// 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
    }
  }
}
```

First things first: create our output data structure

Then we populate **v2f**'s **pos** and **wpos** properties

This function is available because… we **#include**d Unity's **Cg** helper functions

We pass the **vertex** input parameter to this function

Since **SV_POSITION** and **POSITION** have the same **semantic** meaning (*the **vertex**s position in object, aka local, space*)…
passing **vertex** results in our local/object coordinates…
being transformed to *clip space* coordinates

NOTE: Even though the **semantic**s are identical, the values for **o.pos** and **vertex** will be different because…
they are *defined* in *different spaces*;

// 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
    }
  }
}
```

First things first: create our output data structure

Then we populate **v2f**'s **pos** and **wpos** properties

This function is available because… we **#include**d Unity's **Cg** helper functions

We pass the **vertex** input parameter to this function

Since **SV_POSITION** and **POSITION** have the same **semantic** meaning (*the **vertex**s position in object, aka local, space*)…

passing **vertex** results in our local/object coordinates…

being transformed to *clip space* coordinates

NOTE: Even though the **semantic**s are identical, the values for **o.pos** and **vertex** will be different because…

they are *defined* in *different spaces*; *local/object* space…