可编程渲染管线概述

Tim Cooper, 一月 31, 2018

在2018.1 beta版中介绍的可编程渲染管线（SRP）是一种由C＃脚本控制的配置执行Unity中的渲染的方式。在编写自定义渲染管线之前，理解我们在说的渲染管线究竟是什么是很重要的。

**什么是渲染管线**

“渲染管线”是用于将对象放到屏幕上的许多技术的总称。它包含以下的高级概念：

## 剔除

## 渲染对象

## 后期处理

除了这些高级概念之外，每个部分都可以进一步细分，具体取决于您想要如何执行它们。 例如，渲染对象可以使用以下方法执行：

## 多通道渲染

每道光每个物体一个通道

## 单通道渲染

每个物体一个通道

## 延迟渲染

将物体表面属性渲染到G缓冲区，然后再执行屏幕空间照明。

在编写自定义SRP时，这些决定是您需要做的。每种技术都有许多权衡需要去考虑。

**演示项目**

本文讨论的所有功能都包含在GitHub上的演示项目中

**渲染入口点**

在使用SRP时，您需要定义一个控制渲染的类; 这是您将要创建的渲染管线。入口点是一个使用渲染上下文（render context）（下面会介绍到）和相机列表渲染的“Render”调用。

|  |  |
| --- | --- |
| 1  2  3  4 | public class BasicPipeInstance : RenderPipeline  {     public override void Render(ScriptableRenderContext context, Camera[] cameras){}  } |

**渲染管线上下文（Render Pipeline Context）**

SRP使用了延迟执行的概念。你作为用户建立一系列命令（command）然后执行它们。用于构建这些命令的对象就叫做“ScriptableRenderContext”。 当您填充了上下文后，您可以调用“Submit”来提交所有排队等待的绘图调用（draw call）。

例如，使用由渲染上下文执行的命令缓冲来清除渲染目标：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | // Create a new command buffer that can be used  // to issue commands to the render context  var cmd = new CommandBuffer();    // issue a clear render target command  cmd.ClearRenderTarget(true, false, Color.green);    // queue the command buffer  context.ExecuteCommandBuffer(cmd); |

一个并不激动人心的渲染管线例子:)

这是一个简单的清除屏幕的渲染管线例子。

**剔除**

剔除是计算出将什么渲染到屏幕上的过程。

Unity中的剔除包括：

**·**视锥体剔除（Frustum culling）：计算相机近平面和远平面之间的物体。

**·**遮挡剔除（Occlusion culling）：计算哪些对象隐藏在其他对象的后面并从渲染中排除它们。想了解更多信息，请参考[Occlusion Culling docs](https://docs.unity3d.com/Manual/OcclusionCulling.html)。

渲染开始时第一步就是计算需要渲染的东西。这牵涉到相机以及在相机的透视区域执行剔除操作。

剔除操作返回一系列可渲染到相机上的对象和灯光。这些对象稍后将在渲染管线中使用。

**SRP中的剔除**

在SRP中，您通常从相机的透视区域渲染对象。这是Unity用于内置渲染的相机对象。SRP为剔除提供了许多API。通常流程如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | // Create an structure to hold the culling paramaters  ScriptableCullingParameters cullingParams;    //Populate the culling paramaters from the camera  if (!CullResults.GetCullingParameters(camera, stereoEnabled, out cullingParams))      continue;    // if you like you can modify the culling paramaters here  cullingParams.isOrthographic = true;    // Create a structure to hold the cull results  CullResults cullResults = new CullResults();    // Perform the culling operation  CullResults.Cull(ref cullingParams, context, ref cullResults); |

得到填充的剔除结果现在可以用于执行渲染了。

**绘制**

现在我们有一组剔除操作的结果，我们可以将它们渲染到屏幕上。但是有很多可以配置的东西，所以我们需要先做出一些决定。这些决定将由以下方面驱动：

**·**您将执行渲染管线的硬件

**·**您希望实现的特定外观和感觉

**·**您正在制作的项目类型

例如，一个移动2D 横板游戏与PC高端第一人称游戏。这些游戏具有截然不同的要求所以它们的渲染管线也大不相同。会造成下面这些具体的真实的例子：

**·**高动态光照渲染（HDR）和低动态光照渲染（LDR）

**·**线性（Linear）与伽马（Gamma）

**·**多重采样抗锯齿（MSAA）与后期抗锯齿

**·**基于物理的材质和普通材质

**·**光照和无光照

**·**光照技术

**·**阴影技术

在编写渲染管线时做出这些决定将能帮助您确定编辑时的许多设置。

现在，我们将演示一个简单的没有灯光的渲染器，可以渲染某些不透明对象。

**过滤：渲染存储区和图层**

通常，渲染对象具有特定的分类，它们是不透明，透明，子表面或其他一些类别。Unity使用队列的概念来表示何时渲染一个对象，这些队列形成一个对象（物体上的材料）缓存区。从SRP调用渲染时，您可以指定要使用的缓冲区的范围。

除了存储缓存区之外，标准的Unity层也可以用于过滤。

这提供了通过SRP绘制对象时额外过滤的功能。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // Get the opaque rendering filter settings  var opaqueRange = new FilterRenderersSettings();  //Set the range to be the opaque queues  opaqueRange.renderQueueRange = new RenderQueueRange()  {  min = 0,;  max = (int)UnityEngine.Rendering.RenderQueue.GeometryLast,  }  //Include all layers;  opaqueRange.layerMask = ~0; |

**绘制设置：应该如何绘制**

过滤和剔除决定了需要绘制的内容，但是接下来我们需要确定如何渲染它。SRP提供了多种选项来配置渲染通过过滤的对象。'DrawRenderSettings'结构就是用于配置这些数据的结构。这个结构允许配置很多东西：

**·**排序 - 对象渲染的顺序，包括从后到前和从前到后。

**·**渲染器标志 - 从Unity传递到着色器的内置设置，这包括每个对象的光探针，每个对象的光照贴图以及类似物。

**·**渲染标志 - 应该使用什么算法进行批处理，实例化还是非实例化。

**·**着色器通道 – 当前的绘制调用应该使用哪一个着色器通道

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | // note that it takes a shader pass name  var drs = new DrawRendererSettings(Camera.current, new ShaderPassName("Opaque"));  // enable instancing for the draw call  drs.flags = DrawRendererFlags.EnableInstancing;  // pass light probe and lightmap data to each r  drs.rendererConfiguration = RendererConfiguration.PerObjectLightProbe | RendererConfiguration.PerObjectLightmaps;  // sort the objects like normal opaque objects  drs.sorting.flags = SortFlags.CommonOpaque; |

**绘制**

现在我们有三个需要发起绘制调用的东西：

**·**剔除结果

**·**过滤结果

**·**绘制结果

我们可以发起绘制调用！像SRP中的所有东西一样，一个绘制调用作为调用进入到上下文中。在SRP中，您通常不会渲染单个网格物体，而是发起可以一次渲染大量物体的调用。这减少了脚本执行开销，并允许在CPU上快速执行作业。

我们结合我们一直在编辑的功能发起绘制调用。

|  |  |
| --- | --- |
| 1  2  3  4  5 | // draw all of the rend  context.DrawRenderers(cullResults.visibleRenderers, ref drs, opaqueRange);  // submit the context, this will execute all of the queued up commands.  context.Submit(); |

这会将对象绘制到当前绑定的渲染目标中。如果您愿意，您可以使用命令缓冲区来切换渲染目标。

渲染不透明对象的渲染器可以在这里找到：

<https://github.com/stramit/SRPBlog/blob/master/SRP-Demo/Assets/SRP-Demo/2-OpaqueAssetPipe/OpaqueAssetPipe.cs>

这个例子可以进一步扩展为透明渲染：

<https://github.com/stramit/SRPBlog/blob/master/SRP-Demo/Assets/SRP-Demo/3-TransparentAssetPipe/TransparentAssetPipe.cs>

这里要注意的重要一点是，渲染透明时，渲染顺序会更改为从后到前。

我们希望这篇文章能够帮助您开始编写自己的自定义SRP。

**Spotlight团队最佳实践：在Unity**

**中制作真实的视觉效果**

**作为Spotlight团队的一员，我很幸运能够参与一些非常有趣的项目。Unity的Spotlight团队和客户一起开发游戏，而我的主要任务是帮助开发人员实现项目所需的外观和质量。我在业内听到了许多故事并能够确定内容创作者面临的常见问题。我所从事的几个项目旨在实现相当逼真的视觉效果。鉴于该项目的艺术内容，我们如何在Unity中创造一个看起来可信的场景？**

为了制作真实的视觉效果我们需要涵盖许多主题。在这篇文章中，我将讨论光照和渲染设置。 接下来，我还会分享我们的Spotlight隧道示例场景，并解释如何使用它来学习和练习。

准备Unity渲染设置

了解如何使用Unity的渲染功能来模仿真实世界能帮助您实现项目的视觉目标。

**线性渲染模式**

简而言之，这将Unity设置为使用物理上精确的数学进行光照和阴影计算，然后将最终输出转换为最适合显示器的格式。

要指定伽马或线性工作流程，到**Edit** > **Project Settings** > **Player**并打开**Player Settings**。

然后到**Other Settings** > **Rendering**将**Color Space** 改为 **Linear.**

定义色彩空间应该是你项目中最早做的决定之一，因为它对最终的阴影和光照效果影响很大。

**渲染模式**

在Spotlight隧道示例场景中，我们使用延迟渲染路径。这使得内容创作者可以高效地处理多个动态灯光，组合多个反射立方体贴图，并且可以使用Unity 2017+中现有的屏幕空间反射功能。

如果要设置它，请转到Graphic Settings > Rendering Path 或者 Camera > Rendering

你可以在这部分的Unity文档中找到更多关于渲染模式的信息。

**高动态范围（HDR）相机。**

当渲染真实的类似现实生活中照明时，内容创作者将处理亮度高于1（高动态范围）的照明值和发射表面。这些值需要重新映射到适当的屏幕范围（这称为色调映射）。此设置对于允许Unity相机处理这些高值而不剪切它们至关重要。

要启用此功能，请选择场景中的主摄像机，并确保在选定摄像机的检查器选项卡中选中了HDR。