

## // 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);
            }

        }
    }
}
```

We use this **float3** as the first 3 arguments...  
to the **float4** constructor

The 4th argument we hard code to 1; the  
4th channel is the opacity/transparency channel  
Since this is a single-pass shader transparency  
is not supported

Hard coding the opacity/transparency channel to  
1 makes it clear that we do not want this shader  
to support transparency

Now that our **float4** (with a **semantic** of **COLOR**  
and channels for red, green, blue, and alpha) is  
constructed, we return it

**NOTE:** **vert** can manipulate the local/clip/world  
space coordinates of vertices (*it can even add*  
*or remove vertices*) but **frag**'s only purpose is to  
take in data from **vert**...

## // 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);
            }

        }
    }
}
```

We use this **float3** as the first 3 arguments...  
to the **float4** constructor

The 4th argument we hard code to 1; the  
4th channel is the opacity/transparency channel  
Since this is a single-pass shader transparency  
is not supported

Hard coding the opacity/transparency channel to  
1 makes it clear that we do not want this shader  
to support transparency

Now that our **float4** (with a **semantic** of **COLOR**  
and channels for red, green, blue, and alpha) is  
constructed, we return it

**NOTE:** **vert** can manipulate the local/clip/world  
space coordinates of vertices (*it can even add*  
*or remove vertices*) but **frag**'s only purpose is to  
take in data from **vert**...  
and return...