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。