DEVELOPING CUSTOM LIGHTING MODELS IN UNITY

NIGHTMARE: MALARIA

Brian Kehrer

@birdimus

NIGHTMARE: MALARIA

- Over 1,000,000 users
- Developed in 3 months
- Around 9 people-months of work (~ 3 people at any time)

LIGHTING GOALS:

- **DOZENS OF LIGHTS PER SCENE**
- **HIGHLY CONFIGURABLE**
- LOW MEMORY (NO LIGHTMAPS)
- IPAD 2
- ‘UNDERWATER’ FEEL
- VIGNETTE

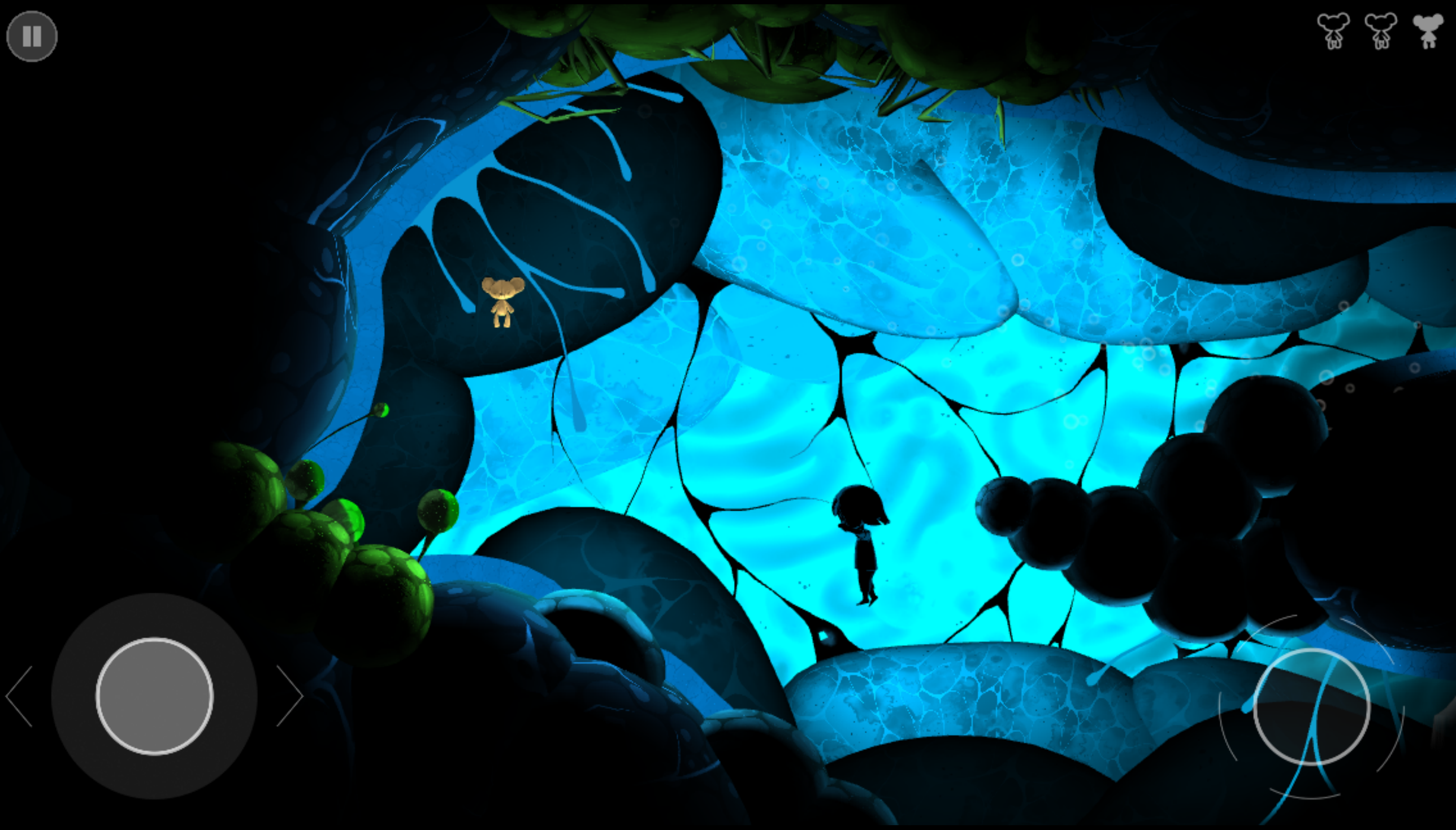
LIMITATIONS

- **UNITY ONLY SUPPORTS 4 VERTEX LIGHTS PER SCENE**
- **WE COULDN'T AFFORD LOTS OF PIXEL LIGHTS**
- LIGHTS 'POP' BETWEEN SPHERICAL HARMONICS AND VERTEX LIGHTS
- LIGHTMAPS REQUIRED TOO MUCH MEMORY, TOO SLOW TO ITERATE

WHERE WE ENDED UP

RESULT

- **4 VERTEX LIGHTS PER OBJECT**
- **HIGHER QUALITY VERTEX LIGHTING**
- **LIGHT LINKING**
- **FULL SCREEN RIPPLE & VIGNETTE**
- **NO RENDER TEXTURES OR LIGHT MAPS**

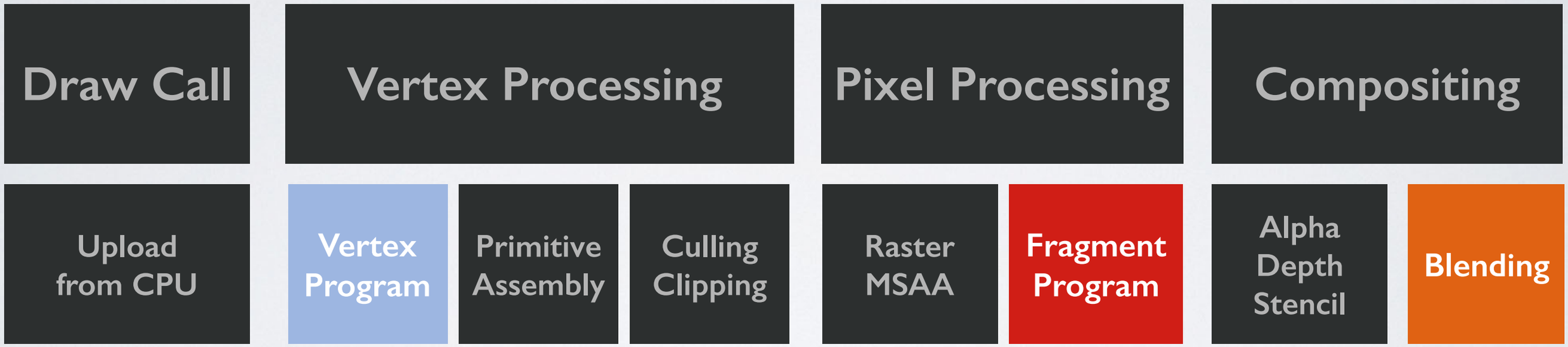


HTTPS://GITHUB.COM/ICOSAHEDRA/SHADERDEMO

A **SHADER** IS A **SET OF INSTRUCTIONS** (CODE), FOLLOWING **RULES** (API - E.G. DX9.0C, DX11, OPEN GL 3.2) IN ORDER TO **PERFORM OPERATIONS** ON THE **GRAPHICS CARD**.

A **SHADER** IS A **PROGRAM** THAT **DRAWS STUFF** ON SCREEN.

GRAPHICS HARDWARE PIPELINE



ANATOMY OF A SHADER

SOMEWHAT UNITY SPECIFIC

```
Shader "Presentation/Simple Shader" {

Properties {
    _Color ("Main Color", Color) = (1,1,1,0.5)
    _MainTex ("Texture", 2D) = "white" { }
}

SubShader {
    Tags {"Queue"="Geometry" "RenderType"="Opaque"}

    Pass {
        Cull Back // default
        ZWrite On //default
        ZTest LEqual //default
        Blend Off
        AlphaTest Off


CGPROGRAM
#pragma vertex VertexProgram
#pragma fragment FragmentProgram

#include "UnityCG.cginc"

float4 _Color;
sampler2D _MainTex;
float4 _MainTex_ST;

struct VertexInput {
    float4  position : POSITION;
    float4  uv : TEXCOORD0;
};

struct VertexToFragment {
    float4  position : POSITION;
    float2  uv : TEXCOORD0;
};

VertexToFragment VertexProgram (VertexInput vertex)
{
    VertexToFragment output;
    output.position = mul (UNITY_MATRIX_MVP, vertex.position);
    output.uv = TRANSFORM_TEX (vertex.uv, _MainTex);
    return output;
}

half4 FragmentProgram (VertexToFragment fragment) : COLOR
{
    half4 texcol = tex2D (_MainTex, fragment.uv);
    return texcol * _Color;
}

ENDCG

    }
}

Fallback "VertexLit"
}
```

ANATOMY OF A SHADER

```
Shader "Presentation/Simple Shader" {

Properties {
    _Color ("Main Color", Color) = (1,1,1,0.5)
    _MainTex ("Texture", 2D) = "white" { }
}

SubShader {
    Tags {"Queue"="Geometry" "RenderType"="Opaque"}

    Pass {
        Cull Back // default
        ZWrite On //default
        ZTest LEqual //default
        Blend Off
        AlphaTest Off


CGPROGRAM
#pragma vertex VertexProgram
#pragma fragment FragmentProgram

#include "UnityCG.cginc"

float4 _Color;
sampler2D _MainTex;
float4 _MainTex_ST;

struct VertexInput {
    float4  position : POSITION;
    float4  uv : TEXCOORD0;
};

struct VertexToFragment {
    float4  position : POSITION;
    float2  uv : TEXCOORD0;
};

VertexToFragment VertexProgram (VertexInput vertex)
{
    VertexToFragment output;
    output.position = mul (UNITY_MATRIX_MVP, vertex.position);
    output.uv = TRANSFORM_TEX (vertex.uv, _MainTex);
    return output;
}

half4 FragmentProgram (VertexToFragment fragment) : COLOR
{
    half4 texcol = tex2D (_MainTex, fragment.uv);
    return texcol * _Color;
}

ENDCG

    }
}

Fallback "VertexLit"
}
```

```
Shader "Presentation/Simple Shader" {

Properties {
    _Color ("Main Color", Color) = (1,1,1,0.5)
    _MainTex ("Texture", 2D) = "white" { }
}

SubShader {
    Tags {"Queue"="Geometry" "RenderType"="Opaque"}

    Pass {
        Cull Back // default
        ZWrite On //default
        ZTest LEqual //default
        Blend Off
        AlphaTest Off
```

ANATOMY OF A SHADER

```
Shader "Presentation/Simple Shader" {

Properties {
    _Color ("Main Color", Color) = (1,1,1,0.5)
    _MainTex ("Texture", 2D) = "white" { }
}

SubShader {
    Tags {"Queue"="Geometry" "RenderType"="Opaque"}

    Pass {
        Cull Back // default
        ZWrite On //default
        ZTest LEqual //default
        Blend Off
        AlphaTest Off

CGPROGRAM
#pragma vertex VertexProgram
#pragma fragment FragmentProgram

#include "UnityCG.cginc"

float4 _Color;
sampler2D _MainTex;
float4 _MainTex_ST;

struct VertexInput {
    float4  position : POSITION;
    float4  uv : TEXCOORD0;
};

struct VertexToFragment {
    float4  position : POSITION;
    float2  uv : TEXCOORD0;
};

VertexToFragment VertexProgram (VertexInput vertex)
{
    VertexToFragment output;
    output.position = mul (UNITY_MATRIX_MVP, vertex.position);
    output.uv = TRANSFORM_TEX (vertex.uv, _MainTex);
    return output;
}

half4 FragmentProgram (VertexToFragment fragment) : COLOR
{
    half4 texcol = tex2D (_MainTex, fragment.uv);
    return texcol * _Color;
}

ENDCG

    }
}

Fallback "VertexLit"
}
```

```
CGPROGRAM
#pragma vertex VertexProgram
#pragma fragment FragmentProgram

#include "UnityCG.cginc"

float4 _Color;
sampler2D _MainTex;
float4 _MainTex_ST;

struct VertexInput {
    float4  position : POSITION;
    float4  uv : TEXCOORD0;
};

struct VertexToFragment {
    float4  position : POSITION;
    float2  uv : TEXCOORD0;
};
```

ANATOMY OF A SHADER

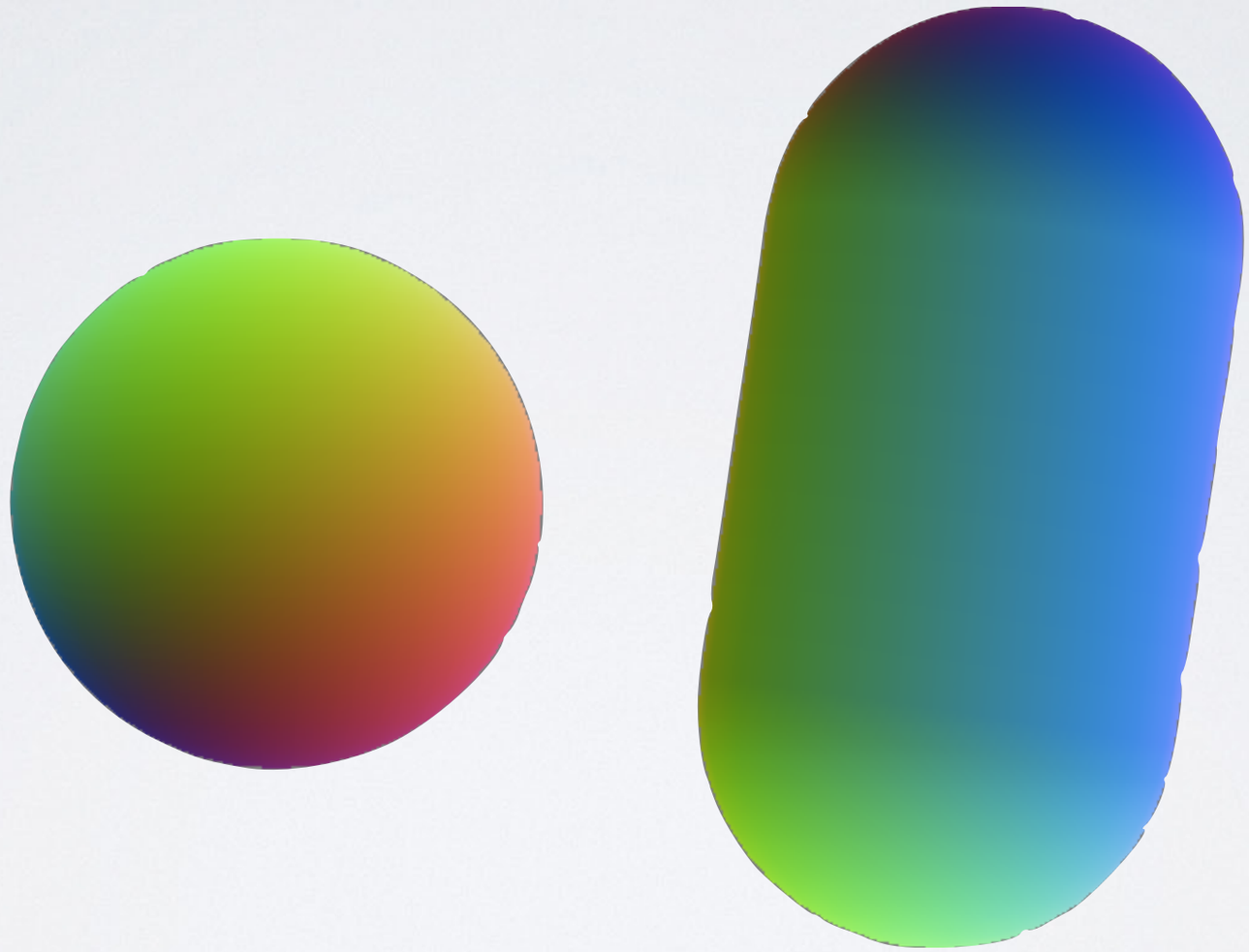
```
Shader "Presentation/Simple Shader" {

Properties {
    _Color ("Main Color", Color) = (1,1,1,0.5)
    _MainTex ("Texture", 2D) = "white" { }
}

SubShader {
    Tags {"Queue"="Geometry" "RenderType"

    Pass {
        Cull Back // default
        ZWrite On //default
        ZTest LEqual //default
        Blend Off
        AlphaTest Off


CGPROGRAM
#pragma vertex VertexProgram
#pragma fragment FragmentProgram

#include "UnityCG.cginc"

float4 _Color;
sampler2D _MainTex;
float4 _MainTex_ST;

struct VertexInput {
    float4  position : POSITION;
    float4  uv : TEXCOORD0;
};

struct VertexToFragment {
    float4  position : POSITION;
    float2  uv : TEXCOORD0;
};

VertexToFragment VertexProgram (VertexInp
{
    VertexToFragment output;
    output.position = mul (UNITY_MATRIX_M
    output.uv = TRANSFORM_TEX (vertex.uv,
    return output;
}

half4 FragmentProgram (VertexToFragment f
{
    half4 texcol = tex2D (_MainTex, fragm
    return texcol * _Color;
}

ENDCG

    }
}

Fallback "VertexLit"
}
```

```
VertexToFragment VertexProgram (VertexInput vertex)
{
    VertexToFragment output;
    output.position = mul (UNITY_MATRIX_MVP, vertex.position);
    output.uv = TRANSFORM_TEX (vertex.uv, _MainTex);
    return output;
}

half4 FragmentProgram (VertexToFragment fragment) : COLOR
{
    half4 texcol = tex2D (_MainTex, fragment.uv);
    return texcol * _Color;
}

ENDCG

    }
}

Fallback "VertexLit"
}
```

DEBUG NORMALS

```
24  struct VertexInput {
25      float4  position : POSITION;
26      float4  normal : NORMAL;
27  };
28
29  struct VertexToFragment {
30      float4  position : POSITION;
31      float4  color : COLOR0;
32  };
33
34  VertexToFragment VertexProgram (VertexInput vertex)
35  {
36      VertexToFragment output;
37      output.position = mul (UNITY_MATRIX_MVP, vertex.position);
38      output.color = 0.5*vertex.normal + 0.5;
39      return output;
40  }
41
42  half4 FragmentProgram (VertexToFragment fragment) : COLOR
43  {
44      return fragment.color;
45  }
```

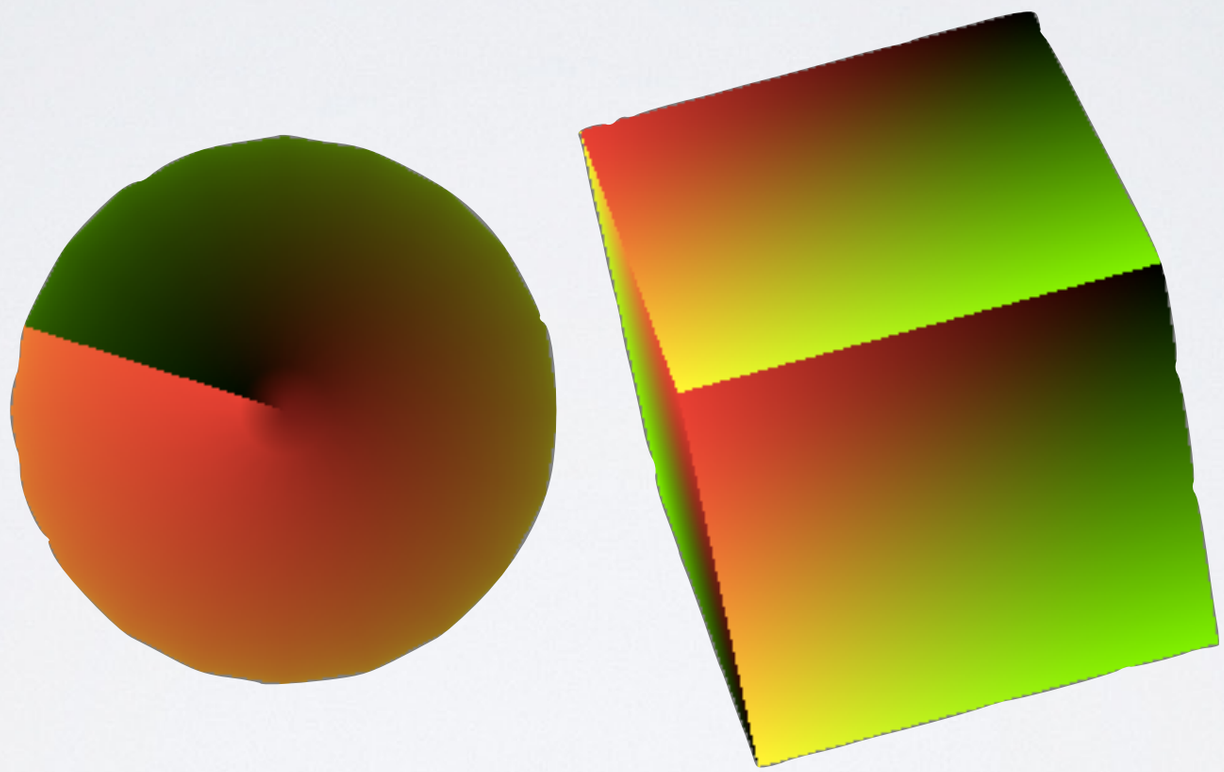
DEBUG NORMALS



DEBUG UV

```
24
25  struct VertexInput {
26      float4  position : POSITION;
27      float4  uv : TEXCOORD0;
28  };
29
30  struct VertexToFragment {
31      float4  position : POSITION;
32      float4  color : COLOR0;
33  };
34
35  VertexToFragment VertexProgram (VertexInput vertex)
36  {
37      VertexToFragment output;
38      output.position = mul (UNITY_MATRIX_MVP, vertex.position);
39      output.color.rg = vertex.uv.xy;
40      return output;
41  }
```

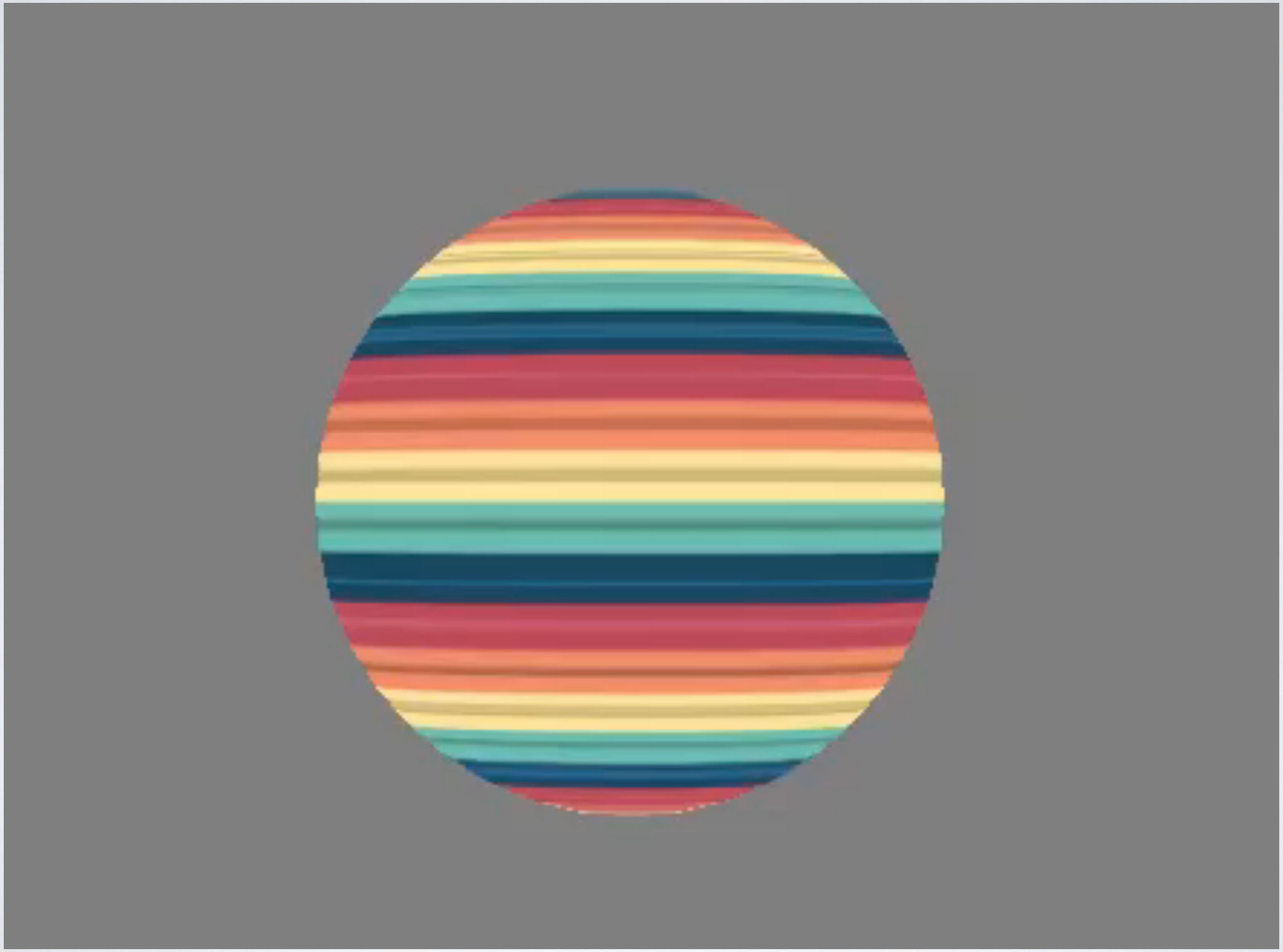
DEBUG UV



UV FLOW

```
37  VertexToFragment VertexProgram (VertexInput vertex)
38  {
39      VertexToFragment output;
40      output.position = mul (UNITY_MATRIX_MVP, vertex.position);
41      vertex.uv.x = fmod(0.5*_Time.y,1);
42      output.uv = vertex.uv.xy;
43      return output;
44  }
45
46  half4 FragmentProgram (VertexToFragment fragment) : COLOR
47  {
48      return tex2D (_MainTex, fragment.uv);
49  }
```

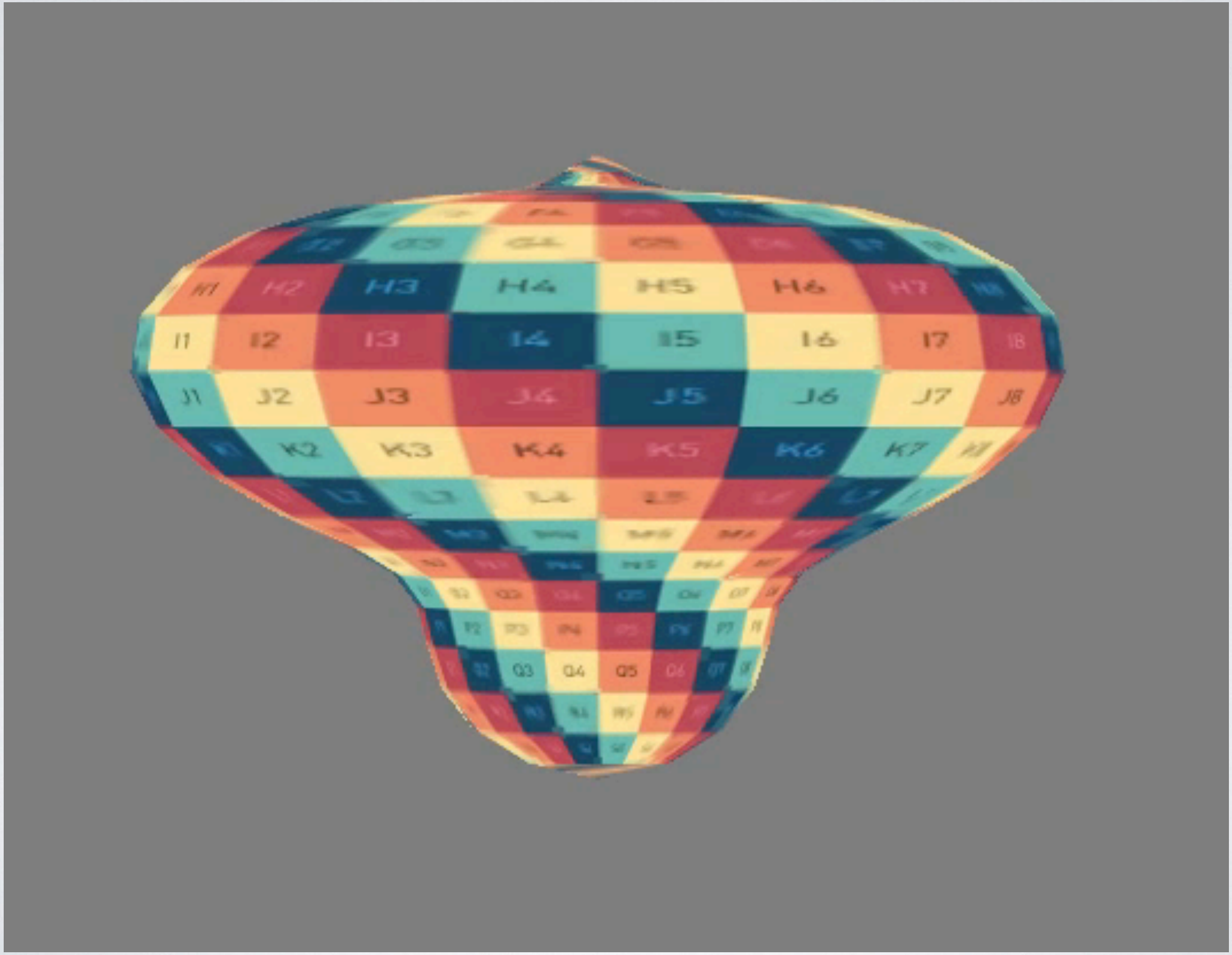
UV FLOW



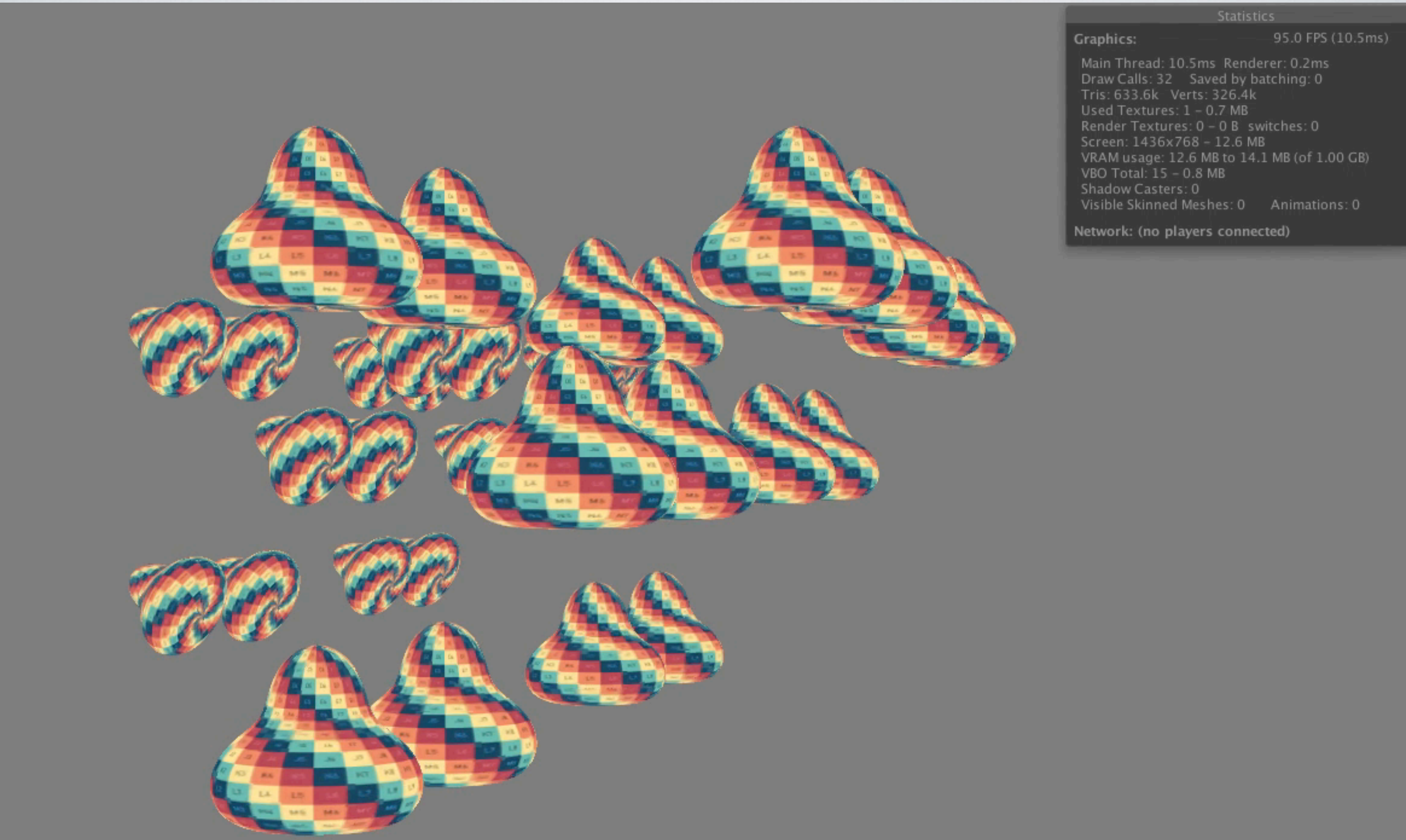
VERTEX POSITION

```
27  struct VertexInput {
28      float4  position : POSITION;
29      float4  uv : TEXCOORD0;
30      float4  normal : NORMAL;
31  };
32
33  struct VertexToFragment {
34      float4  position : POSITION;
35      float2  uv : TEXCOORD0;
36  };
37
38  VertexToFragment VertexProgram (VertexInput vertex)
39  {
40      VertexToFragment output;
41      output.position = mul (UNITY_MATRIX_MVP, vertex.position);
42      output.uv = vertex.uv.xy;
43      output.position += vertex.normal*(1+sin(_Time.y + 10*output.uv.y));
44      return output;
45  }
46
47  half4 FragmentProgram (VertexToFragment fragment) : COLOR
48  {
49      return tex2D (_MainTex, fragment.uv);
50  }
```

VERTEX POSITION



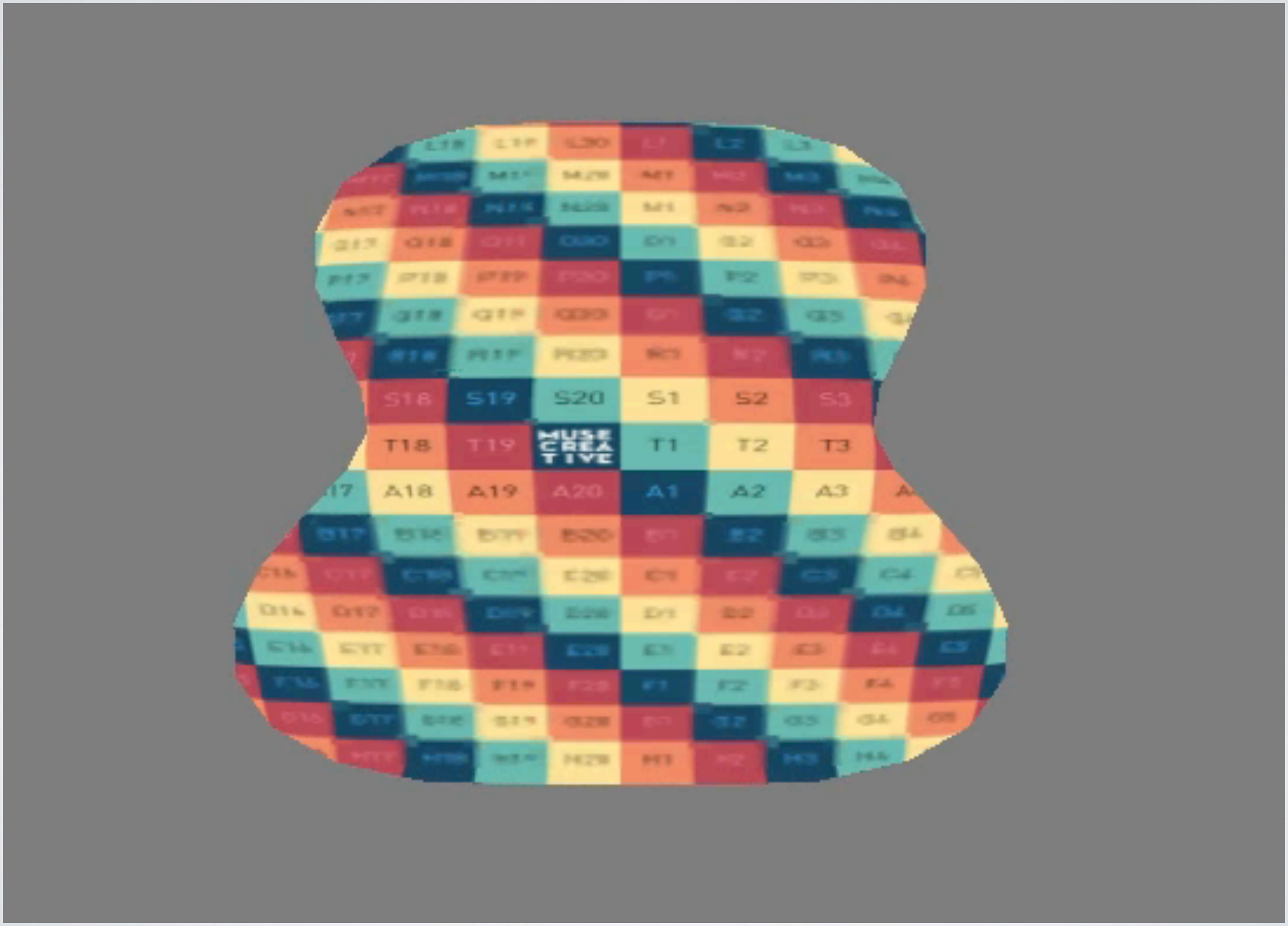
600,000 ANIMATED VERTS (MB AIR)



OBJECT SPACE UV

```
38  VertexToFragment VertexProgram (VertexInput vertex)
39  {
40      VertexToFragment output;
41      output.position = mul (UNITY_MATRIX_MVP, vertex.position);
42      output.uv = vertex.uv.xy;
43      output.position += vertex.normal*(1+sin(_Time.y + 10*output.uv.y));
44      output.uv = output.position/10;
45      return output;
46  }
```

OBJECT SPACE UV



MORE VERTEX DATA!

WE CAN USE IT ANY WAY WE WANT

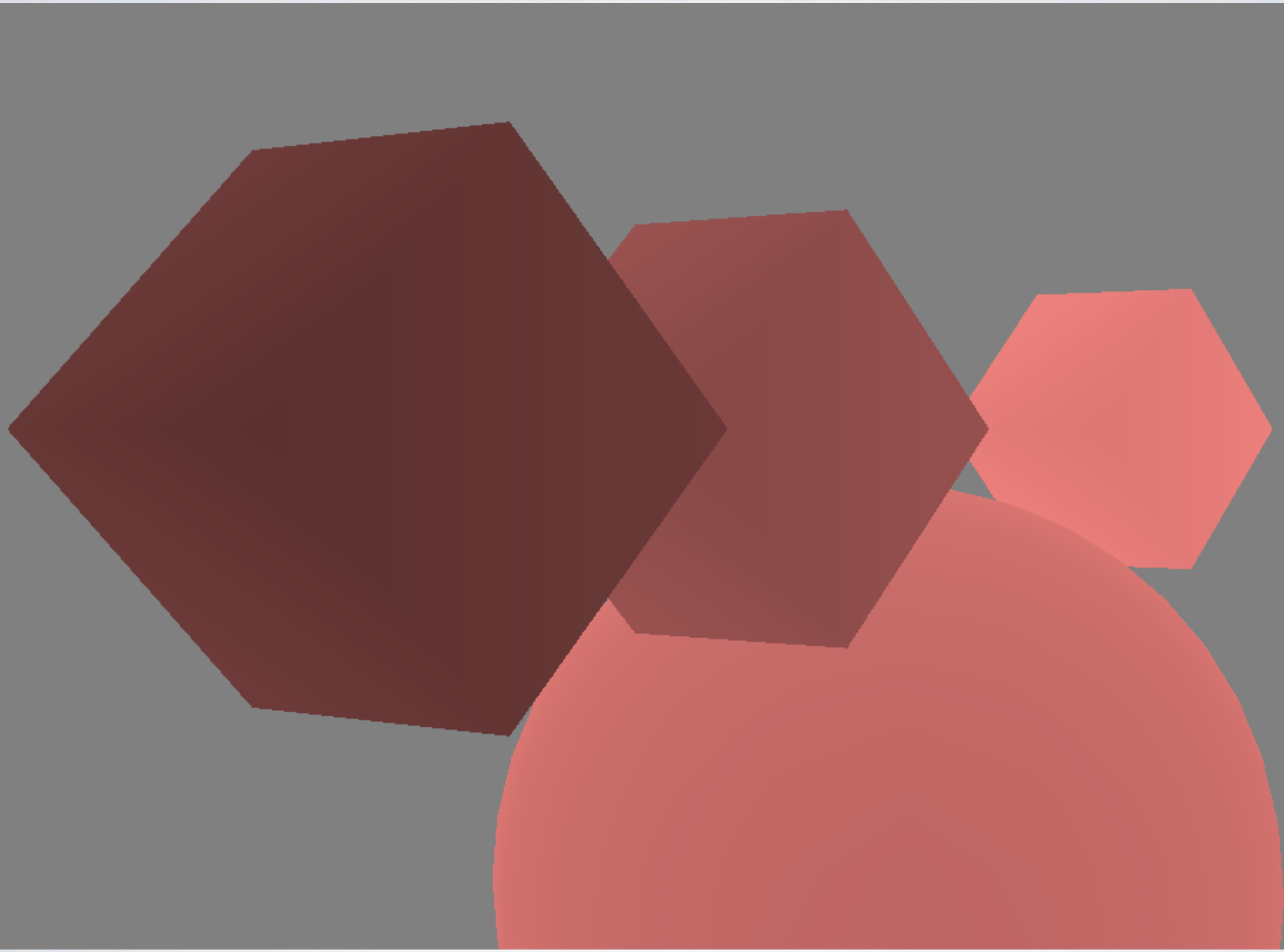
```
27  struct VertexInput {
28      float4  position : POSITION;
29      float4  uv : TEXCOORD0;
30      float4  uv2 : TEXCOORD1;
31      float4  uv3 : TEXCOORD2;
32      float4  uv4 : TEXCOORD3;
33      float4  color : COLOR0;
34      float4  color2 : COLOR1;
35      float4  color3 : COLOR2;
36      float4  normal : NORMAL;
37      float4  tangent : TANGENT0;
38  };
```

8 UV channels,
3 colors,
Normals,
Tangents,
Positions

DISTANCE

```
40
41  VertexToFragment VertexProgram (VertexInput vertex)
42  {
43      VertexToFragment output;
44      output.position = output.screenPosition = mul (UNITY_MATRIX_MVP, vertex.position);
45      output.worldPosition = mul(_Object2World, vertex.position);
46      output.uv = TRANSFORM_TEX (vertex.uv, _MainTex);
47      return output;
48  }
49
50  half4 FragmentProgram (VertexToFragment fragment) : COLOR
51  {
52      float distanceToPoint = distance(_WorldSpaceCameraPos.xyz, fragment.worldPosition.xyz)/10;
53      half4 texcol = half4(fragment.screenPosition.z)/10;
54      return texcol * _Color;
55  }
56
```

DISTANCE



BLENDING
FIXED FUNCTION

SHADER VALUE * BlendFactor

BlendOp (+)

SCREEN BUFFER * BlendFactor

RESULT

BLENDING FACTORS

- One
- Zero
- SrcColor
- SrcAlpha
- DstColor
- DstAlpha
- OneMinusSrcColor
- OneMinusSrcAlpha
- OneMinusDstColor
- OneMinusDstAlpha

BLENDING OPERATIONS

- Add - Addition (default)
- Sub - Subtract source from destination
- RevSub - Subtract destination from source
- Min - Minimum value
- Max - Maximum value

ALPHA BLENDING

SHADER VALUE * **SrcAlpha**

\+

SCREEN BUFFER * **OneMinusSrcAlpha**

RESULT

ADDITIVE BLENDING

SHADER VALUE * **One**

\+

SCREEN BUFFER * **One**

RESULT

MULTIPLICATIVE BLENDING

SHADER VALUE * **DstColor**

\+

SCREEN BUFFER * **Zero**

RESULT

LIGHTING GOALS:

- **DOZENS OF LIGHTS PER SCENE**
- **HIGHLY CONFIGURABLE**
- LOW MEMORY (NO LIGHTMAPS)
- IPAD 2
- ‘UNDERWATER’ FEEL
- VIGNETTE

LIMITATIONS

- **UNITY ONLY SUPPORTS 4 VERTEX LIGHTS PER SCENE**
- **WE COULDN'T AFFORD LOTS OF PIXEL LIGHTS**
- LIGHTS 'POP' BETWEEN SPHERICAL HARMONICS AND VERTEX LIGHTS
- LIGHTMAPS REQUIRED TOO MUCH MEMORY, TOO SLOW TO ITERATE

RESULT

- **4 VERTEX LIGHTS PER OBJECT**
- **HIGHER QUALITY VERTEX LIGHTING**
- **LIGHT LINKING**
- **FULL SCREEN RIPPLE & VIGNETTE**
- **NO RENDER TEXTURES OR LIGHT MAPS**