Normals

Normals are vectors which point outwards from a surface.

Normals are a special animal when it comes to space conversions and other matrix operations.

Vertex Normals

These are saved automatically by the Unity Engine. You can grab them via the NORMAL semantic when making your vertex input struct,

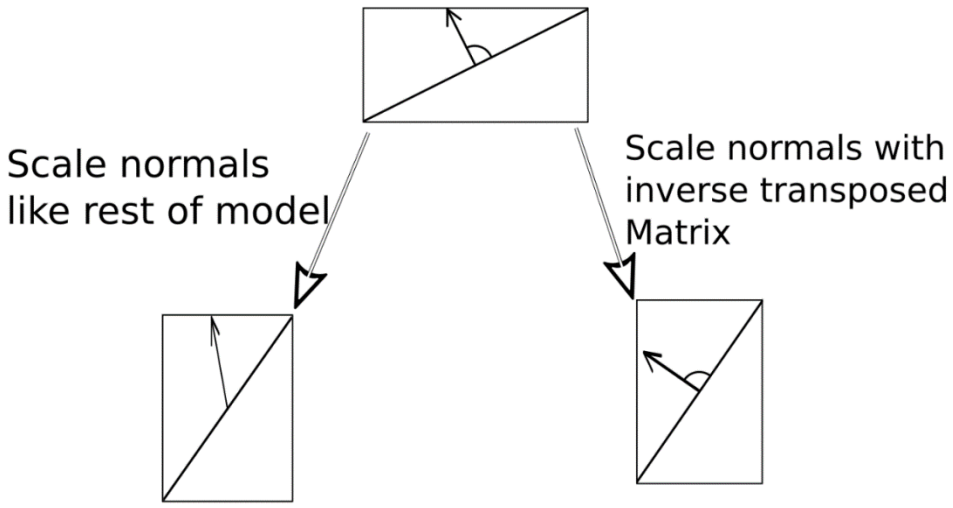
i.e.

```
// Engine -> Vertex
struct VertexInput
{
    float4 vertexPosition : POSITION;

    float3 vertexNormal : NORMAL;
};
```

Note that unlike the vertex position, vertex normals are of the type float3.

World Normals

You cannot simply multiply a vertex normal by the object to world matrix. See the figure below:

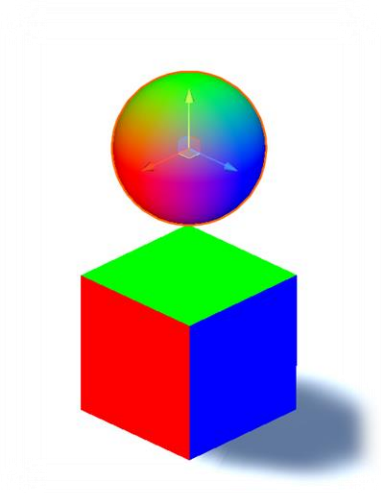


Instead, you need to multiply by the inverse transpose matrix, AKA just multiply by the unity_WorldToObject matrix.

i.e.

```
// Vertex Stage  (Object Space -> Clip Space)
VertexToFragment vert(VertexInput input)
{
    // Create a new VertexToFragment.
    VertexToFragment instance;

    // Convert vertex object space position to clip space position.
    instance.position = UnityObjectToClipPos(input.vertexPosition);
    // Grab the world position of the vertex in object space position.
    float4 worldPosition = mul(unity_ObjectToWorld, input.vertexPosition);
    /* The VertexToFragment struct uses a float3 for world position, so we
    swizzle to get the first 3 components from the earlier conversion.*/
    instance.worldPosition = worldPosition.xyz;

    /* Calculate world normals.  Notice we multiply by unity_WorldToObject
    rather than the usual unity_ObjectToWorld.  Not doing this will result
    in the normals being incorrect.*/
    float3 worldNormal = mul(input.vertexNormal, unity_WorldToObject);
    // Normalize the world normal vectors.  This means make the magnitude 1.
    worldNormal = normalize(worldNormal);


    instance.worldNormal = worldNormal;

    return instance;
}
```

Using VertexNormal to WorldNormal Shader, the normals (colored) will look something like this:



Red, Green, Blue here correspondings to the world axis directions X,Y,Z respectively.

Notice that the coloration is based on the normals of the object surfaces.

VertexNormal to WorldNormal Shader

```
Shader "MyShaders/TriPlanarMapping" {
    Properties
    {
        _Color ("Color", Color) = (0,0,0,1)
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Tags
        {
            "RenderType" = "Opaque"
            "Queue" = "Geometry"
        }

        Pass
        {
           CGPROGRAM
            //include useful shader functions
            #include "UnityCG.cginc"
            //define vertex and fragment shader
            #pragma vertex vert
            #pragma fragment frag
            // Properties
            float4 _Color;
            sampler2D _MainTex;
            float4 _MainTex_ST;
            // Engine -> Vertex
            struct VertexInput
            {
                float4 vertexPosition : POSITION;

                float3 vertexNormal : NORMAL;
            };
            // Vertex -> Fragment
            struct VertexToFragment
            {
                float4 position : SV_POSITION;
                float3 worldPosition : TEXCOORD0;

                float3 worldNormal : NORMAL;
            };

            // Vertex Stage  (Object Space -> Clip Space)
            VertexToFragment vert(VertexInput input)
            {
                // Create a new VertexToFragment.
                VertexToFragment instance;

                // Convert vertex object space position to clip space position.
                instance.position = UnityObjectToClipPos(input.vertexPosition);
                // Grab the world position of the vertex in object space position.
                float4 worldPosition = mul(unity_ObjectToWorld, input.vertexPosition);
                /* The VertexToFragment struct uses a float3 for world position, so we
                swizzle to get the first 3 components from the earlier conversion.*/
                instance.worldPosition = worldPosition.xyz;
```

```
            /* Calculate world normals.  Notice we multiply by unity_WorldToObject
            rather than the usual unity_ObjectToWorld.  Not doing this will result
            in the normals being incorrect.*/
            float3 worldNormal = mul(input.vertexNormal, unity_WorldToObject);
            // Normalize the world normal vectors.  This means make the magnitude 1.
            worldNormal = normalize(worldNormal);
            instance.worldNormal = worldNormal;

            return instance;
        }

        // Fragment Stage (Outputing colors for pixels, hence why we use fixed4)
        fixed4 frag(VertexToFragment input) : SV_TARGET
        {
            // Pixel color is based on the world normal vector.
            fixed4 finalPixel = fixed4(input.worldNormal.xyz,1);

            finalPixel *= _Color;
            return finalPixel;
        }

        ENDCG
    }
  }
  Fallback "VertexLit"
}
```