(剑三)性能管理器

一个资源受限平台上的性能自平衡系统

副标题

一个资源受限平台上的性能自平衡系统

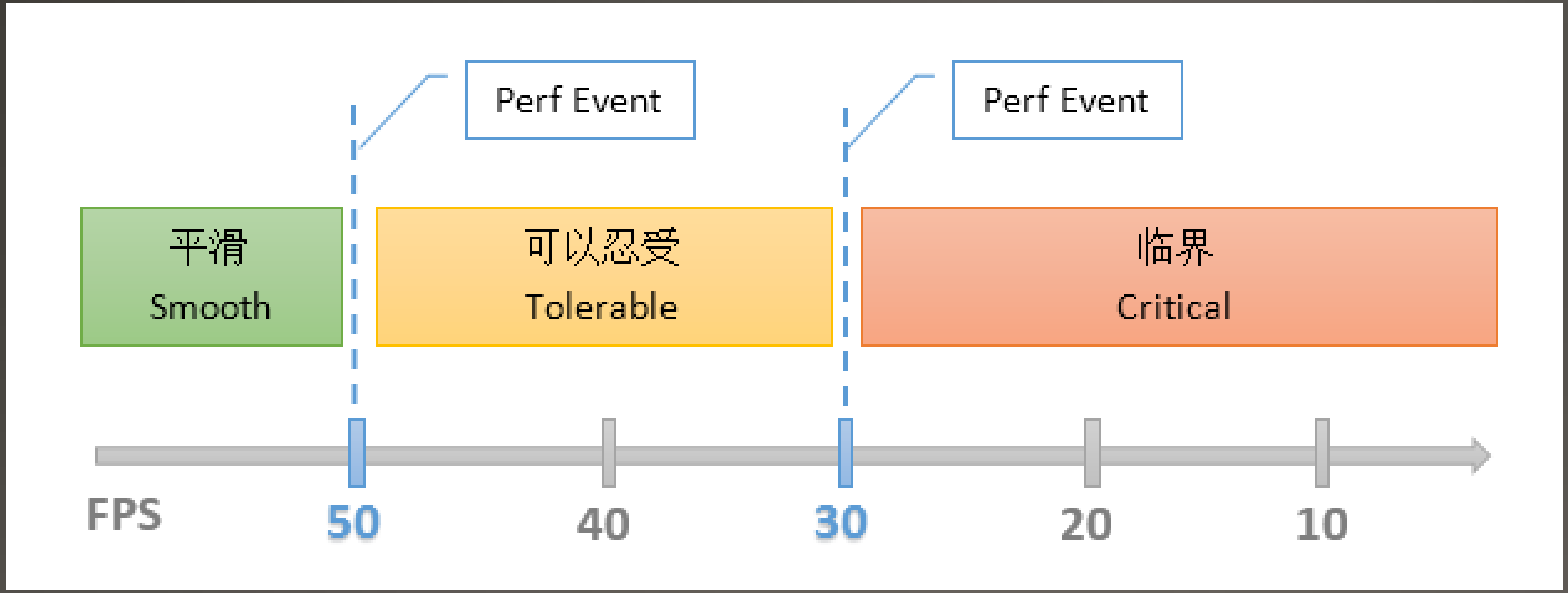
自平衡

- 自平衡 = 自适应 + 负反馈
- 自适应：当性能指标变化时，调整系统的整体负载来适应该变化，以免进一步恶化
- 负反馈：系统输出反过来影响系统输入，循环往复，使输出与目标的误差减小，系统趋于稳定
- 自平衡：两者的结合，采用各种手段，调动各个模块，提高并稳定系统的性能表现。

具体做法

检测系统当前运行帧数(fps)，根据系统当前表现，动态做出反应，调整各个子系统的负载以降低压力，提高帧数。当系统恢复到健康平滑状态时，将负载逐步还给系统。

性能分档


Perf Event
Perf Event
平滑
Smooth
可以忍受
Tolerable
临界
Critical
FPS
50
40
30
20
10

事件响应

```
interface IPerfEventHandler
{
    ...
    virtual void OnPerfLevelChanged(ePerfLevel newLevel, ePerfLevel oldLevel) = 0;
    virtual void OnPerformanceWarning(ePerfLevel newLevel, double fTime) = 0;
    ...
};
```

任意子系统，继承此类并改写这两个函数，就可以对系统的性能变化做出反应。

警告 OnPerformanceWarning()

- 什么是警告?
  - 程序判断出就快要跌到下一档之前，预先以每3秒一次的频率向整个系统发出警告，这时，如果各个子系统积极响应，妥善调整，系统就有很大的机会在进一步的恶化前恢复到之前的状态。
- 抢先干预
  - “从没感觉到卡过”vs“先变卡后恢复了”
- 缓解颠簸

这么做有啥牺牲?

- 玩家视野范围内一些相对不重要，或本来用于丰富场景的东西会被有选择地干掉（或简化）。

- 干掉（及简化）的原则是（重要性依次降低）
  - 当前系统的性能
  - 目标与玩家的距离
  - 目标对性能的影响程度
  - 目标对画面效果的提升程度

这么做有啥好处？

- 平均帧数提高并倾向于稳定在某一档
- 程序在不同地图，不同情境下会跑出不同的参数集，总是倾向于自动地找到当下最优的方案。
  - 换句话说，小白用户不用折腾系统设置了，只要知道系统会替他操心就可以了
- 提高整个系统在性能上的伸缩性
  - 通过响应 OnPerfEvent() 和 OnPerfWarning()，优化的职责分担到了开发具体模块的同事

应用场景

- 角色
- 裁剪
- 物件

角色性能控制第一版(v1)实现

- 分页系统(Page In / Page Out)
- 优先级缓存队列
- 急进缓出(Greedy In / Gentle Out)

第一版(V1)问题与思考

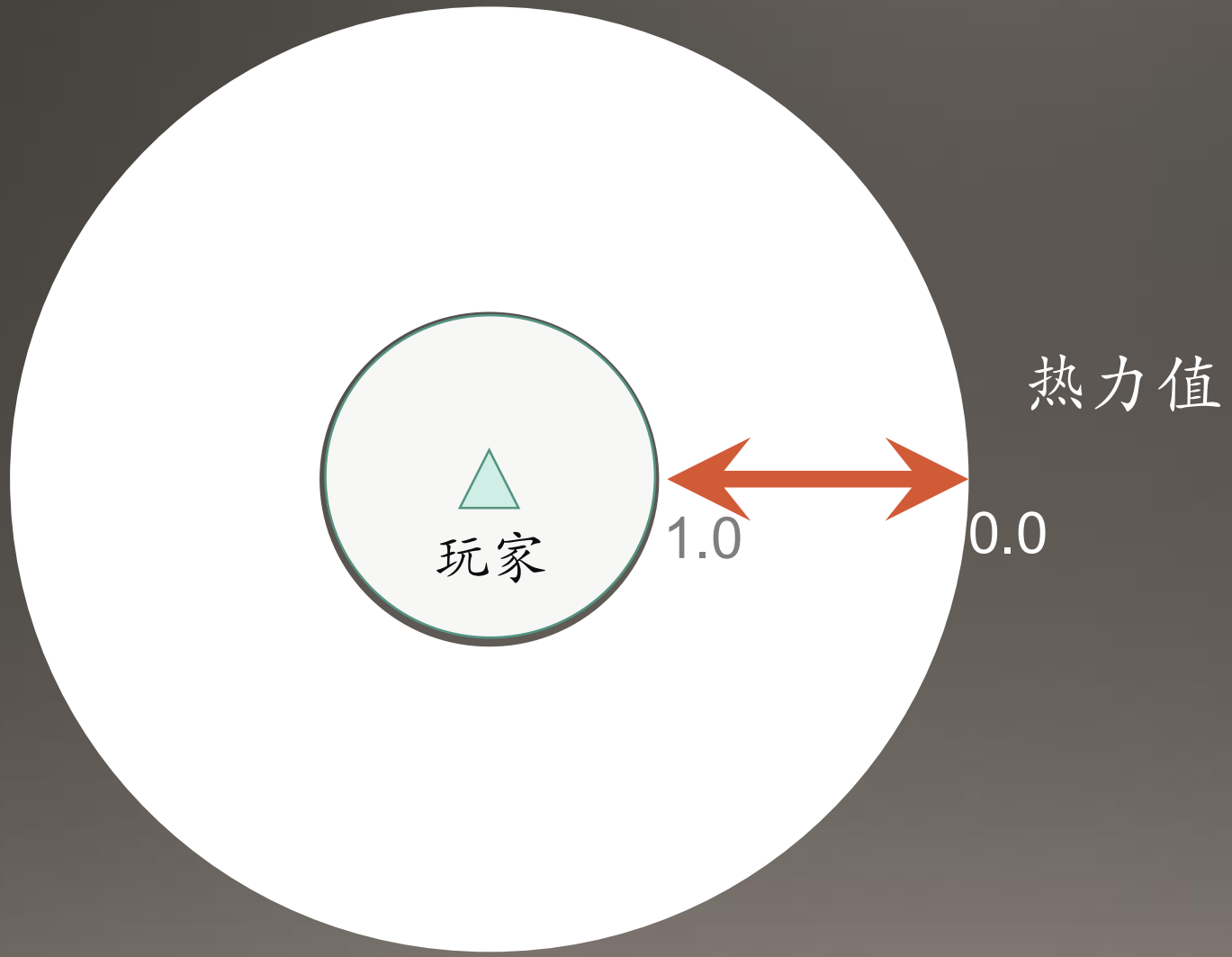
1. 非黑即白，缺乏更细粒度
2. 性能控制和游戏逻辑互相影响，不够正交
3. 行为不够直观，出bug不好调试

- 结论：根本性问题，无法修修补补解决

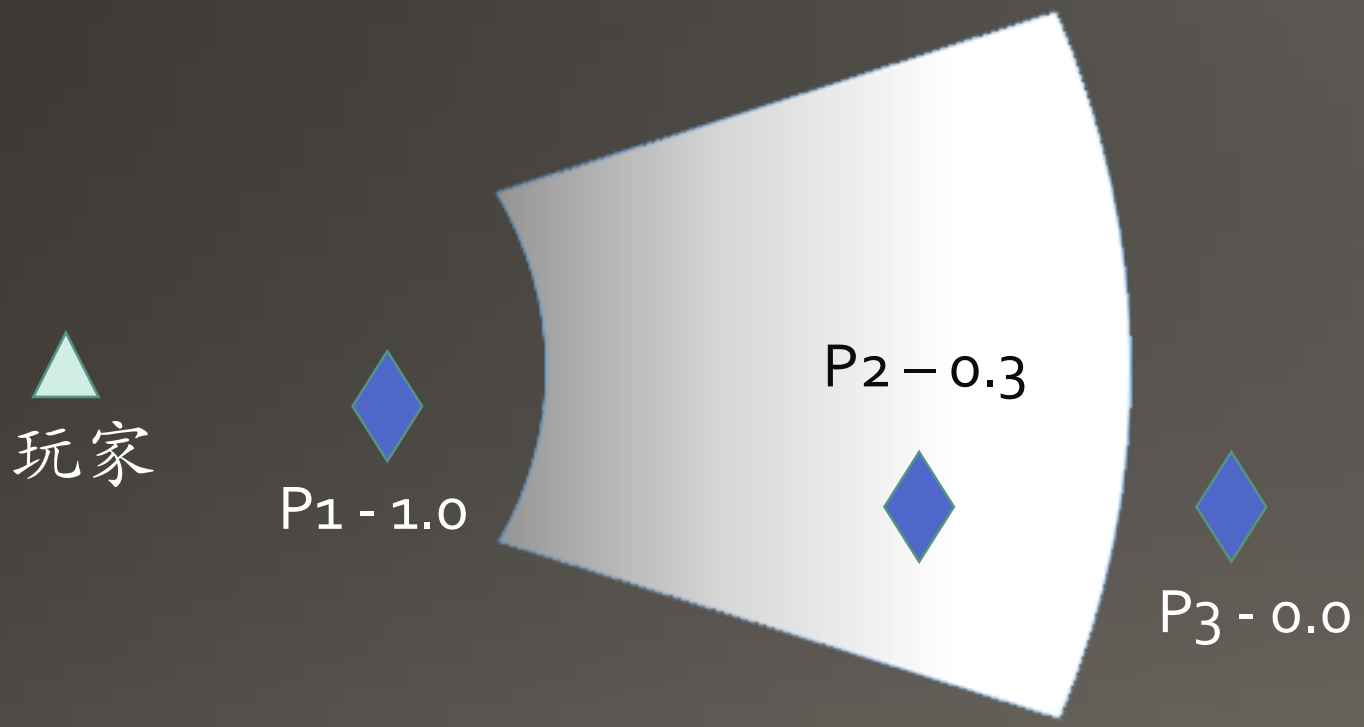
第二版(V2)实现

- 如何在解决这些根本性问题的同时，保留好处呢?
- 答案：“热度环”系统

“热度环”

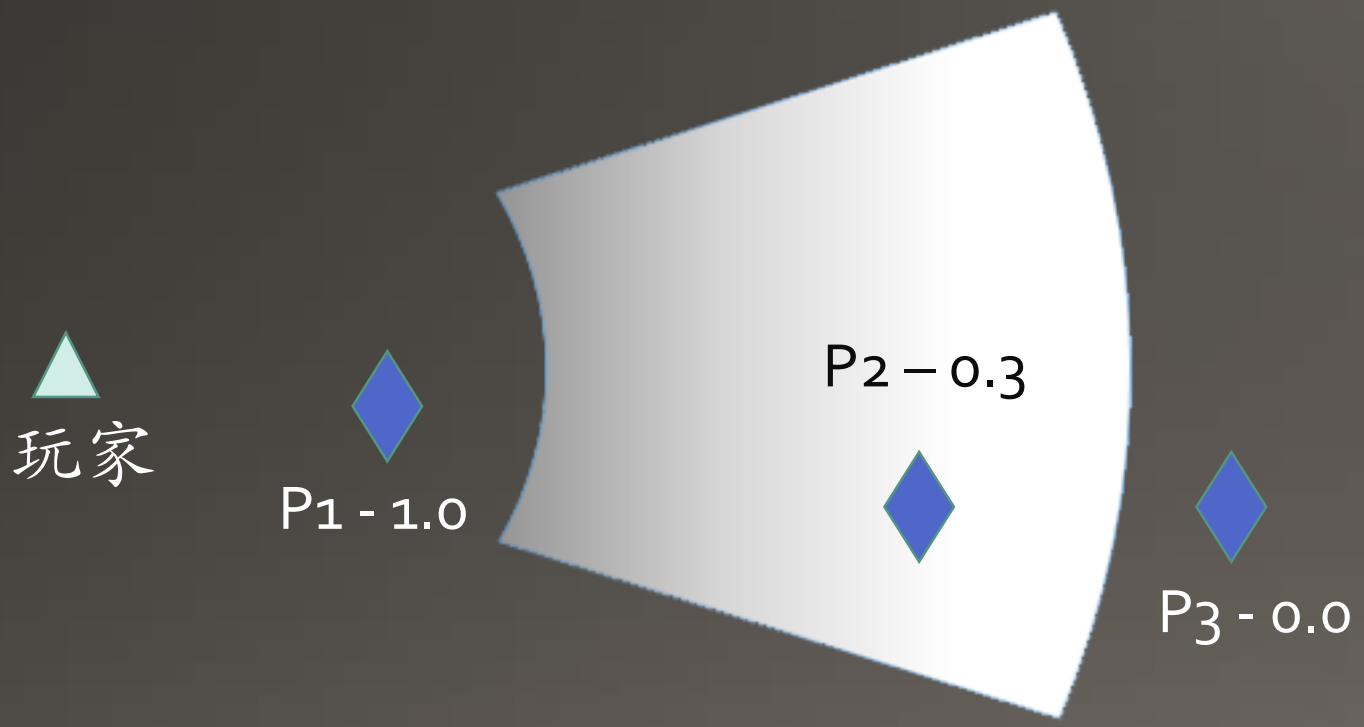


热度环的实际应用



- 本体模型
- PhysX驱动的装备
- 装备武器
- 装备武器特效
- 本体特效
- 姓名板，气泡，任务图标等

热度环在系统架构上的关键作用



1. “怎么来的”与“怎么用的”隔离
2. 各自独立演化

- 本体模型
- PhysX驱动的装备
- 装备武器
- 装备武器特效
- 本体特效
- 姓名板，气泡，任务图标等

Remote Character 上的新增标记

```
11 enum ePerfStatusFlag
12 {
13     PerfStatus_Nothing          = 0,
14     PerfStatus_ShowBody         = 1,            // 本体，包括脸，头发，手，腿，身体等部分
15     PerfStatus_ShowBodySFX      = 1 << 1,       // 脚印等特效
16     PerfStatus_ShowEquips       = 1 << 2,       // 挂载类装备（武器等）
17     PerfStatus_ShowEquipsSFX    = 1 << 3,       // 挂载类装备的特效
18     PerfStatus_ShowSocketSFX    = 1 << 4,       // 刀光
19     PerfStatus_Full             = 0xffffffff,
20 };
```

```
一些约定：
1.  比现有的游戏逻辑内对应的设置（如 m_eShowLevel）的优先级要低。也就是说，只有在游戏里被确认开启的情况下，
    才会进一步考虑这里的设置。这样的物理隔离是为了从根源上避免与已有逻辑互相影响。
2.  这些设置应该是非侵入的，只影响视觉表现，不应修改RL内各种对象之间的逻辑关系，以免破坏已有的游戏逻辑的假定。
    比如是否使用空模型（Empty Model），是否有 Attach / Bind 的关系，等等。
*/
DWORD m_dwPerfStatus;
```

(v2)对(v1)各种问题的解决情况

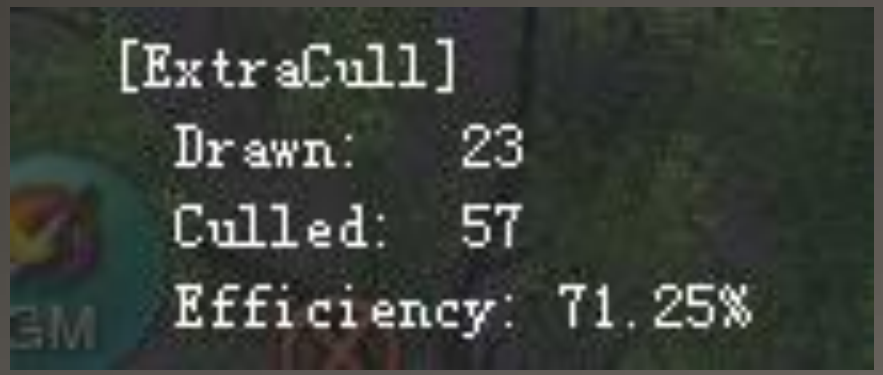
- 重新设计的热度环解决了“缺乏细粒度控制”的问题，同时改善了系统架构。
- 单独的性能标记解决了“性能控制跟游戏逻辑控制相互干扰”的问题
- 核心代码不到30行，大幅简化了原有系统。只要打印出任意角色的热力值即可随时监视，调试无压力。

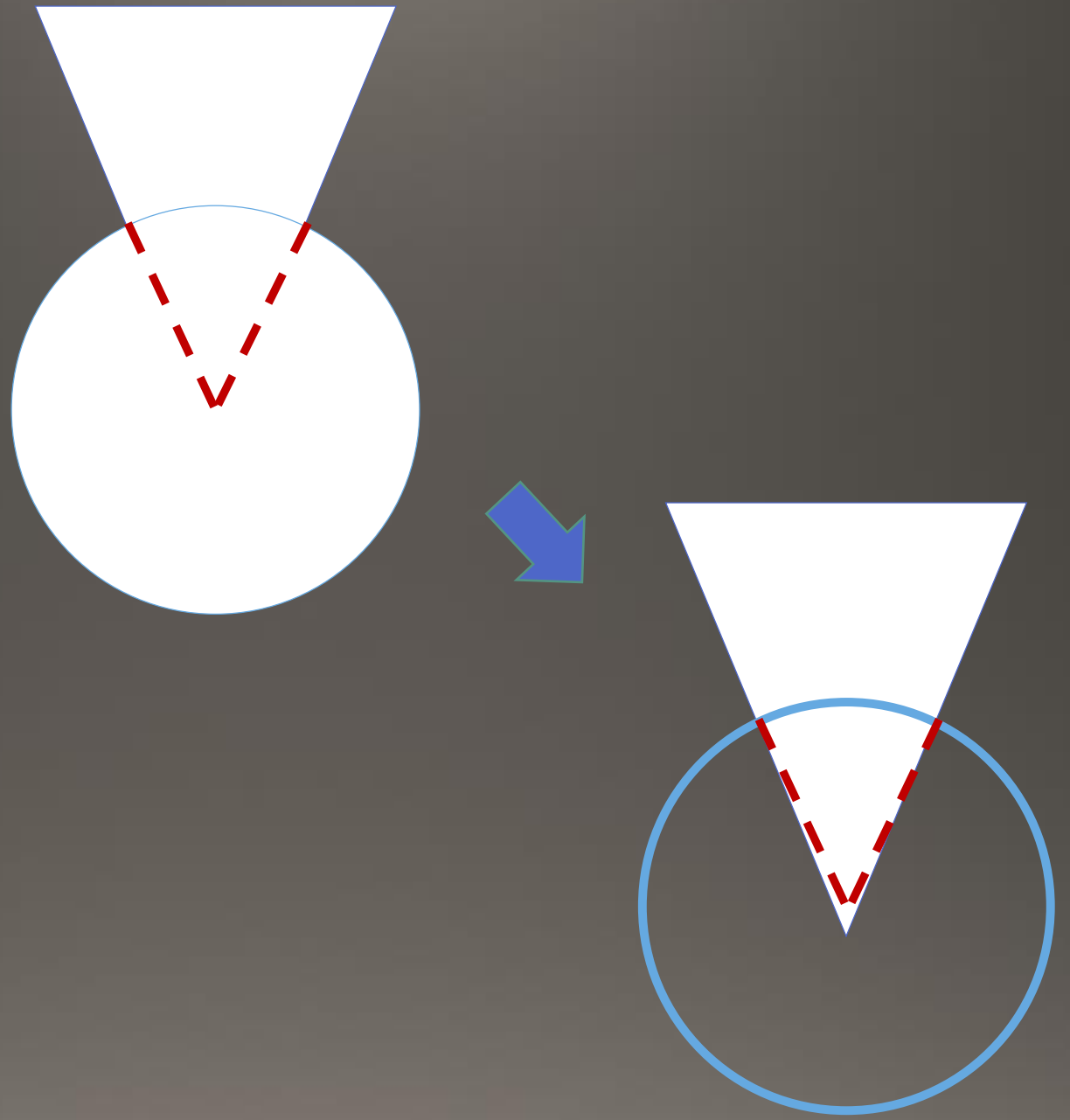
应用场景

- 角色
- 裁剪
- 物件

裁剪小插曲

- 二次裁剪效率
  - 典型场景 50% - 80%





应用场景

- 角色
- 裁剪
- 物件

物件系统优化

- 为什么是物件系统?

- 跟其它子系统相比，物件系统
  - drawcall 偏多
  - 状态切换多
  - 同屏贴图和材质总量较大

物件系统优化（运行时分析）

- X 轴为距离玩家的距离
- Y 轴为物件尺寸
- 颜色用来标识
  - Model Type
  - Runtime Shader Type
  - 美术资源包的信息
  - 被过滤掉的物体（红色）

实际过滤范围（分段）

图中的折线为边界
下方的物体（所有标为红色的点）都被过滤

- 距离从30开始
- 分段的目的
- 150外的处理

(播放视频)

运行时分析，按美术资源包分组

资源分组内显示的具体信息

1. 颜色区分
2. 资源包路径
3. 可视数量

美术辅助

- 沙盘设计帮助
  - 通过运行此分析功能，可直观地看到一张地图上，实际运行时，不同区域内的资源使用情况，对整个场景做出更合理的规划

- 后期地图优化
  - 跑地图时，更有效地找到热点
  - 注意控制资源的局部性（不同贴图/材质的资源在区域内的混杂程度）

改进空间

- 考虑远裁剪面，雾效和环境光
  - 远裁剪面越近，雾效衰减越快，环境光约弱，过滤机制就可以越激进
- 更细的粒度
  - 简化shader而不是直接干掉

谢谢

问题？

西山居技术中心，顾露
gulu@kingsoft.com