HLSL-Fragment Shader

The fragment shader is responsible for determining the colour of the pixel output (including alpha) . For unlit shaders this can be a fairly simple solid colour or a colour obtained from sampling an input texture . For lit shaders , it's a bit more complicated but URP provides some handy functions which I'll be going through in the Lighting section .
For now since our shader is Unlit , all we need is :

```
half4 UnlitPassFragment(Varyings IN) : SV_Target
{
    // Sample BaseMap Texture :
    half4 baseMap = SAMPLE_TEXTURE2D(_BaseMap, sampler_BaseMap, IN.uv);
    // Tint texture result with Color property and vertex colours :
    return baseMap * _BaseColor * IN.color;
}
```

This produces a shader which outputs a half4 colour , based on the sampled `_BaseMap` texture , which is also tinted by the `_BaseColor` property and interpolated vertex colour . The `SAMPLE_TEXTURE2D` macro is provided by the ShaderLibrary and returns the colour at the given uv coordinate , since the shader runs per-fragment/pixel .
As mentioned in the FragmentOutput section , `SV_Target` is used to write the fragment/pixel colour to the current render target .
Something that we might also want to do , is discard pixels if their alpha value is below a certain threshold , so that the entire mesh isn't visible - e.g. for grass/leaf textures on quads . This can be done in opaque shaders as well as transparent , and is usually referred to as **Alpha Clip/Cutout/Cutoff** . If you are familiar with **Shader Graph** , it's handled with the **Alpha Clip Threshold** . In Shader Code this commonly involves a Float property named `_Cutoff` (added to Shaderlab Properties as well as the UnityPerMaterial CBUFFER for SRP Batcher-compatibility) . This can then be used in the fragment shader :

```
if (_BaseMap.a < _Curoff)
{
    discard;
}
// OR
clip(_BaseMap.a - _Cutoff);
// inside the fragment function , before returning
```

This is essentially the Unlit Shader Code complete .