// 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
    }
  }
}
```

This is the initial version of our new shader. In subsequent commits well add useful features, and we'll end up with terrain that finally looks… like terrain!

For now, let's break what we have here apart and work through understanding it step-by-step.

NOTE: In this commit the shader has not yet been applied to the terrain. In the next commit, 7a0dfc4, the material and shader will have been applied to the terrain - so we can see what our shader's output is.

```
// 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
    }
  }
}
```

These are the *minimum required sections* for **ShaderLab**

In **Unity**, **ShaderLab** *wraps* **Cg** code

ShaderLab provides a structured way to define *single pass* (*what our shader is*) and *multi-pass* shaders (*shaders with multiple Pass blocks*)

ShaderLab also allows us to specify *multiple* **SubShader**s

SubShaders allow us to *target* and *optimize* for *specific platforms* - *i.e:* a **SubShader** *optimized* for PS4, another **SubShader** *optimized* for mobile, and yet another **SubShader** *optimized* for high-end PCs

NOTE: Most **vertex**/**fragment** shader examples show the *Properties* section

The *Properties* section is not required, most tutorials and examples include it because they want a texture to use in their shader

We are generating everything procedurally (*including textures - a few Tuts down the road*) so we have no use for a texture property