**着色器变体**

**着色器变体基本概念**

**什么是着色器变体？**

可以编写着色器代码片段来共享通用代码，但在启用或禁用给定关键字时具有不同功能。Unity 编译这些着色器代码片段时，它将为已启用和已禁用关键字的不同组合创建单独的着色器程序。这些各个着色器程序被称为着色器变体。

**如何创建着色器变体？**

要创建着色器变体，请使用以下 pragma 指令之一：

#pragma multi\_compile

#pragma multi\_compile\_local

#pragma shader\_feature

#pragma shader\_feature\_local

**下划线(\_)的作用？**

为了生成未定义预处理器宏的着色器变体，请添加一个只有下划线 (\_) 的名称。这是避免用完两个关键字的常用方法，因为在一个项目中可以使用的关键字数量有限。

# pragma multi\_compile \_ FOO\_ON和# pragma multi\_compile FOO\_ON等价。此指令生成两个着色器变体：一个未定义任何关键字(\_)，另一个定义了FOO\_ON。

**特定于阶段的关键字**

**特定于阶段的关键字指令有哪些？**

\_vertex、\_fragment、\_hull、\_domain、\_geometry 和 \_raytracing。

**阶段的关键字的作用？**

如果关键字仅影响片元阶段，则编辑器会为每个变体生成相同的顶点阶段。Unity会在之后识别并删除重复项，因此这种冗余工作**不会影响构建大小或运行时性能**；但是，如果您有很多阶段和/或变体，着色器**编译期间浪费的时间**会非常明显。

**阶段的关键字支持的图形 API**

Unity 并不完全支持在所有图形 API 中使用特定于阶段的关键字指令。

为 OpenGL 和 Vulkan 编译着色器时，编辑器会自动将任何特定于阶段的关键字指令恢复为常规关键字指令。

为 Metal 编译着色器时，任何针对顶点阶段的关键字也会影响曲面细分阶段，反之亦然。

**全局关键字和局部关键字**

**本地关键字与全局关键字**

本地关键字：shader\_feature\_local和multi\_compile\_local

全局关键字：shader\_feature和multi\_compile

在理想设置中，多用本地关键字和少用全局关键字可以减少每个着色器的关键字总数。

如果全局关键字和本地关键字同名，Unity 会优先考虑本地关键字。

**关键字数量限制**

Unity 中的关键字数量上限是384，Unity将大约60个关键字保留供内部使用（因此降低了可用上限），外加 64 个本地关键字。

**如何启用和禁用着色器关键字？**

全局关键字

Shader.EnableKeyword

Shader.DisableKeyword

CommandBuffer.EnableShaderKeyword

CommandBuffer.DisableShaderKeyword

本地关键字

Material.EnableKeyword

Material.DisableKeyword

ComputeShader.EnableKeyword

ComputeShader.DisableKeyword

**shader\_feature与multi\_compile**

**shader\_feature与multi\_compile 之间的区别？**

shader\_feature与multi\_compile 非常相似。唯一的区别是 Unity 没有将 shader\_feature 着色器的未用变体包含在最终构建中。因此，应该将 shader\_feature 用于材质中设置的关键字，而 multi\_compile 更适合通过代码来全局设置的关键字。

**变体优化**

**变体数量如何计算？**

# pragma multi\_compile A B C

# pragma multi\_compile D E

这会为第一行生成三个变体，为第二行生成两个变体。总共生成六个着色器变体（A+D、B+D、C+D、A+E、B+E、C+E）。

**如何在构建时剥离着色器变体？**

常规着色器：IPreprocessShaders.OnProcessShader；

计算着色器：IPreprocessComputeShaders.OnProcessComputeShader；

**#pragma指令使用规范**

1，