

// 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 assign the result of this calculation to the variable **p**...

which has a type of **float**

We then use p to build...

a **float3** representing the red, green, and blue color channels

We then assign this **float3** to the variable **y**; making it clear that our **float3** color comes from our **i.wpos.y** value

NOTE: Since the red, green, and blue channels all have the same value (p) this is a greyscale shader - as **p approaches 0** the color will be darker shades of grey (until the color becomes black when p equals 0) and as **p approaches 1** the color will be lighter shades of grey (until the color becomes white when p equals 1)

// 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 assign the result of this calculation to the variable **p**...

which has a type of **float**

We then use p to build...

a **float3** representing the red, green, and blue color channels

We then assign this **float3** to the variable **y**; making it clear that our **float3** color comes from our **i.wpos.y** value

NOTE: Since the red, green, and blue channels all have the same value (**p**) this is a greyscale shader - as **p** approaches 0 the color will be darker shades of grey (until the color becomes black when p equals 0) and as **p** approaches 1 the color will be lighter shades of grey (until the color becomes white when p equals 1)

NOTE: This is why it's rare to let a **normalized** value go beyond 1: **float3** represents colors as red, green, and blue values between 0 and 1 - values greater than 1 don't make any sense to a graphics card