

详解轻量级渲染管线 (LWRP)

成亮
大中华区技术讲师



内容概要

- Unity新闻快报
- LWRP 原理介绍
- LWRP 光照
- LWRP Shader
- LWRP 扩展



2

Unity 新闻快报



unity

Unity 2018.3推出Project Tiny小游戏开发套件

- **功能:** 专注于构建即时2D游戏和可玩式广告。在未来的版本中，会添加额外的功能以构建即时性3D和AR的游戏和体验
- **性能:** 面向数据的ECS架构可以实现卓越的性能。例如：在iPhone 6S上，Tiny运行时可以显示的移动动画精灵数量是其它面向Web的2D引擎的3~4倍，同时还能保持60 帧每秒的运行速度
- **使用:**
Project Tiny通过名为“Tiny Mode”的资源包提供。
安装时，请打开Unity 2018.3 Beta的资源包管理器，选择Preview Packages，然后安装Tiny Mode。



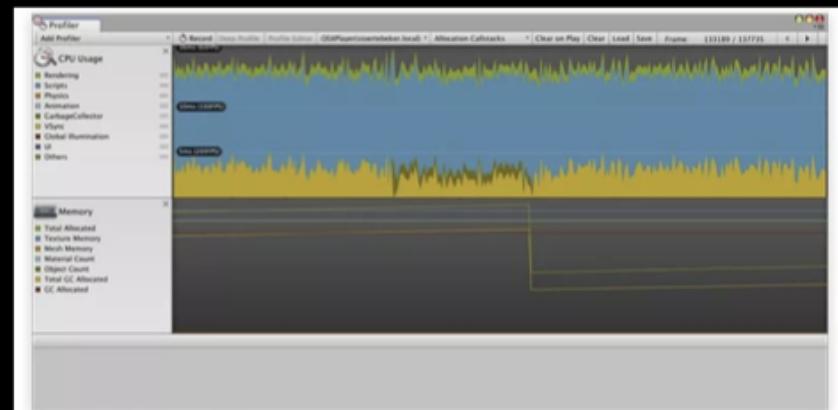
Unity 2019.1 Alpha新功能：增量式垃圾回收

- **功能:** 该方法从整体上不会让垃圾回收过程变快，但它能通过分配工作量到多个帧，显著减少GC峰值对动画流畅性的影响问题

- **性能:** 启用了增量式垃圾回收的情况，相同的项目维持在稳定的60 fps帧率，因为垃圾回收操作分到多个帧完成，只在这几帧使用了少量时间切片

- **使用:**

增量式垃圾回收目前是实验性选项，位于Player Settings窗口的“Other settings”部分，勾选Use incremental GC (Experimental)后，就可以构建版本进行尝试。



BIM数据可视化

- **功能:** 将Revit数据直接导入Unity，让数据能够从世界领先的建筑信息建模工具，导入到行业领先的真实体验3D设计工具，转换为最终可视化解决方案以获得实时体验

- **性能:** 可以加速决策的过程，让客户更加确信他们正在获得理想中的建筑。

- **使用:**

现在你就能通过使用AEC工具包中的PiXYZ工具，导入BIM数据、开发交互式真实体验并发布到多个平台。下载AEC工具包：

<https://unity.com/solutions/unity-aec/industry-bundle>

预计在2019年秋，我们将提供功能全面的解决方案，用于将Revit转换到Unity。



LWRP 原理介绍



“什么是可编程渲染管线（**SRP**）？”

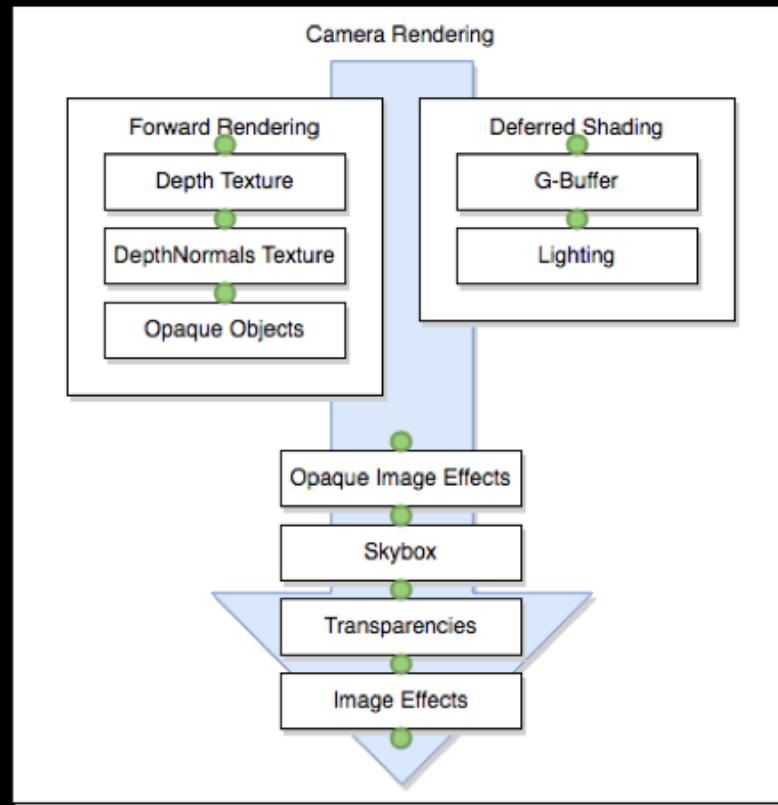


“什么是可编程渲染管线（SRP）？”

“一种通过C#脚本在Unity中配置
和执行渲染的方法”

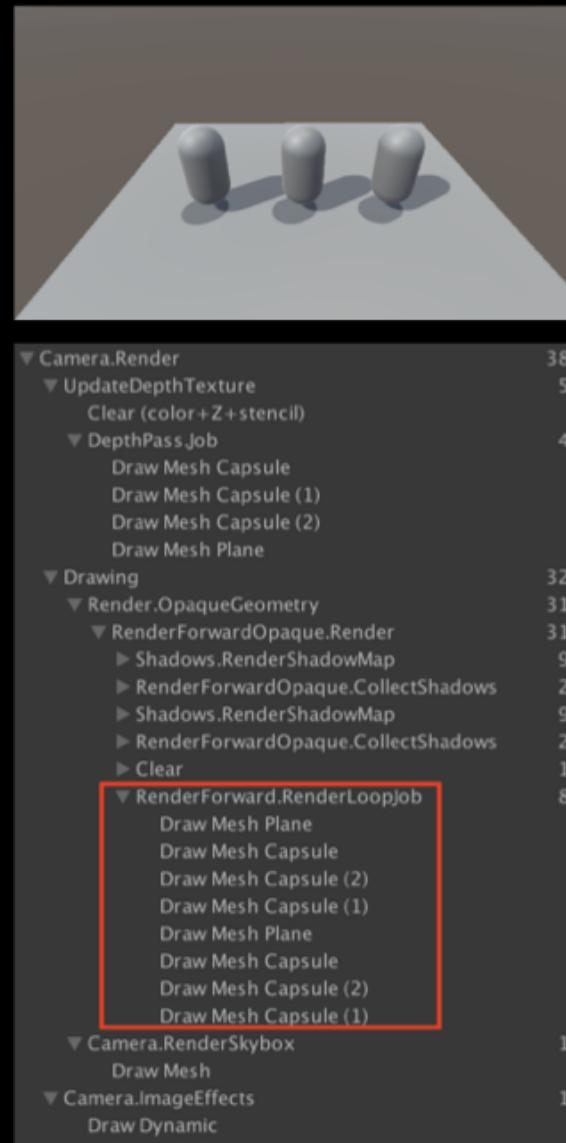


Render Pipeline



Forward Shading

- **原理:** 每个作用于物体的像素光都会单独计算一次，因此draw call会随着物体和光照数量的增加而成倍增加
- **优点:** 不受硬件限制，可以使用MSAA等等
- **缺点:**
 - 1) 光照计算开销会因为光源和物体数量成倍增加
 - 2) 每个物体接受的光照数量有限，通常除了主方向光外最多接受4栈像素光，逼真度受到限制



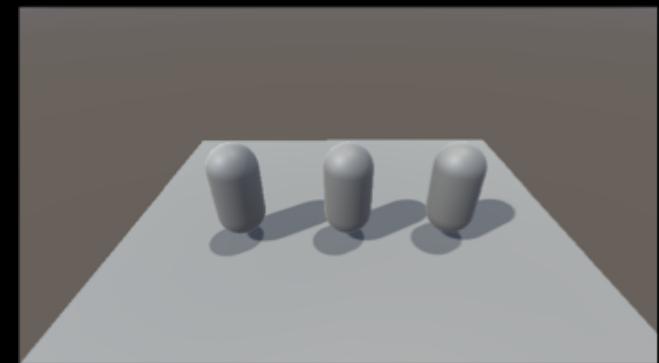
Deferred Shading

- **原理:** 物体颜色，法线，材质等信息先渲染到G-Buffer中，光照最后单独渲染，从而避免了每个物体多个光照批次的问题。

- **优点:** 作用于每个物体的光照数量不再受到限制，而且光照计算不会随着物体增加而增加

- **缺点:**

- 1) 移动设备需要支持OpenGL3.0
- 2) 不支持MSAA，可以使用后处理AA
- 3) 半透明物体仍然使用前向渲染
- 4) 还有其它限制，可以参阅文档



▼ Camera.Render	36
▼ Drawing	35
▼ RenderOpaqueGeometry	34
▼ RenderDeferred.GBuffer	6
► Clear	1
Clear (color+Z+stencil)	
Draw Mesh Plane	
Draw Mesh Capsule	
Draw Mesh Capsule (2)	
Draw Mesh Capsule (1)	
► RenderDeferred.CopyDepth	1
► RenderDeferred.Relections	2
► RenderDeferred.RelectionsToEmissive	1
▼ RenderDeferred.Lighting	24
▼ RenderDeferred.Light	24
► Shadows.RenderShadowMap	9
► Shadows.CollectShadows	2
Draw GL	
► Shadows.RenderShadowMap	9
► Shadows.CollectShadows	2
Draw GL	
▼ Camera.RenderSkybox	1
Draw Mesh	
▼ Camera.ImageEffects	1
Draw Dynamic	

案例



Cities:Skylines 由于场景中有大量的小范围光照，使用了Deferred Shading，确保了游戏性能

Command Buffer

Command Buffer: 用于扩展Unity的渲染管线。

Command Buffer包含一系列渲染命令，比如设置渲染目标，绘制网格等等，并且可以设置为在摄像机期间的各个点执行渲染。

```
buf = new CommandBuffer();
buf.name = "Grab screen and blur";
m_Cameras[cam] = buf;

// copy screen into temporary RT
int screenCopyID = Shader.PropertyToID("_ScreenCopyTexture");
buf.GetTemporaryRT (screenCopyID, -1, -1, 0, FilterMode.Bilinear);
buf.Blit (BuiltinRenderTextureType.CurrentActive, screenCopyID);

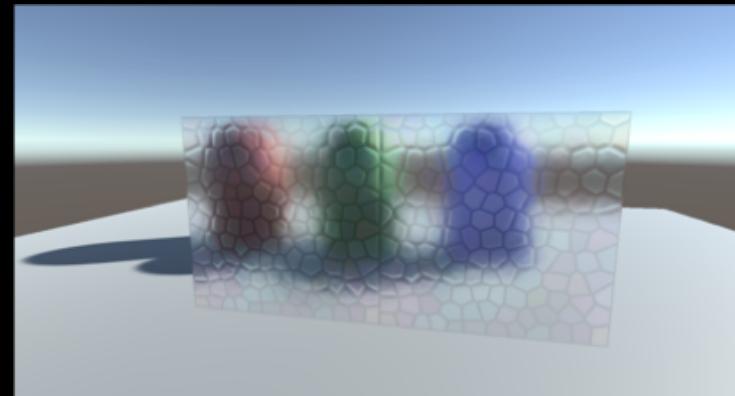
// get two smaller RTs
int blurredID = Shader.PropertyToID("_Temp1");
int blurredID2 = Shader.PropertyToID("_Temp2");
buf.GetTemporaryRT (blurredID, -2, -2, 0, FilterMode.Bilinear);
buf.GetTemporaryRT (blurredID2, -2, -2, 0, FilterMode.Bilinear);

// downsample screen copy into smaller RT, release screen RT
buf.Blit (screenCopyID, blurredID);
buf.ReleaseTemporaryRT (screenCopyID);

// horizontal blur
buf.SetGlobalVector("offsets", new Vector4(2.0f/Screen.width,0,0,0));
buf.Blit (blurredID, blurredID2, m_Material);
// vertical blur
buf.SetGlobalVector("offsets", new Vector4(0,2.0f/Screen.height,0,0));
buf.Blit (blurredID2, blurredID, m_Material);
// horizontal blur
buf.SetGlobalVector("offsets", new Vector4(4.0f/Screen.width,0,0,0));
buf.Blit (blurredID2, blurredID, m_Material);
// vertical blur
buf.SetGlobalVector("offsets", new Vector4(0,4.0f/Screen.height,0,0));
buf.Blit (blurredID2, blurredID, m_Material);

buf.SetGlobalTexture("GrabBlurTexture", blurredID);

cam.AddCommandBuffer (CameraEvent.AfterSkybox, buf);
```



▼ Camera.Render	29
► UpdateDepthTexture	5
▼ Drawing	22
► RenderOpaqueGeometry	13
► Camera.RenderSkybox	1
▼ CommandBuffer.AfterSkybox	7
▼ Grab screen and blur	7
▼ RenderTexture.ResolveAA	1
Resolve Color	
Draw Dynamic	
► Render.TransparentGeometry	1
► Camera.ImageEffects	2

为什么SRP?

- 使Unity的渲染不再是黑盒子!
- 为游戏/项目选择更多的渲染方式
- 通过公开的方法和参数进行更多定制
- 对于渲染执行的时机和方式有更多的控制



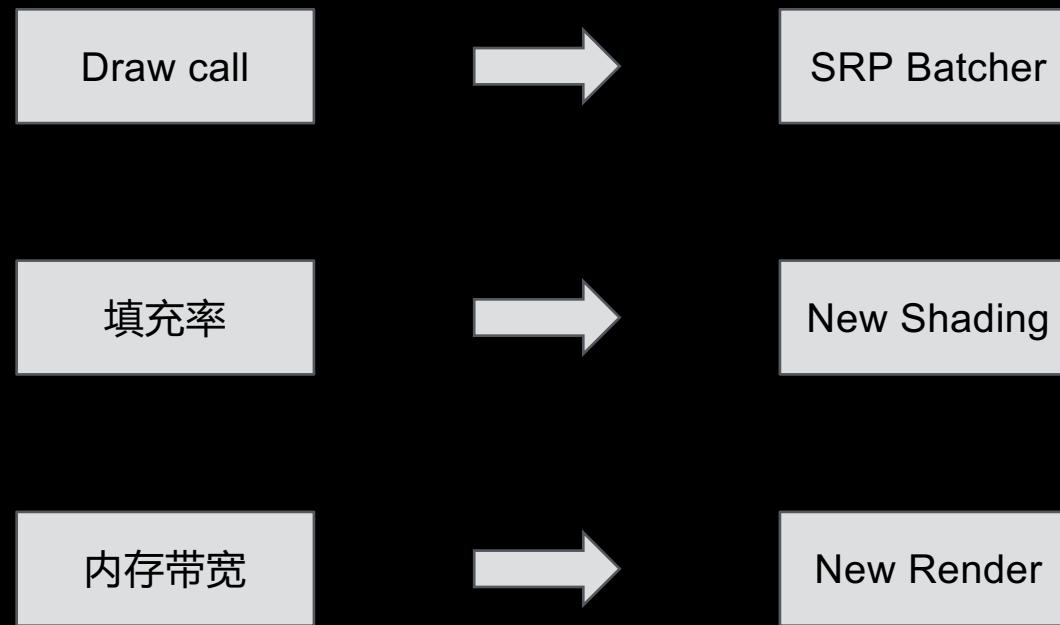
可编程渲染管线(SRP)

- Built-In Renderer
 - Forward & Deferred
 - 当前默认管线
- 可编程渲染管线 (SRP) API
 - 高清渲染管线 (HDRP)
 - 轻量级渲染管线 (LWRP)
 - 自定义渲染管线

可编程渲染管线(SRP)

- Built-In Renderer
 - Forward & Deferred
 - 当前默认管线
- 可编程渲染管线 (SRP) API
 - 高清渲染管线 (HDRP)
 - 轻量级渲染管线 (LWRP)
 - 自定义渲染管线

LWRP对性能的改进



LWRP特性

- 精简优化过的渲染管线
 - 聚焦于性能
 - 针对移动和XR平台
- Single-Pass Forward Rendering
 - 高性能和高一致性的 PBR
 - 新的Shader库
 - 一次渲染多个实时光!
 - 可插入Scriptable Render Passes的API

LWRP特性

- 基于C# 的源码
 - 渲染不再是黑盒
 - 渲染过程可读
 - 可以在GitHub获取!
- 图形功能可定制化
 - 修改
 - 增加
 - 删除
 - 学习!



Boat Attack - LWRP Demo Project



github.com/Verasl/BoatAttack

Trash Dash - LWRP Version



github.com/Unity-Technologies/EndlessRunnerSampleGame

Shader Graph Demo Projects



blogs.unity3d.com/2018/10/05/art-that-moves-creating-animated-materials-with-shader-graph
github.com/UnityTechnologies/ShaderGraph_ExampleLibrary

Demo Time!

How to setup a project to use Lightweight Render Pipeline

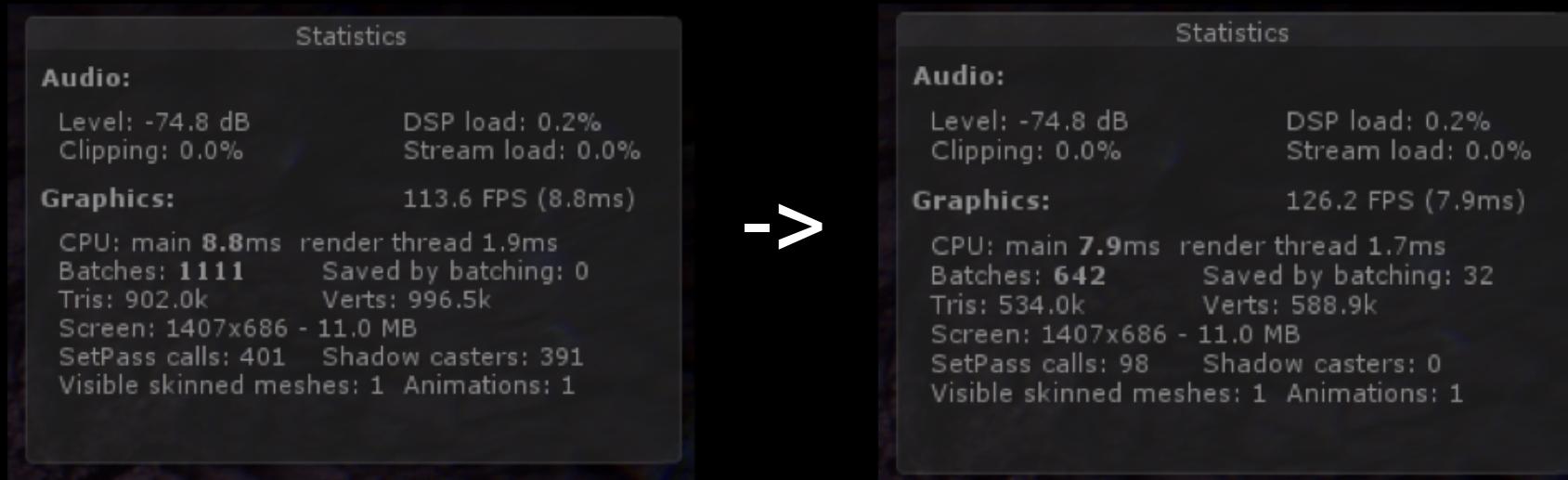
Built-In Renderer -> LWRP



->



Built-In Renderer -> LWRP



光照

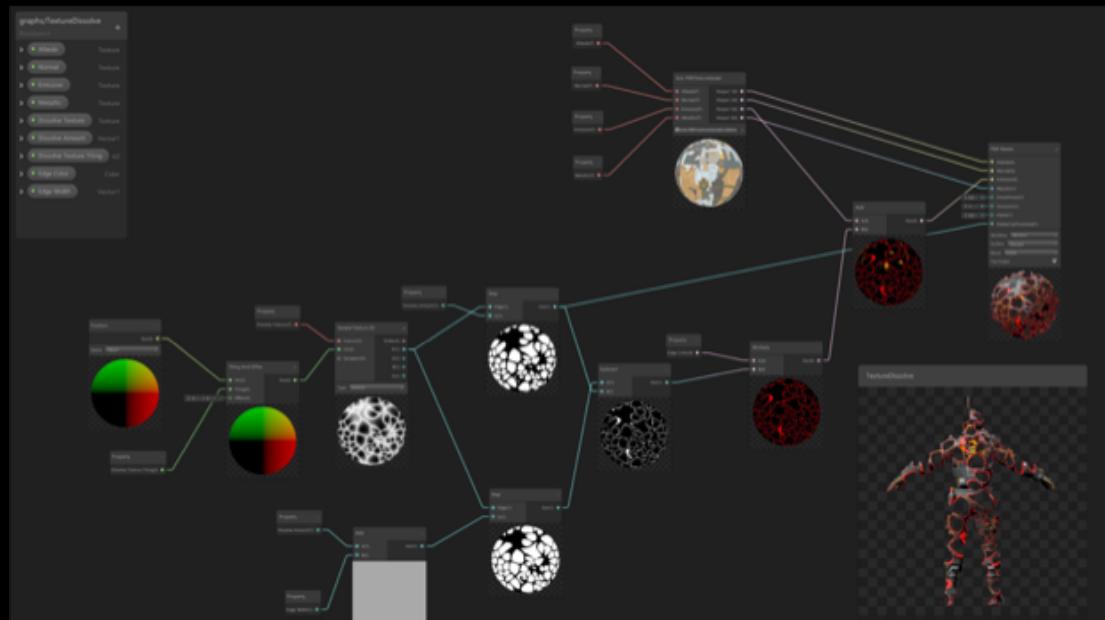
- 实时光照
 - 按照每个物体进行光照射剔除
 - 在一个批次进行渲染
 - 针对每个物体的限制:
 - 1 盏主方向光
 - 4 盏附加光 (Point/Spot)
 - 每个相机中总共16盏可见光
- 基于物理的光线衰减
 - 只受强度控制，不受范围影响

LWRP Shader



Shaders

- 新的Shader库!
 - Core 和 Lightweight RP 库
 - 更小的‘chunks’
- 兼容性
 - 插件中默认提供LWRP兼容Shaders
 - 没有提供 Surface Shaders
 - Unlit Built-In Shaders 仍然可以渲染!
- Shader Graph
 - 自动生成 LWRP Shaders!
 - 可以 查看/编辑 生成的Shader代码!

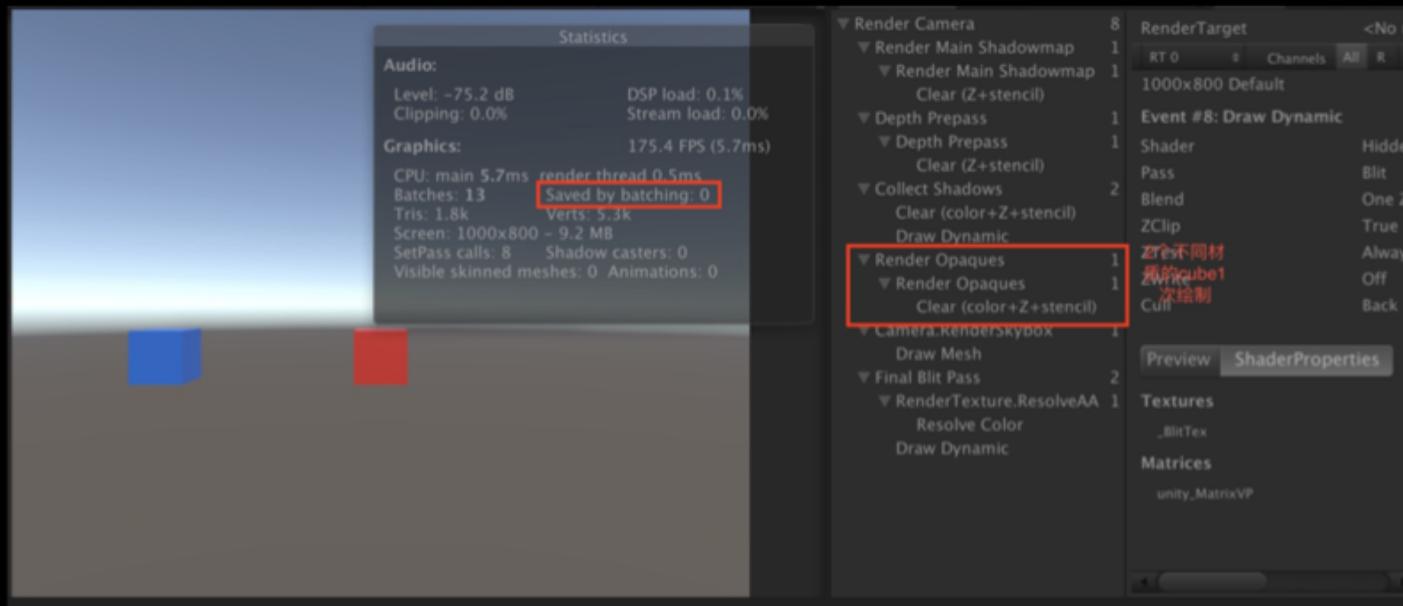


自定义LWRP Shader

- 暂时无法在工程中直接创建，可以使用官方提供了一个LWRP PBR的示例来扩展
<https://gist.github.com/phi-lira/225cd7c5e8545be602dca4eb5ed111ba>
- 如果需要对默认Shader Keywords进行删除，应该在管线设置中取消掉相应设置，这样Keywords就会在构建对时候被Strip掉。
- 在所有着色器中包含“Packages / com.unity.render-pipelines.lightweight / Core.hlsl”。这将包括着色器变量和您需要的大多数实用程序功能。类似于UnityCG.cginc。
- 如果声明属性为UnityPerMaterial，则属性会保存于CBUFFER中，SRP就可以在帧之间缓存材质属性，并显着降低每个绘制调用的成本。每个LWRP着色器在* Input.hlsl中定义它们的常量。您可以查看LitInput.hlsl以获取示例。

SRP Batcher(Experimental)

- 对于不同材质但 shader 相同但材质进行合批。利用 CBuffer 存储材质中的属性，避免每次渲染的时候创建，渲染的时候将不同材质的 CBuffer 一并提交 GPU。
- 需要在运行时调用 `GraphicsSettings.useScriptableRenderPipelineBatching = true;`



Shaders stripping

LWRP增加了如下**Stripping**的规则：

- 基于LWRP的设置。比如：如果禁用Additional Lights，LWRP会从构建中删除所有相关变体。
- 剥离无效的变体组合。比如：已定义 _MAIN_LIGHT_SHADOWS_CASCADE但是 _MAIN_LIGHT_SHADOWS无效，因此会被剥离
- 剥离未使用的Pass。如果您在配置中选择不支持阴影，则会从构建中剥离所有Shadow caster pass。

Shader Keywords

- **定义:** 用于预处理指令`#pragma multi_compile` 或者 `#pragma shader_feature` 中，通过编译产生不同的Shader Variants。比如：`#pragma multi_compile FANCY_STUFF_OFF FANCY_STUFF_ON` 预处理指令中，`FANCY_STUFF_OFF` 和 `FANCY_STUFF_ON` 就是两个Shader Keywords，在编译后会生成两个Shader Variants。
- **用法:** 在运行时，从Material关键字（`Material.EnableKeyword`和`DisableKeyword`）或全局着色器关键字（`Shader.EnableKeyword`和`DisableKeyword`）中选取适当的着色器变体。
- **优点:** 提升Shader 代码的复用性
- **缺点:** 过多的关键字组合会导致编译出的过多冗余的Shader Variants，造成编译时间过长，包体过大，和加载变慢等问题

Shader Variants

- 项目中 Shader Variant 总量计算公式：

$$TotalShadersVariants = \sum_{a=1}^{ShaderAssets} \sum_{s=1}^{SubShaders} \sum_{p=1}^{Passes} \left(Stages_{(a,s,p)} \prod_{d=1}^{Directives_{(a,s,p)}} Keywords_{(a,s,p,d)} \right)$$

- 单个 Shader 的 Variant 数量计算公式：

$$ShaderVariants = 1 \sum_{s=1}^{SubShaders} \sum_{p=1}^{Passes} \left(Stages_{(s,p)} \prod_{d=1}^{Directives_{(s,p)}} Keywords_{(s,p,d)} \right)$$

Shader Variants计算示例

```
SubShader
{
    Pass
    {
        Name "ShaderVariantsStripping/Pass"

        CGPROGRAM
        #pragma vertex vert
        #pragma fragment frag
        #pragma multi_compile COLOR_ORANGE COLOR_VIOLET COLOR_GREEN COLOR_GRAY
        #pragma multi_compile OP_ADD OP_MUL OP_SUB

        struct appdata
        {
            float4 vertex : POSITION;
            float2 uv : TEXCOORD0;
        };

        struct v2f
        {
            float2 uv : TEXCOORD0;
            float4 vertex : SV_POSITION;
        };

        sampler2D _MainTex;
        float4 _MainTex_ST;

        v2f vert (appdata v)
        {
            v2f o;
            o.vertex = UnityObjectToClipPos(v.vertex);
            o.uv = v.uv;
            return o;
        }
    }

    fixed4 get_color()
    {
        #if defined(COLOR_ORANGE)
            return fixed4(1.0, 0.5, 0.0, 1.0);
        #elif defined(COLOR_VIOLET)
            return fixed4(0.8, 0.2, 0.8, 1.0);
        #elif defined(COLOR_GREEN)
            return fixed4(0.5, 0.9, 0.3, 1.0);
        #elif defined(COLOR_GRAY)
            return fixed4(0.5, 0.9, 0.3, 1.0);
        #else
            #error "Unknown 'color' keyword"
        #endif
    }

    fixed4 frag (v2f i) : SV_Target
    {
        fixed4 diffuse = tex2D(_MainTex, i.uv);
        fixed4 color = get_color();

        #if defined(OP_ADD)
            return diffuse + color;
        #elif defined(OP_MUL)
            return diffuse * color;
        #elif defined(OP_SUB)
            return diffuse - color;
        #else
            #error "Unknown 'op' keyword"
        #endif
    }
} ENDCG
}
```

$$\text{ColorPass} = 2 \times (4 \times 3) = 24 \text{ shader variants}$$

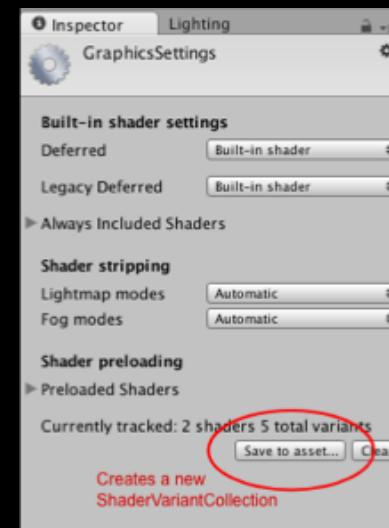
优化Shader加载时间

- **Shader build time stripping:** 1) 使用#pragma

shader_feature的着色器，如果所使用的Material都不使用特定Variant，则它不会包含在构建中，标准着色器就使用此功能。2) 如果场景中没有用到Fog 和 Lightmapping，则相应的 Shader variants 不会包含在构建中。通过这些方式，可以很大程度减少shader variants的数量。例如，完全编译的标准着色器需要几百兆字节，但在典型的项目中，它通常最终只占用几兆字节（并且通常由应用程序打包过程进一步压缩）。

- **Shader Variants Collection:** ShaderVariantCollection

是一种资产，包含着色器列表，以及每个要加载Shader的Pass类型和Shader keywords组合的列表。主要用于提前加载该列表中的shader variants，从而避免在运行时使用到该variant再进行加载造成卡顿。

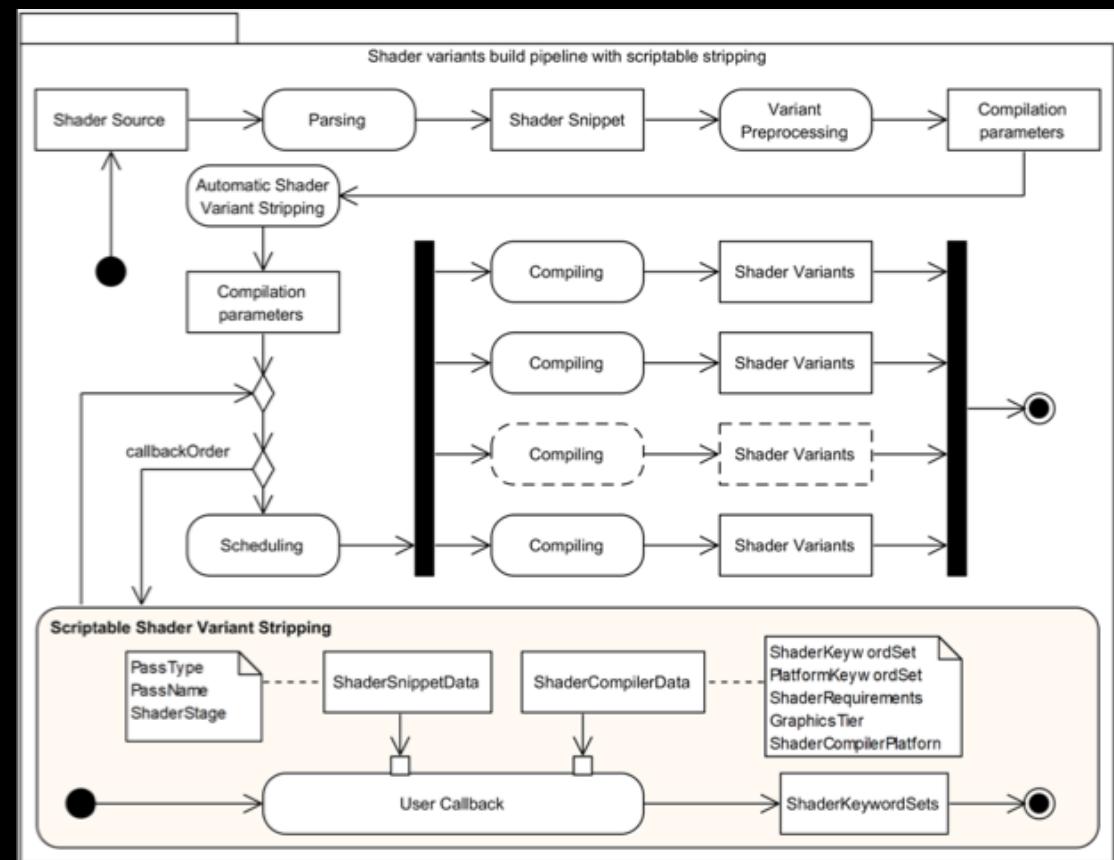


从编辑器使用的着色器创建ShaderVariantCollection

Scriptable Shader Variants Stripping

Scriptable shader variants

stripping提供了一套API给用户，可以让用户在系统提供的Shader variant stripping之后，自定义stripping过程，比如删除掉某些Shader Keywords。这种方式可以让用户更大程度的减少Shader variants的数量。



Scriptable Shader Variants Stripping 示例

该脚本可以剥离与“DEBUG”配置关联的所有着色器变体，这些变量由开发Player构建中使用的“DEBUG”关键字标识

```
2  using UnityEditor;
3  using UnityEditor.Build;
4  using UnityEditor.Rendering;
5  using UnityEngine;
6  using UnityEngine.Rendering;
7
8 // Simple example of stripping of a debug build configuration
9 class ShaderDebugBuildProcessor : IPreprocessShaders
10 {
11     ShaderKeyword m_KeywordDebug;
12
13     public ShaderDebugBuildProcessor()
14     {
15         m_KeywordDebug = new ShaderKeyword("DEBUG");
16     }
17
18     // Multiple callback may be implemented.
19     // The first one executed is the one where callbackOrder is returning the smallest number.
20     public int callbackOrder { get { return 0; } }
21
22     public void OnProcessShader(
23         Shader shader, ShaderSnippetData snippet, IList<ShaderCompilerData> shaderCompilerData)
24     {
25         // In development, don't strip debug variants
26         if (EditorUserBuildSettings.development)
27             return;
28
29         for (int i = 0; i < shaderCompilerData.Count; ++i)
30         {
31             if (shaderCompilerData[i].shaderKeywordSet.IsEnabled(m_KeywordDebug))
32             {
33                 shaderCompilerData.RemoveAt(i);
34                 --i;
35             }
36         }
37     }
38 }
```

LWRP 扩展



unity

LWRP源码

- 在GitHub上获取!

github.com/Unity-Technologies/ScriptableRenderPipeline

- 基于 C# 编写

- 可读!
 - 可下载可修改
 - 可作为自定义管线的基础

- DefaultRendererSetup.cs

- 负责渲染批次的注册逻辑.

Scriptable Render Pass

- 获取 Render State, 修改, 插入!
 - 不需要另外的相机来专门渲染特效!
 - 使用Command Buffers
- 特定插入点:
 - IAfterDepthPrePass
 - IAfterOpaquePass
 - IAfterTransparentPass
 - IAfterSkyboxPass
 - IAfterRender



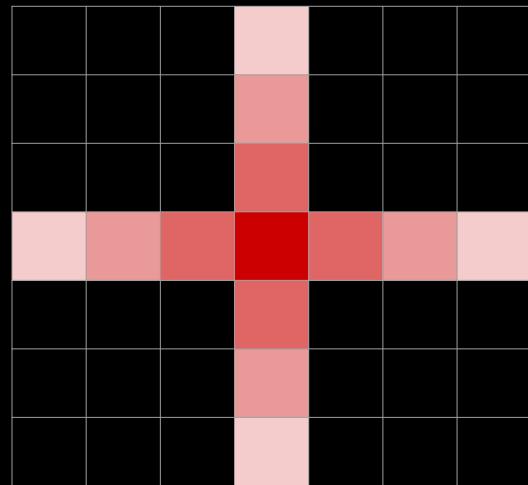
github.com/johnsietsma/ExtendingLWRP

Demo Time!

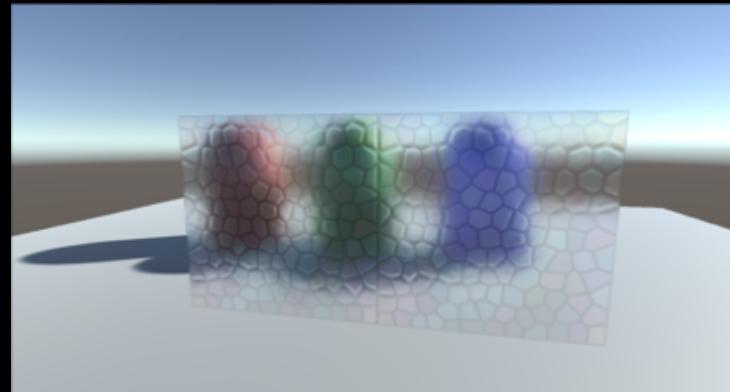
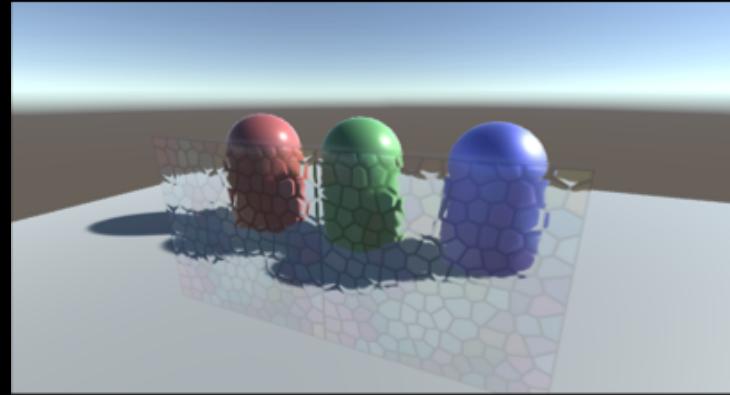
Implementing Lightweight Render Pipeline features!

简单高斯模糊

高斯模糊: 从数学的角度来看, 图像的高斯模糊过程就是图像与[正态分布](#)做卷积。

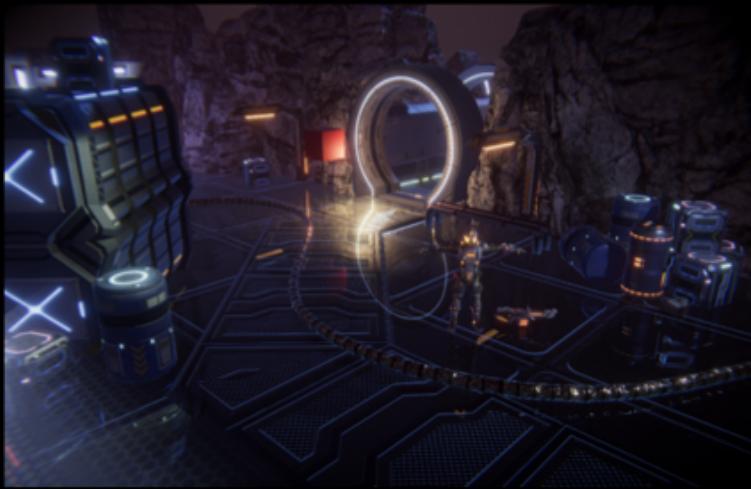


将每个像素点的颜色通过高斯分布到左右上下点从而达到模糊的效果



Planar Reflection

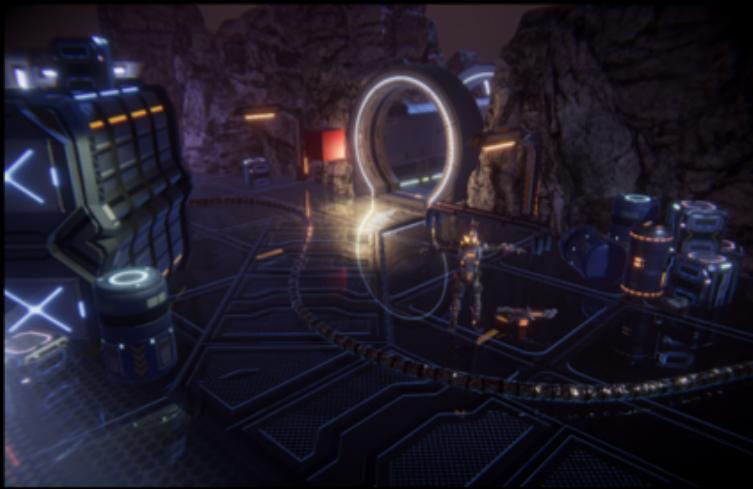
- 继承`IBeforeCameraRender`接口，可以在主相机渲染之前执行操作
- 创建镜像相机，渲染出镜像效果到`RenderTexture`中。



▼ Render Camera	581
► Render Opaques	133
► Camera.RenderSkybox	6
► Render Transparents	5
► Render Main Shadowmap	185
► Render Additional Shadows	82
► Render Opaques	136
► Custom Blur Pass	4
► Camera.RenderSkybox	6
► Copy Depth	1
► Copy Color	1
► Render Transparents	8
► Render PostProcess Effects	14

Planar Reflection

- 继承`IBeforeCameraRender`接口，可以在主相机渲染之前执行操作
- 创建镜像相机，渲染出镜像效果到`RenderTexture`中。



▼ Render Camera	
▶ Render Opaques	133
▶ Camera.RenderSkybox	6
▶ Render Transparents	5
▶ Render Main Shadowmap	185
▶ Render Additional Shadows	82
▶ Render Opaques	136
▶ Custom Blur Pass	4
▶ Camera.RenderSkybox	6
▶ Copy Depth	1
▶ Copy Color	1
▶ Render Transparents	8
▶ Render PostProcess Effects	14

学习资料

资源:

- Scriptable Render Pipeline: github.com/Unity-Technologies/ScriptableRenderPipeline
- LWRP 4.0.1 Forum Release Notes: forum.unity.com/threads/lwrp-4-0-1-preview-is-out.562291/
- LWRP Shader Differences Guide: github.com/johnsietsma/ExtendingLWRP/wiki/Shaders
- Custom Render Pipeline Tutorial: catlikecoding.com/unity/tutorials/scriptable-render-pipeline/

项目:

- Boat Attack Demo: github.com/Verasl/BoatAttack/
- Extending LWRP: Blurry Refractions Example: github.com/johnsietsma/ExtendingLWRP
- Trash Dash - LWRP Version: github.com/Unity-Technologies/EndlessRunnerSampleGame

Unity开发者圣诞派对 邀请函

地点

Unity大中华区总部（上海）

时间

2018年12月28日

18:00 - 20:30

嘉宾

杨栋 Unity平台部技术总监

张黎明 Unity技术总监

Unity开发者圣诞派对是一场
针对Unity订阅用户的专属交流会。
每一位受邀的开发者都有机会
和特邀嘉宾交流开发难题
提问自己最关心的开发难题
交换创作思路

资源插件 圣诞礼包

(福利截止日期：12月19日)



起步开发者圣诞礼包

（总价最高达1500元）

- Unity圣诞派对邀请函
（仅限每日前5位订阅用户）
- 礼包中任选1件资源插件
- 圣诞惊喜折扣
（优惠码XMAS2018）

**Unity Plus加强版新用户
每预付订阅一个席位可获得**

专业开发者圣诞礼包

（总价超过5000元）

- Unity圣诞派对邀请函
- 6件资源插件圣诞礼包
- Unity Icon Collective
第一期资源包

**Unity Pro专业版新用户
每订阅一个席位可获赠**

Unity Icon Collective 资源包赠礼



最新推出的第一期明星资源包——由游戏界资深人士
开发的**Unity Icon Collective**（总价超过1700元），
活动期间订阅**Unity Pro**专业版免费获取，助您实
现AAA级场景。

12月1日 - 12月25日

订阅成功的用户可获得包含

Unity开发者

圣诞派对邀请函

在内的圣诞大礼包



Unity在线客服微信

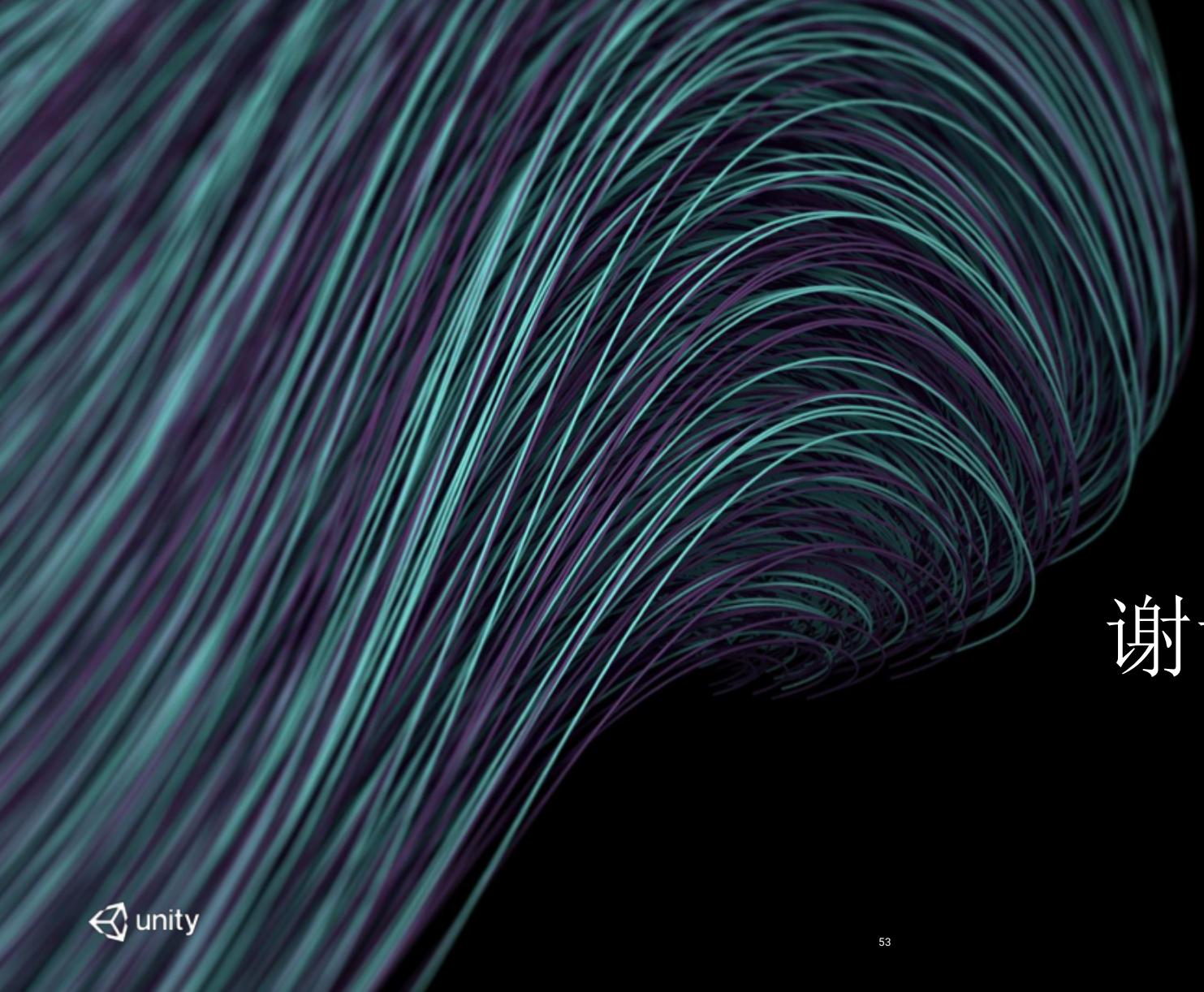


Unity线上商店



有奖调查问卷





谢谢大家！

