
Unity®
10w量级
粒子的模拟与渲染
2023

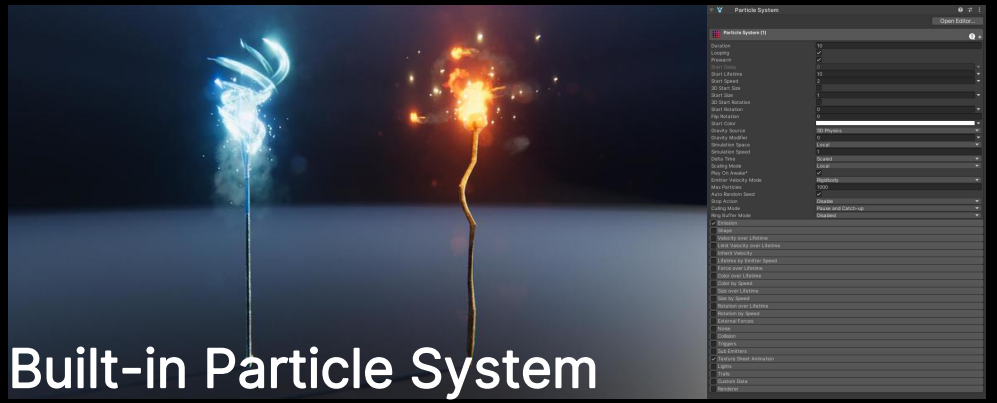
Particle systems

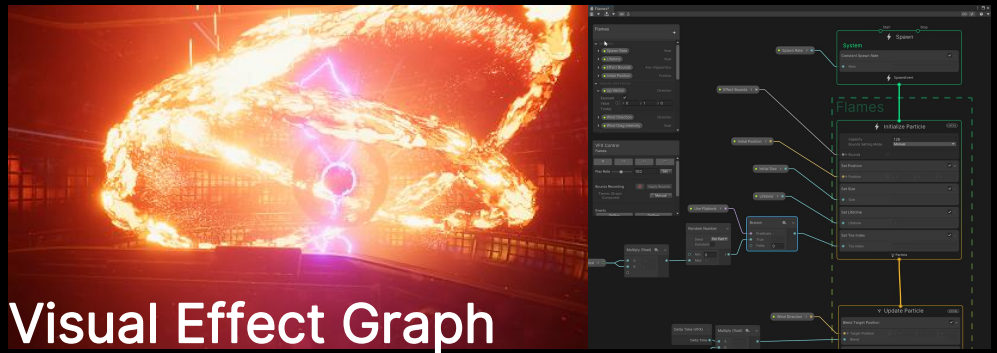
→ 数据量大

→ 相似的简单对象

→ 高频率的对象产生和销毁

→ 大量相同半透明物体渲染





Goal

→ Unity内置粒子系统

→ 广泛应用

→ 社区活跃
- Assetstore约4000款资源

→ 历史悠久unity5.x时代（近10年历史）

→ 基于OpenGL ES 2.0

→ 单线程模型
- 粒子系统内
- 1W粒子量级

Goal

→ 重写
- 功能相似
- 表现更好

→ 性能
- 1W vs 10W


Simulation

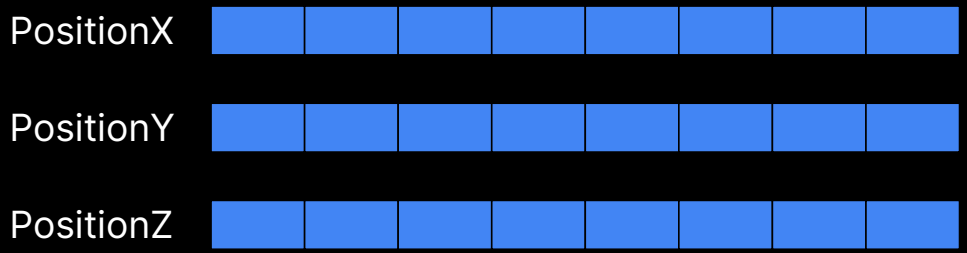
Simulation

→ 10W量级粒子

→ 量变产生质变
- 粒子的生成与销毁
- 曲线采样

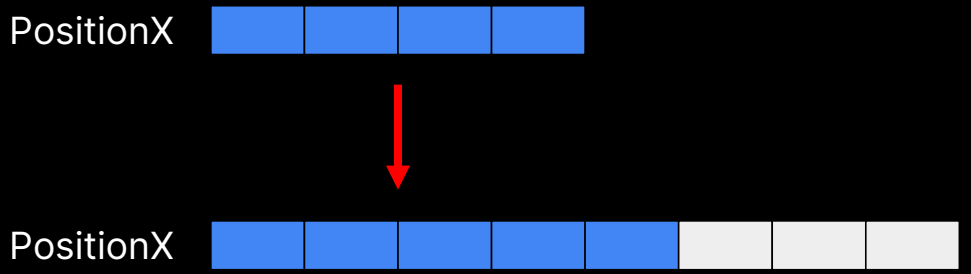
Built-in Simulation

- → 数组
- → RingBuffer



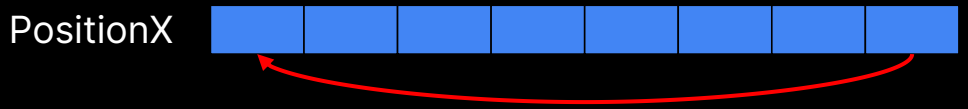
Built-in Simulation

- → 创建
- → 数组
  - 固定容量
  - 到达上限需要进行拷贝
  - 当粒子数量越来越多时，拷贝的开销线性增加



Built-in Simulation

- → 创建
- → RingBuffer
- 到达上限直接复用前面的



Built-in Simulation

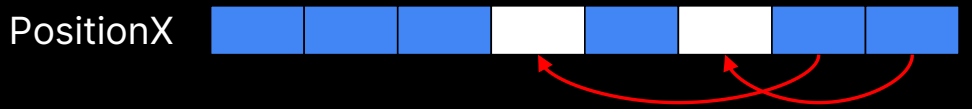
→ 销毁

→ 数组

- Swapback
- 单线程，开销随着销毁粒子数量线性增长

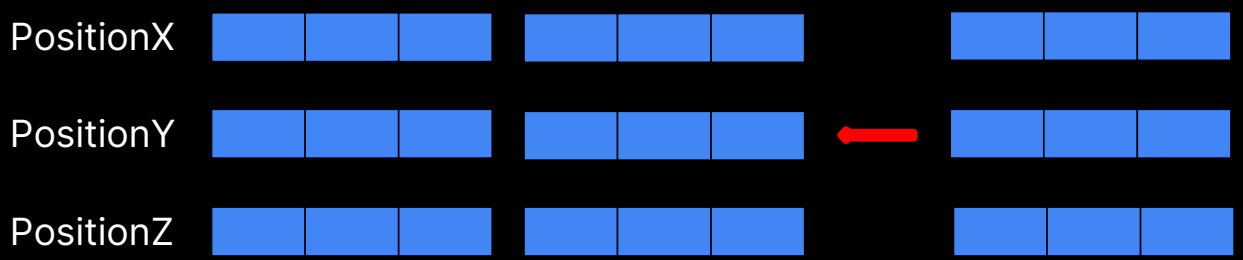
→ RingBuffer

- 躺平
- Pause
- Loop



Simulation

- → 创建
- 分治
- 128个粒子为单位
- 10W = ~780组



Simulation

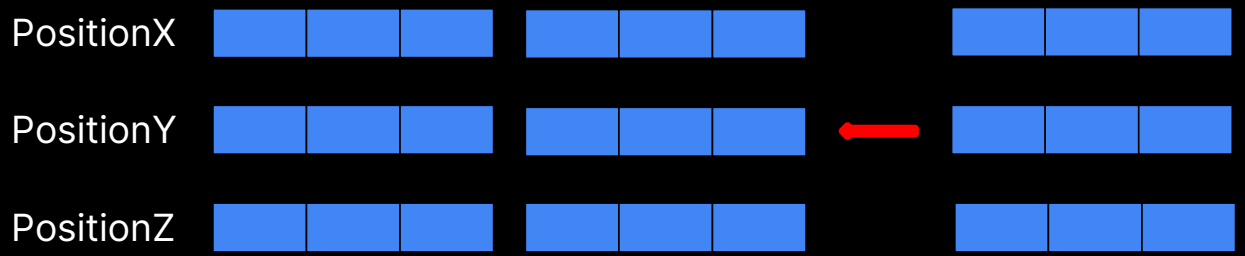
→ 创建

- 代价
- 更复杂的代码
- 对内存管理要求更高

Simulation

→ 内存管理

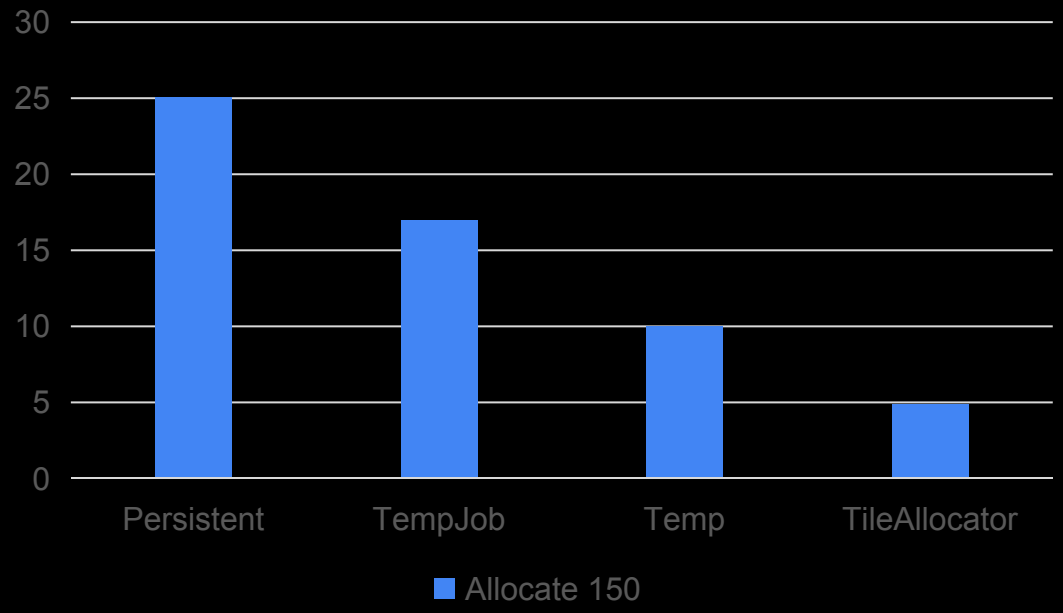
- 128个粒子
- SOA
- PositionX[128]
- PositionY[128]
- PositionZ[128]



Simulation

→ 内存管理

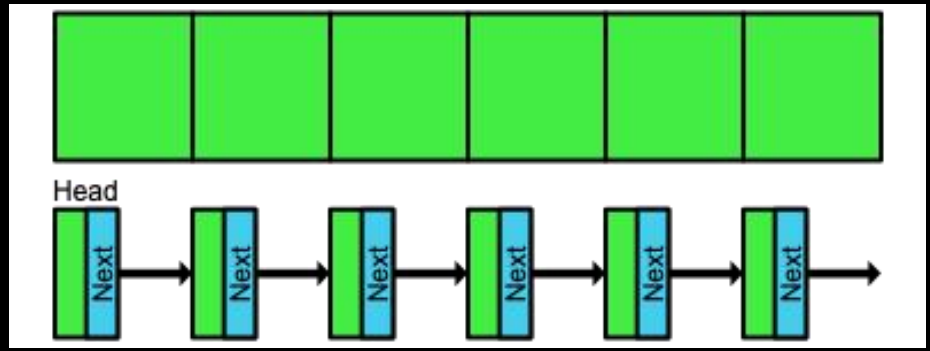
- 自定义Allocator



Simulation

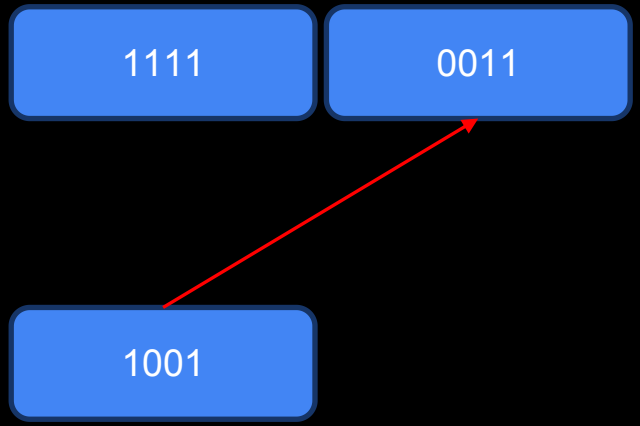
→ 内存管理

- 自定义Pool Allocator



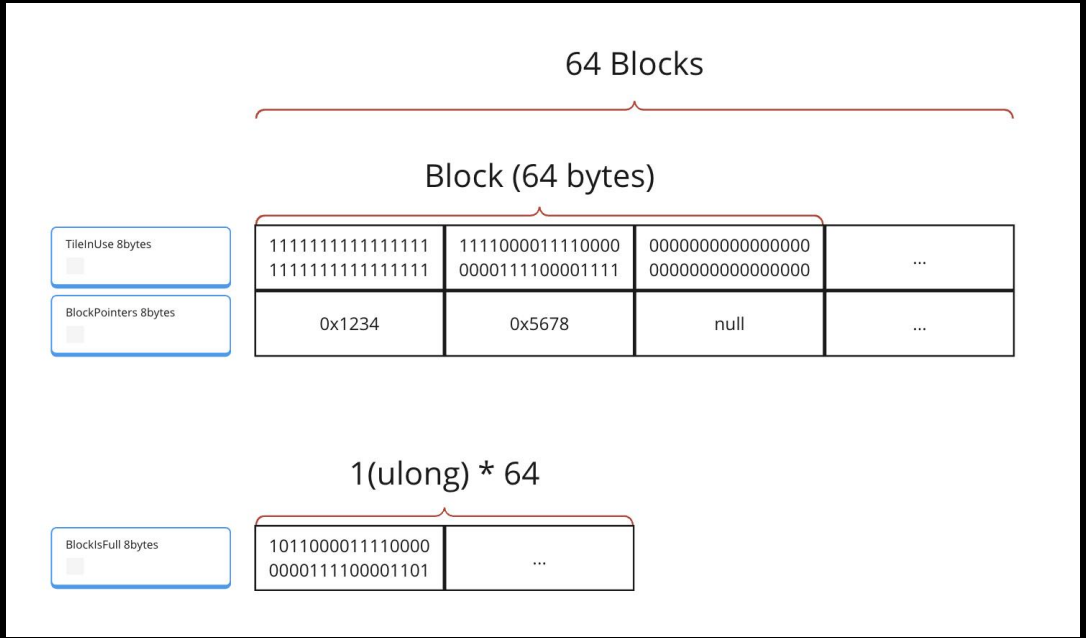
Simulation

- → 内存管理
- 自定义Allocator
- 两层bitmap
- 第一层查询是否有空闲Block
- 第二层查询Block内哪一段可用



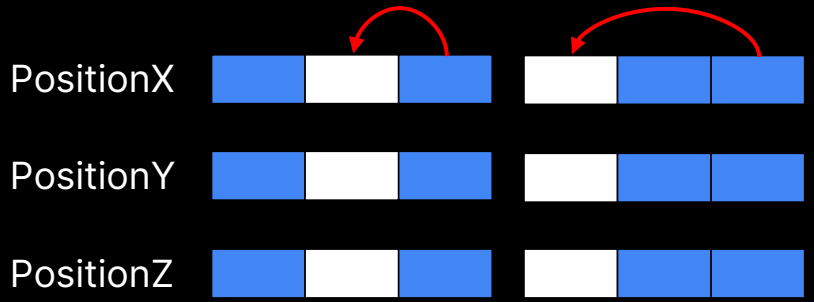
Simulation

- → 内存管理
- 自定义Allocator
- math.tzcnt()
- math.countbits()
- Cache line对齐
- 自旋锁+Interlocked



Simulation

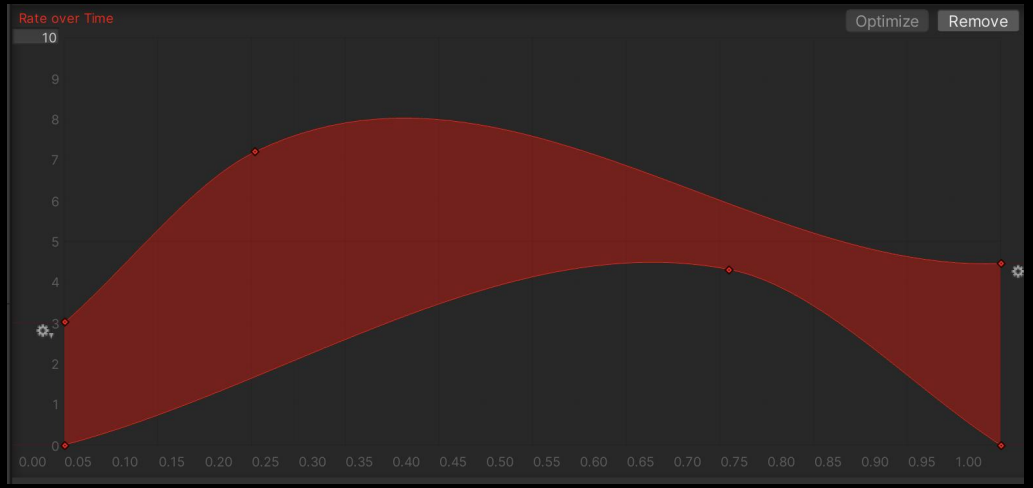
- → 销毁
- → 多线程swapback



Simulation

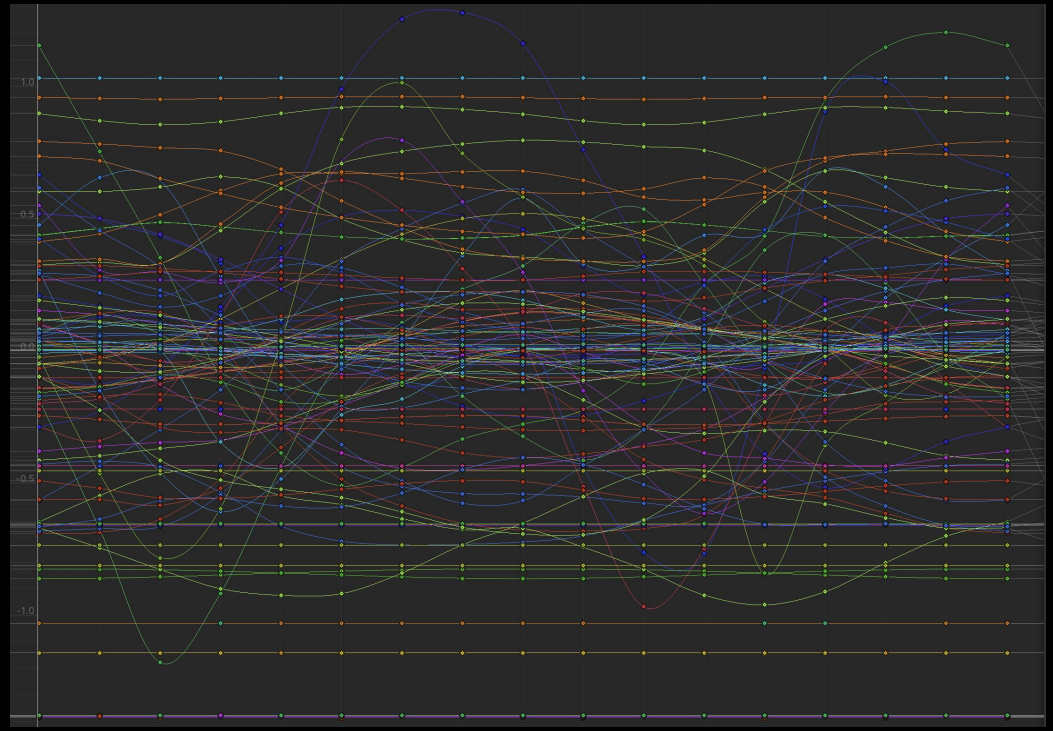
→ 曲线采样

- 大量的曲线采样计算
- Key比较少
- 采样频率很高
- Unity提供了Optimize选项
- 限定[0-1], 必须是clamp模式
- 否则降级到AnimationCurve



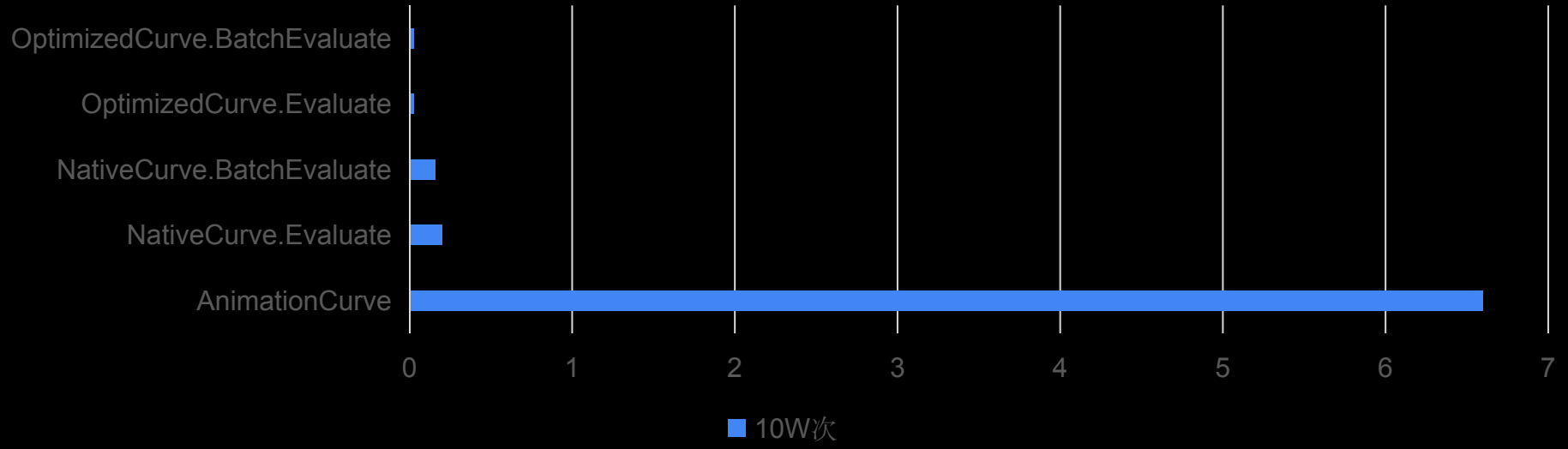
Simulation

- → 曲线采样
- → AnimationCurve
- 曲线很多
- key也很多
- 共享Segments



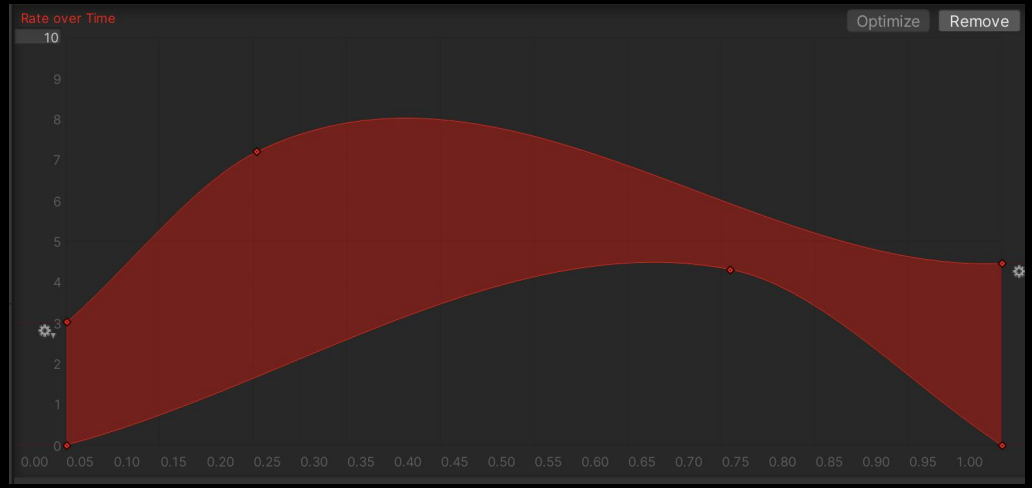
Simulation

- → 曲线采样
- → 为粒子系统重新实现曲线采样



Simulation

- → 曲线采样
- → 思路
- → 本质是针对曲线的每一段进行求解多项式
- → $F_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$



Simulation

- 曲线采样
- 思路
- 预计算a,b,c,d
- 采样
- $ax^3+bx^2+cx+d = [(ax + b)x + c]x + d$

```
math.dot(x: m_Factors, y: timeSerial);
```

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
6 usages  Zhongyuan Li
public float Sample(float time)
{
    return mad(a: mad(a: mad(a: m_Factors.x, b: time, c: m_Factors.y), b: time, c: m_Factors.z), b: time, c: m_Factors.w);
}
```

3X faster!

Simulation

→ 曲线采样

- Tradeoff
- 预计算导致curve创建开销比较高
- 3000条曲线创建2.57ms


Rendering

Built-in Particle System : 50wCube

- Core i5-13600KF
- 4070
- Direct3D 12

- 11 Modules
- Lit

Built-in Particle System 的渲染方案

- → 生成 Mesh
- → 单线程

新的渲染方案

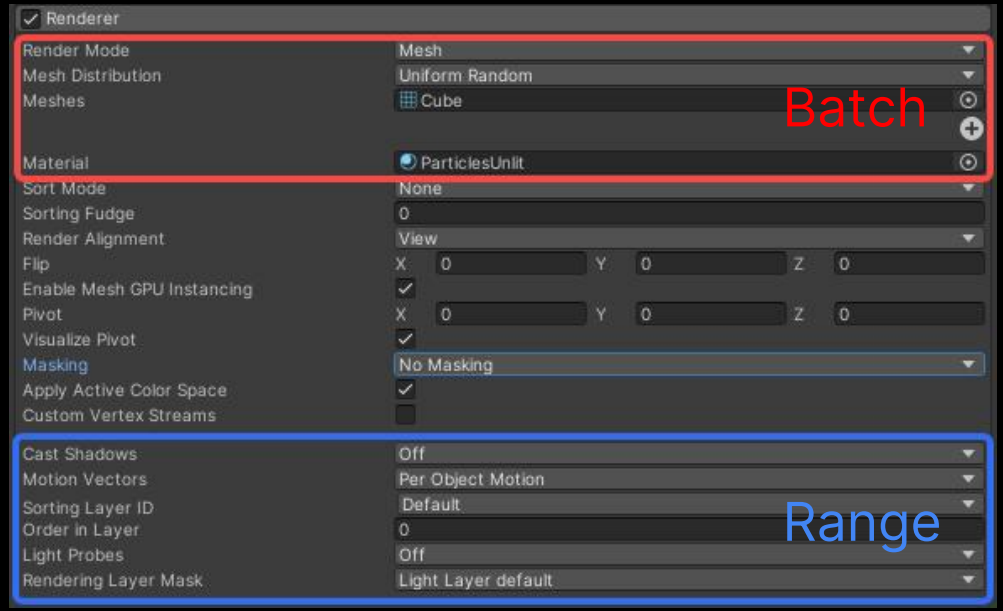
- → 基于BatchRendererGroup
- → RendererManager
  - — 组织场景内所有 Particles 和 Trails
  - — Cpu to Gpu 数据上传
  - — Culling
  - — 组织 DrawCommand

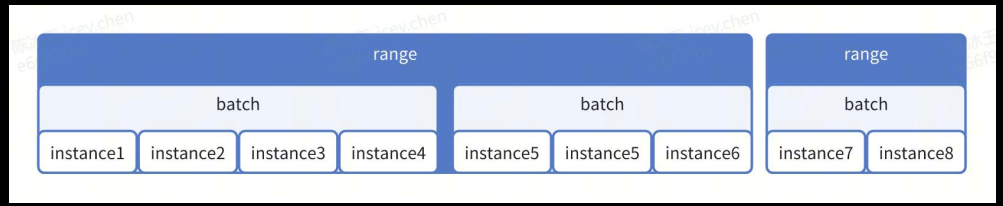
粒子系统的合批策略

→ 影响合批的选项
- Mesh
- Material
- Filter Setting

→ 一个粒子系统的所有粒子
- Same Range
- Batch：1-4

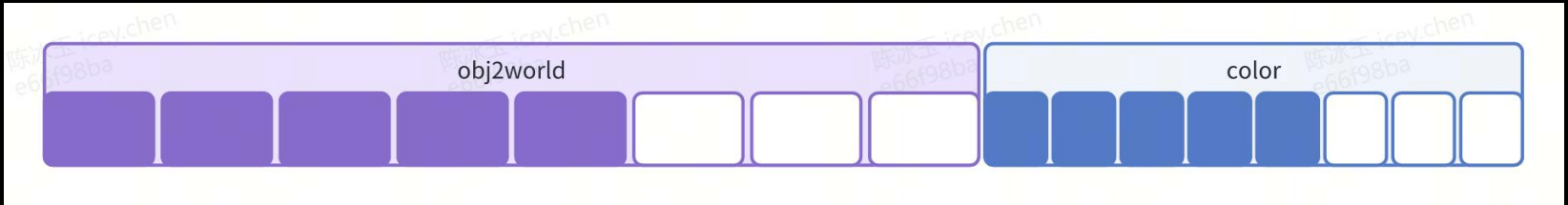
→ 参数确定最佳批次





GPU内存管理

- SOA组织数据
- 持久的 GPU 内存
  - capacity：粒子数量上限
  - update：实际的粒子数量



减少上传数据量

→ 区分Shared数据和Instance数据

→ 灵活修改上传数据的内存布局

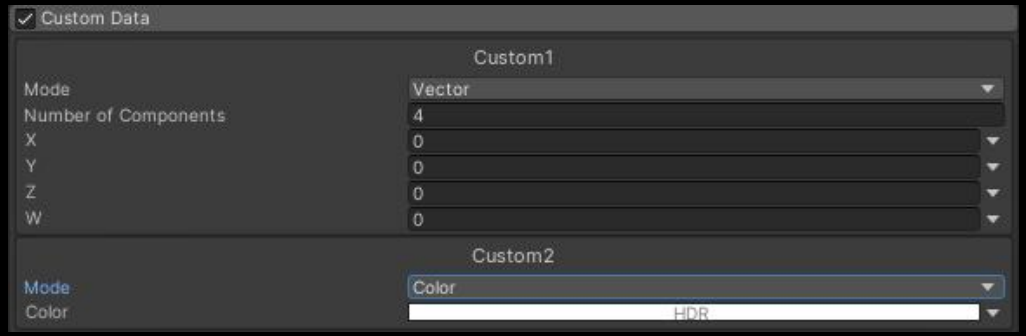
部分模拟转移到GPU

→ 矩阵变换、UV动画、Random

→ 加速计算

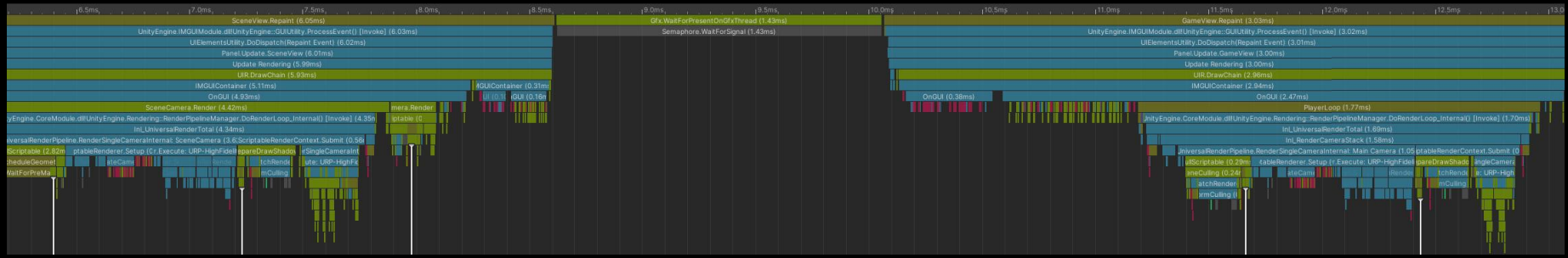
— 通常粒子数量远大于顶点数量

→ 用更少的数据描述更多的属性



CPU to GPU

- → 计算结果直接写入GPU
- → 多线程并行写入
- → 写入时机优化



相比Built-in Particle System省了多少?

- → Shared数据量级：40w vs 1
- → Instance数据量级：40w vs 10w
- → Instance数据类型减少 40%

- → 启用阴影时，数据总量减少 50%

Culling

- per particle systems / per particle
  - 1 / 10w

- 一个粒子一帧内需要和多少个裁剪面求交
  - camera: 6
  - SpotLight: 7-9
  - PointLight: 42-54
  - DirectionalLight: 6-48
  - SceneView+GameView: *viewNum

Culling加速

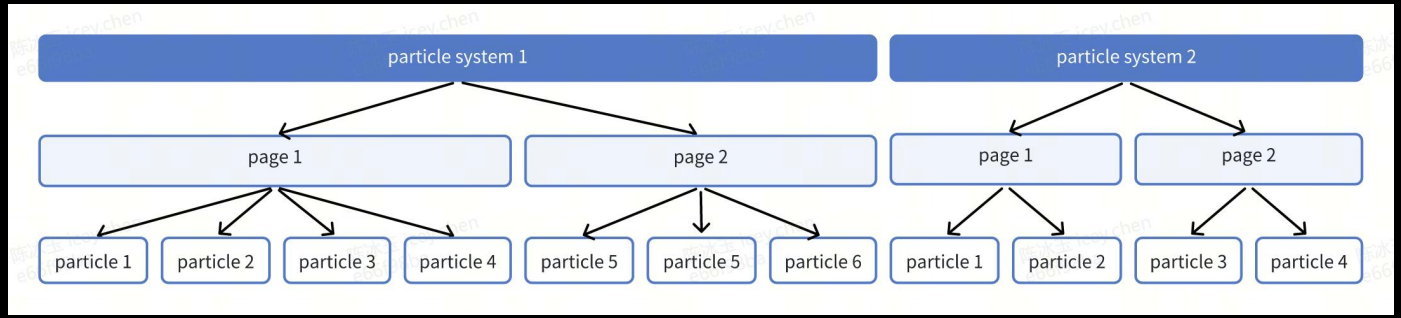
→ SIMD

```
public struct PlanePacket4
{
    public float4 Xs;  Serializable
    public float4 Ys;  Serializable
    public float4 Zs;  Serializable
    public float4 Distances;  Serializable
}
```

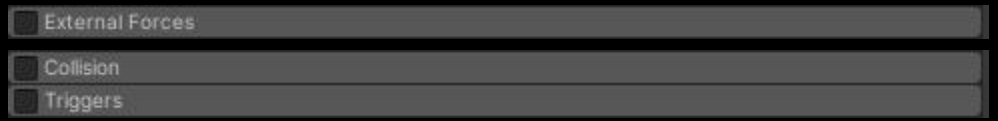
→ 多线程

→ 层次结构

- particle systems
- page
- particle



→ other module



Rewrite : 50wCube

- Core i5-13600KF
- 4070
- Direct3D 12

- 11 Modules
- Lit


Unity®
The world is a better place with more creators.
世 界 会 因 有 更 多 创 作 者 而 变 得 更 美 好

Unity®

2023

Unity官方B站

Unity官方
开发者服务平台

Connect App

Thank you

https://developer.unity.cn/