CARLA同步测试

模式	时间步长	控制权	优点	缺点	适用场景
异步 + 可变步长（默认）	自动（由服务器实际性能决定）	服务器主导	速度最快，能充分利用硬件	不可复现，物理与传感器不同步，掉帧/乱序风险	快速可视化，demo，交互式测试
异步 + 固定步长	fixed_delta_seconds 固定	服务器主导	信息容易对应到仿真时刻，运行速度不受实时限制（可快于实时）	仍然不可复现（浮点误差 & 不同步），不同机器结果可能不同	长时间仿真、不需要精确复现，只要数据带时间戳
同步 + 可变步长	客户端 tick，但每步长短长不定	客户端主导	能停等慢客户端，防止数据丢失	时间步长可能过大 → 物理失真，不可靠	几乎没有实际用途
同步 + 固定步长（当前设置）	fixed_delta_seconds 固定	客户端 tick 决定推进	可复现（Deterministic）传感器/物理/控制完全对齐结果一致性高	速度受限于最慢的客户端（可能低于实时）	研究实验、训练、仿真验证、需要传感器精确同步的任务

实验环境：
笔记本配置

ubuntu20.04

GPU：4070

CARLA 0.9.12 编译版本

测试内容：

3.1 仿真循环抖动 (Jitter) 分析 （一票否决项）	准备工作： 1. 对标准Ubuntu系统进行性能调优。使用isolcpus内核启动参数，隔离出2-4个CPU物理核心，专用于运行仿真任务，以避免操作系统自身的“噪音”干扰。 2. 使用taskset命令，将待测试的仿真进程**强行绑定（Pinning）到被隔离的CP核心上运行。 测试执行： 3. 以固定步长（dt=10ms）驱动仿真空跑10分钟，收集至少60,000个循环耗时数据点。

在**同步 + 固定步长**下：

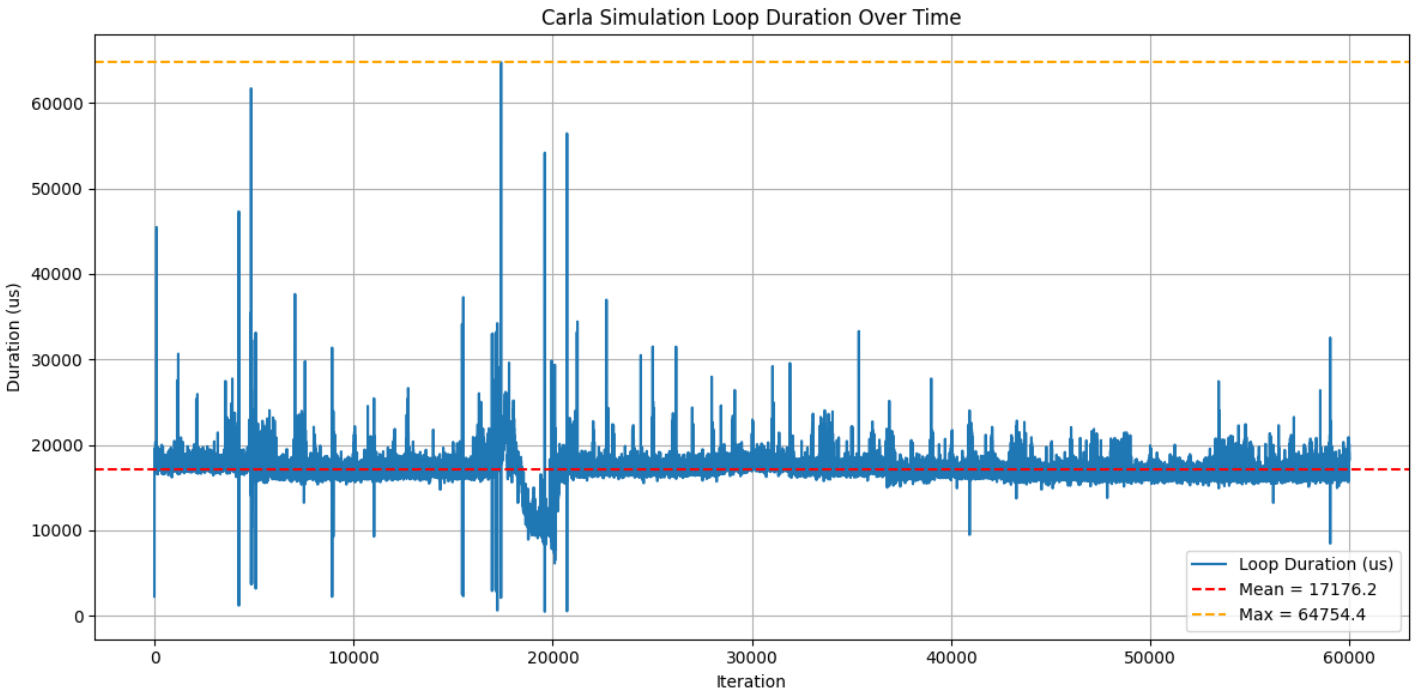
每次 `world.tick()` 都让**仿真时间**前进固定的 10ms；

但这一帧**实际算完**要花多久（0.3ms、12ms、17ms…）取决于渲染/物理/传感器/系统调度等，这就是**tick 耗时**。

仿真时钟（Sim clock）：CARLA 世界里的时间线。设定 `fixed_delta_seconds=0.01` **100HZ**，每个 tick 世界时间就前进 10ms。

墙钟（Wall clock）：现实中电脑真的跑完这一帧所花的时间，`time.perf_counter()` 量到的数值。

正常渲染模式



横坐标(X 轴)= 迭代次数（0–60000）。

纵坐标 (Y 轴) = 每次 `world.tick()` 的墙钟耗时（单位：微秒 µs）

“步长”是虚拟世界的时间；“tick 耗时”是现实世界跑完这一步需要的时间。

均值 (Mean)：≈ **17.2 ms**（17176 µs）

每步比设定的 10 ms 固定步长要长，说明渲染/传感器计算消耗大，仿真实际跑不到 100 Hz 实时。

最大值 (Max)：≈ **64.7 ms**

出现了严重尖峰，可能是渲染帧卡顿、资源加载、传感器回调阻塞。

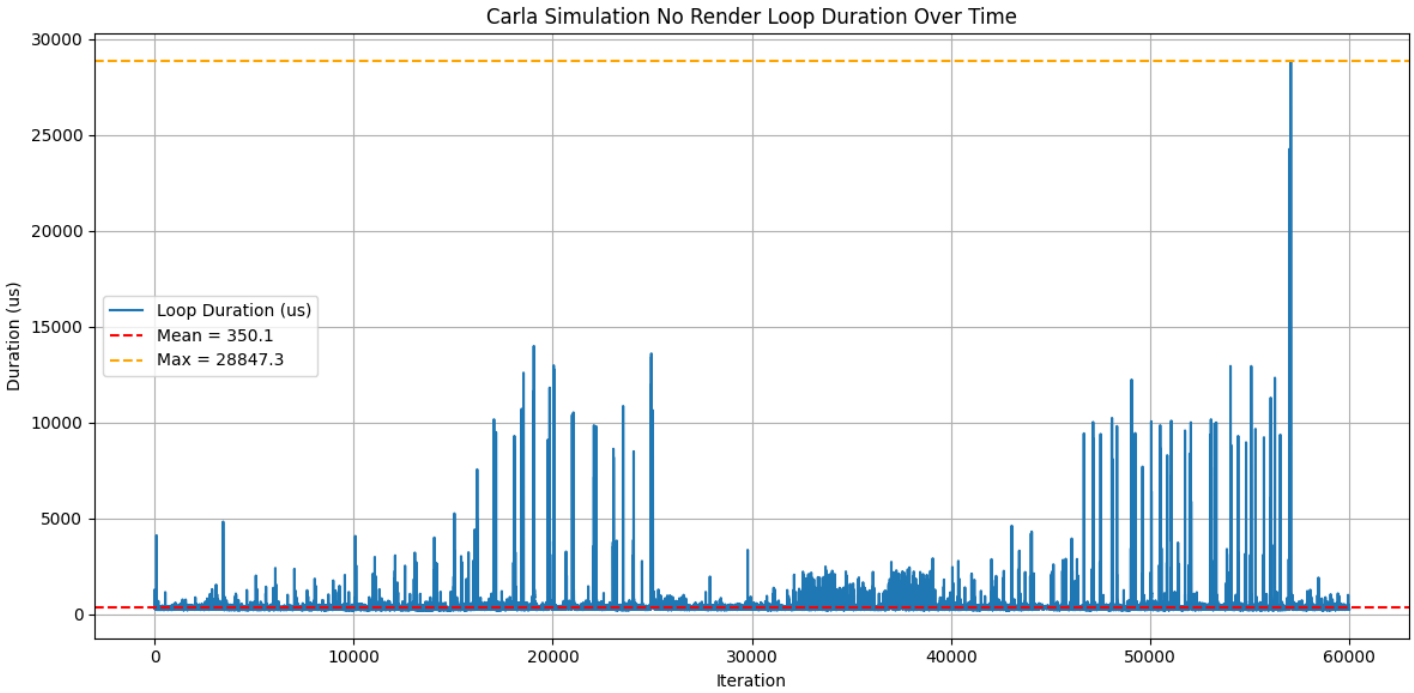
抖动情况：

主体分布在 15–20 ms 区间

尖峰（30–60 ms），说明 GPU 渲染和同步开销导致不稳定。

在渲染模式下就算没挂任何传感器，窗口中有默认场景，UE的整套渲染管线也在正常跑。主要耗时来自于渲染本身。

无渲染模式



横坐标(X 轴)= 迭代次数（0–60000）。

纵坐标 (Y 轴) = 每次 `world.tick()` 的墙钟耗时（单位：微秒 µs）

均值 (Mean)：≈ **0.35 ms**（350 µs）

tick 极快，远小于 10 ms 仿真步长 → 可以 **比实时快几十倍**。

最大值 (Max)：≈ **28.8 ms**

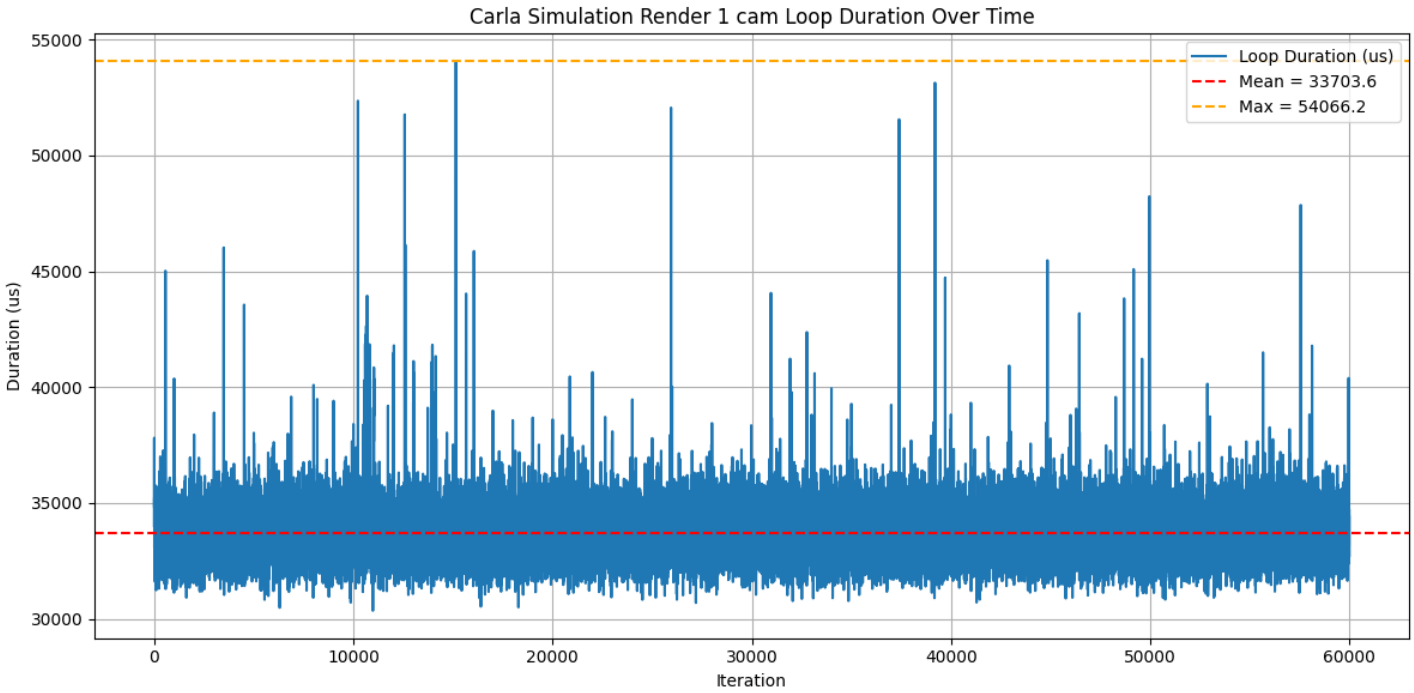
仍有少量尖峰，但比渲染状态的 64 ms 小得多。

抖动情况：

大部分迭代稳定在 < 1 ms

零星出现 5–20 ms 的波动（可能是 Python GC、OS 调度或者少量内部任务）。

桌面渲染1路camera



横坐标(X 轴)= 迭代次数（0–60000）。

纵坐标 (Y 轴) = 每次 `world.tick()` 的墙钟耗时（单位：微秒 µs）

均值 ≈ 33.7 ms（33703 µs）

最大 ≈ 54.1 ms

主体分布在 **31–37 ms**，伴随不少 **40–54 ms** 的尖峰

多了一次 SceneCapture 渲染通道，成本接近再加一帧。