Unity Optimization

- 1 脚本 策略
  - 1.Cache Component reference
  - 2.仔细选择获取Component的方法
    - 4.x GetComponent(typeof())
    - 5.x GetComponent(typeof())和GetComponent<T>()没有区别
  - 3.Removing empty callback declarations
    - Starup和Update,这些函数如果不用就不要声明
  - 4.避免使用Find和SendMessage函数
  - 5.避免从GameObject获取String
  - 6.Cache Transform Changes
    - only commit change to transform in FixedUpdate for once during one whole frame
  - 7.Fast GameObject null check
    - System Object.ReferenceEquals(gameObject,null)
- 2 合并批次
  - 1.Draw Call
    - 准备资源
    - 设置Render状态
    - CPU Bound操作
  - 2.Dynamic Batching
    - 条件
      - 所有的物理使用相同的材质
      - 只能Batch粒子和Mesh,其他的如蒙皮网络不能Batch
      - Shader的Mertex Attribute的限制.每次Batch只能使用900哥,也就是说,过多的顶点个数,或者顶点的属性太多,都会导致不能Batch
      - 所有的Mesh使用相同的UniformScale,或者每个Mesh单独使用一个唯一的Scale
        - 使用相同的Uniform Scale,Unity所有的版本都会Batch
        - 不同的Scale,Unity 5可以Batch,但是Unity 4不可以
      - 使用相同的光照贴图
      - 不能接受实时阴影
      - 不能使用多个Pass的Shader
      - 没有文档,但事实上确实是
        - 不能操作300个顶点
          - 上面操作了900,也就是一个顶点,UV,顶点发现
          - 不能超过32000个顶点Index
  - Static Batching
    - 构建一个大的Mesh,存入所有的顶点
    - Static flag
    - 消耗多余的内存
      - 同一个Mesh渲染多次的情况,会消耗n倍的内存
    - 使用相同的材质
    - 缺点
      - 无法移动中间某一个物体
      - 某一个物体可见,会导致整个Mesh被渲染
- 3 美术资源管理
  - 1.Audio
  - 2.Texture
    - 压缩格式
    - 纹理效率增强
      - 纹理文件大小
      - MipMap
      - 使用外部工具来调整纹理分辨率
      - 调整Anisotrpic
      - 图片集使用
      - 非正方形的纹理的压缩
  - 3.Mesh And Animation
    - 减少多边形数量
      - 纹理压缩
    - Read/Write enable
    - 谨慎的选择顶点数据格式
    - 考虑顶点动画
      - 顶点动画只记录每帧之间位置变化的顶点位置
      - 如果顶点数量不多,可以考虑
    - Optimize Meshes Option
      - 重组顶点数据格式
      - 重组渲染格式
      - 如果不是有明显的副作用,大多数是有应该使用
  - 4.合并Mesh
    - 减少DC
    - 优先使用Static Batching
- 4 物理相关计算
- 5 图形相关
  - Brute force testing
    - CPU-bound
      - 1.Disable Vsync in profiler
      - 2.Reduce Draw call
        - 如果不能减少,则尝试禁止那些减少DC的功能,增加DC,看看会不会导致帧率下降
      - 3.Ocdusion Culling
      - 4.Multithread rendering
      - 5.GPU Skinning
  - Front end bottlenecks
    - 简化Mesh
      - LOD
        - 调整Mesh
        - 调整动画
        - 对开过工作流有较大影响
        - 消耗内存
    - Disable GPU Skinning
  - Back end bottlenecks
    - 降低填充率
      - brute force test
        - Reduce resolution
        - Reduce texture quality
      - shader Optimization
        - 如果降低分辨率导致帧率上升,则填充率是平静下来
        - 用Mobile shader
        - 控制浮点数精度
        - 截取数据的时候不要改变精度
        - 熟悉CG函数,尽量使用库函数
        - 删除不需要的特性
        - 控制输入数据大小
        - 减少导出变量的数量
        - 减少数学计算复杂度,少用pow,exp,log等
        - 控制纹理采样次数
        - 控制条件语句
      - Oclussion Culling
    - 内存带宽
      - 减少纹理采样
      - 选择的合适的纹理压缩算法
      - 减少纹理切换
  - 移动设备优化
    - 减少DC
    - 减少材质数量
    - 降低纹理大小和材质数量
    - 使用2的n次方的纹理
    - 在shader内尽可能使用低精度的浮点数
    - 避免AlphaTesting
- 6 内存管理
  - Mono
    - 强制Mono, 一个具体的方法
      - 通过Method的method.MethodHandle.GetFunctionPointer()
      - .Net上RuntimerHelpers.PrepareMethod()在Mono下有Bug
  - 内存使用优化
    - Unity内存划分
      - Unity引擎
      - Mono虚拟机
      - 插件
  - 引用类型和值类型
  - Prefab池
- 7 使用技巧