

Anti-Aliasing 技术盘点



原亮 ✅

腾讯 游戏引擎工程师

179 人赞同了该文章

先说一下什么是 Anti-Aliasing。Anti-Aliasing 是指抗锯齿，在图像处理上的定义是去除图像的抖动部分。什么是图像的抖动部分？可以看一下下面几张手游的截图：

▲ 赞同 179



● 11 条评论



分享



喜欢

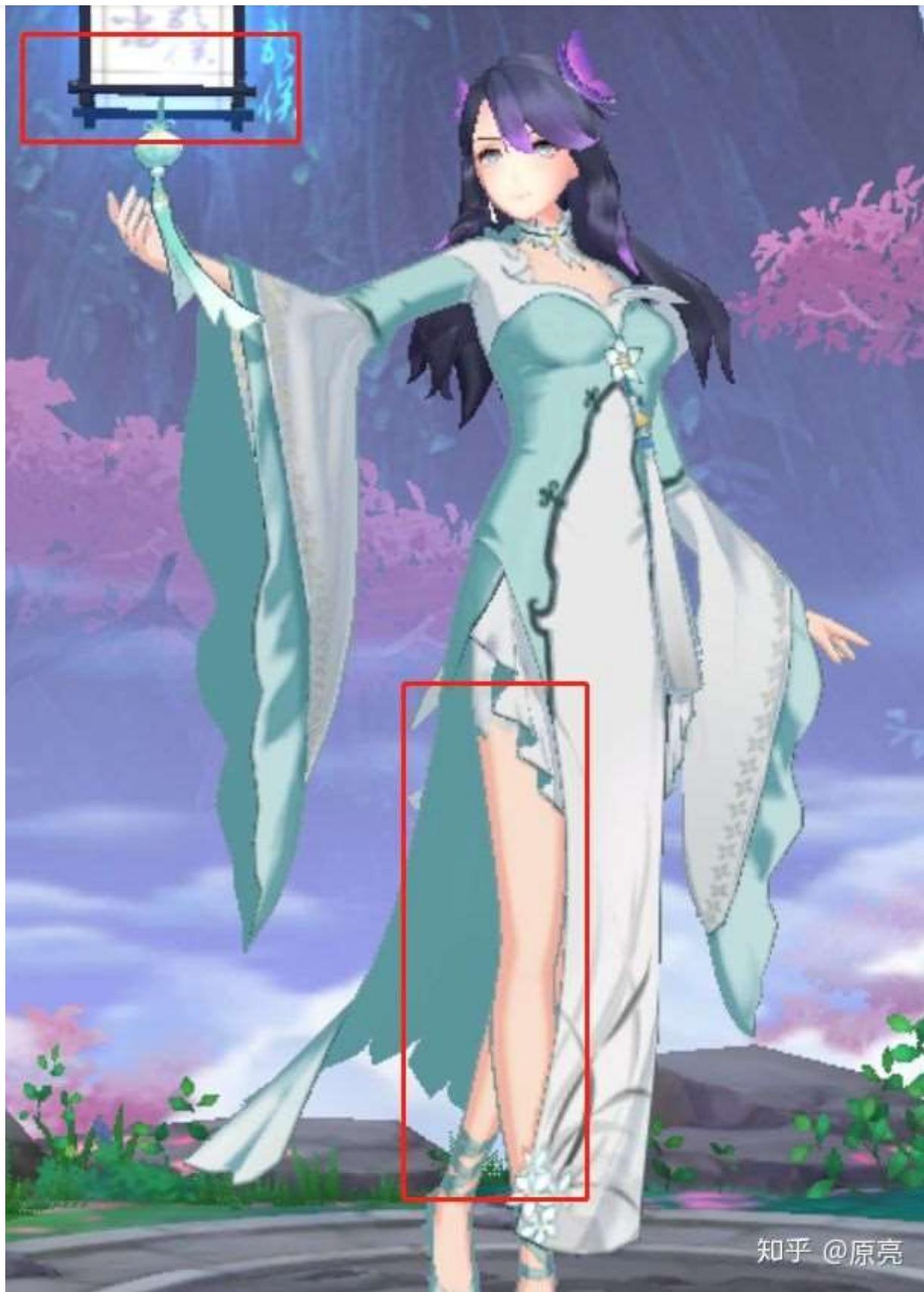


收藏



申请转载





▲ 赞同 179



● 11 条评论



喜欢



申请转载





这是两款近两个月比较火的手游，我也在玩，可以看到模型有明显的锯齿状边缘，甚至描边都是一块儿一块儿的。可能在以前的游戏中感觉不太明显，但在现在的大屏手机上看起来还是对画面质量有比较大的影响的。随着玩家对画面质量要求的提高，移动端游戏之后也应该有抗锯齿的画面设置。（这里感谢知友 [@Jeffrey Zhuang](#) 的指正）

那么锯齿是怎么产生的呢？其实是采样率的问题。一般来说引擎对在画面渲染时都是 Per-Pixel Shading（逐像素着色），等于是1: 1的采样率，这其实是不够的，在边缘处的随机性就会导致锯齿状噪音的出现。不只是模型的边缘，半透明物体的边缘和着色的高频部分（比如比较锐利的高光）也同样会出现抖动。所以说早期的Anti-Aliasing方法都是基于超采样做的，就是以1: 4, 1: 16或者更高的采样率对图像进行着色，然后对同一像素的采样结果做平均，DownSample 到最终输出大小。这种做法效果当然是最好的，可是 GPU 的带宽，执行次数和内存占用会直接成倍的增

▲ 赞同 179

● 11 条评论

▼ 分享

喜欢

收藏

申请转载

...

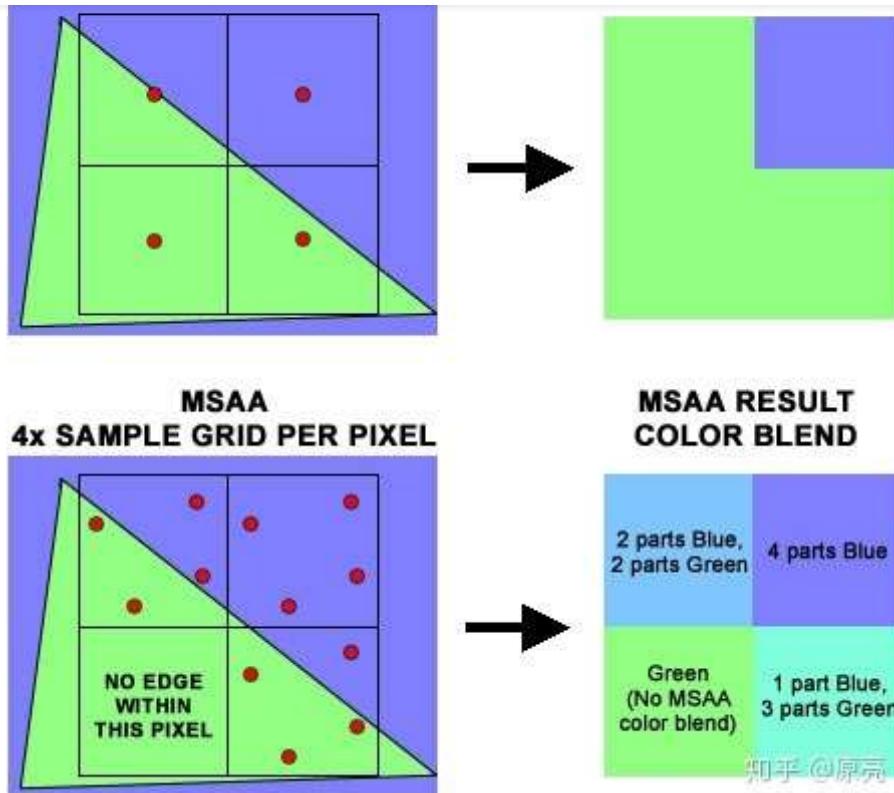
SSAA (Supersample Anti-Aliasing)

超采样抗锯齿就是最朴素的也是最根本的抗锯齿方法，直接提高图片的采样率。之后介绍的抗锯齿基本都会拿 SSAA 来作为效果标杆。SSAA 的技术实现没什么好讲的，不过这里我想提一个概念，就是 **Per-Pixel Shading** 和 **Per-Sample Shading**。这两个东西对大多数人来说比较陌生，因为其实我们都在用 Per-Pixel Shading，所以一般不会特地提到这两个概念。Per-Pixel Shading 和 Per-Sample Shading 实际上指 GPU 上的 PixelShader 的执行频率，是每像素一次还是每个采样点一次。但实际上在 GPU 在执行 DrawCall 时，可以设置一个目标像素对应几次 Sample，在 SSAA 中这些 Sample 都会执行从光栅化 (Rasterization) 开始的所有流程，包括 PixelShader 计算，所以 SSAA 是 Per-Sample Shading。至于 SSAA 的性能消耗，自己要就取决于设置的 Sample 数了。比如 SSAA 16x 在光栅化时每个像素就会进行16次采样，之后也会执行16次 PixelShader，最后平均出该像素的结果，简单粗暴，效果拔群。当然代价就是效率十分的捉鸡，尤其是在线程数不足的中低端 GPU 上基本上渲染时间会线性的上涨，手机平台就更不用考虑了。

MSAA (Multisample Anti-Aliasing)

多样本抗锯齿是目前最泛用的抗锯齿方法，基本上所有平台的GPU都会支持。MSAA其实和SSAA一样，也是基于硬件的实现，只是在增加采样点和平均的时候用了一些 Trick，大幅优化了 SSAA 的效率，当然代价就是对特定的锯齿完全没有优化效果。

首先我们来了解一下 MSAA 的原理。和 SSAA 不同，MSAA 其实是 Per-Pixel Shading。GPU 打开 MSAA 后，会在光栅化阶段根据图元边缘的计算出其覆盖的像素，和对该像素内每一个 Sample 的覆盖关系 (coverage mask)，在执行 PixelShader 后写入颜色时，会根据覆盖关系把 PS 输出的颜色和 BackBuffer 颜色进行平均后作为最终结果。以上所有的步骤都是 GPU 硬件实现的，所以效率上有一定的保证。MSAA 4x 中的 4 其实就是计算 coverage mask 时的采样数。



但是这样做同样会有一些问题，就是针对在PS执行后产生的锯齿没有任何效果，比如高频高光。还有一个大问题是半透明物体如树叶，草这些，也需要抗锯齿。但是这些物体的边缘是在执行 PS 计算时对贴图中的 Alpha 值进行采样后得到的，所以光栅化阶段得到的 coverage mask 对这些物体也没有效果，因为计算 coverage mask 时它们不过是一堆 Quad 罢了。对于这些物体，要想知道他们的边缘只能在执行 PS 时做workaround，就是 Alpha to Coverage。简单来说就是在执行 PS 的时候，对 Alpha 值和 AlphaTest 的 Clip Value 进行比较，根据差值修改该像素的 Coverage Mask。但是，要注意的是这个方法并不健壮。如果一个 DrawCall 中，有多层的半透明物体，比如说头发，那么就需要多次的修改同一个 CoverageMask，而且这个顺序是完全不可控的，如果对于发片这种很细的物体使用的话，最后的效果往往是稀疏的头发。这篇文章里有对这个问题的分析和进一步的workaround：

<https://medium.com/@bgolus/anti-aliased-alpha-test-the-esoteric-alpha...>
medium.com



主要就是在计算alpha的时候加下面这一句Hack代码：

```
col.a = (col.a - _Cutoff) / max(fwidth(col.a), 0.0001) + 0.5;
```

具体的原理上文中有介绍，这里就不再赘述。如果要实现高质量的 MSAA 这些细节都是必不可少

▲ 赞同 179

● 11 条评论

▼ 分享

♥ 喜欢

★ 收藏

✉ 申请转载

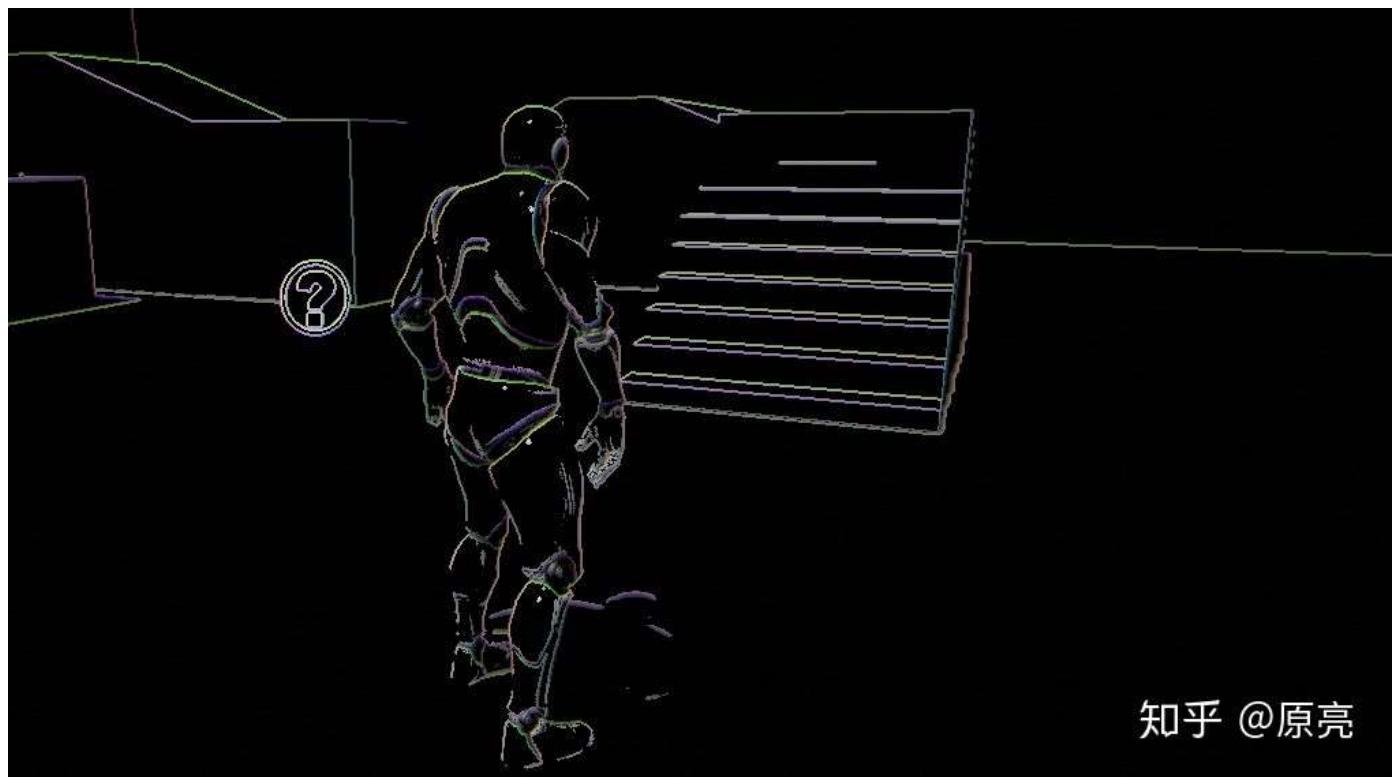
...

总的来说虽然 MSAA 有诸多不足，但是对早期游戏简单的场景来说还是可以的，而且移动平台上也有硬件支持，效率很高。但是，随着 Deferred Shading 的出现和流行，光栅化时期图元数据很难保持到着色阶段，再加上 Deferred 管线本身就存在的 memory impact，导致 SSAA、MSAA 很难再用到 Deferred 管线中。这也引出了新一代的基于后处理的抗锯齿方法。

FXAA (Fast Approximate Anti-Aliasing)

FXAA 是由 NVIDIA 发明的高效后处理抗锯齿方案，也是目前所有后处理抗锯齿方案里面同等设置下效率最高的抗锯齿算法。

边缘提取



如上图，FXAA 的边缘提取是根据 SceneColor 的 Luma 值进行检测的，所以可以提取到高光、阴影，几何边缘，半透明物体。这是 FXAA 相对于 MSAA 最大的优势。

边缘混合

边缘混合的策略其实会根据所选 FXAA 质量的不同有很大的区别。最高效的 FXAA 在混合时不会有 ForLoop，只会对 ColorTex 进行四次采样，而最高质量的 FXAA 在混合时最多会对 ColorTex 进行20余次的采样以确定最优的周围像素进行混合。

▲ 赞同 179



● 11 条评论



▼ 分享

♥ 喜欢

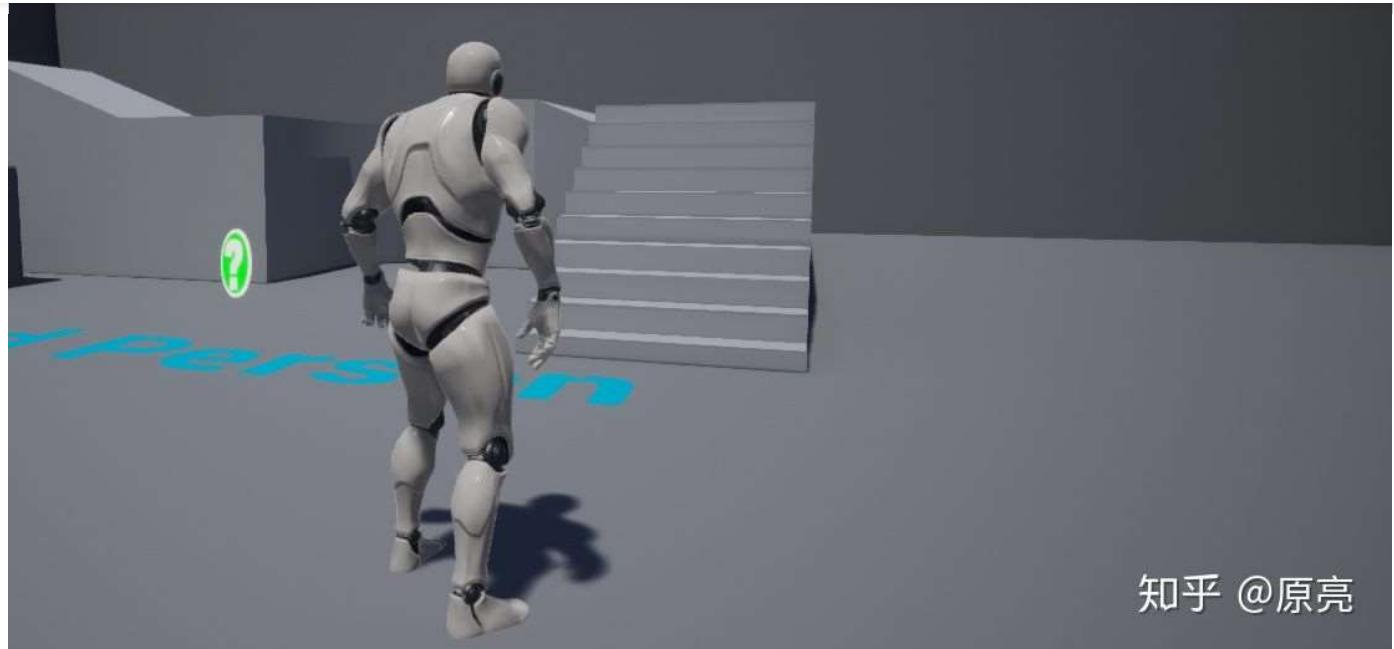


★ 收藏

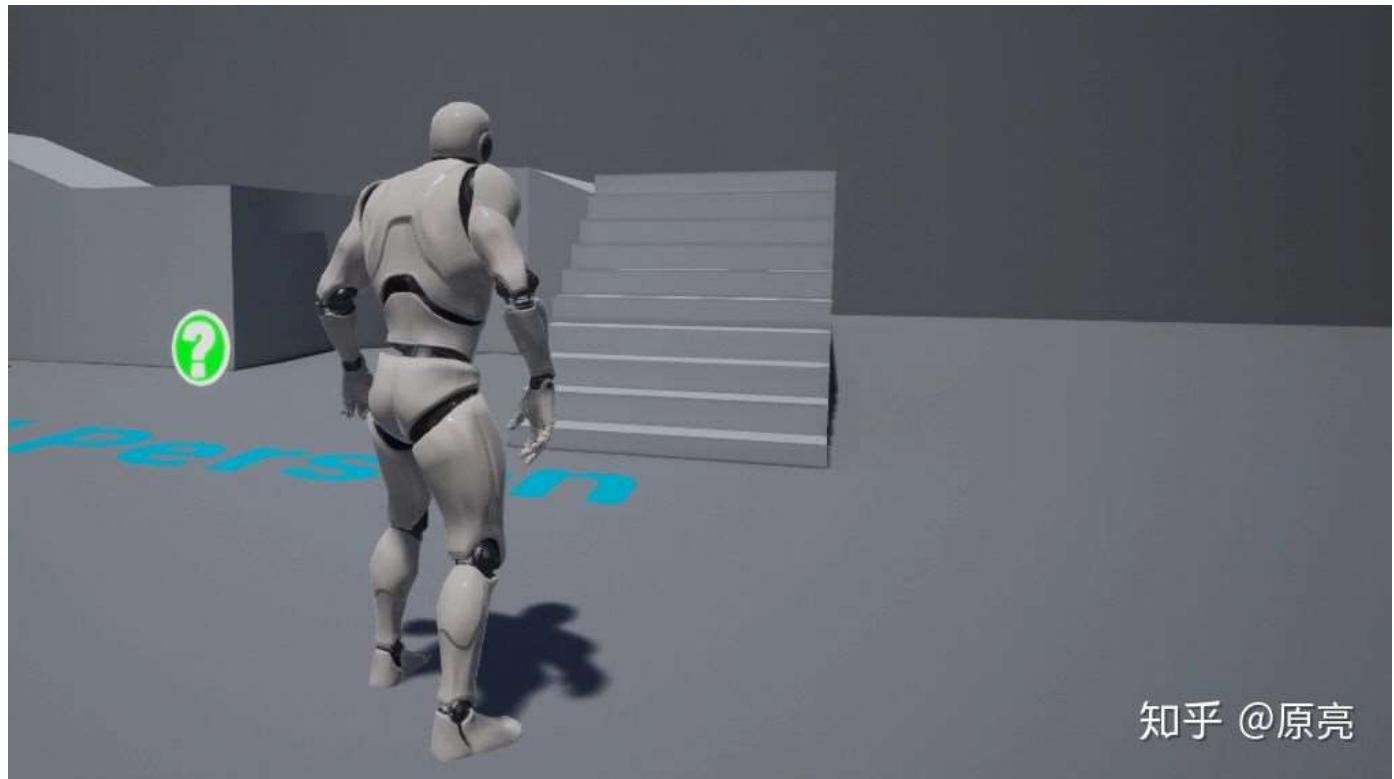


✉ 申请转载





FXAA Low



FXAA Ultra

因为 FXAA 对直角边缘的混合策略，一些细长物体的中间段因为混合时采样到了两侧的空隙（如上图中台阶之间的点），看起来像是断裂的。这也是 FXAA 的不足之一。FXAA 的优势在于其对 Alpha Test，阴影，高光噪点都有效，而且只需要一个单独的后处理 Pass，很好集成。

皇江六田的 CVAA 大业新山吉治珍二十九岁 1994 年 12 月 12 日 12:00:00 小八立应亥吉

▲ 赞同 179

● 11 条评论

▼ 分享

♥ 喜欢

★ 收藏

✉ 申请转载

...

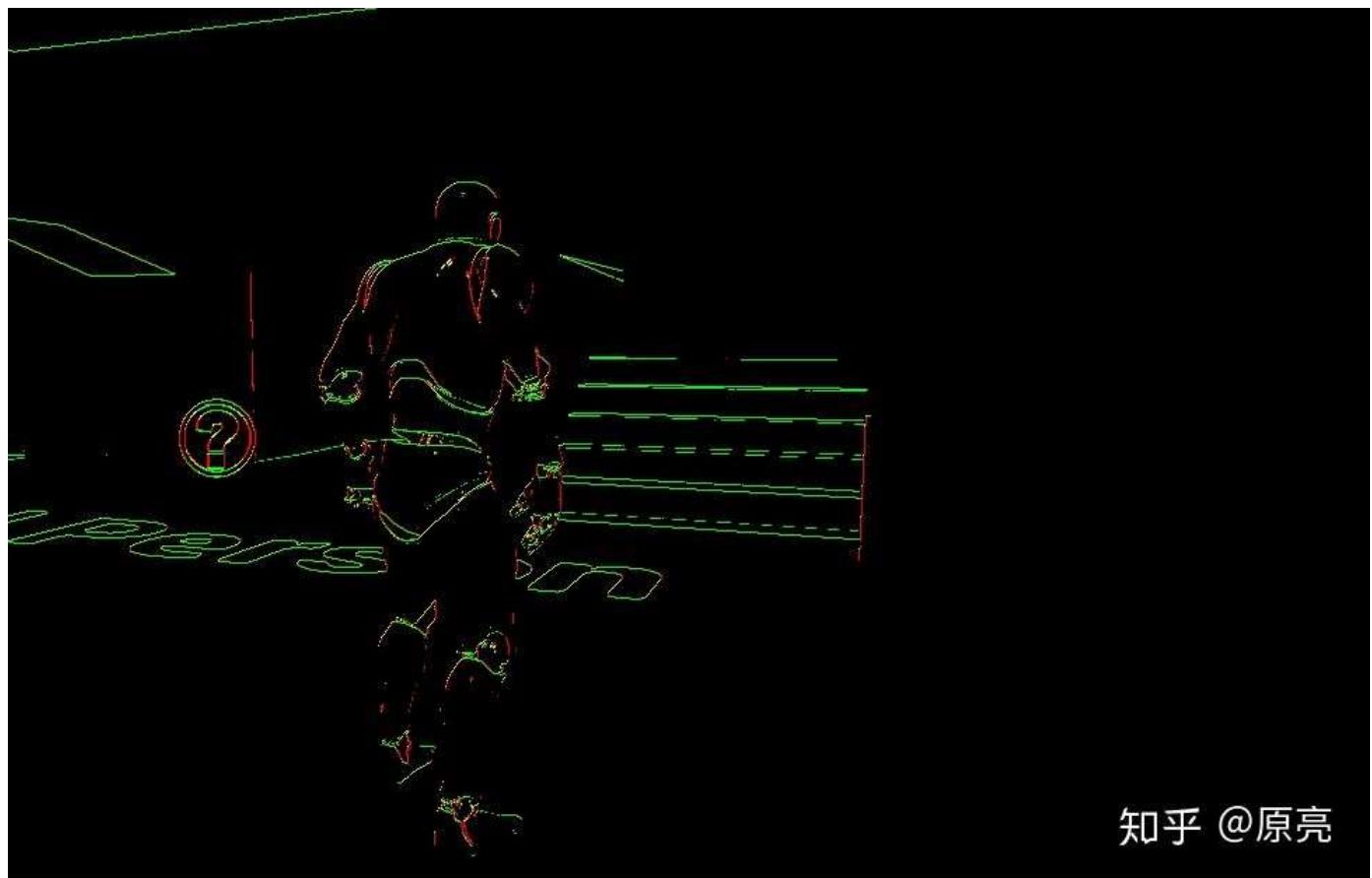
SMAA 是由 CryEngine 开发的更注重效果的后处理抗锯齿方案，当然其效率在大多数移动平台上也是很难被接受的。它和FXAA的最大区别是会更具边缘的形状选择不同的周围像素进行模糊，尽量还原出合理的SubPixels。

和FXAA相比，它不仅本像素是不是锯齿像素，还会关心本周围像素周围的像素，从而推断出该像素处于那种边缘，应该取哪个方向的周围像素来做混合消除这个锯齿像素。所以，SMAA的提取边缘像素这部操作和混合操作是无法在一个Pass中完成的，因为混合时需要边缘提取的计算结果。

SMAA 1x 由以下三个 Pass 组成：

边缘检测

SMAA在提取边缘时会严格区分边缘的形状，低质量的边缘提取和高质量的边缘提取结果会有很大的差别。所以 SMAA 在低质量 (SMAA 1x) 的设定下效果反而不如同等级的 FXAA。



SMAA Low

▲ 赞同 179

▼

● 11 条评论

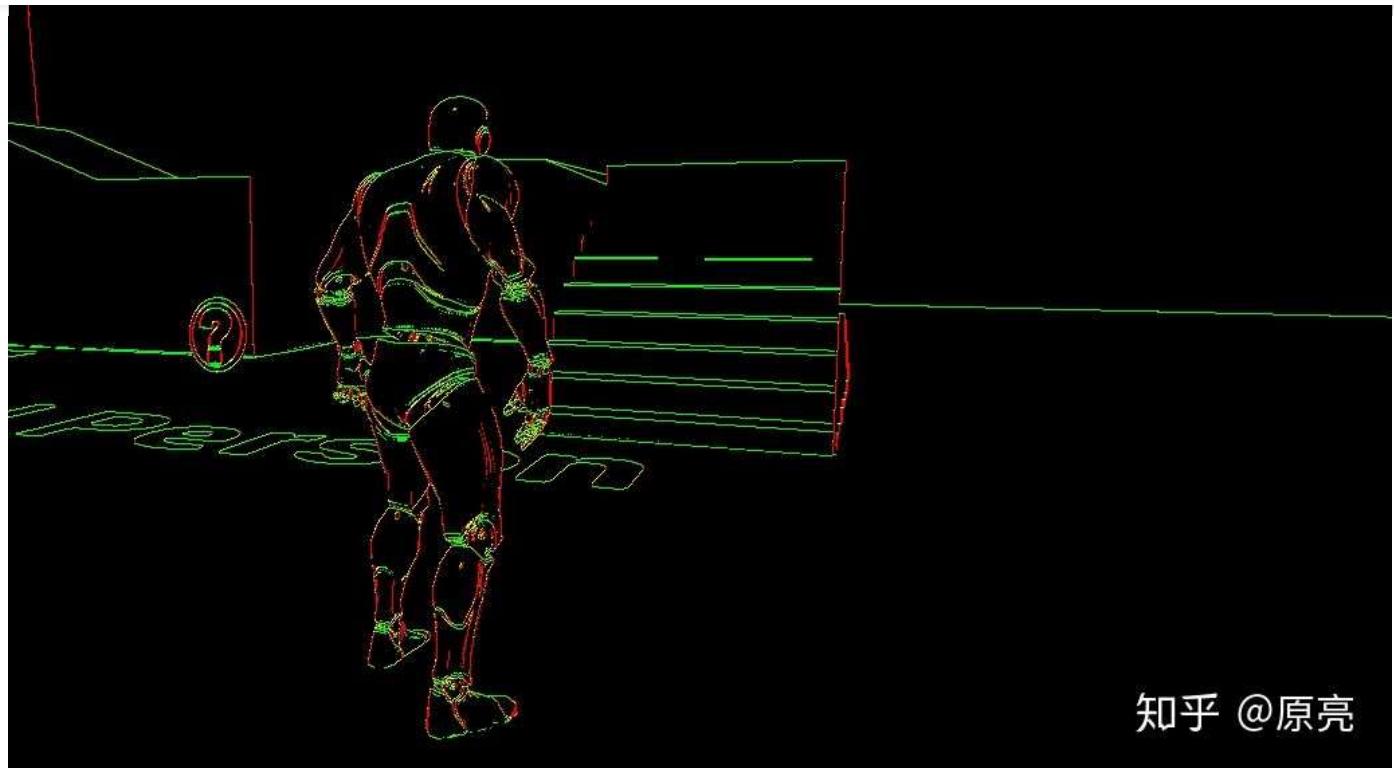
▼ 分享

♥ 喜欢

★ 收藏

✉ 申请转载

...



SMAA Ultra

官方给出的质量定义是这样的：

- * SMAA_PRESET_LOW (%60 of the quality)
- * SMAA_PRESET_MEDIUM (%80 of the quality)
- * SMAA_PRESET_HIGH (%95 of the quality)
- * SMAA_PRESET_ULTRA (%99 of the quality)

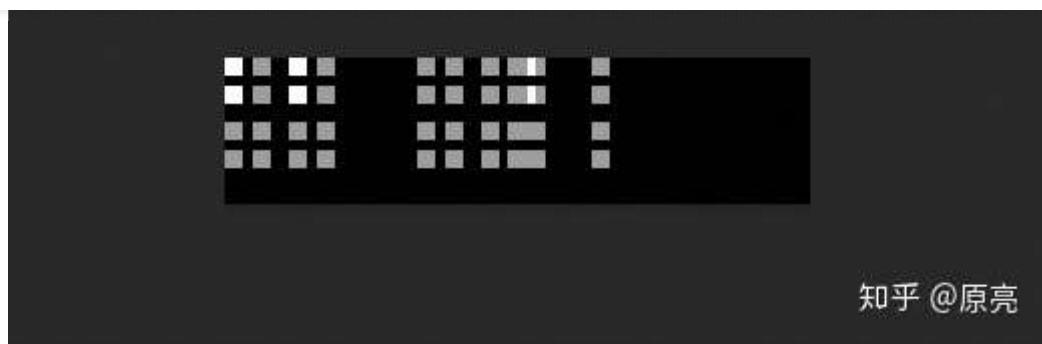
但实际上 SMAA_PRESET_LOW 和 SMAA_PRESET_MEDIUM 效果都很有限。但从这里来看 SMAA 的边缘提取策略其实比 FXAA 更保守。

混合权重计算

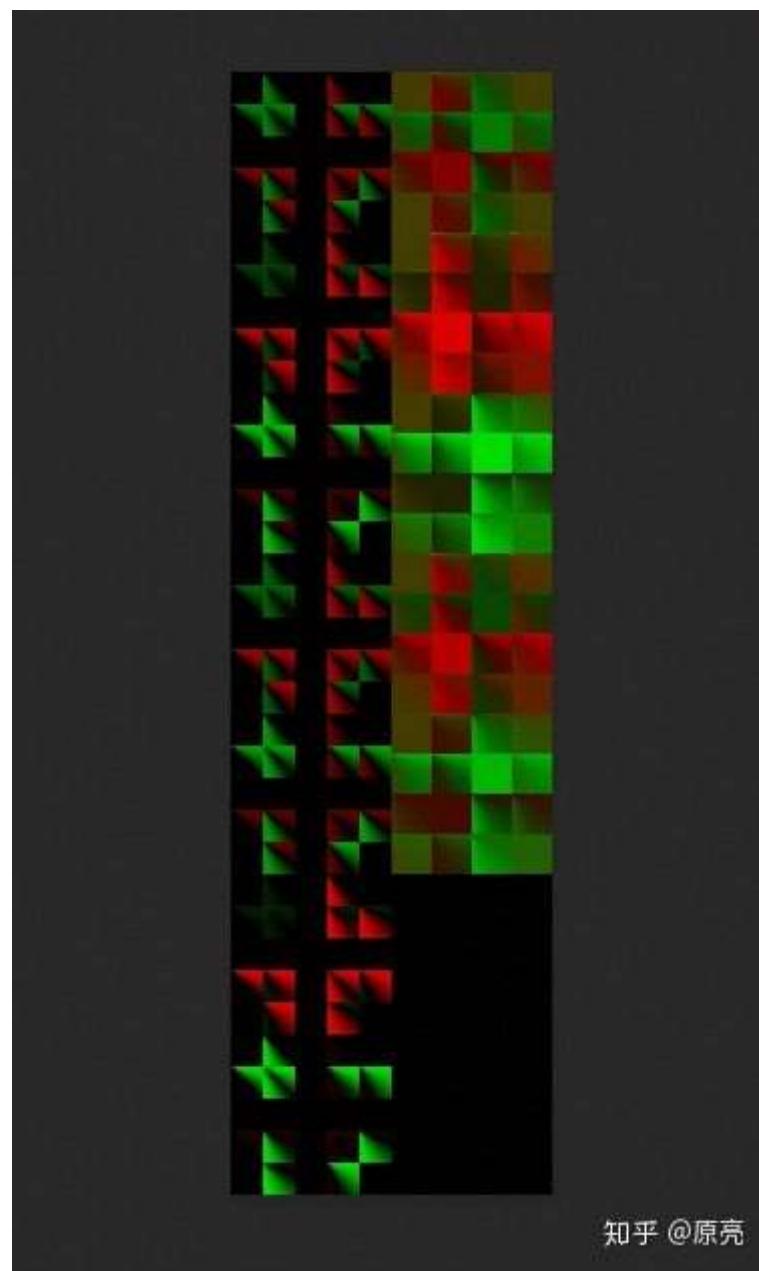
混合权重的计算是为了决定检测出的边缘在下一步中如何混合。这一步需要周围像素在上一步边缘检测中的计算结果，所以必须分出一个单独的Pass。

这里会用到两张张预计算的 Texture：

知乎

首发于
GraphiCon图形控

SearchTex



AreaTex

AreaTex 和 SearchTex 是用于线段特征检测和混合偏移的预计算贴图，用于减少在 PixelShader

▲ 赞同 179



● 11 条评论



分享



喜欢

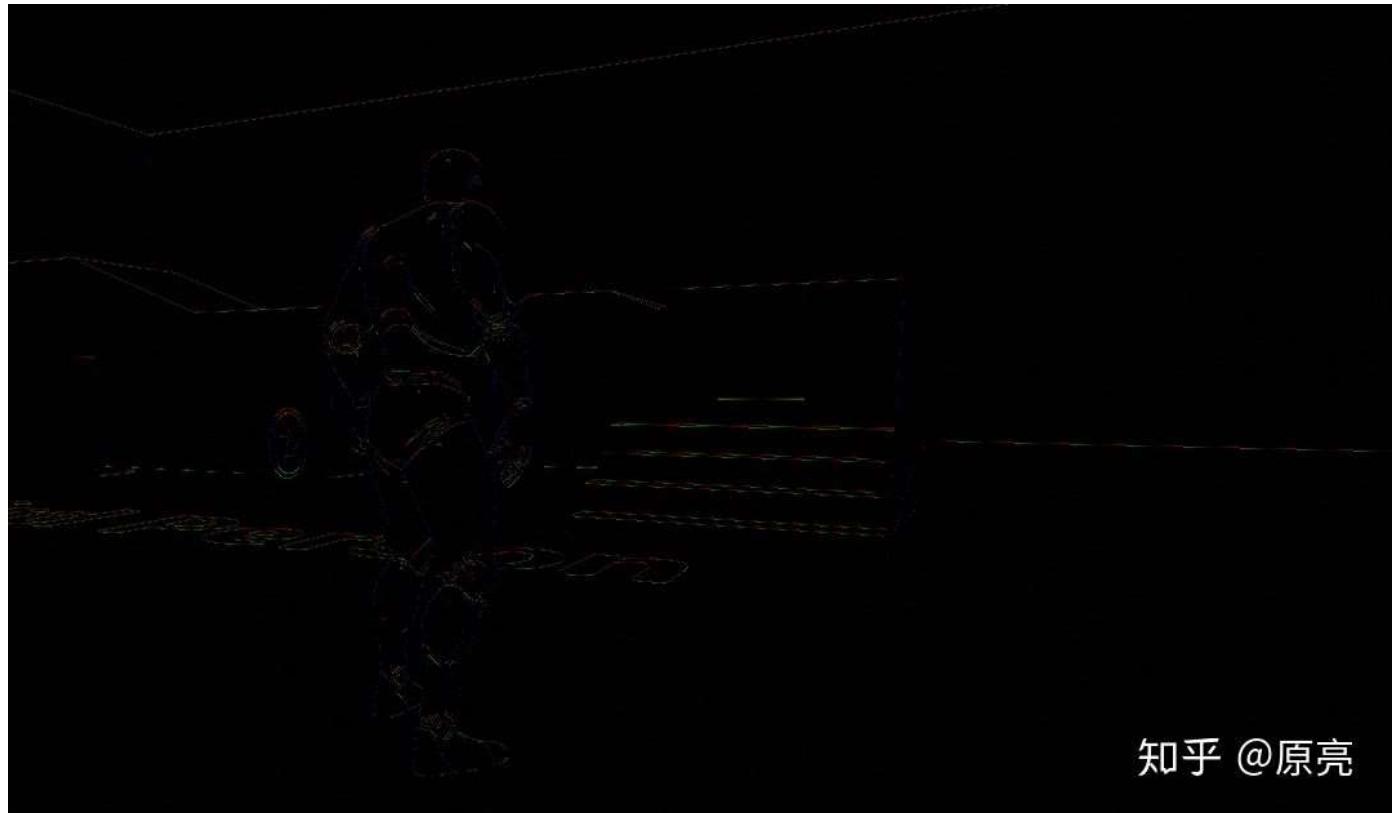


收藏



申请转载





Blend Weight Calculation

可以看到这一步只需要对边缘像素进行操作，所以可以在上一步边缘检测中写好 Stencil，这一步开启 Stencil Test 避免耗时的全屏处理。因为实际上计算的像素很少，所以这一步的耗时非常短。

像素混合

▲ 赞同 179



● 11 条评论



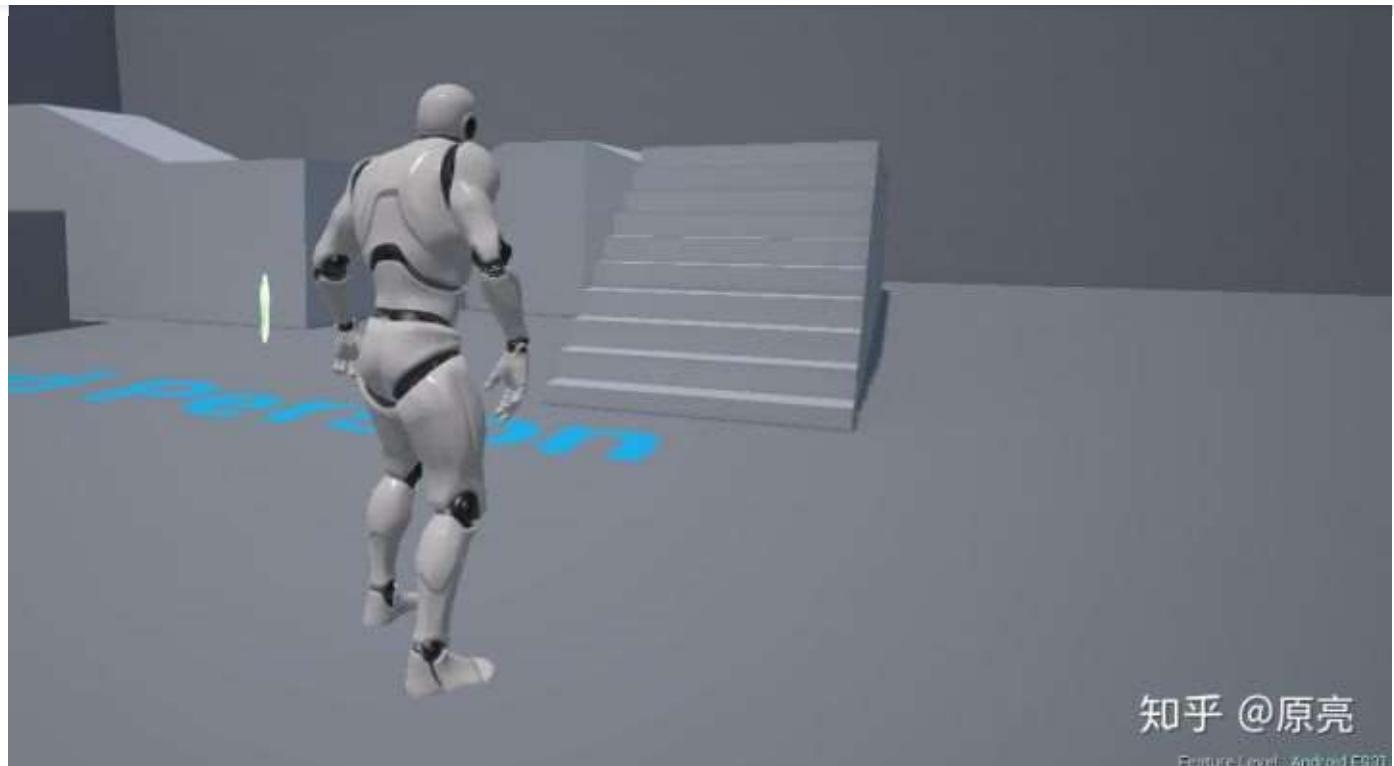
喜欢

收藏

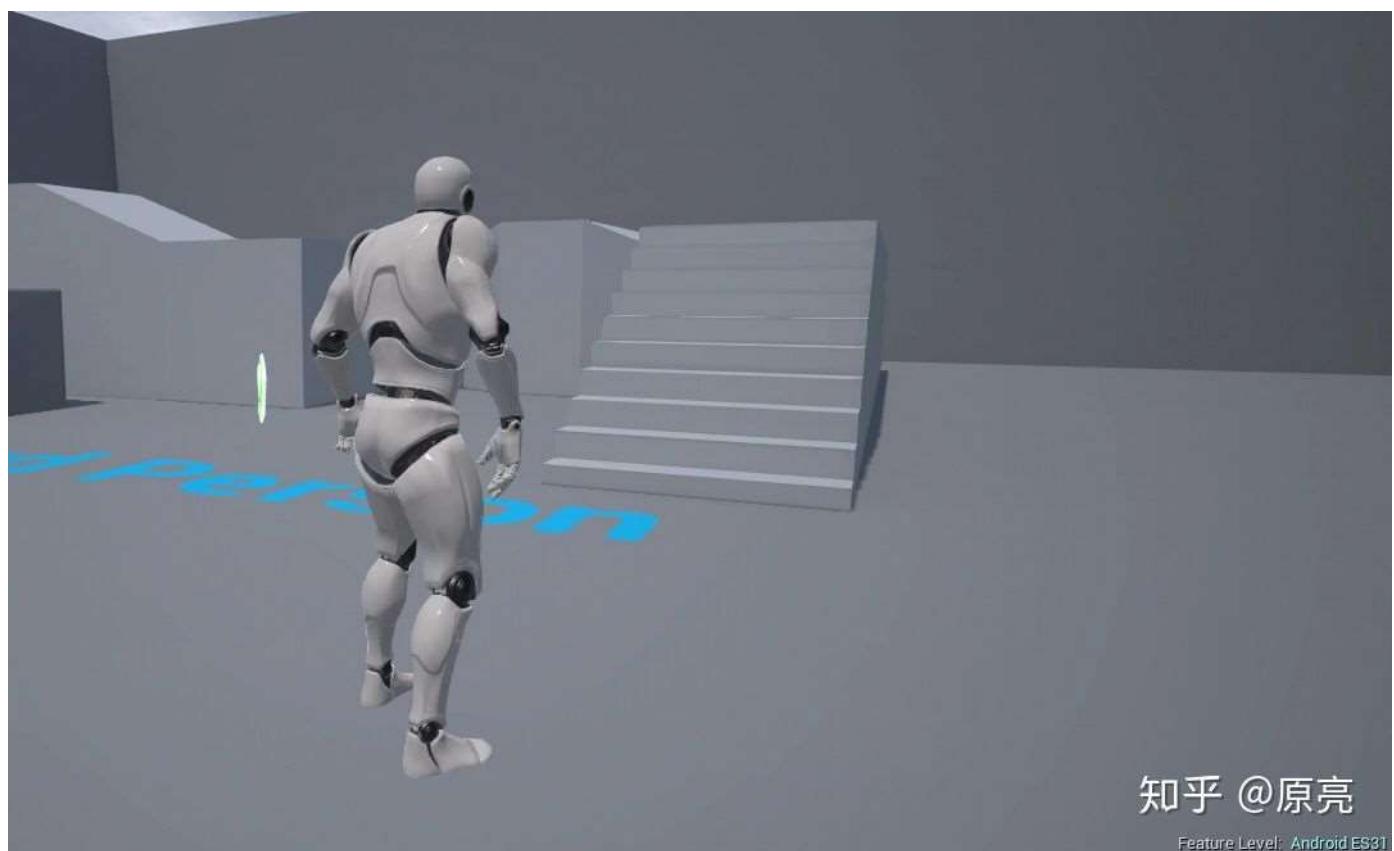
申请转载



知乎

首发于
GraphiCon图形控

SMAA Low



SMAA Ultra

可以看到高质量的 SMAA 确实比 FXAA 更接近超采样 AA 的效果，看上去更加自然。边缘处也没有

▲ 赞同 179



● 11 条评论



喜欢



收藏



申请转载



势，所以就不做介绍了。具体的可以参考官方网站以及 git 上的 demo：

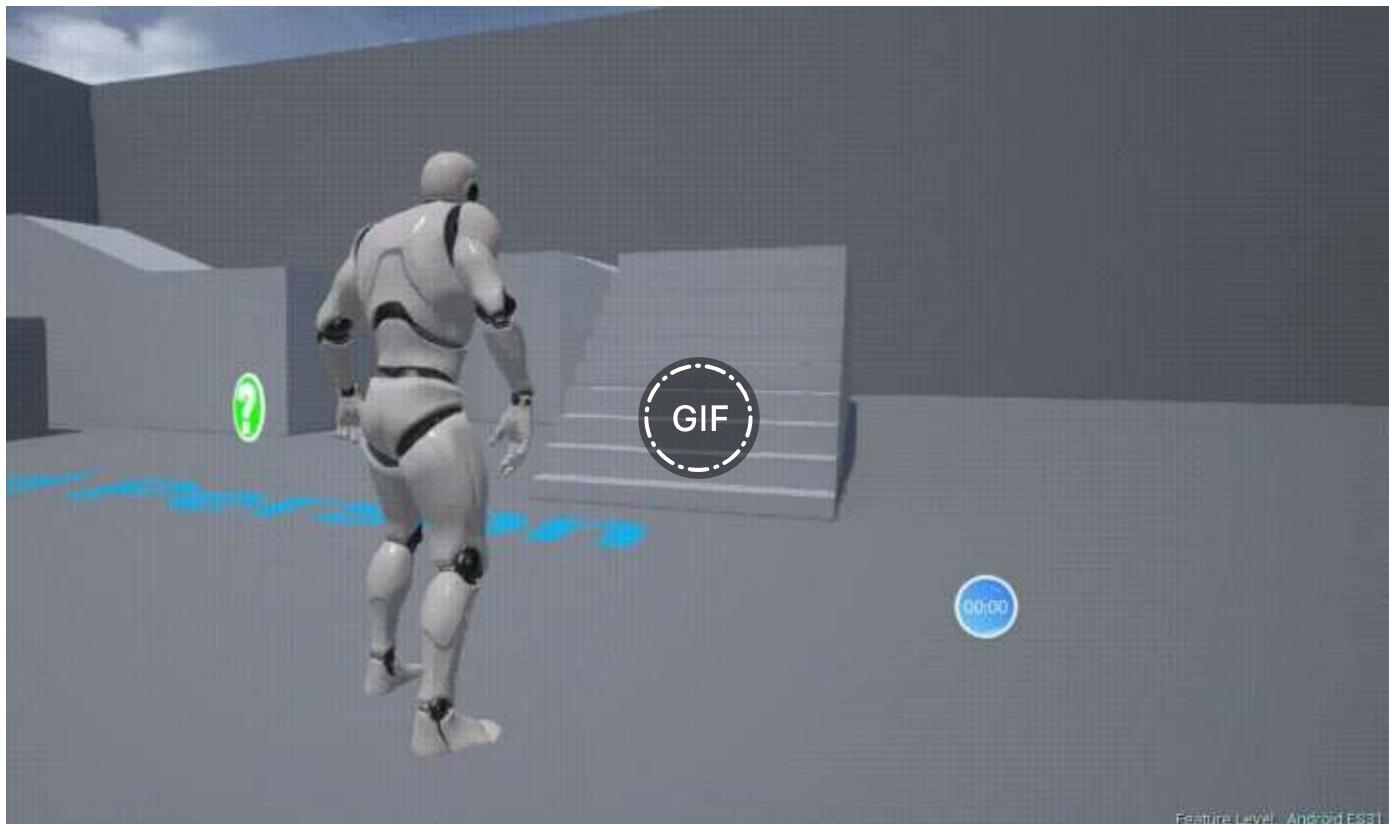
SMAA: Enhanced Subpixel Morphological Antialiasing
www.iryoku.com



对于目前的大多数移动平台来说，SMAA 的性价比是很低的。因为手机本身的屏幕尺寸比较小，而DPI会比显示器高很多，所以 SMAA 在 SubPixel 细节上的改进看起来并不明显。但是在 iPad 和超大屏手机上，SMAA 的效果是明显优于 FXAA 的。而且这两种 AA 方法都是纯后处理方法，切换起来基本没有成本。所以如果有能力的话，根据不同的平台和场景动态切换是最理想的方案。

Temporal AA (Temporal Anti-Aliasing)

基于历史帧的抗锯齿算法 Temporal AA 的概念其实很早就被提出了，目前最常用的 Temporal AA 实现是由 Unreal Engine 开发的。与 FXAA 和 SMAA 不同的是，它本质上是超采样的AA方法。与 MSAA/SSAA 不同的是，Temporal AA 利用了画面在时间上的连续性，不是在当前着色阶段超采样，而是通过保留历史帧数据的方法，在计算时采样历史数据实现超采样。



▲ 赞同 179

● 11 条评论

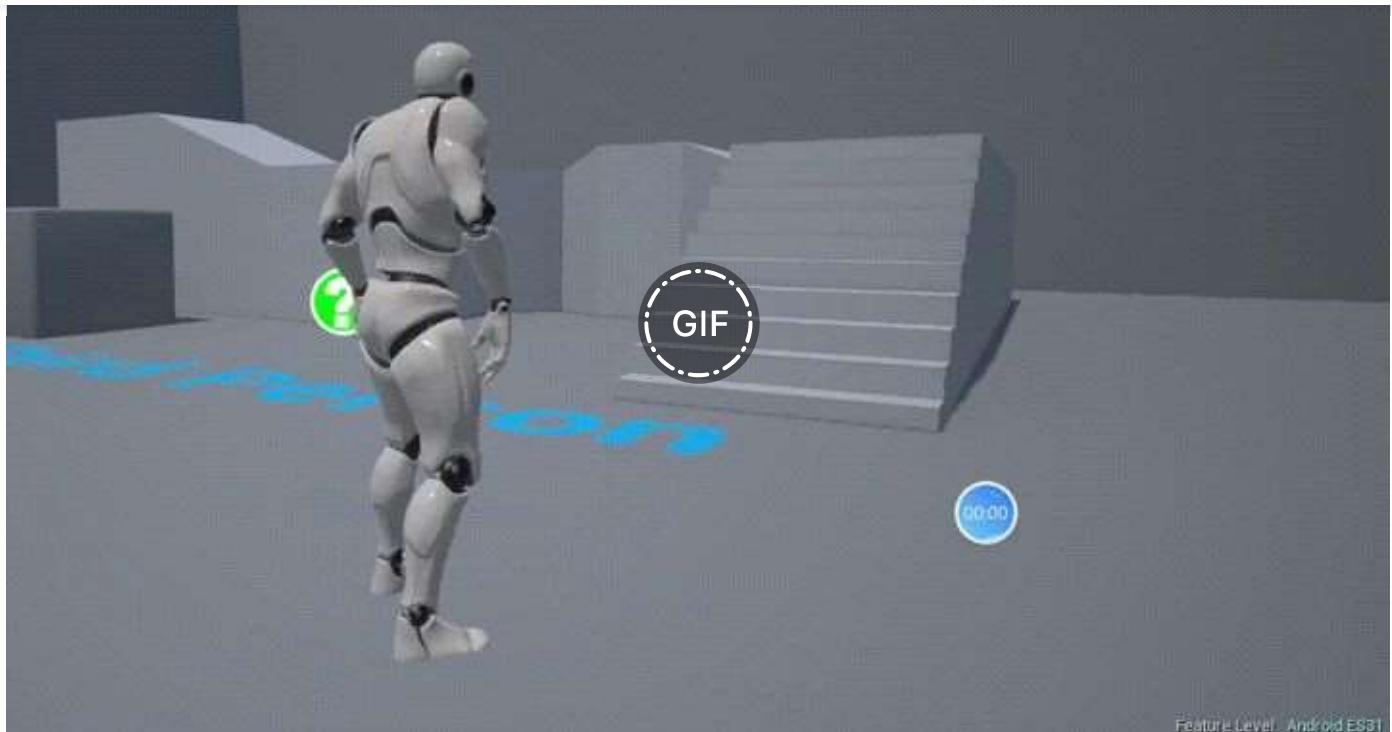
▼ 分享

♥ 喜欢

★ 收藏

✉ 申请转载

...



Temporal AA 在动态画面下

因为 SMAA 1x, FXAA 只根据当前画面进行处理，所以很可能两帧相近的画面间会因为锯齿处细节的差别产生微妙的不同，比如上面台阶间的抖动。可以看到上面两个简陋的录屏在台阶细部的不同：FXAA 在细部会因为两帧间计算的不同结果产生闪烁，而 Temporal AA 即使在快速运动时细节也不会产生抖动。这就是 Epic 提出的“Temporal Aliasing”。一个“Temporal Aliasing”很常见的例子就是远处栅栏在移动时会有很规律的斜向条纹抖动。

Velocity Buffer

因为要重用历史数据，所以必须排除运动造成的影响。这里普遍的做法就是每一帧把 Motion Vector 计算出来存储在一张 Texture 里，一般称为 Velocity Buffer，再根据 Velocity Buffer 找到对应像素上一帧的颜色。一般来说运动来源于五个方面：相机移动、物体移动、骨骼动画、顶点动画和相机切换。其中相机移动最好处理，只需要存储两帧的 View Matrix 即可，不用写入 VelocityBuffer，只需要在使用 VelocityBuffer 时根据 View Matrix 计算一下即可。物体移动和骨骼动画可以通过保存 Model/Animation Matrix 在 Vertex Shader 中计算得到，所以 Temporal AA 需要 Skinning 这一步在 VertexShader 中完成。同样，有 Vertex Animation 的物体需在 VertexShader 中计算两帧间 Vertex 的 MotionVector 并写入 VelocityBuffer。相机切换时，所有历史数据都需要重新计算，VelocityBuffer 也没有了用处，所以在相机切换时 GPU 要执行完全不同的逻辑。



知乎 @原亮

VelocityBuffer

要注意的一点是，为了适配移动速度很快的物体，VelocityBuffer的精度要求会非常高，一般采用R16B16。上图是UnrealEngine的Velocity Buffer在0.498~0.501区间内的颜色，其实这些值的差都在0.002以内。

历史数据重建

历史数据重建分为三个部分，第一步是找到本像素对应的历史帧屏幕坐标，要注意的是，为了在几何边缘有更好的抗锯齿效果，需要采样周围X形的Depth，找到附近最前面的像素作为采样目标，并计算目标坐标是否在屏幕内决定是否要叠算历史帧数据。第二步是采样并过滤历史颜色，首先要计算Neighborhood Bounding Box，这里需要采样当前像素周围的9个像素。然后要采样历史帧数据，这里可以用Catmull-Rom的方式采样五次历史颜色然后进行混合减少运动模糊。最后需要根据Neighborhood Bounding Box对History进行Clamp。第三步是根据当前像素、Neighborhood和History的Luma值混合出最终的着色结果。

对粒子特效的影响

▲ 赞同 179



● 11 条评论



▼ 分享



♥ 喜欢



★ 收藏



✉ 申请转载



...

知乎

首发于
GraphiCon图形控

知乎 @原亮

Responsive AA

因为 TAA 在采样当前像素和历史帧像素时都采取了采样周围多个像素并进行混合的方法，所以对细碎的着色锯齿有非常好的抵消，这是 FXAA 和 SMAA 无法做到的。但是这同样也带来了问题，就是一些细小的粒子特效同样也会受到 TAA 的影响变得模糊。这里 Unreal 提供的解决方案是 Responsive AA，对于指定的物体可以选择开启 Responsive AA，这样在 TAA 计算最终混合结果时会强制历史帧数据只有 0.25 的权重。

从效果上来说，Temporal AA 是目前后处理抗锯齿方法中除了只能用在 NVIDIA 高端显卡上的 DLSS 之外最好的 AA 方案了，如果是注重画质的 PC/主机游戏，用了 Deferred 管线，那么 Temporal AA 就是最优的选择。手机上的话，TAA 与 SMAA 性能消耗差不多，都无法适用于目前的中低端机型。对于动态画面来说，比起时间上的锯齿，帧率的稳定更加重要，所以如果不是机能上有余裕的话，FXAA 就足够了。

以上，现在主流的实时抗锯齿方案就介绍完了。和其他效果有些不同，虽然这些方案都是针对同一个问题，但各有特点，很难相互取代。开发者应该根据游戏平台、类型，甚至是单独的场景选择最适合的方案。遗憾的是，UE4 和 Unity 中目前都没有完全实现这些功能，所以无法直观的进行对比后做技术选型。希望能通过这篇文章，让大家对各种 AA 的特点、可以解决的问题和性能消耗有较为全面的了解，为自己的项目选择最适合的 AA 方案。

编辑于 2020-01-10

▲ 赞同 179

▼ 11 条评论

分享

喜欢

收藏

申请转载

...