GUI背景模糊效果优化

**<茜色世界中与君咏唱>项目组 客户端工程师 – 文立**

**1. 导语**

在游戏界面显示时，通常会对背景进行模糊，使显示界面更加清楚。此外，在处理景深（Depth of Field）和泛光（Bloom）等后处理特效时，模糊效果也是必不可少的。本文针对移动平台，结合实际项目中遇到的UI背景模糊显示效果，讨论了模糊效果的优化解决方案。

为了测试不同方案的实际模糊性能，新建了一个较为简单的测试工程，并使用了2款较为低端的手机进行测试：红米4，处理器为骁龙625和三星Galaxy C5，处理器为骁龙617。在没有背景模糊效果时，2款手机的帧率都稳定在60FPS左右。

**2. 高斯模糊**

说到模糊效果，必然提及高斯模糊。

高斯模糊通常用来减少图像噪声，降低细节层次和图像模糊等，其利用高斯滤波函数处理图像。

简单地讲，高斯模糊就是对图像中的每个像素进行加权平均，利用高斯核函数对像素周围的多个采样点加权相加，结果作为新的像素值。在实现时，可以预先计算归一化的二维高斯核函数。当采用阶高斯核函数时，算法的复杂度是。

一种常见的优化方案是将二维运算拆分成两次一维运算，先在水平方向做一维高斯运算，再在竖直方向做一次高斯运算，以7阶二维高斯运算为例，二维运算需要49次计算，拆分成两次一维运算只需要14次计算，大大降低了计算消耗。

一维0均值高斯函数数学表示如下：



此外，因为背景图片本来就要进行模糊，没有必要按照全分辨率进行采样，对背景图片进行降采样可以提升模糊计算效率。

最初项目里采用的模糊方案就是优化高斯模糊算法。

*GrabPass {}*

*Pass{ … } // 处理水平方向*

*GrabPass {}*

*Pass{ … } // 处理垂直方向*

利用2次GrabPass进行截屏，截屏结果一次进行水平方向模糊处理，一次进行垂直方向模糊处理，2次处理都用了9个采样点。由于进行了2次截屏，然后进行高斯处理，性能消耗挺大的。游戏在一些手机上开启模糊时较为卡顿，性能较差。

利用测试工程进行测试，结果在红米上帧率只有40FPS，在三星上帧率只有20FPS。可以看到，在低端机上这样处理的性能损耗还是很大的，比较严重的影响了游戏体验。

初步优化方案就是减少2次处理的采样点数，从9个减为7个，并且发现原来的采样坐标计算是在fragment shader中计算，会对性能产生一定影响，改在vertex shader中计算，然后传到fragment shader中。

再次测试，在红米上帧率提升到46FPS，在三星上仍然只有22FPS，比之前结果稍好一点。

因为移动端使用GrabPass性能消耗较大，考虑只用一次截屏，利用原始高斯模糊算法进行测试，因为原始计算采样点数较多，所以只使用了5阶高斯核函数。

*fixed4 col = fixed(0, 0, 0, 0);*

*for (int s = 0; s < 5; ++s){*

*for (int t = 0; t < 5; ++t){*

*half4 uv = i.grabuv + \_GrabTexture\_TexelSize \* half4(s - 2, t - 2, 0, 0) \* \_BlurRadius;*

*col += tex2Dproj(\_GrabTexture, uv) \* GaussWeight[s][t];*

*}*

*}*

*return col;*

测试结果在红米上帧率降到20FPS，三星上帧率降到13FPS。果然还是采样点数过多，计算消耗太大。

后来考虑过使用后处理或者RenderTexture，但是这样会对前景界面产生影响，不能很好地产生背景模糊效果。

**3. Scriptable Render Pipeline**

为了不使用GrabPass，减少性能开销，尝试从Unity的渲染管线看看能不能在渲染背景遮罩之前进行截屏处理。

在2018年，Unity官方推出了Scriptable Render Pipeline（可编程渲染管线），用户可以根据自己项目的实际需要自定义渲染管线，或者使用官方开放的2个模板：LightweightRenderPipeline和HighDifinitionRenderPipeline，目前这2个渲染管线还处于测试当中，会有一些变动。在LightweightRP中，可以自定义ScriptableRenderPass，并可以自定义RenderPassEvents，目前预设了BeforeRendering，Before/AfterRenderingShadows，Before/AfterRenderingPrepasses，Before/AfterRenderingOpaques，…,AfterRendering等系列事件。

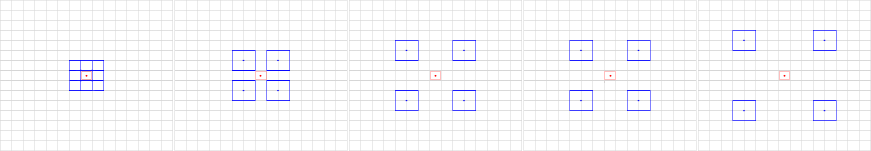
因为UI界面都是在Transparent阶段进行绘制的，如何在背景遮罩之前增加一个pass进行模糊处理，仍未找到一个好的办法，可能是我对可编程渲染管线理解尚不深刻，虽然没有找到方法，还是做个记录，后面可以继续研究。

**4. Kawase模糊**

最后还是采用2个GrabPass，但是要对高斯模糊算法再次优化，后面发现了一种优化高斯模糊算法的Kawase模糊算法。

Kawase模糊算法最先是由Masaki Kawase在GDC2003上提出的，是一种性能优于高斯模糊算法，但是模糊效果稍差的解决方案。该算法最初用于泛光效果，效果十分接近高斯模糊。

Kawase滤波器是一个多通道滤波器，每个通道都会利用上一通道的结果，通过累加足够的滤波通道逼近高斯滤波器。在每个通道中，对距离当前像素一定距离的矩形范围的采样点进行平均更新当前像素值。处理通道的数量和距离像素中心的距离可以根据结果灵活调整。



5通道Kawase模糊，采样距离分别为0, 1, 2, 2, 3

测试采用2个通道，每个通道处理4个采样点，通道采样距离逐渐增加，具体数值可以调整。

采样代码如下：

*fixed4 col = fixed4(0, 0, 0, 0);*

*col += tex2Dproj(\_GrabTexture, UNITY\_PROJ\_COORD(i.uvgrab[0]));*

*col += tex2Dproj(\_GrabTexture, UNITY\_PROJ\_COORD(i.uvgrab[1]));*

*col += tex2Dproj(\_GrabTexture, UNITY\_PROJ\_COORD(i.uvgrab[2]));*

*col += tex2Dproj(\_GrabTexture, UNITY\_PROJ\_COORD(i.uvgrab[3]));*

*col \*= 0.25;*

*return col;*

测试结果在红米上帧率未见明显下降，在三星上稳定在28FPS左右，达到还算流畅的效果。模糊效果相比高斯模糊稍差一点，但是仍然可以接受。Kawase模糊只用4个采样点，能够获得比较不错的模糊效果，是一种不错的解决方案。

最后将项目中使用的9阶高斯模糊算法和Kawase模糊算法进行实际对比，如图所示

（a）优化高斯算法 （b）Kawase算法

从图中可以看出，Kawase算法的模糊效果稍差于高斯算法，但是考虑到Kawase算法每个通道只用了4个采样点，算法消耗明显降低，模糊效果也是可以接受的，仍是一种不错的模糊方案。

**5. 总结**

采用原始高斯模糊算法，虽然只用一次截屏，但是采样点数过多，即使只是5阶，计算消耗仍然太高，移动端性能承受不了。采用2次截屏方案，优化高斯算法分成2次一维模糊处理，大大减少了采样点，性能有所提升，但仍对帧率有较大影响，通道采样点数可以根据模糊效果和机器性能进行选择，一般采用5个或7个采样点；Kawase模糊算法每个通道只用4个采样点，减少了计算消耗，性能最好，并且显示效果比较接近高斯模糊，是一种不错的模糊方案。

另外，对于Shader编写有一些常见的优化：

1. 使用合适的数值精度：

0.0-1.0范围内的颜色值通常使用低精度变量fixed表示；

位置数据通常使用高精度float；

光照计算中的法线和向量通常使用中等精度half；

不清楚时，数据默认采用高精度float。

2. 延后向量计算

先进行标量计算，再进行向量计算

3. 使用Uniforms或Constants变量，而不是在Shader中计算

4. 避免动态查找

避免在fragment shader中对纹理坐标进行数学计算，在vertex shader中计算。

*uniform sampler2D textureSampler;*

*fixed4 frag(v2f i){*

*half2 modifiedTexCoord = half2(1 – i.vTexCoord.x, 1 – i.vTexCoord.y);*

*fixed4 col = texture2D(textureSampler, modifiedTexCoord);*

*return col;*

*}*

**6. 参考资料**

<https://blog.csdn.net/poem_qianmo/article/details/51871531>

<https://software.intel.com/en-us/blogs/2014/07/15/an-investigation-of-fast-real-time-gpu-based-image-blur-algorithms>

<http://www.daionet.gr.jp/~masa/archives/GDC2003_DSTEAL.ppt>

<https://developer.apple.com/library/archive/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/BestPracticesforShaders/BestPracticesforShaders.html>