文本

描述已自动生成

CPU：主线程花费了31.7ms，渲染线程花费了29.2ms。

Batches：总共有30,003批次。

Saved by batching：分批处理节省的批次为0。

Shadow casters：图包含了10,000个点。

SetPass calls：渲染6次（所以每个点都被渲染了三次。一次用于深度测试，一次用于阴影投射(也分别列出)，还有一次用于渲染最终的立方体。其他三个批处理用于额外的工作，如天空框和阴影处理，它们独立于我们的图。还有6个set-pass calls，这可以被认为是GPU重新配置以不同的方式渲染，比如使用不同的材料。）

图示

描述已自动生成

启用 SRP Batcher 并不会减少 Draw Call，而会减少每 Batch 所需要做的设置工作，此外 SRP Batcher 还能减少另一种开销较大的调用：SetPass calls

动态批处理：URP还有另一个用于动态批处理的开关。这是一个老的技术，动态地将小的网格组合成一个大的网格，然后进行渲染。

Edit/Project setting/Player的Dynamic Batching为动态批处理开关

GPU实例化：另一种提高渲染性能的方法是启用GPU实例化。这使得使用一个绘制命令告诉GPU用相同材质绘制一个网格的多个实例成为可能，提供一个转换矩阵阵列和其他可选的实例数据。

对于URP来说，GPU实例化是最好的，其次是动态批处理，然后是SRP批处理。但是差别很小，所以它们在我们的图中是等价的。唯一明确的结论是应该使用GPU实例化或SRP批处理。与动态批处理相比，BRP的GPU实例化的批处理数量更多，但帧率略高。

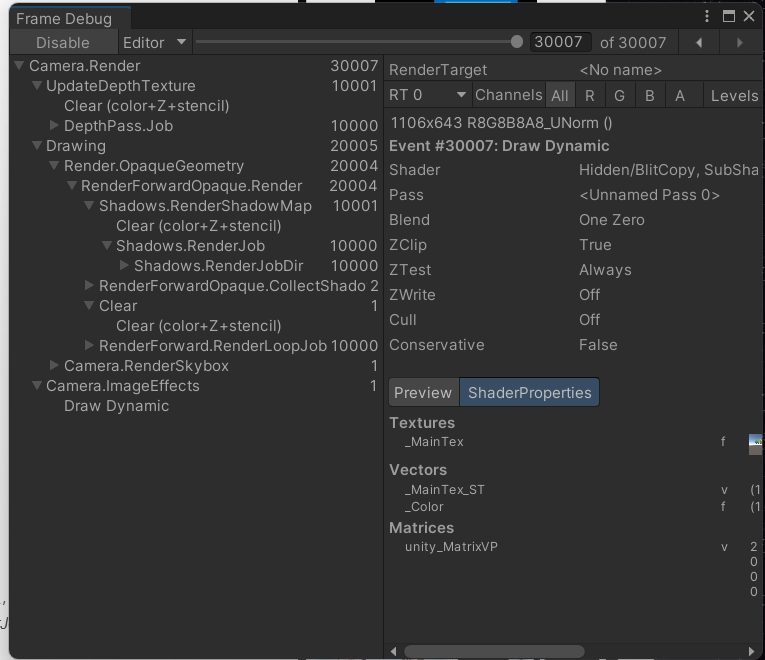
Window / Analysis / Frame Debugger打开的帧调试器

URP不使用单独的深度测试来进行定向阴影

动态批处理似乎不适用于阴影映射，这解释了为什么它对URP不太有效。

URP材质比标准BRP材质依赖更多的网格顶点数据，因此在单个批次中适合的点更少。与动态批处理不同的是，GPU实例化确实适用于阴影，所以它在这种情况下是优越的。

BRP调试器（平行光）：



文本

描述已自动生成

实际的渲染次数比数据面板上的次数多（4次）

BRP调试器开启动态批处理（平行光）：

文本

描述已自动生成

BRP调试器关闭动态批处理（平行光加点光源）：

文本

描述已自动生成

BRP调试器打开动态批处理（平行光加点光源）：

文本

描述已自动生成

BRP调试器打开GPU实例化（平行光）：

文本

描述已自动生成

BRP调试器打开GPU实例化（平行光加点光源）：

文本

描述已自动生成

存在点光源时，GPU实例化和动态批处理不能减少光照的渲染次数，场景中有多少Gameobject需要额外被渲染多少次。

带有额外的光的BRP现在用额外的时间绘制所有的点。帧调试器显示了RenderForward.RenderLoopJob渲染比以前多一倍(场景中需要渲染的物体数目)。更糟糕的是，动态批处理现在只适用于深度和阴影通道，而不是前向通道。

URP的 Dynamic Batching被默认隐藏了，在Edit选项卡的Preference中的Core Render Pipeline，把Visibility由All Hidden改为All Visible即可在URL的Rendering中看到SRP Batching。

URP调试器什么都不开：：

图形用户界面, 文本

描述已自动生成

URP调试器开动态批处理：

文本

描述已自动生成

URP调试器开GPU实例化：

图形用户界面, 文本

描述已自动生成

URP调试器开SRP：

图形用户界面, 文本

描述已自动生成

URP开启第二个点光源：

文本

描述已自动生成

与BRP不同的是，开启第二个点光源对URP的Batch数没有任何影响。

第二个光源似乎对URP没有影响，因为它是一个现代正向渲染器，在一个通道中应用所有的光源。所以命令列表保持不变，即使GPU每次绘制需要执行更多的光照计算。

了解CPU方面发生了什么，可以打开分析器窗口。通过Window / Analysis / Profiler打开窗口。它会在播放模式下记录性能数据并存储以备后续检查。

URP分析器窗口：

图形用户界面

描述已自动生成

图形用户界面, 文本

描述已自动生成

BRP分析器窗口：

图形用户界面, 日程表

描述已自动生成

图片包含 图形用户界面

描述已自动生成

从上图中可以看到，无论是URP还是BRP，在运行时EditorLoop的占比都很高，为程序增加了许多额外开销。因此需要构建一个专用于监测的应用程序。File / Build Settings

勾选

图形用户界面

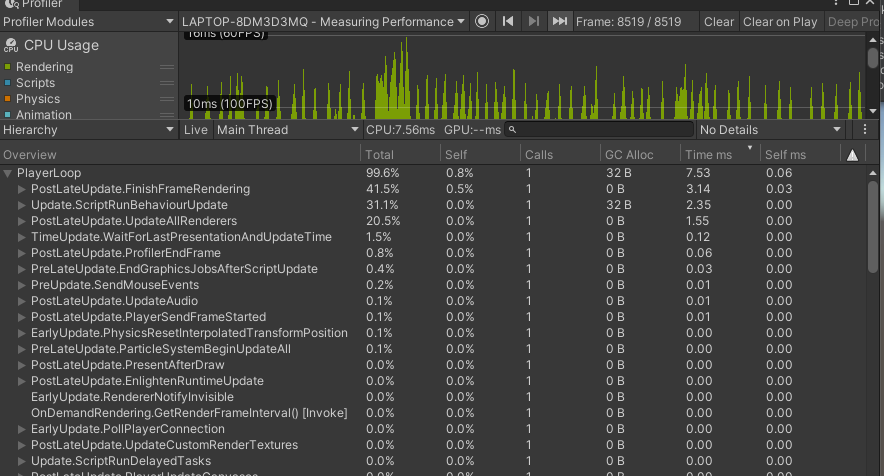
描述已自动生成

按Build按钮构建，或者Build and Run构建运行，或者通过File / Build and Run选择已有构建运行。

URP分析构建：

日程表

描述已自动生成



比起UGUI的Text控件，TextMeshPro有着许多优势。它使用Signed Distance Field（SDF）即有向距离场作为主要的文本渲染管线，在不同的分辨率和不同的位置下都能渲染出清晰的文本，而且它通过不同的着色器来实现描边，阴影，发光等效果，表现更出色的同时性能也更好。

Time.ununscaledDeltaTime：从上一帧到当前帧的独立于 timeScale的时间间隔（以秒为单位）（只读）。与deltaTime不同是，该值不受timeScale（1.0为现实时间流速，0.5是2倍速）的影响

TMPro.TextMeshProUGUI字段保存用于显示其数据的文本组件的引用

通过在0后面加上冒号和想要的数字来指示文本四舍五入到小数点后的特定位数。比如四舍五入到一个整数，冒号后的数字则应为0。

帧率(Frame rate)=帧数(Frames)/时间(Time)

帧数就是在1秒钟时间⾥传输的图⽚的量

在函数间插值，使得函数间的切换更加平滑。



Lerp是线性插值的简写。它将在函数之间产生一个直线的匀速转换。我们可以通过放慢开始和结束附近的进程让它看起来更平滑。这是通过用0、1和progress作为参数，调用Mathf.Smoothstep替换原始的progress。它适用于3 x 2 − 2 x3函数，俗称Smoothstep。

lerp函数就是在a和b之间过渡，a和b可以是一个值，也可以是一个函数。

图表, 折线图

描述已自动生成