简介

对于VR应用来说，如果想要让用户获得好的用户体验，特别是免除恶心眩晕的困扰，在VR开发中进行优化是必不可少的，惟其如此才能达到我们期望的游戏运行帧速。和其它平台上的开发不同，对VR应用的优化应该在项目启动的前期就开始，而且应该贯穿始终，而不是像传统项目那样把优化的工作留到最后去做。此外，在目标设备上进行实际测试也是非常有必要的。

相比非VR项目来说，VR项目是非常昂贵的，其主要原因就是所有的画面都必须为每只眼睛单独渲染一次。因此，在开发VR应用的过程中需要时刻想到这些问题。如果我们能在开启之前就想到这些问题，那么会节省大量的时间。

对于移动VR来说，优化工作就显得尤为重要。不仅仅是因为要运行VR应用，还因为移动设备的运算性能和散热性相比桌面电脑来说都要差上不少。

考虑到实现目标帧速是如此重要，所有的优化方法都必须考虑在内。我们需要在所有可能的地方优化项目代码，关于优化代码，可以参考Unity的guide。

Oculus的相关资源

在Oculus的官方网站上提供了大量关于如何优化VR应用的信息，在阅读我们的教程之前，强烈建议大家仔细阅读这些文档。

https://developer.oculus.com/documentation/

http://static.oculus.com/sdk-downloads/documents/Oculus_Best_Practices_Guide.pdf

https://developer.oculus.com/blog/squeezing-performance-out-of-your-unity-gear-vr-game/

https://developer.oculus.com/blog/squeezing-performance-out-of-your-unity-gear-vr-game-continued/

Unity Editor优化工具

Unity提供了一系列有用的工具和方法，可以帮助我们来优化VR内容。

The Profiler

profiler可以帮助开发者了解游戏中渲染每一帧所耗费的时间，并将其分为CPU、渲染、内存、音频、物理引擎和网络。学会如何使用Profiler对于检测游戏运行性能是至关重要的，

关于Profiler的相关信息，可以参考以下链接：

http://docs.unity3d.com/Manual/Profiler.html

https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/profiler-overview

https://unity3d.com/learn/tutorials/modules/intermediate/editor/intro-to-profiler

Frame Debugger

使用Frame Debugger可以让我们冻结某一帧，然后通过单独的draw调用来查看场景是如何生成的，然后来发现需要进行优化的地方。在这个过程中，我们可能会发现渲染了一些不必要进行渲染的对象，这样可以大幅度降低每帧的draw 调用。

关于使用Frame Debugger的更多信息，请参考这里：

http://docs.unity3d.com/Manual/FrameDebugger.html

https://unity3d.com/learn/tutorials/modules/intermediate/graphics/frame-debugger

VR应用优化的基础

考虑到对应用进行优化是个庞大的话题，对不同的平台有不同的要求，我们也提供了延伸阅读的相关信息。

通常来收，现有的优化技巧对VR开发也是适用的，因此这些知识也用得上。

Geometry（几何体）

在VR应用我们应尽量删除几何体中用户永远也不会注意到的面。我们没必要在场景中渲染出用户根本看不到的东西。比如，如果某个杯子背靠着墙壁，那么用户可能永远也不会看到它的背面，因此我们可以不必显示模型中的这些面。

对于3D美术设计人员，应该尽可能的简化模型设计。根据目标平台的不同，我们可能会需要查看纹理细节，或许还会希望查看视差映射贴图，和曲面细分。虽然这种方法可能会影响游戏性能，也可能对特定的平台根本无法使用。

Overdraw

Overdraw可以让开发者查看哪些对象绘制在其它对象的顶部，但其实是在浪费GPU时间。我们应尽可能的减少使用overdraw。我们可以使用Scene View Control Bar来查看场景视图中的overdraw。

正常的着色视图如下：

启用Overdraw之后的着色视图：

Level of Detail(LOD)细节层次

通过使用LOD，可以随着物体和摄像机之间的距离来减少物体渲染的三角形数目。除非所有的物体都离摄像机同样远，否则我们都可以使用LOD来减少硬件的负担。我们可以添加一个LOD组件，然后对远离摄像机的物体提供低精度模型。

使用Simplygon可以自动完成对大多数asset的LOD预处理。

Draw Call 批处理

我们应尽可能的通过Static Batching和Dynamic Batching来实现Draw Call 批处理。Draw Call批处理可以极大的提升游戏性能。具体请参考Unity官方指南的Draw Call Batching。

Light mapping

尽可能减少动态光照，尽量多使用光照烘焙，尽量避免实时阴影。

关于这部分的具体内容，请参考Unity官方的Lighting and Rendering。

Light Probes

使用Light probes可以让我们对场景中的光照点取样，然后应用到动态物体上。使用light probes通常更快，而且也能产生绝佳的视觉效果。

Reflection Probes

Reflection probes可以保存其周围的立方图，从而实现真实反射效果，而且也会对游戏性能产生影响。需要注意的是，目前在VR中使用实时reflection probes会导致游戏性能大幅降低。

Occlusion Culling

Occlusion Culling（遮挡剔除）可以避免渲染那些不可见的物体。例如，如果玩家正处于某个房间中，而另外一个房间的门是关闭的，那么对玩家来说另外一个房间中的所有物体都是不可见的，也就完全没必要进行渲染。

根据项目和目标平台的不同，我们可能会希望实现Occlusion Culling，从而大幅提升游戏性能。

下图是一个frustum culling（视锥体剔除）的示例：

下图是Occlusion Culling（遮挡剔除）的示例：

Anti-Aliasing（抗锯齿）

抗锯齿对VR应用来说非常重要，因为使用这种技术可以让图像的边缘显得更加平滑，并减少毛边线下。如果我们在项目中使用Forward Rendering，那么就需要在Quality Setting中启用MSAA。而对于Gear VR项目来说，任何时候我们都需要启用该选项。

当然，在使用Deferred Rendering时我们无法启用MSAA，此时需要启用AntiAliasing作为后处理特效（所谓的“反走样”），或者考虑使用SMAA。
这里提供了一个相关的示例。

Textures

通常来说，在VR项目中我们应尽可能的使用Texture Atlasing（纹理贴图），以减少单独纹理和材质的使用量。
为简化和加速这个过程，我们可以考虑使用MeshBaker来烘焙游戏中所使用的纹理、模型和材质。
在Oculus Connect 2开发者大会上，来自Turbo Button的Holden曾分享过优化应用以及使用MeshBaker的相关经验。

有一点需要注意的是，在VR项目中normal maps看起来效果并不好，因此我们应该避免使用。关于纹理的更多知识，请参考Oculus documents。

Shaders

在VR项目中，我们应尽可能使用最基本的shader。在Gear VR上，我们可能会需要考虑使用不那么消耗资源的Mobile>Unlit(Supports Lightmap) shader，并使用lightmap来给场景提供光照。

Fullscreen Effects（全屏特效）

对VR项目来说Fullscreen Effects过于奢侈，因此我们应在Gear VR项目中完全避免使用。

Quality Settings

Quality Settings中的选项将直接影响项目的视觉效果。通过调整这些属性，可以某种程度的提升游戏性能，当然代价就是牺牲了部分视觉效果。

RenderScale

调整VRSetting.renderScale可以牺牲画质换取更高的游戏性能。具体可以参考Getting Started。

Asynchronous Loading

为了提升性能，我们可以考虑把游戏场景分成诸多小的场景。不过这样做需要注意的是，在加载下一个场景的内容时，应该避免锁定对头部的跟踪，以免产生nausea恶心现象。

为避免出现这种情况，我们可以考虑设计一个允许头部运动跟踪的加载场景，让游戏异步加载新的场景，具体的方法是使用SceneManager.LoadSceneAsync。

示例场景中所用到的优化技巧

为了让用户在DK2和Gear VR上面获得更好的体验，我们在示例场景中使用了一系列的优化技巧。

考虑到我们需要让同一个项目支持两个凭条，因此需要考虑对最低端性能设备的支持，也就是Gear VR。我们选择了低多边形的艺术风格，并使用少量的基本色彩，让物体从环境中脱颖而出。

在使用Forward Rendering时，我们需要在Edit > Project Settings > Quality Settings 中启用4x MSAA，以便获得更好的视觉效果：

让我们简单看看这些场景中所使用的优化技术：

Menu 场景中使用的优化：

跟该项目中所有的场景一样，Menu场景中使用了低多边形的美术资源，而且避免使用实时光照。

我们在菜单面板上使用了定制的shader，名为SeparableAlpha，可以为一系列的图像定义独立的alpha通道。这就意味着不是每一帧都需要自己的alpha通道。这样做可以节省文件大小，并去掉某些贴图。

Flyer 场景中使用的优化：

我们在Flyer场景中动态启用了fog，从而避免让物体突然跳进玩家的视野，并缩短了视距，这也就意味着减少了所需渲染的物体数量。

场景中陨石的顶点数较低，从而可以通过Dynamic Batching来减少draw call。

为了重用某些物体，我们创建了一个对象池，以处理激光、陨石和星门这样的对象。通过这种方式，可以避免昂贵的初始化调用。

对于Flyer场景中的飞船纹理我们同样做了优化，通过使用Detail Map slot中的次级UV 通道，可以只需使用更少的色块。这样一来我们就可以缩减总体的纹理大小。

Maze场景中的优化

Maze场景中使用了lightmap，从而在运行时获得更好的性能，特别是在Gear VR上。除此之外，该场景没有任何的实时光照和特效。

Shooter180(Target Gallery)和Shooter360(Target Arena)中的场景优化

和其它游戏一样，我们在这些场景中沿用了低多边形风格，并为目标对象创建了object pooling。同时我们使用了低顶点数以启用Dymaic Batching。

看完本篇教程，大家对VR游戏优化应该有了整体的印象，也大概了解了我们应如何使用Unity内置的工具来分析游戏性能，以及如何通过某些技巧来获得更好的游戏表现。

Oculus官方网站有很多关于这一点的内容：

https://developer.oculus.com/documentation/

http://static.oculus.com/sdk-downloads/documents/Oculus_Best_Practices_Guide.pdf

https://developer.oculus.com/blog/squeezing-performance-out-of-your-unity-gear-vr-game/

https://developer.oculus.com/blog/squeezing-performance-out-of-your-unity-gear-vr-game-continued/

在教程的最后一部分，我们将提供一系列的参考资料，供大家深入学习。

对VR开发感兴趣的朋友可以通过邮件(eseedo@gmail.com)或微信(iseedo)联系我，希望跟大家一起学习。另外在赛隆网(http://www.cylonspace.com)和我的个人博客(http://blog.sina.com.cn/eseedo)上也会放VR/AR开发的相关内容。