

// MinTuts/Procedural Terrain.shader

```
Shader "MinTuts/Procedural Terrain" {
    SubShader {
        Pass {
            CGPROGRAM

            #pragma vertex    vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            struct v2f {
                float4 pos    : SV_POSITION;
                float3 wpos   : POSITION1;
            };

            v2f vert(float4 vertex : POSITION) {
                v2f o;

                o.pos = UnityObjectToClipPos(vertex);
                o.wpos = mul(unity_ObjectToWorld, vertex);

                return o;
            }

            float4 frag(v2f i) : COLOR {
                float  p = i.wpos.y * 0.015;
                float3 y = float3(p, p, p);

                return float4(y, 1);
            }

            ENDCG
        }
    }
}
```

Next, we call **mul** - a function built into HLSL

We pass the UnityCG constant **unity_ObjectToWorld**

We also pass our **vertex** input parameter

unity_ObjectToWorld is a transform matrix

Multiplying **unity_ObjectToWorld** and **vertex** our local/object space coordinates are transformed to world space coordinates

// MinTuts/Procedural Terrain.shader

```
Shader "MinTuts/Procedural Terrain" {
    SubShader {
        Pass {
            CGPROGRAM

            #pragma vertex    vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            struct v2f {
                float4 pos    : SV_POSITION;
                float3 wpos   : POSITION1;
            };

            v2f vert(float4 vertex : POSITION) {
                v2f o;

                o.pos = UnityObjectToClipPos(vertex);
                o.wpos = mul(unity_ObjectToWorld, vertex);

                return o;
            }

            float4 frag(v2f i) : COLOR {
                float  p = i.wpos.y * 0.015;
                float3 y = float3(p, p, p);

                return float4(y, 1);
            }

            ENDCG
        }
    }
}
```

Next, we call **mul** - a function built into HLSL

We pass the UnityCG constant **unity_ObjectToWorld**

We also pass our **vertex** input parameter

unity_ObjectToWorld is a transform matrix

Multiplying **unity_ObjectToWorld** and **vertex** our local/object space coordinates are transformed to world space coordinates

The transformed coordinates are then assigned to **o.wpos**