**CustomRenderPipelineAsset**

渲染资产

# 序列化资产

## 批处理相关

### useDynamicBatching

### useGPUInstancing

### useSRPBatcher

## 阴影设置

### shadows

# 创建渲染管线

## CreatePipeline()

### 返回一个pipline管线

# CustomRenderPipeline

渲染管线

## 成员

### renderer

* 渲染器

### useDynamicBatching

### useGPUInstancing

### shadowSettings

## 构造函数

### 设置系统对SRP Batch的支持

* GraphicsSettings.useScriptableRenderPipelineBatching = useSRPBatcher;

### 灯光使用线性强度

* GraphicsSettings.lightsUseLinearIntensity = true;

## 每一帧都会调用这个方法进行画面渲染 该方法是SRP的入口，调用渲染器进行渲染

### Render(ScriptableRenderContext context, Camera[] cameras)

# CameraRenderer

相机渲染管理类：单独控制每个相机的渲染

## 成员

### context

* 渲染上下文

### camera

* 当前渲染的相机

### bufferName

### buffer

* CommandBuff，默认名字为bufferName

### cullingResults

* 存储相机剔除后的结果，核心方法

### light mode tag

* unlitShaderTagId

• ShaderTagId("SRPDefaultUnlit")

* litShaderTagId

• ShaderTagId("CustomLit")

### lighting

* 光照实例

## 核心渲染方法，每帧都会调用

### Render()

* 初始化context
* 初始化camera
* PrepareBuffer()

• 如果编辑器模式采用相机名作为采样名，否则用bufferName

* PrepareForSceneWindow()

• 再game视图绘制的几何体也绘制到Scene试图中

* Cull()

• 进行空间剔除，将场景中不会被相机扫描到的空间里的物体剔除，同时将ShadowDistance之外的物体的ShadowCaster属性设置为false，剔除后剩下的渲染实体放在cullingResults对象中

* 采样（相机名）
* lighting.Setup(context, cullingResults, shadowSetting);

• 光源数据和阴影数据发送到GPU计算光照

* 结束采样
* Setup()

• 设置相机的属性和矩阵，同时根据flag清理深度，模版，颜色缓冲等

* 采样（相机名）
* DrawVisibleGeometry(useDynamicBatching, useGPUInstancing);

• 绘制tag为unlitShaderTagId和litShaderTagId的几何体

• 1.绘制不透明物体

• 2.绘制天空盒

• 3.绘制半透明物体

* DrawUnsupportedShaders();

• 绘制SRP不支持的着色器类型

* DrawGizmos()

• 绘制Gizmos

* lighting.Cleanup()

• 释放阴影深度缓冲

* Submit()

• 结束采样，context提交

# ShadowSettings

## MaxDistance

### 阴影最大距离

## TextureSize

### 阴影贴图大小

## Directional

### 方向光数据

### atlasSize

## directional

### 默认为1024

# Lighting

## 成员

### maxDirLightCount

* 最大可见平行光数量

### shaderTagId

* dirLightCountId

• PropertyToID("\_DirectionalLightCount")

* dirLightColorsId

• PropertyToID("\_DirectionalLightColors")

• 每个方向光的颜色

* dirLightDirectionsId

• PropertyToID("\_DirectionalLightDirections")

• 每个方向光的方向

* dirLightShadowDataId

• PropertyToID("\_DirectionalLightShadowData")

• 每个灯光的阴影数据

### cullingResults

* 剔除后的渲染实体对象

### shadows

* 阴影对象

## Setup

初始化设置

### 1. 存储cullingResults

### 2.采样Lighting

### 3.shadows.Setup(context, cullingResults, shadowSetting);

* 传递阴影数据

### 4.SetupLights();

* 存储并发送所有光源数据

• 1.获取当前所有可见光

• 2.遍历所有可见光

• 存储光颜色

desc

• 存储光到方向

desc

• shadows.ReserveDirectionalShadows

desc

• 存储阴影数据

• 3.存储光源数量dirLightCountId

desc

• 4.存储光源颜色dirLightColorsId

desc

• 5.存储光源方向dirLightDirectionsId

desc

• 6.存储阴影数据dirLightShadowDataId（偏移量，光源强度）

desc

### 5 shadows.Render();

* 渲染阴影

### 6. 结束采样Lighting

### 7 执行buffer并清理

## Cleanup

### 清理阴影

# Shadows

## SetUp

### 初始化 context

### 初始化 cullingResults

### 初始化shadowsetting

### 初始化阴影光源的数量

## ReserveDirectionalShadows

### 是否超出最大支持的阴影数量

### 光源是否支持阴影

### 光源强度是否大于0

### 是否在最大投影范围内

### 初始化ShadowedDirectionalLights

### 返回Vector3

* light.shadowStrength
* shadowSetting.directional.cascadeCount \* ShadowedDirectionalLightCount++

• 集联阴影的第一个位置

* light.shadowNormalBias

• 法线偏移量（顶点在世界空间沿着法线偏移，修正阴影瑕疵的一种手段）

## Render

### 渲染定向光阴影

* 1.创建RT， 并指定为阴影贴图
* 2.阴影贴图绑定到dirShadowAtlasId
* 3.清理深度缓冲
* 4.采样“Shadows”
* 5.根据支持阴影的光源数量对阴影贴图进行切片

• 1.

* 6.遍历所有方向光渲染阴影

• 1.计算该方向光对view，projection矩阵，spilitdata

• 2. 创建ShadowDrawingSettings，根据光源，cullingresults，splitdata进行渲染物体

• 子主题 1

• 3.计算阴影空间转换矩阵，存储到dirShadowMatrices

• 4. 设置VP矩阵给buffer

• 5.执行buffer

• 6.正式开始渲染需要castershadow的物体

desc

* 7.存储阴影空间转换矩阵到dirShadowMatricesId

desc

* 8 结束采样
* 9.执行buffer

# Shadow.ShadowedDirectionalLights

## visibleLightIndex

### 可见光索引，这个索引是指第几个方向可见光，非cullingresult里的索引

## slopeScaleBias

### todo

## nearPlaneOffset

### todo

# Light.hlsl

## CBUFFER

### \_DirectionalLightCount

* 方向可见光数量

### \_DirectionalLightColors[MAX\_DIRECTIONAL\_LIGHT\_COUNT]

* 每个方向光的颜色

### \_DirectionalLightDirections[MAX\_DIRECTIONAL\_LIGHT\_COUNT]

* 每个方向光的方向

### \_DirectionalLightShadowData[MAX\_DIRECTIONAL\_LIGHT\_COUNT]

* 阴影数据