bilibili 游戏

你的幻想世界

移动平台性能优化

深入理解优化技巧

本次分享核心

美术优化比程序优化见效快
理解优化原理，自己为性能把关
道阻且长,行则将至

Pt. 01 流畅度优化

Pt. 02 续航优化

Pt. 03 内存优化

Pt. 04 其他优化

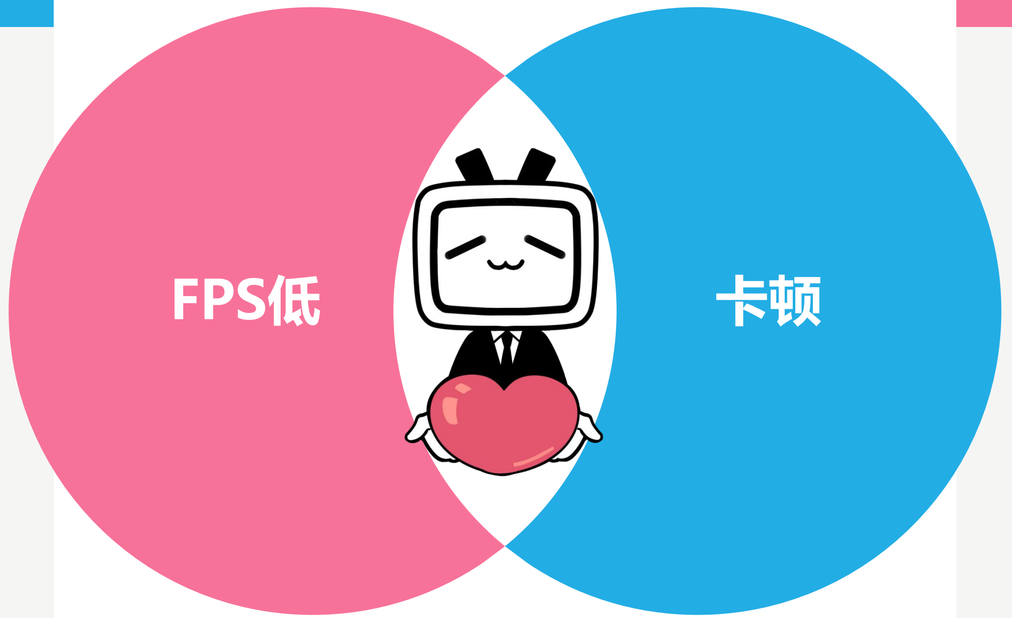
Pt. 1

流畅度优化

影响流畅度的因素

FPS低

每一帧耗时都过长



卡顿

周期性的某一帧耗时特别长

或者连续几帧耗时偏长

FPS低

CPU逻辑处理 + CPU将渲染数据写入显存

GPU从显存读取需要的数据 + GPU图形计算 + GPU将渲染结果写入显存

FPS低

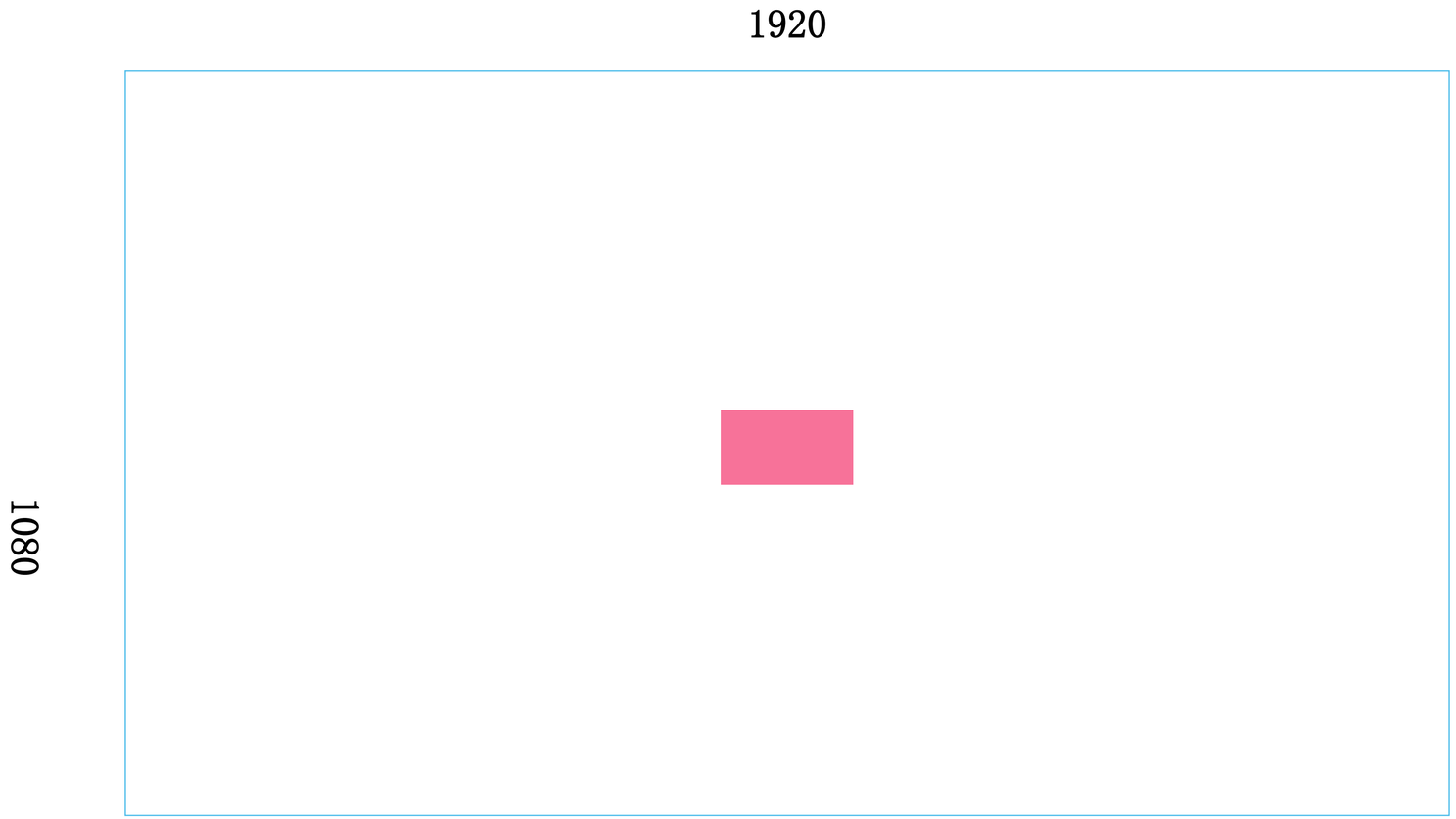
GPU计算瓶颈	CPU计算瓶颈	带宽瓶颈

GPU算力瓶颈

- 顶点处理,像素处理共用计算单元
- Iphone 12 A14 芯片算力:824GFLOPS
- 一个顶点处理:40次浮点运算
- 一个三角面平均2个顶点,80次浮点运算
- 一个像素处理:20-1000次浮点运算,平均估算200次
- 1920*1080分辨率填充一次2,073,600个像素,414,720,000(4亿)次浮点运算
- 60FPS,一帧耗时16ms,A14芯片可以计算13.8G次浮点运算

- Iphone12在1080P,60FPS环境下
- 不考虑顶点计算,一帧可以填充13.8G÷(2073600*200)=33.3个屏幕
- 不考虑像素着色,一帧可以处理13.8G÷(40*2)=172.5M(1.7亿)个三角面
- 填充一个屏幕消耗的像素计算可以处理414,720,000÷80=5,184,000(5百万)个三角面的顶点

GPU算力瓶颈



顶点计算:4*40=160次浮点计算

像素计算:1920*1080*20=41,472,000(4千万)次浮点计算

GPU算力瓶颈

减少面数不是优化重点
减少屏幕上的渲染像素才是关键

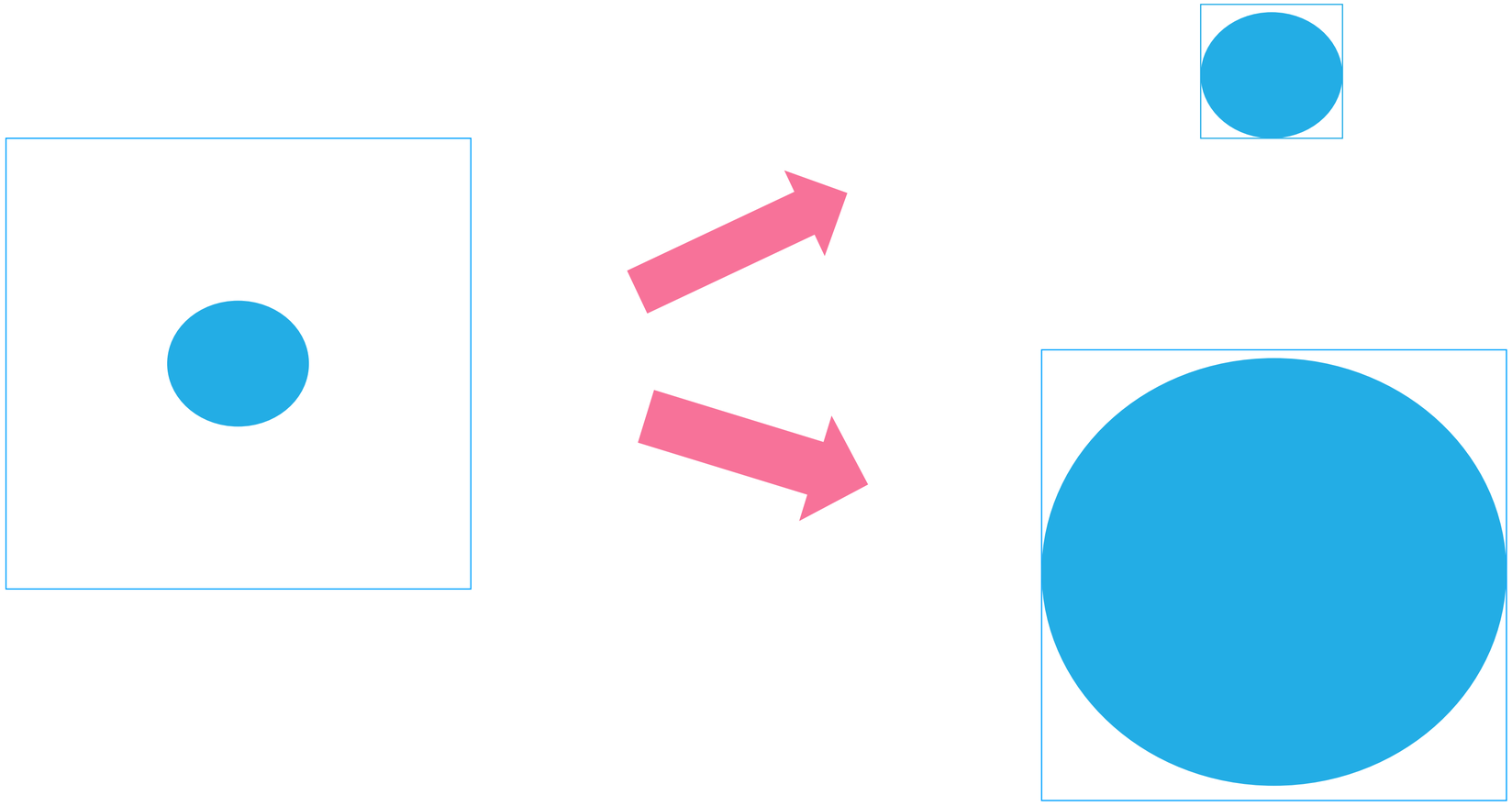
减少像素计算

1. 贴图优化，减少面积

贴图优化

2:1

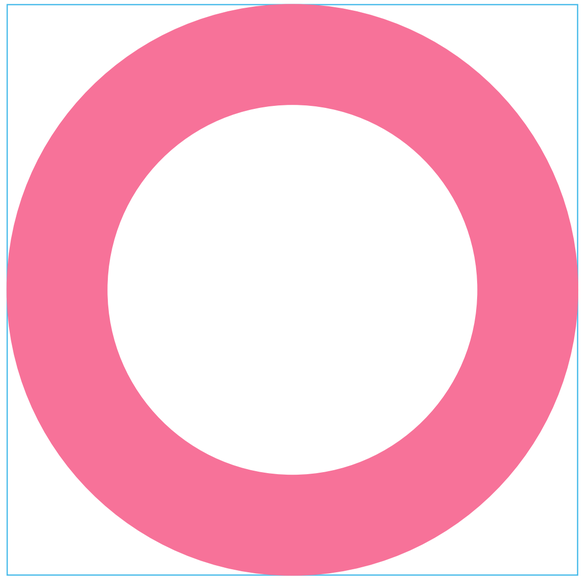
贴图优化

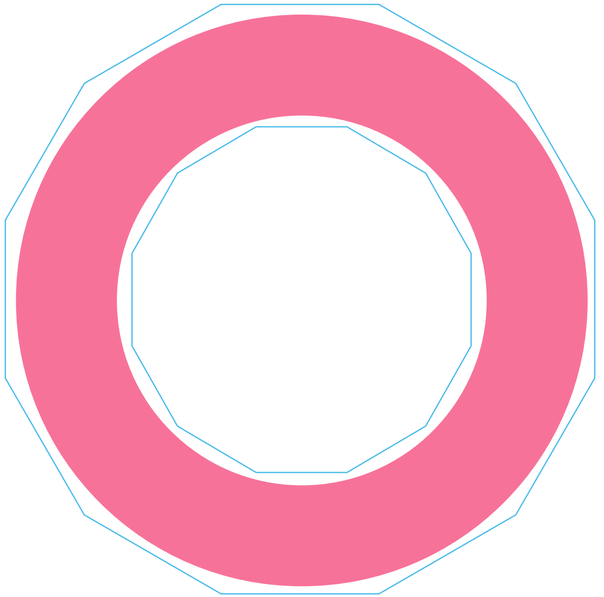


减少像素计算

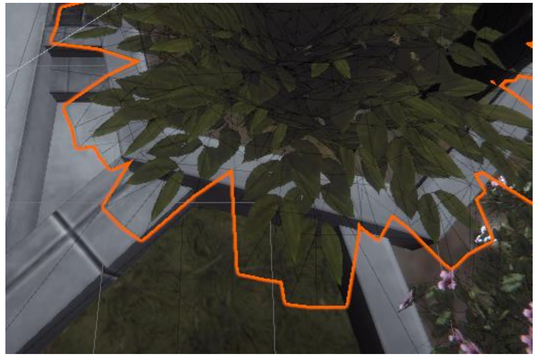
1. 贴图优化，减少面积
2. 增加顶点，减少面积

网格优化





网格优化



网格优化

减少像素计算

1. 贴图优化，减少面积
2. 增加顶点，减少面积
3. 定制Shader，减少面数

定制Shader

避免多张贴图的效果，避免用多个面片叠加的方法
使用定制shader，采样多张贴图来实现

减少像素计算

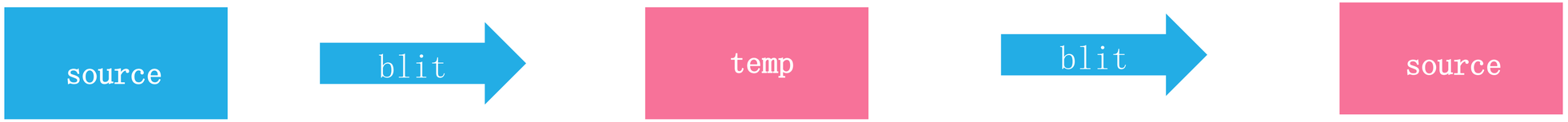
1. 贴图优化,减少面积
2. 增加顶点,减少面积
3. 定制Shader,减少面数
4. 改变特效制作思路,从夸大转向精致
5. UI避免堆叠,多利用九宫格的镂空效果
6. 异形UI,需要切分组合,或者自定义网格
7. 避免使用Mask，用RectMask2D代替
8. UI上一些不可见的点击触发区域，一定要勾选Cull Transparent Mesh

UI优化

减少像素计算

1. 贴图优化,减少面积
2. 增加顶点,减少面积
3. 定制Shader,减少面数
4. 改变特效制作思路,从夸大转向精致
5. UI避免堆叠,多利用九宫格的镂空效果
6. 异形UI,需要切分组合,或者自定义网格
7. 避免使用Mask，用RectMask2D代替
8. UI上一些不可见的点击触发区域，一定要勾选Cull Transparent Mesh
9. 减少不必要的后期处理

减少后处理



200万像素
4亿次计算

省掉一次后处理
场景可以添加5百万个不透明三角形

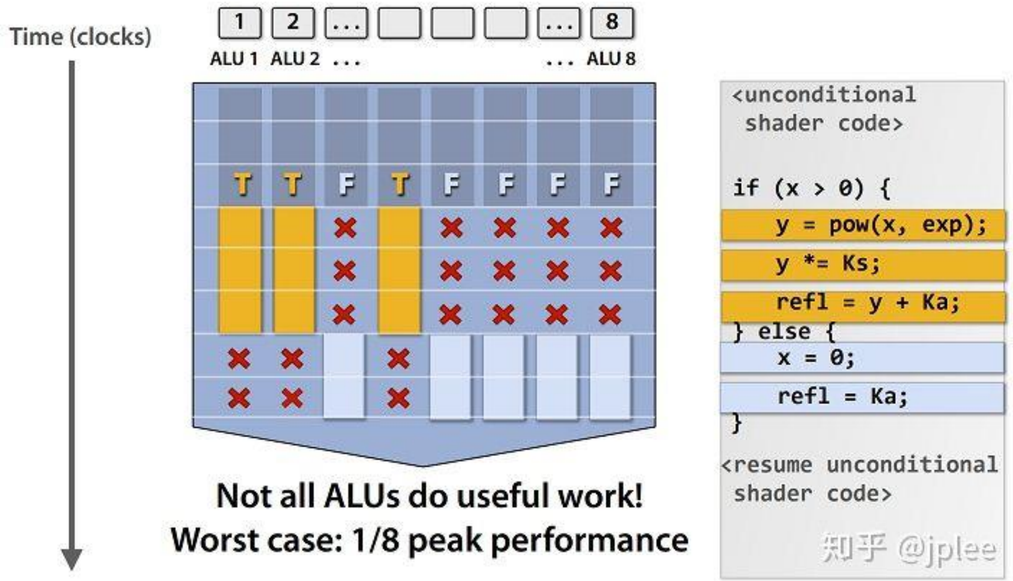
减少像素计算

1. 贴图优化,减少面积
2. 增加顶点,减少面积
3. 定制Shader,减少面数
4. 改变特效制作思路,从夸大转向精致
5. UI避免堆叠,多利用九宫格的镂空效果
6. 异形UI,需要切分组合,或者自定义网格
7. 避免使用Mask，用RectMask2D代替
8. UI上一些不可见的点击触发区域，一定要勾选Cull Transparent Mesh
9. 减少不必要的后期处理
10. 合并后期处理
11. 在更小的屏幕空间做后处理，DLSS,FSR
12. 用贴花代替投影
13. 用阴影投影面片代替全地面接收阴影
14. 使用Occlusion Culling
15. 使用SMAA代替MSAA

减少Shader复杂度

1. 避免动态分支

动态分支的执行


Time (clocks)
1
2
...
8
ALU 1
ALU 2
...
... ALU 8
T T F T F F F F
Not all ALUs do useful work!
Worst case: 1/8 peak performance
<unconditional shader code>
if (x > 0) {
y = pow(x, exp);
y *= Ks;
refl = y + Ka;
} else {
x = 0;
refl = Ka;
}
<resume unconditional shader code>
知乎 @jplee

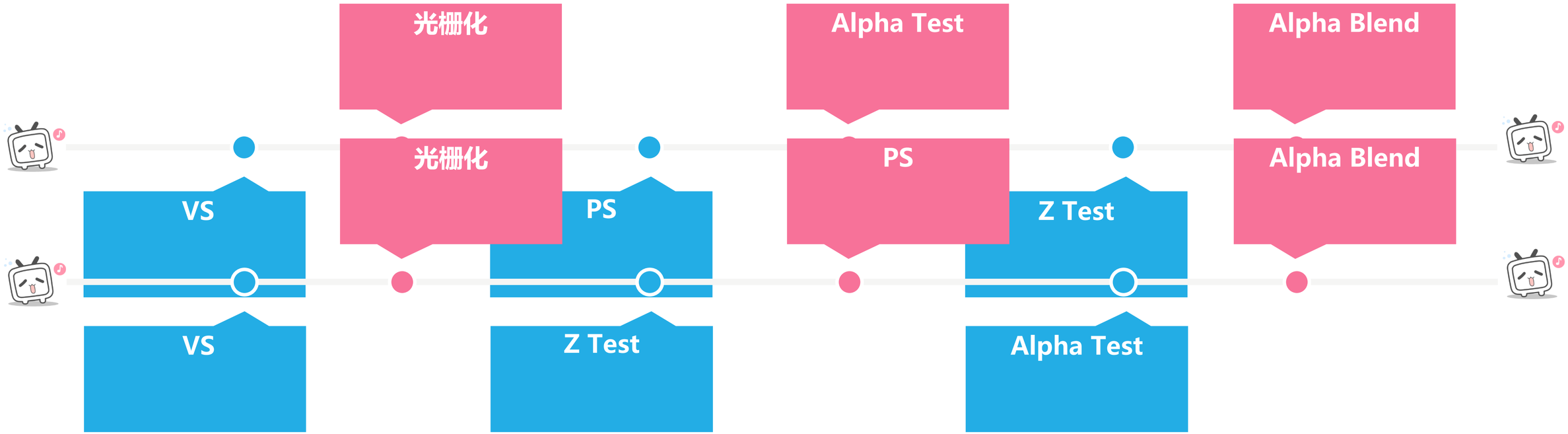
减少Shader复杂度

1. 避免动态分支
2. 有些Fragment Shader中可以简单计算的，不要放在Vertex Shader中计算，通过增加像素属性传到ps
3. 去掉用不到的定点属性
4. 如果可以很早discard像素,可以写动态分支discard
5. 如果精度允许，使用half代替float，提高GPU运算能力，降低寄存器占用
6. UV 的TillingOffset明确是1,0不会变的，Shader中去掉UV的平移缩放计算

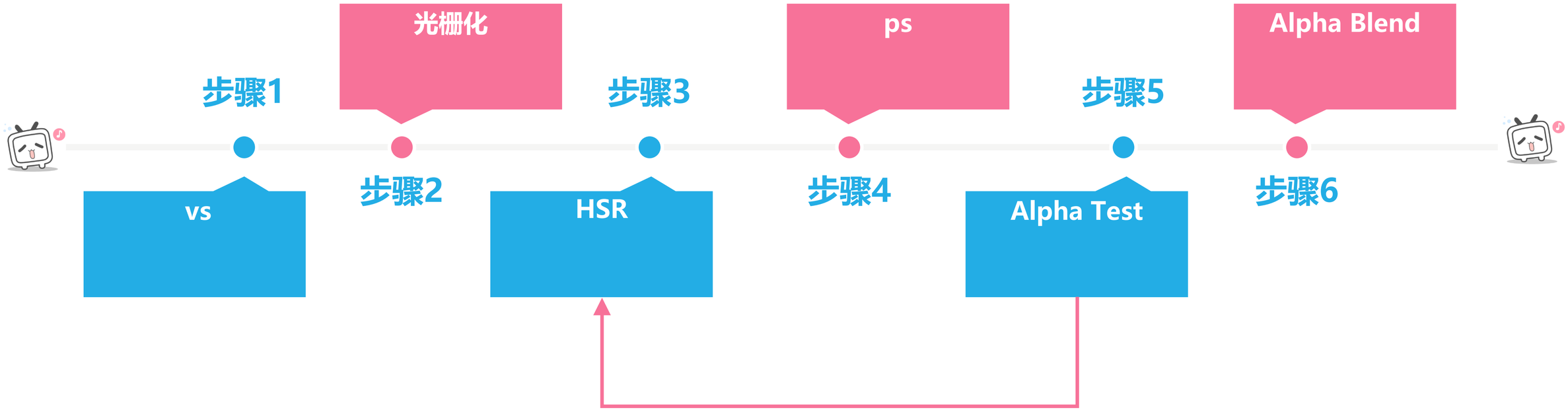
HSR优化

1. Early Z，TBDR，FPK

Early Z


光栅化
Alpha Test
Alpha Blend
光栅化
PS
Alpha Blend
VS
PS
Z Test
VS
Z Test
Alpha Test

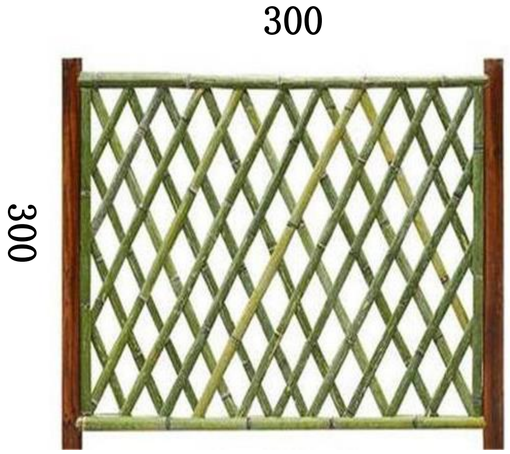
TBDR



HSR优化

1. Early Z，TBDR，FPK
2. 严格遵守Opaque->Alpha Test->Alpha Blend渲染顺序
3. Opaque从近到远渲染
4. 尽量用Mesh不透明材质代替Alpha Test

Mesh代替Alpha Test



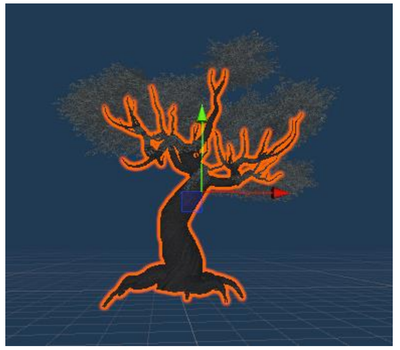
Alpha test:300*300*200=18,000,000次浮点计算
Mesh:20000*80=1,600,000次浮点计算

Mesh代替Alpha Test

网格草降低 OverDraw

Mesh代替Alpha Test

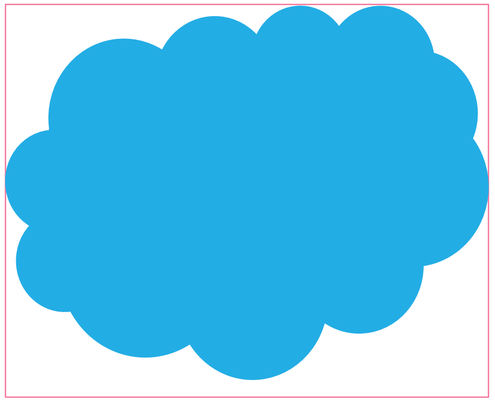
树叶允许用alpha test，树干必须使用opaque材质



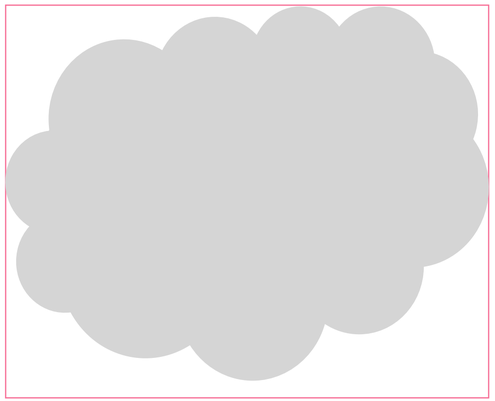
HSR优化

1. Early Z，TBDR，FPK
2. 严格遵守Opaque->Alpha Test->Alpha Blend渲染顺序
3. Opaque从近到远渲染
4. 尽量用Mesh不透明材质代替Alpha Test
5. shader计算很复杂的Alpha Test可以采用prez优化方法

prez



100*100*1000=10, 000, 000



100*100*12=120, 000

+

8000*1000=8, 000, 000

=

8, 120, 000

HSR优化

1. Early Z，TBDR，FPK
2. 严格遵守Opaque->Alpha Test->Alpha Blend渲染顺序
3. Opaque从近到远渲染
4. 尽量用Mesh不透明材质代替Alpha Test
5. shader计算很复杂的Alpha Test可以采用prez优化方法
6. Alpha Blend会使TBDR的hsr失效，但是不影响earlyz。但是earlyz下也不建议用alpha blend代替alpha test，实在没招了可以尝试

CPU算力瓶颈

1. 热点函数优化，算法优化
2. 分支预测优化
3. 大循环放在小循环里面
4. 利用多线程，jobsystem，task，将每个核心算力发挥出来
5. 采用ECS框架
6. 降低DrawCall
7. 大量可并行计算采用Computer Shader，发挥GPU优势
8. Android上利用jit生成机器码，提高执行效率
9. 尽量使用数组代替链表，顺序处理数据
10. 可控范围内，数据尽量做内存对齐
11. 对于骨骼动画，使用OptimizeTransformHierachy优化

带宽瓶颈

1. 减少Cache miss

减少GPU Cache Miss

1. 避免使用Texture来传instancing数据
2. Shader中避免采样不相邻的像素
3. 避免3D Texture
4. 能接受双线性过滤的效果，就避免采用三线性和各向异性过滤
5. 开启mipmap

带宽瓶颈

1. 减少Cache miss
2. 开启mipmap
3. 使用硬件压缩格式，在可接受范围尽量用更大的压缩率
4. 使用九宫格sprite减少UI贴图大小
5. 检查贴图，避免过大的贴图
6. 减小Lightmap 尺寸
7. 使用Occlusion culling减少渲染目标
8. 使用LOD 网格
9. 使用gpu instancing代替网格合并
10. 降低分辨率
11. 尽量使用常量寄存器实现gpu instancing

某一帧耗时过长

1. 检查热点函数，优化函数
2. 使用多线程处理耗时过长的任务
3. 使用协程，分帧处理耗时过长的任务

Pt. 2

续航优化

电都去哪了

1. 数据传输
2. CPU计算
3. GPU计算
4. WIFI模块
5. 屏幕

CPU如何省电

$$P = CfV^2$$

C是常量，和CPU设计有关,f是CPU工作频率,V是CPU工作电压
f和V是正比关系

CPU耗电量和工作频率是3次方关系

暗黑风格的游戏更省电

LCD屏，可以整体调暗，实现省电

OLED屏，像素单独发光，越暗的像素越省电

Thanks

bilibili 游戏

你的幻想世界