上移动。

为了解决这个困境，简单的拖拽控制器采用了一个看似合理、但实际上有巨大缺陷的方案：**它在三维空间中，创建了一个“永远平行于你电脑屏幕”的、看不见的虚拟移动平面。**

·







自动吸附：

接触面候选：由“仓库边界”和“其他货物”的包围盒生成。要求在“其余两轴”上与目标物的投影重叠（接触也算

重叠）。

轴向吸附（与拖拽解耦）：先得到用户期望的轴向位移；若期望位置距某候选接触面在 snapTolerance 范围内，则把该轴位置直接修正为接触面坐标；然后用严格相交检查验证能否放置。

次轴吸附（可选）：当你进行“单轴拖动”（Shift+轴键）时，除了主轴以外的两条轴也会各自尝试在容差内贴到最近接触面，利于一次动作实现三向对齐。

碰撞边界：使用 AABB（轴对齐包围盒）严格重叠判断，接触不算穿透；同时保证结果仍在仓库包围盒内。

调参：

tolerance 越大越容易贴上附近面，但过大可能“吸错面”；一般与数据单位、建模网格或测量误差同量级；

secondaryAxis 仅在单轴拖动时生效，便于快速“卡紧”。

受阻侧向滑移：

触发时机：按住 Shift 锁单轴移动（X 或 Z）并被障碍挡住时，侧向滑移启动；非单轴或 Y 轴不启用。

搜索策略（次轴局部试探）：

在次轴上，以当前位置为中心，左右各 radius 的范围内，按 samples（奇数）均匀取点，形成候选次轴

位置。

对每个候选位置，计算沿主轴能到达的“极限接触位置”（不穿透墙/物体）；将该主轴极限与候选次轴组合成一个候选落点。

对候选落点做“严格相交 + 仓界”校验，过滤不可行点；从可行点中选择“主轴前进距离最大”的那个，作为本次移动结果。

- 设计初衷：模仿人手“贴着边往前蹭”的操作；它并不改变物体尺寸，只是在次轴上小幅调整路线来寻找能通过的入口。

调参：

radius 控制横向探索范围（越大越容易找到入口，但也更易跑偏或多算）；

samples 控制探索精细度（越大越细致，耗时越高，常用 7/9/11）。

一些问题/细节：

1. 是否要保留 受重自动下落功能？利/弊 不需要 需要判断悬空
2. 需要添加：旋转货物
3. 导出时，执行覆盖/另存为，or 让用户自己选择？ 另存为
4. 修改下重置的逻辑
5. 物流系统+可视化系统 文档