HLSL Keywords & Shader Variants

Before we go over lighting , we need to talk about keywords and shader variants first . In shaders we can specify the *#pragma multi_compile* and *#pragma shader_feature* directives which are used to specify keywords for toggling certain parts of the shader code "on" or "off" . The shader actually gets compiled into multiple versions of the shader , known as **shader variants** . In Unity , we can then enable and disable keywords per material to select which variant gets used .

This is useful as it allows us to write a single shader , but create different versions of it with some features off to save on performance . This needs to be used carefully however , as different shader variants will not batch together . URP uses some of these keywords for toggling features like lighting (i.e. *#pragma multi_compile _ _MAIN_LIGHT_SHADOWS* prior to v11) and for (which uses the slightly special *#pragma multi_compile_fog* , same as in the built-in RP).

Multi Compile

```
#pragma multi_compile _A _B _C (...etc)
```

In this example we are producing three variants of the shader , where _A , _B , and _C are keywords . We can then use *#if defined(KEYWORD)/#ifdef KEYWORD* to determine which code is toggled by the keyword . For example :

```
#ifdef _A
// Compile this code if A is enabled
#endif

#ifndef _B
// Compile this code if B is disabled , aka only in A and C .
// Note the extra "n" in the #ifndef , for "if not defined"
#else
// Compile this code if B is enabled
#endif

// There's also #elif , for an "else if" statement
```

URP uses a bunch of multi_compiles , but here is some common ones . Not every shader needs to include all of these , but some of the functions in the ShaderLibrary rely on these keywords being included , otherwise they may skip calculations .

```
// Additional Lights (e.g. Point, Spotlights)
#pragma multi_compile _ _ADDITIONAL_LIGHTS_VERTEX _ADDITIONAL_LIGHTS

// Shadows
#pragma multi_compile _ _MAIN_LIGHT_SHADOWS
#pragma multi_compile _ _MAIN_LIGHT_SHADOWS_CASCADE
// Note, v11 changes this to :
// #pragma multi_compile _ _MAIN_LIGHT_SHADOWS _MAIN_LIGHT_SHADOWS_CASCADE _MAIN_LIGHT_SHADOWS_S
#pragma multi_compile _ _ADDITIONAL_LIGHT_SHADOWS
#pragma multi_compile _ _SHADOWS_SOFT

// Baked Lightmap
#pragma multi_compile _ LIGHTMAP_ON
#pragma multi_compile _ DIRLIGHTMAP_COMBINED
#pragma multi_compile _ LIGHTMAP_SHADOW_MIXING
#pragma multi_compile _ SHADOWS_SHADOWMASK

// Other
#pragma multi_compile_fog
#pragma multi_compile_instancing
#pragma multi_compile _ DOTS_INSTANCING_ON
#pragma multi_compile _ _SCREEN_SPACE_OCCLUSION
```

Shader Feature

Shader Features are similar to Multi-Compile , but an additional variant is generated with all keywords disabled and any **unused variants will be not be included in the final build**. This can be useful to keep build times down , but it's not good to enable/disable these keywords at runtime , since the shader it needs might not be included in the build ! If you need to handle keywords at runtime , multi_compile should be used instead .

```
#pragma shader_feature _A _B (...etc)
```

The above code generates **three** variants , where _A and _B are keywords . While there's only two keywords , an additional variant where both are disabled is also generated . When using Multi-Compile we can also do this , by specifying the first keyword as blank by using one or more underscores (_) . e.g.

```
#pragma multi_compile _ _A _B
```

Shader Variants

With each added multi_compile and shader_feature , it produces more and more shader variants for each possible combination of enabled/disabled keywords . Take the following for example :

```
#pragma multi_compile _A _B _C
#pragma multi_compile _D _E
#pragma shader_feature _F
```

Here , the first line is producing 3 shader variants . But the second line , needs to produce 2 shader variants for those variants where _D or _E is already enabled . So , A & D , A & E , B & D , B & E , C & D , C & E . That's now 6 variants .

Third line , is another 2 variants for each of those 6 , so we now have a total of 12 shader variants . (While it's only keyword , it has the additional variant with it disabled since that line is a shader_feature . Some of those variants might also not be included in the build depending on what is used by materials) .

Each added multi_compile with 2 keywords will double the amount of variants produced , so a shader that contains 10 of these will result in 1024 shader variants! It'll need to compile each shader variant that needs to be included in the final build , so will increase build time as well as the size of the build .

If you want to see how many shader variants a shader produces , click the shader and in the inspector there's a "Compile and Show Code" button , next to that is a small dropdown arrow where it lists the number of included variants . If you click the "skip unused shader_features" you can toggle to see the total number of variants instead .

To assist with reducing the number of variants produced , there is also "vertex" and "fragment" versions of these directives . For example:

```
#pragma multi_compile_vertex _ _A
#pragma multi_compile_fragment _ _B
#pragma shader_feature_vertex _C
#pragma shader_feature_fragment _D
```

In this example , the _A and _C keywords are only being used for the vertex program and _B and _D only for the fragment . Unity tells us that this produces 2 shader variants , although it's more like one shader variant where both are disabled and two "half" variants when you look at the actual compiled code it seems .

The documentation has some more information on shader variants .

Keyword Limits

An important note is there is also a maximum of **256 global keywords per project** , so it can be good to stick to the naming conventions of other shaders to ensure the same keywords are reused rather than defining new ones .

You'll also notice for many Multi-Compile the first keyword is usually left as just "_" . By leaving the keyword blank , it leaves more space available for other keywords in the 256 maximum . For Shader Feature , this is done automatically .

```
#pragma multi_compile _ _KEYWORD
#pragma shader_feature _KEYWORD

// If you need to know if that keyword is disabled
// We can then just do :

#ifndef _KEYWORD
// aka "#if !defined(_KEYWORD)"
// or "#ifdef _KEYWORD #else" also works too

// ... code ...

#endif
```

We can also avoid using up the maximum keyword count by using **local versions** of the *multi_compile* and *shader_feature* . These produce keywords that are local to that shader , but there's also a maximum of **64 local keywords per shader** .

```
#pragma multi_compile_local _ _KEYWORD
#pragma shader_feature_local _KEYWORD

// There's also local_fragment/vertex ones too!
#pragma multi_compile_local_vertex _ _KEYWORD
#pragma multi_compile_local_fragment _ _KEYWORD
#pragma shader_feature_local_vertex _KEYWORD
#pragma shader_feature_local_fragment _KEYWORD
```