HLSL - Vertex Shader

Vertex Shader

The main thing that our vertex shader needs to do is convert the object space position from the mesh into a clip space position . This is required in order to correctly render fragments/pixels in the intended screen position .

In built-in shaders you would do this with the *`UnityObjectToClipPos`* function , but this has been renamed to *`TransformObjectToHClip`* (which you can find in the SRP-core SpaceTransforms.hlsl) . That said , there's another way to handle the transform in URP as shown below which makes conversions to other spaces much easier too .

```
Varyings UnlitPassVertex(Attributes IN)
{
    Varyings OUT;
    // alternatively , Varyings OUT = (Varyings)0;
    // to initalise all struct inputs to 0.
    // otherwise, every variable in the struct must be set.

    // OUT.positionCS = TransformObjectToHClip(IN.positionOS.xyz);
    // Or :
    VertexPositionInputs positionInputs = GetVertexPositionInputs(
        IN.positionOS.xyz);
    OUT.positionCS = positionInputs.positionCS;
    // which also contains .positionWS, .positionVS and .positionNDS
    // (aka screen position)

    // Pass through UV/TEXCOORD0 with texture tiling and offset
    // (_BaseMap_ST) applied :
    OUT.uv = TRANSFORM_TEX(IN.uv, _BaseMap);

    // Pass through Vertex Colours :
    OUT.color = IN.color;

    return OUT;
}
```

`GetVertexPositionInputs` computes the position in each of the commonly used spaces . It used to be a part of Core.hlsl , but was separated into it's own file - ShaderVariablesFunctions.hlsl in URP v9 , but this file is automatically included when we include Core.hlsl anyway .

The function uses the object space postion from the `Attributes` as an input and returns a `VertexPositionInputs` struct , which contains :

- *positionWS* : the position in **World** space
- *positionVS* : the position in **View** space
- *positionCS* : the position in **Clip** space
- *positionNDS* : the position in **Normalised Device Coordinates** , aka Screen Position . (0 , 0) in bottom left , (w , w) in top right . Of note , we would pass the position to the fragment stage , then handle the perspective divide (`positionNDC.xy / positionNDC.w`) so (1 , 1) is top right instead .

For our current unlit shader , we don't need these other coordinate spaces , but this function is useful for shaders where we do . The unused ones also won't be included in the compiled shader so there isn't any unnecessary calculations .

The vertex shader is also responsible for passing data to the fragment , such as the texture coordinates (UV) and vertex colours . The values get interpolated across the triangle , as discussed in the Intro to Shaders post . For the UVs , we could just do `OUT.uv = IN.uv;` assuming both are set to `float2` in the structs , but it's common to include the Tiling and Offset values for the texture which Unity passes into a `float4` with the texture name + _ST (s referring to scale , and t for translate) . In this case , `_BaseMap_ST` which is also included in our UnityPerMaterial CBUFFER from earlier . In order to apply this to the UV , we could do :

```
OUT.uv = IN.uv * _BaseMap_ST.xy + _BaseMap_ST.zw;
```

But the `TRANSFORM_TEX` macro can also be used instead , which is included int the Built-in RP as well as URP .

While we don't need any normal/tangent data for our Unlit shader , there is also `GetVertexNormalInputs` which can obtain the World space position of the normal , tangent and generated bitangent vectors .

```
VertexNormalInputs normalInputs = GetVertexNormalInputs(IN.normalOS,
    IN.tangentOS);
OUT.normalWS = normalInputs.normalWS;
OUT.tangentWS = normalInputs.tangentWS;
OUT.bitangentWS = normalInputs.bitangentWS;
```

This will be useful later when Lighting is needed . There's also a version of the function which takes only the `normalOS` , which leaves `tangentWS` as (1, 0, 0) and `bitangentWS` as (0, 1, 0) , or you could use `positionWS = TransformObjectToWorldNormal(IN.normalOS)` instead , which is useful if the tangent/bitangent isn't needed (e.g. No normal/bump or parallax mapping effects) .