目录

[1. Lwrp概述，内容概述 1](#_Toc28898)

[2. Lwrp shader写法 4](#_Toc18141)

[3. SRP Batcher 8](#_Toc13491)

# Lwrp概述，内容概述

Shemen动画短片

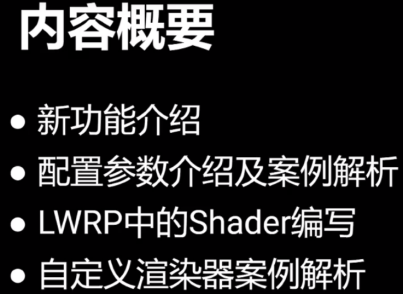
<https://on.unity.com/2K1PZ4O> 申请下载

Demo链接：

1). <https://github.com/Unity-Technologies/LWRP-CustomRendererExamples>

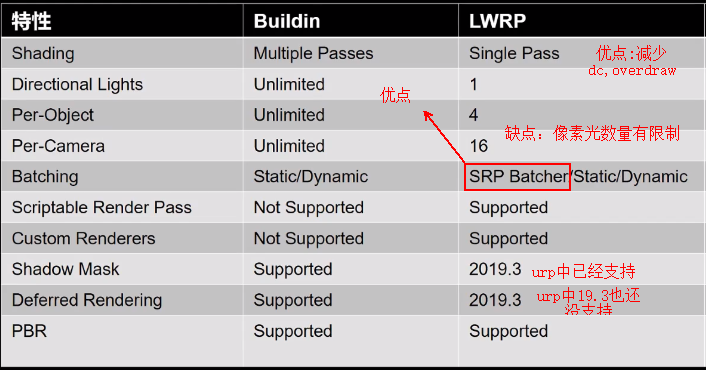
2). <https://github.com/Verasl/BoatAttack>

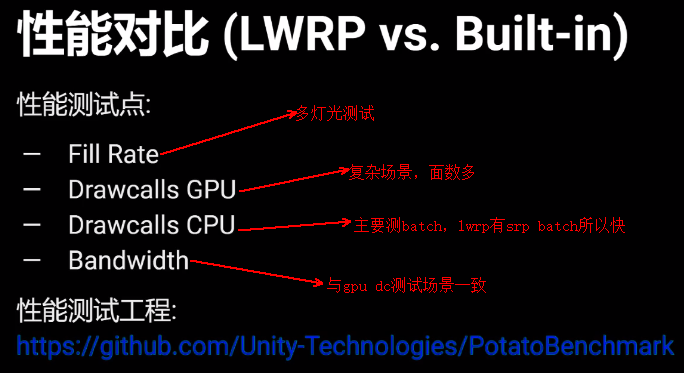
第三个是将code shade，非shader graph



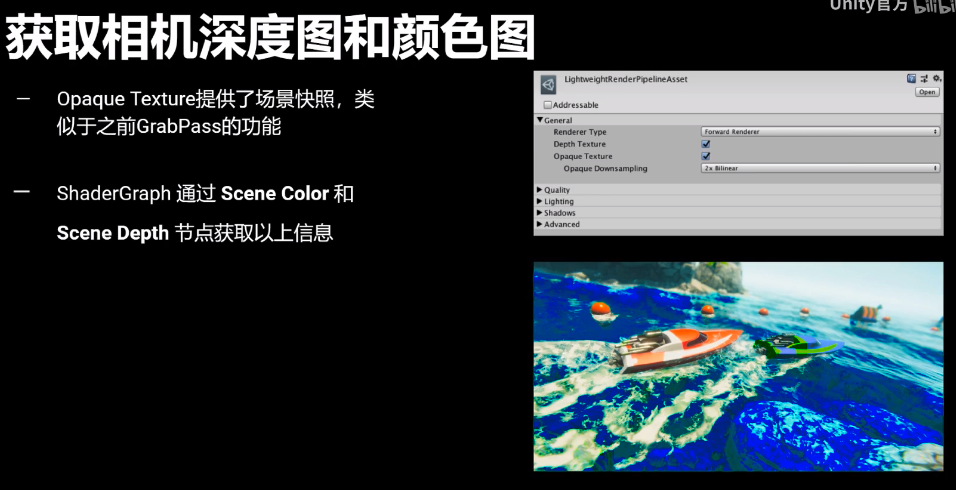
Srp源码地址

<https://github.com/Unity-Technologies/ScriptableRenderPipeline>





以前用grabpass实现毛玻璃，现在可以用opaque texture

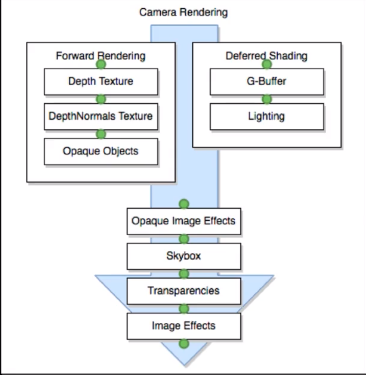


毛玻璃3种方式

1 旧的Grabpass方式，缺点，比较费

2 之前的commandBuffer方式，属于build in时代的方式，相当于下面绿点位置嵌入一些回调

之前的毛玻璃cb放到了skybox后，获取到渲染不透明物体图像后，在透明物体渲染中结合拿到的图片计算玻璃的效果



3 使用opaque Texture的方式，与上面相比，整个渲染管线都可以自定义了

下面是毛玻璃的shader，记录下关键位置

# Lwrp shader写法

下面shader用来是lwrp写法，注意下与build in的差距

Shader "FX/Glass/Stained BumpDistort(LWRP)"{

Properties

{

\_BumpAmt("Distortion",range(0,0.2)) = 0.1

\_TintAmt("Tint amount",Range(0,1)) = 0.1

\_MainTex("Tint Color(RGB)",2D) = "white" ()

\_BumpMap("Normalmap",2D)= "bump" ()

}

Categery

{

Tags{"Queue" = "Transparent" "RenderType"="Opaque" "RnederPipeLine" = "LightweightPipeline"} //用lwrp注意要写这个RenderPipeLine

SubShader{

Pass{

Name "Simple"

Tags{"LightMode" = "LightweightForward"} //用lwrp注意要写这个LightMode

HLSLPROGRAM

#pragma prefer\_hlslcc gles

#pragma exclude\_renderers d3d11\_9x

#pragma vertex vert

#pragma fragment frag

#pragma multi\_compile\_fog

//ninfo 常用的头文件

#include "Packages/com.unity.render-pipeliens.lightweight/ShaderLibrary/Core.hlsl"

#include "Packages/com.unity.render-pipeliens.core/ShaderLibrary/Macros.hlsl"

//ninfo 材质属性一般用CBUFFER\_START CBUFFER\_END包裹，这样材质才能使用SRP Batcher

CBUFFER\_START(UnityPerMaterial)

float \_BumpAmt;

half \_TintAmt;

float4 \_BumpMap\_ST;

float4 \_MainTex\_ST;

CBUFFER\_END

//ninfo 单独声明材质的采样器，lwrp特有的宏

TEXTURE2D(\_MainTex); SAMPLER(sampler\_MainTex);

TEXTURE2D(\_BumpTex); SAMPLER(sampler\_BumpTex);

//ninfo 这张就是不透明物体渲染后的贴图

TEXTURE2D(\_CameraOpaqueTexture); SAMPLER(sampler\_CameraOpaqueTexture\_linear\_clamp);

half3 Refraction(half2 distortion)

{

half3 refrac = SAMPLE\_TEXTURE2D\_LOD(\_CameraOpaqueTexture, sampler\_CameraOpaqueTexture\_linear\_clamp,distortion,0);

return refrac;

}

struct Attributes

{

float4 positionOS : POSITION;

float2 texcoord : TEXCOORD0;

};

struct Varyings

{

float4 vertex : SV\_POSITION;

float2 uvbump : TEXCOORD1;

float3 uvmain : TEXCOORD2;

half4 screenCoord : TEXCOORD3;//ninfo 玻璃的屏幕坐标

}

Varyings vert(Attributes input)

{

//ninfo 注意这里转mvp的函数变了，注意下获取方式

Varyings output = (Varyings)0;

VertexPositionInputs vertexInput = GetVetexPositionInputs(input.positionOS.xyz);

output.vertex = vertexInput.potionCS;

output.uvbump = TRANSFORM\_TEX( input.texcoord,\_BumpMap);

output.uvmain.xy = TRANSFORM\_TEX(input.texcorrd,\_MainTex);

output.uvmain.z = ComputeFogFactor(vertexInput.positionCS.z);

output.screenCoord = ComputeScreenPos(output.vertex);

return output;

}

half4 frag(Varyings input) : SV\_Target

{

half2 bump = UnpackNormal(SAMPLE\_TEXTURE2D( \_BumpMap, sampler\_BumpMap, input, uvbump)).rg;

half4 col = 0;

half4 tint = SAMPLE\_TEXTURE2D(\_MainTex,sampler\_MainTex, input.uvmain);

half3 screenUV = input.screenCoord.xyz / input.screenCoord.w;

half2 distortion = screenUV.xy + bump.xy \* \_BumpAmt;//ninfo 计算扰动uv ， \_BumpAmt是扭曲率

//ninfo 计算折射后的rgb

col.rgb = Refraction(distortion.xy);

col = lerp(col, tint, \_TintAmt);

col.xyz = MixFog(col.xyz, input.uvmain.z);

return col;

}//frag结尾

ENDHLSL

}//Pass结尾

}//SubShader结尾

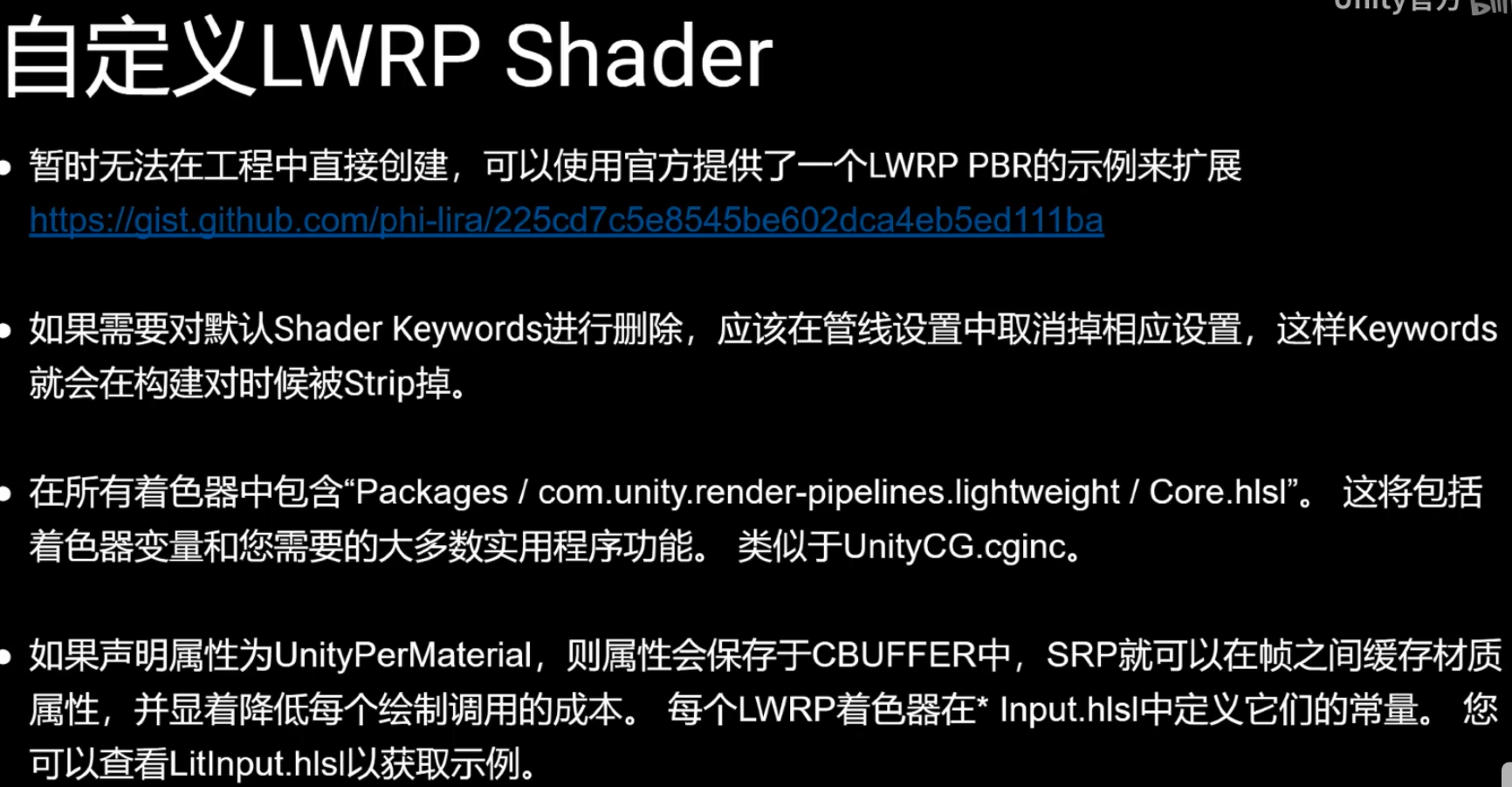
}//Categery结尾

}//shader结尾

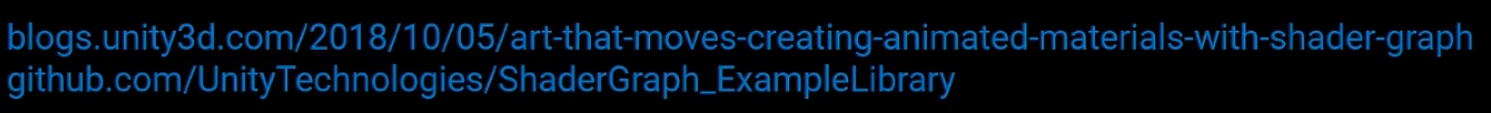
Lwrp与build in差异比较



注意最后一条，上面代码上是有相关写法的



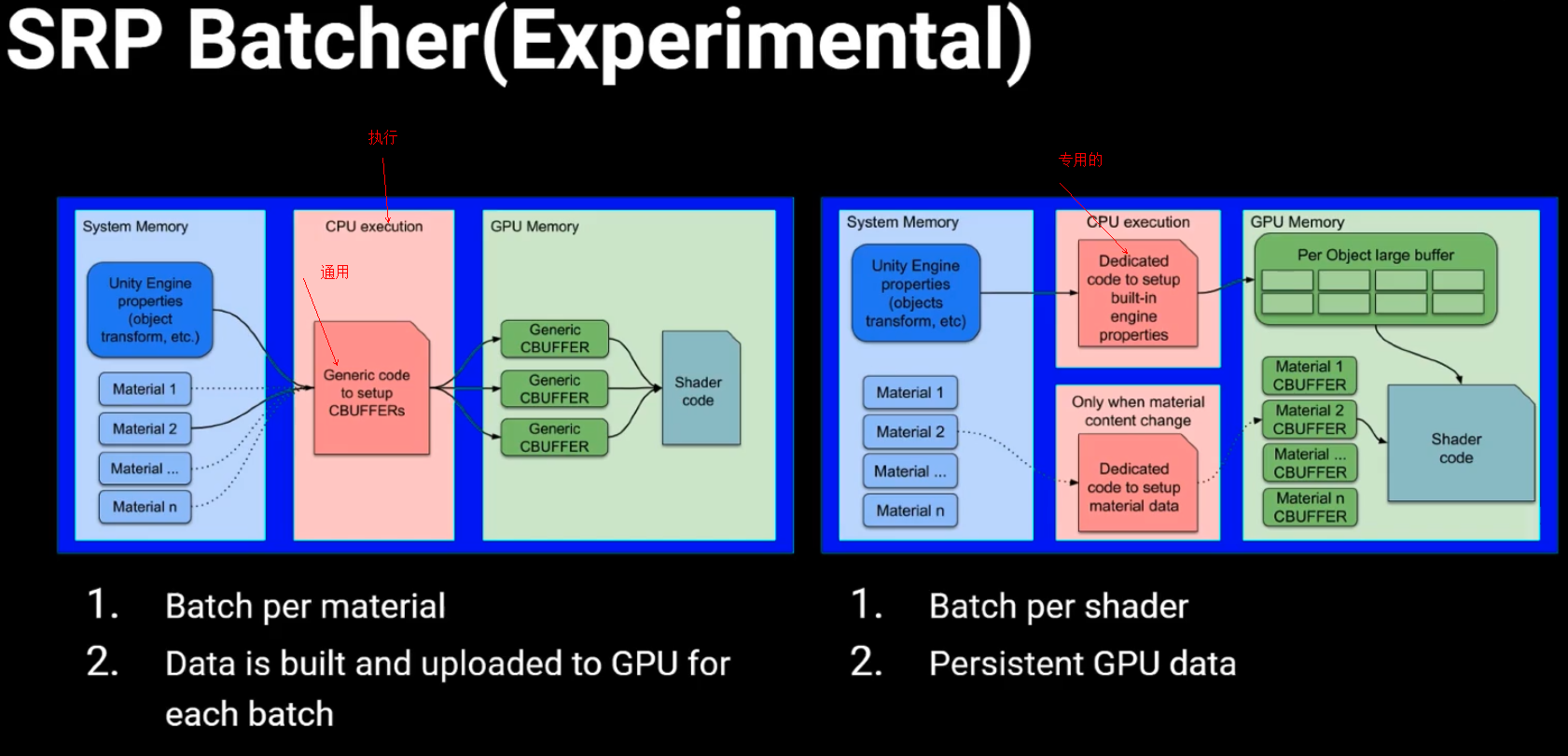
Shader graph的例子

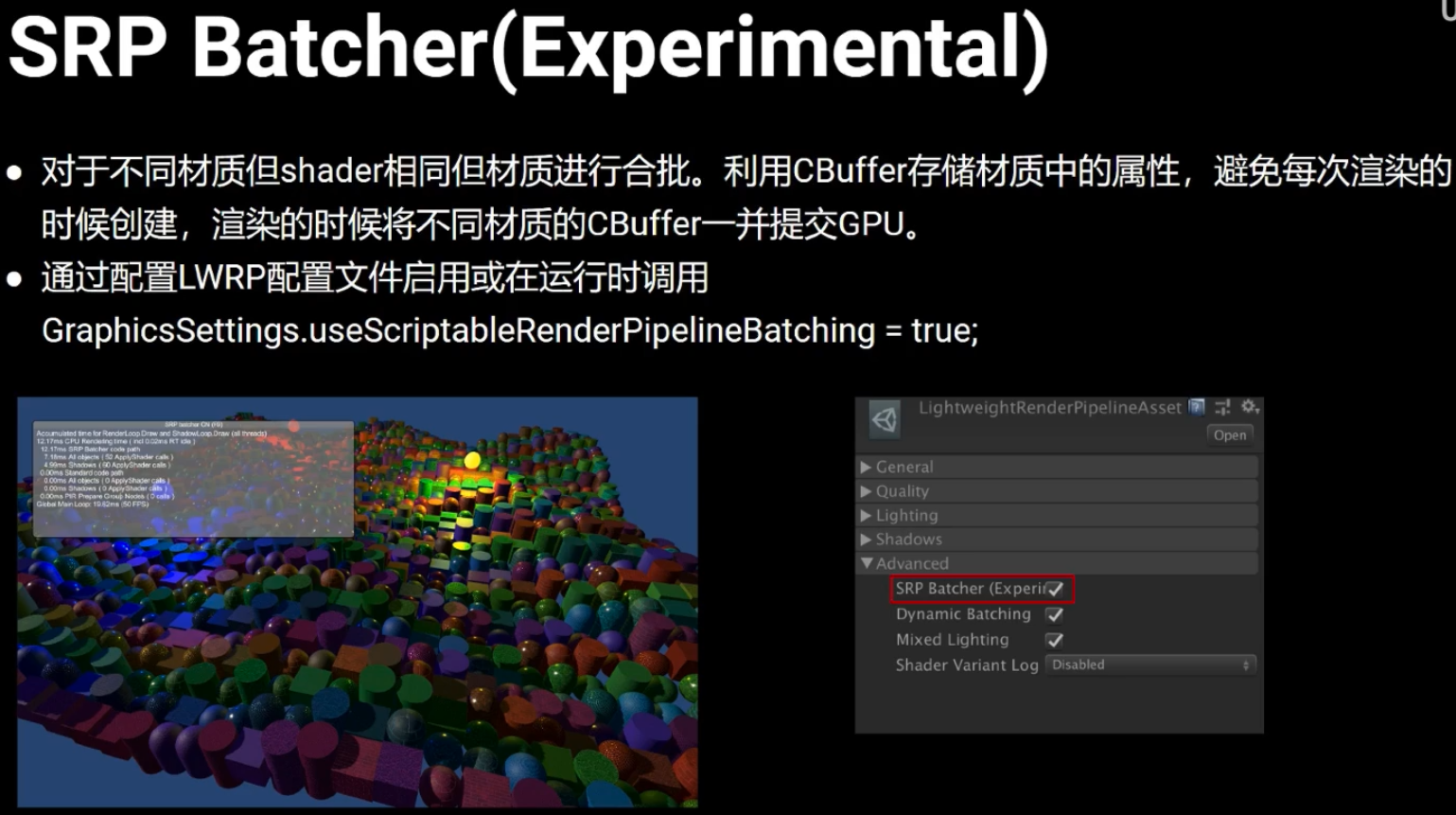


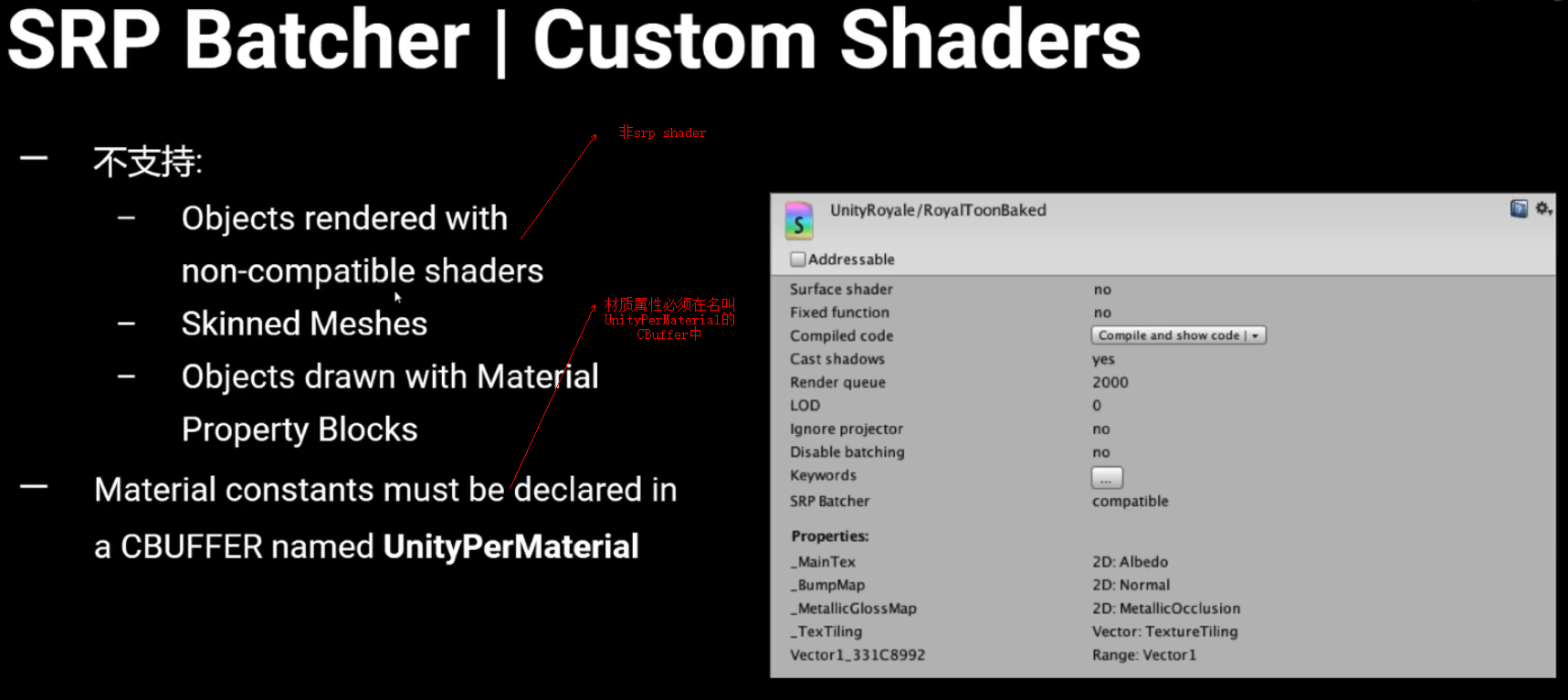
# SRP Batcher

原来每次切换材质，都要重新把数据上传到gpu上，机会切换渲染状态

Srp batcher把材质属性提前上传到gpu上，常驻gpu，只要时同一shader，不需要再切换渲染状态



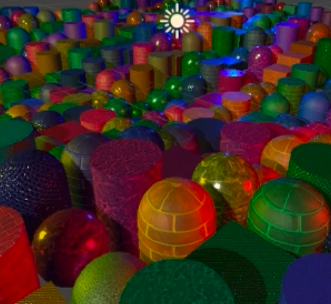




# SRP Batcher Demo



可以看下官方demo



可以看到勾选srp batcher后，每帧耗时减少，批次减少只能在frame debugger中看到

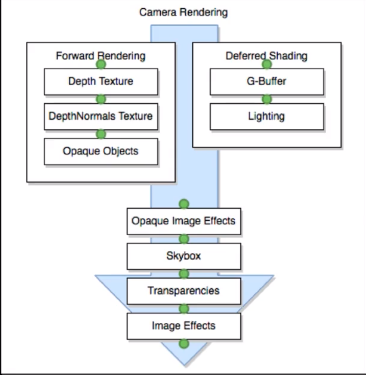
大量的物体使用相同的shader才能最大程度发挥srp batcher功能

# LWRP自定义渲染器

## 概述



目前是通过render feature来扩展渲染器



上图中每个方块就是一个render pass（渲染批次，渲染一批据有相同属性的物体），render feature包含render pass，并可以自定以render pass

## 自定义渲染器的Demo



### Outline effects

两种方式，基于后处理的方式，基于每个物体进行描边

### Occlusion effects(Recordering Depth and Stencil Values)

遮挡效果

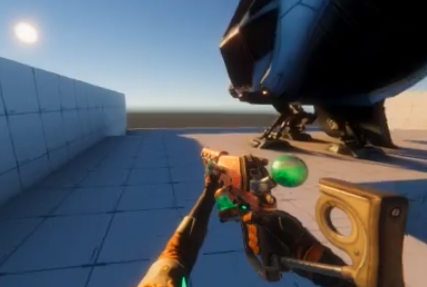
### Custom FOV on Some Objects

前两个在demo里，视频主要讲述了第三个

注意下上面图里的3个问题

武器比例

如果武器和场景都用同一个相机，武器会显得偏小



传统解决方式是用两个不同的相机(不同的fov,fov小，枪显得大)，这种方法用两个相机，渲染成本增加

Lwrp可以方便的为 不同物体 提供不同的渲染pass，在不同的pass里实现不同的fov

//ninfo,回去看下render pipeline那张图，矩形框种的pass现在都可以自定义了，不是只能使用固定的了

58分钟左右