Shaders

theory and examples with Unity

What is a shader?

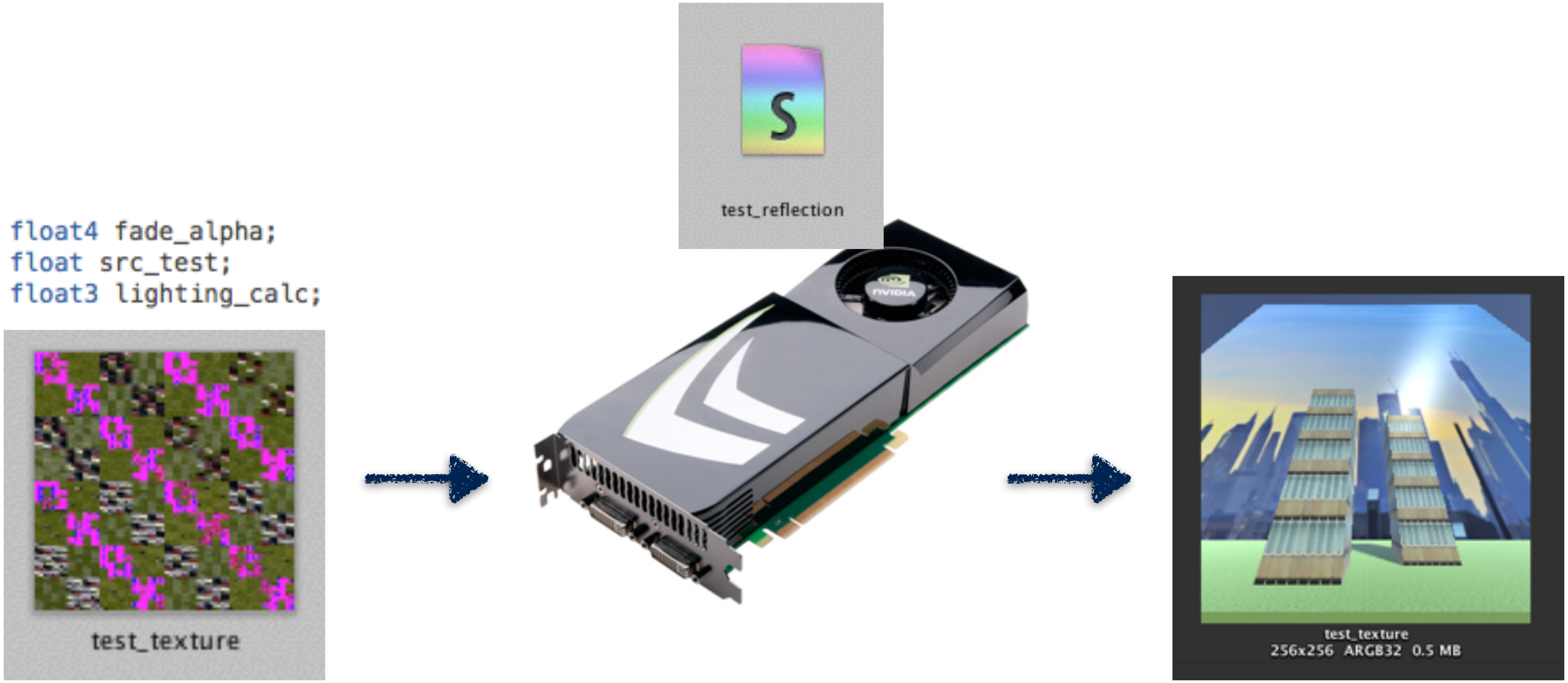
It’s a special program you send to your GPU to render a texture.

What is a GPU?

It’s a processor that’s very efficient at computing pixel data.

You could do everything you do
in shaders in « regular » code…
just not at 60 FPS.

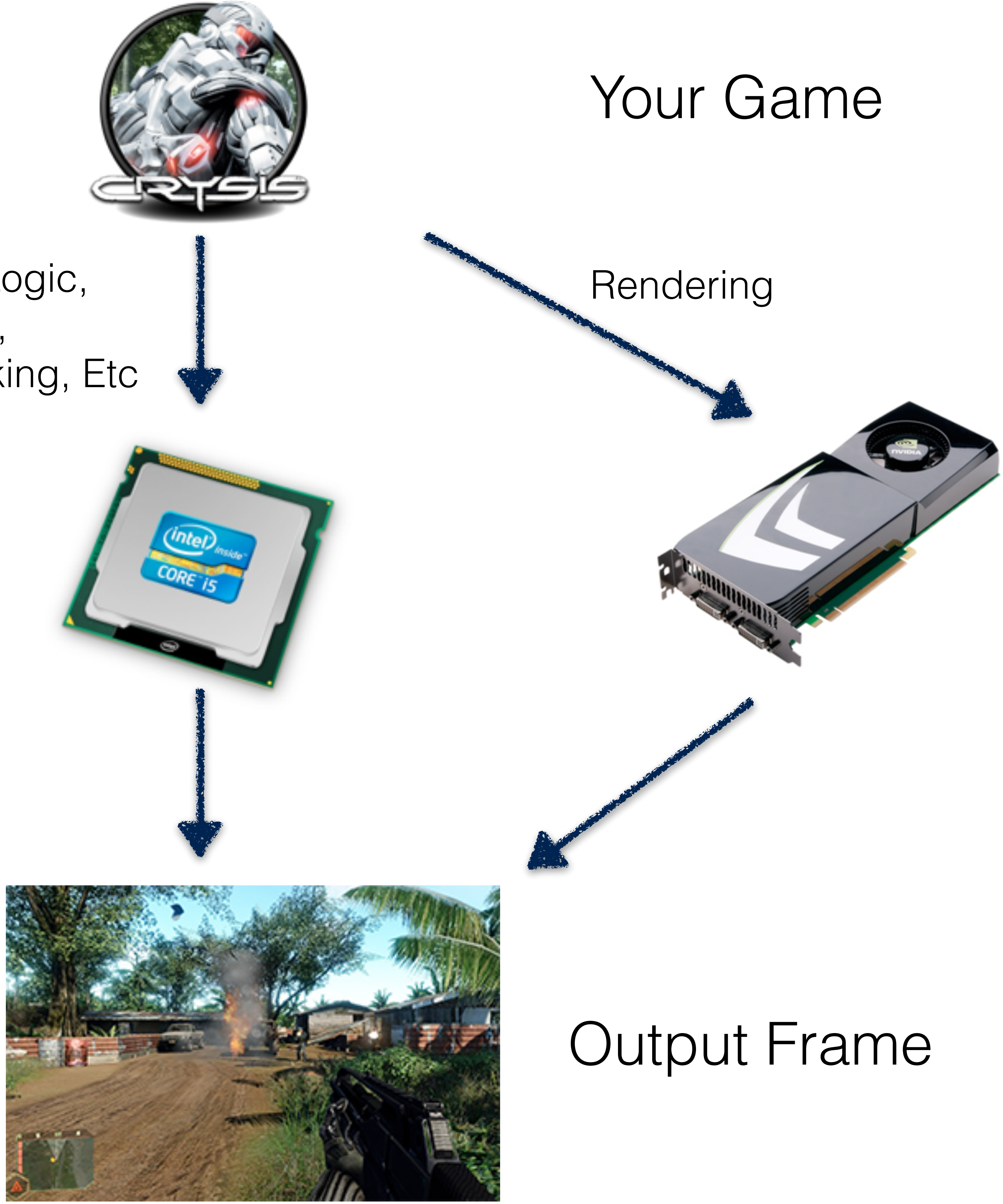
What is a shader?



```
float4 fade_alpha;
float src_test;
float3 lighting_calc;
```

It’s a program that takes numbers and textures and outputs a texture.

Big Picture



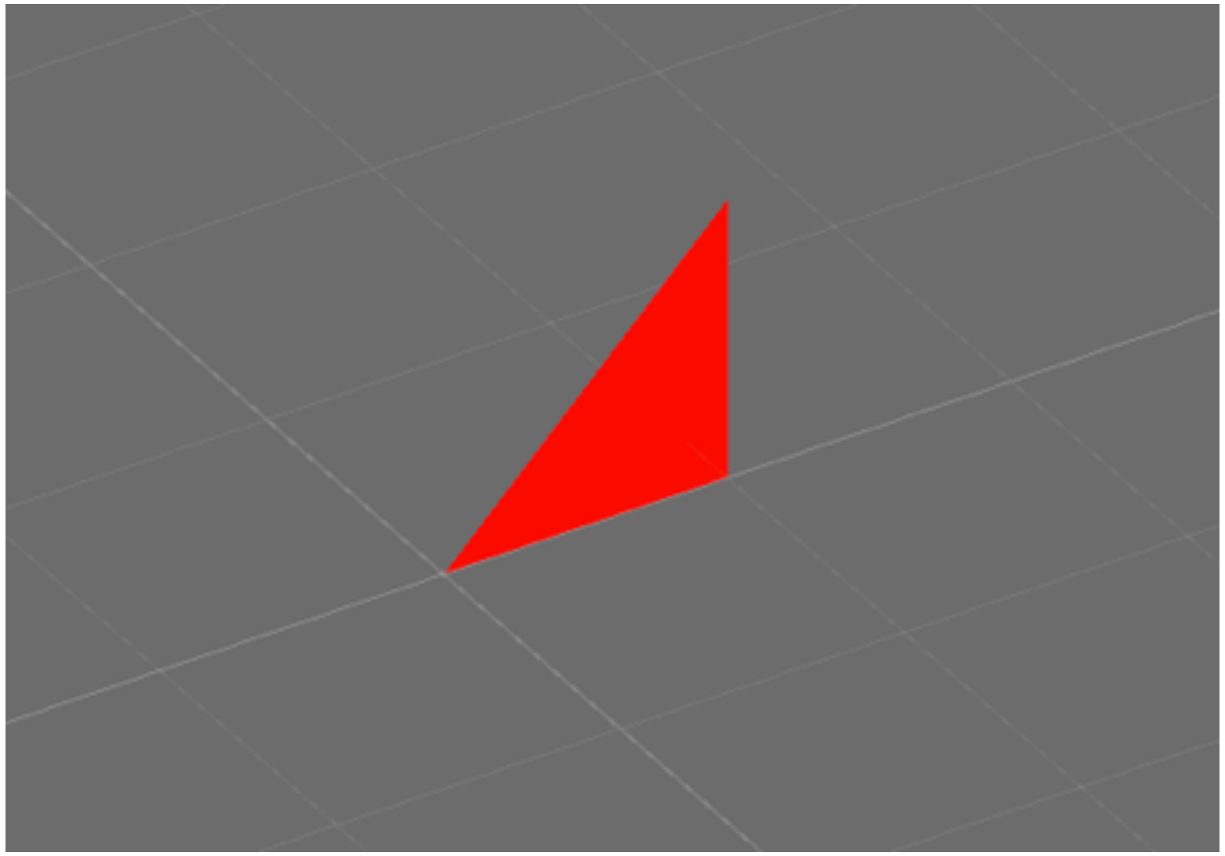
Your Game

Game Logic,
Physics,
Networking, Etc

Rendering

Output Frame

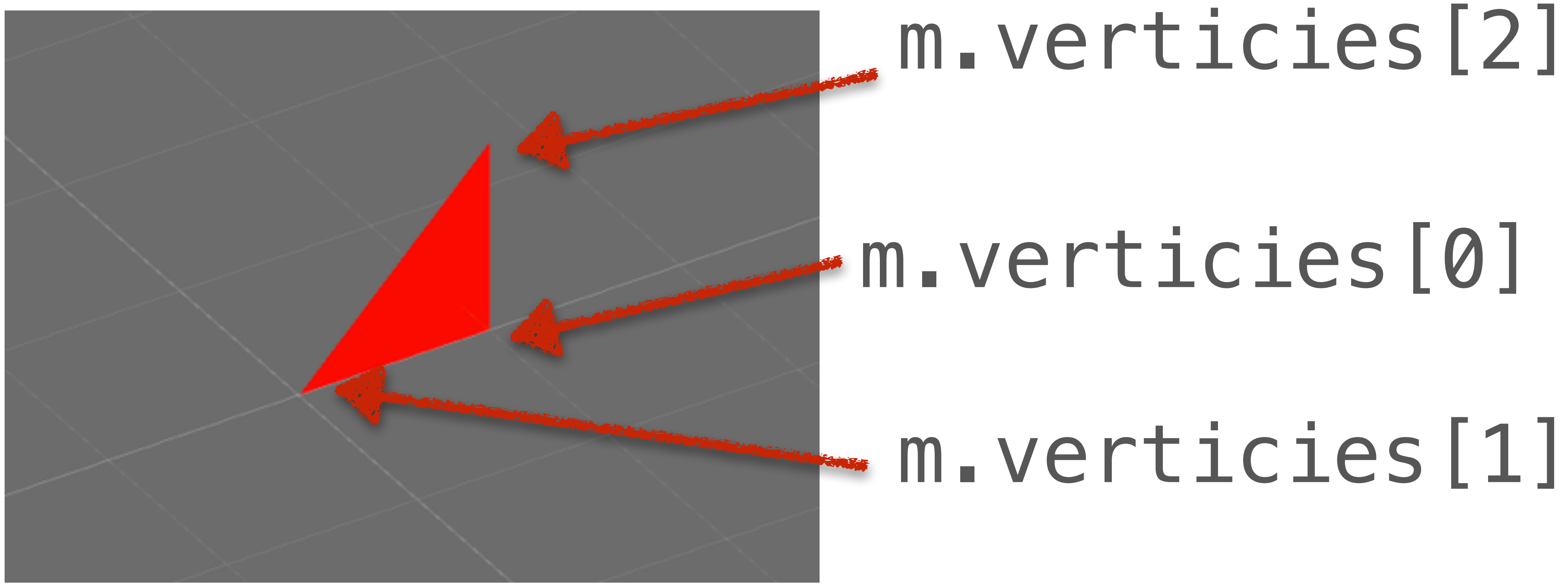
Representation of Geometry



How is a triangle like this represented?

Representation of Geometry

Through a series of points, or « verticies » in « object space ».



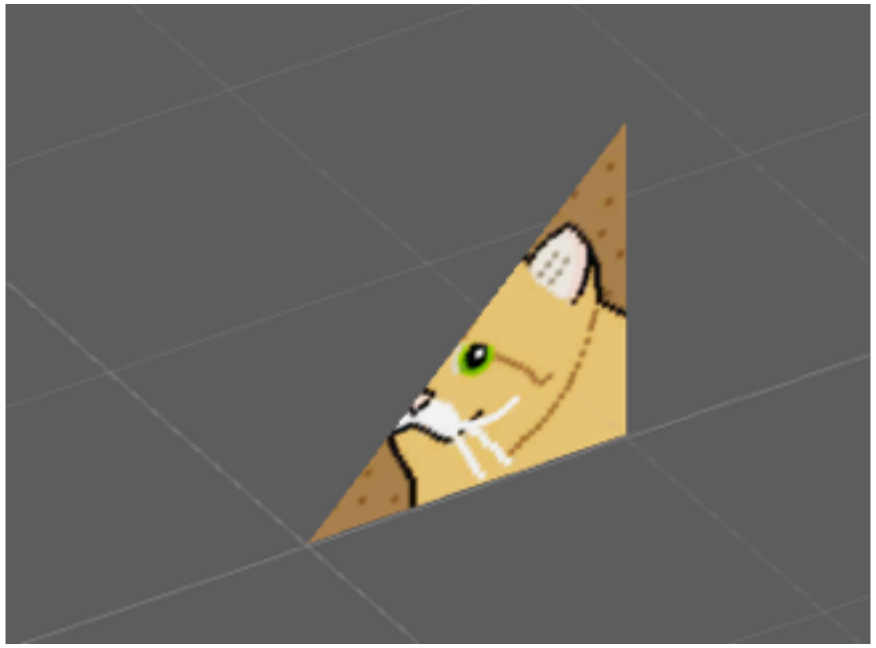
m.verticies[2]

m.verticies[0]

m.verticies[1]

```
m.vertices = new Vector3[] {
  new Vector3(0.0f,0.0f,0.0f),
  new Vector3(1.0f,0.0f,0.0f),
  new Vector3(0.0f,1.0f,0.0f),
} ;
```

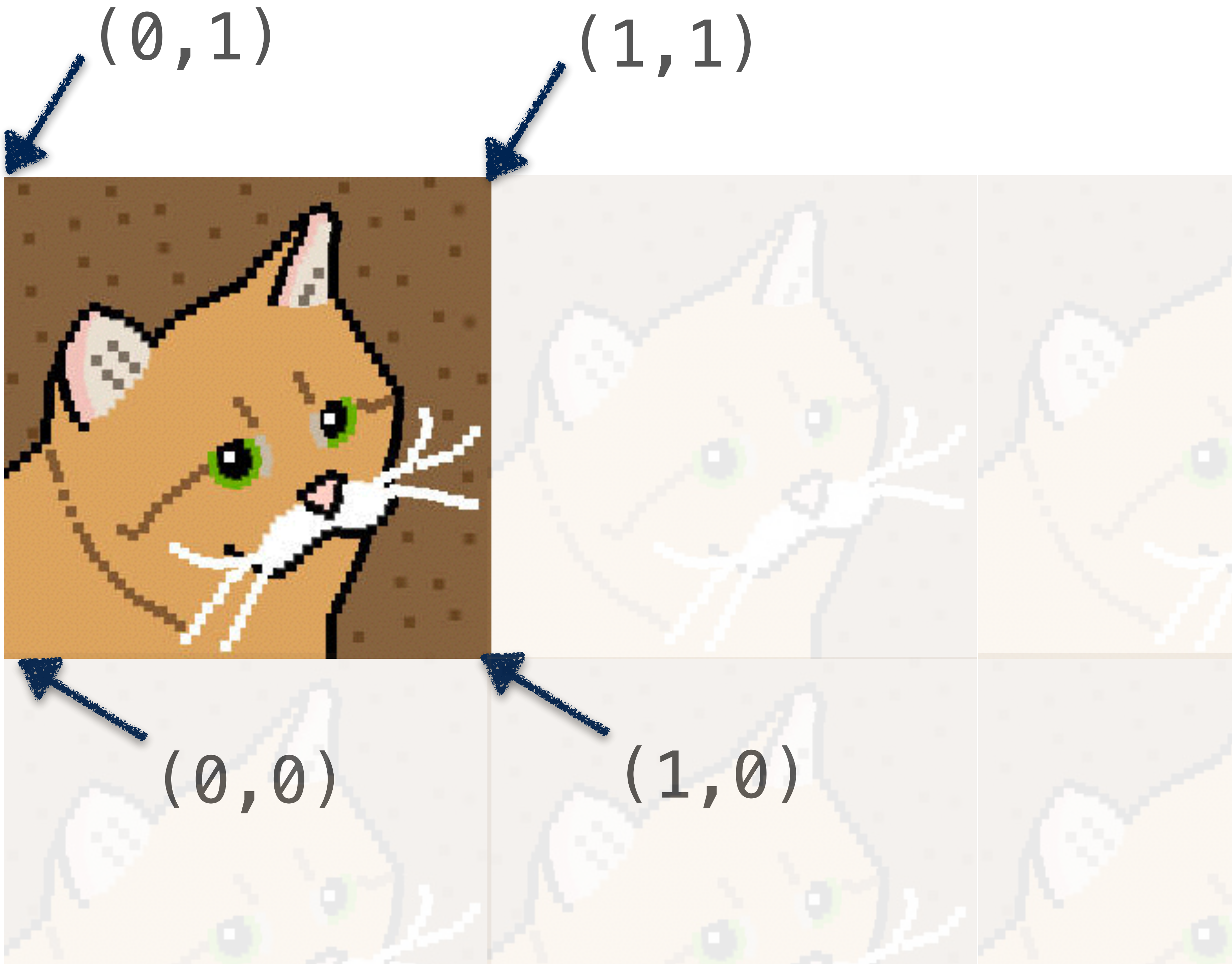
Representation of Objects



How is an image like this represented?

Textures

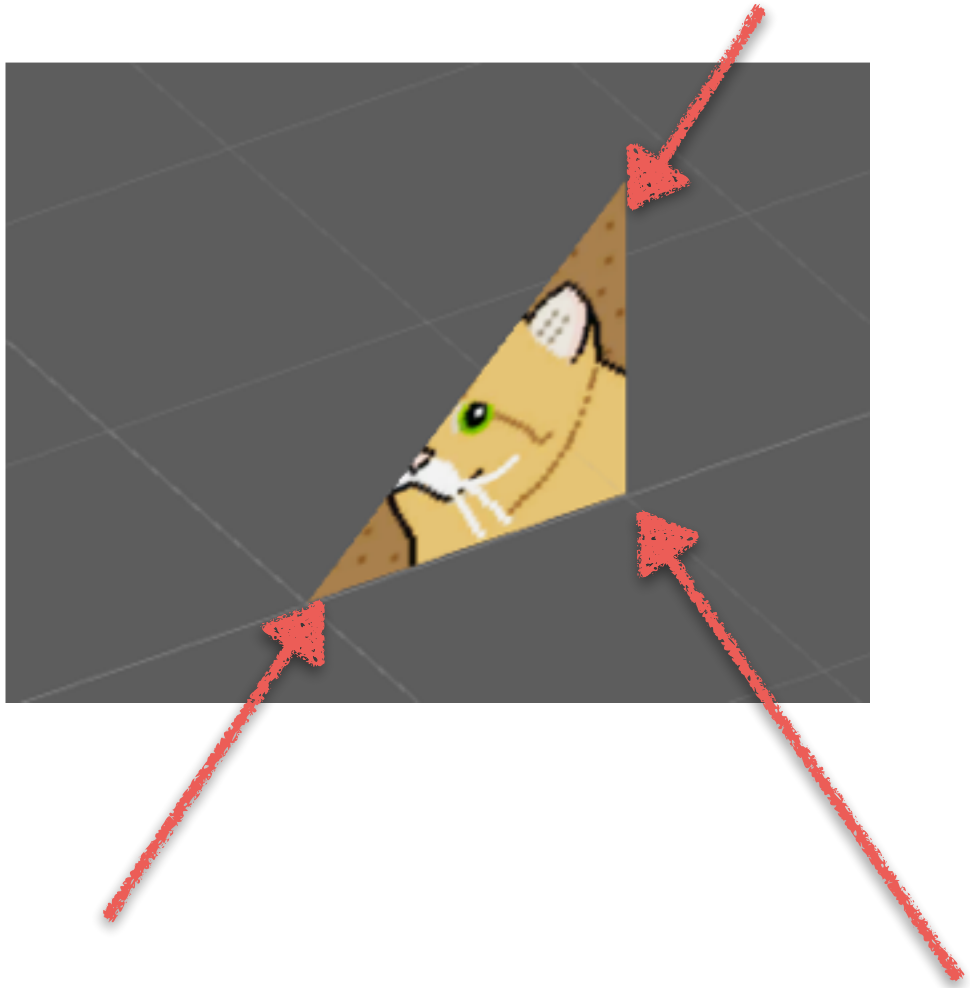
First, a texture is defined in « uv space ».



Per-vertex uv values

Then, every vertex is given a uv value.

```
vertex: <0,1,0>
uv: <0,1>
```



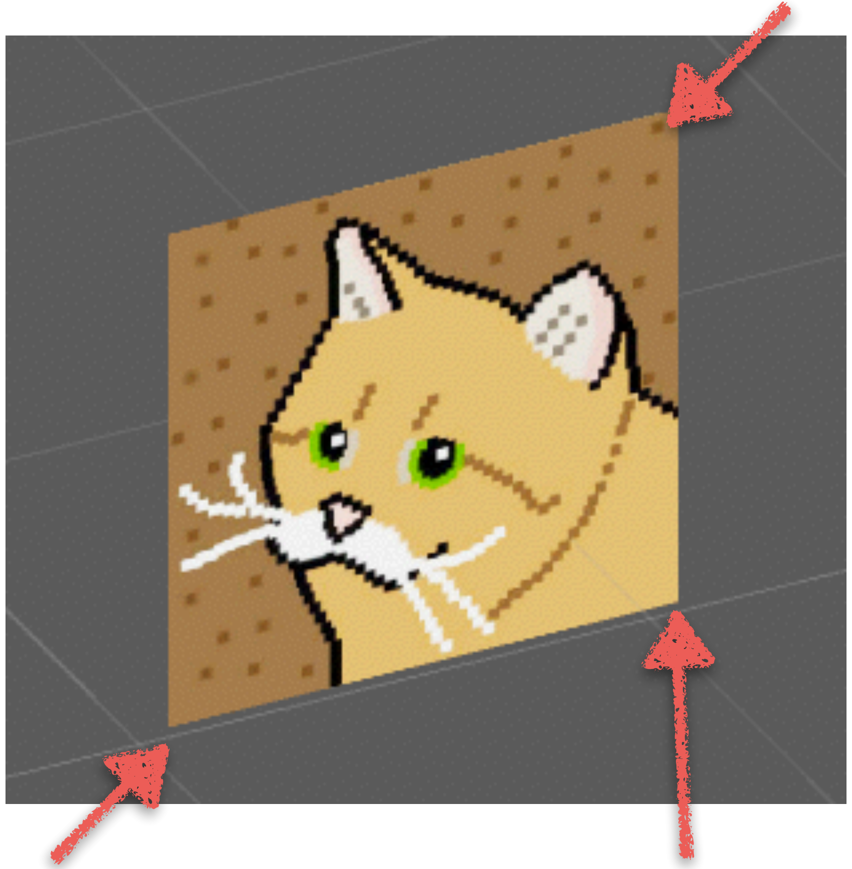
```
vertex: <1,0,0>
uv: <1,0>
```

```
vertex: <0,0,0>
uv: <0,0>
```

```
m.vertices = new Vector3[] {
    new Vector3(0.0f,0.0f,0.0f),
    new Vector3(1.0f,0.0f,0.0f),
    new Vector3(0.0f,1.0f,0.0f),
} ;
```

```
m.uv = new Vector2[] {
    new Vector2 (0, 0),
    new Vector2 (1, 0),
    new Vector2(0, 1),
} ;
```

Just checking.



vertex: <0,1,0>
uv: <0,1>

vertex: <1,0,0>
uv: <1,0>

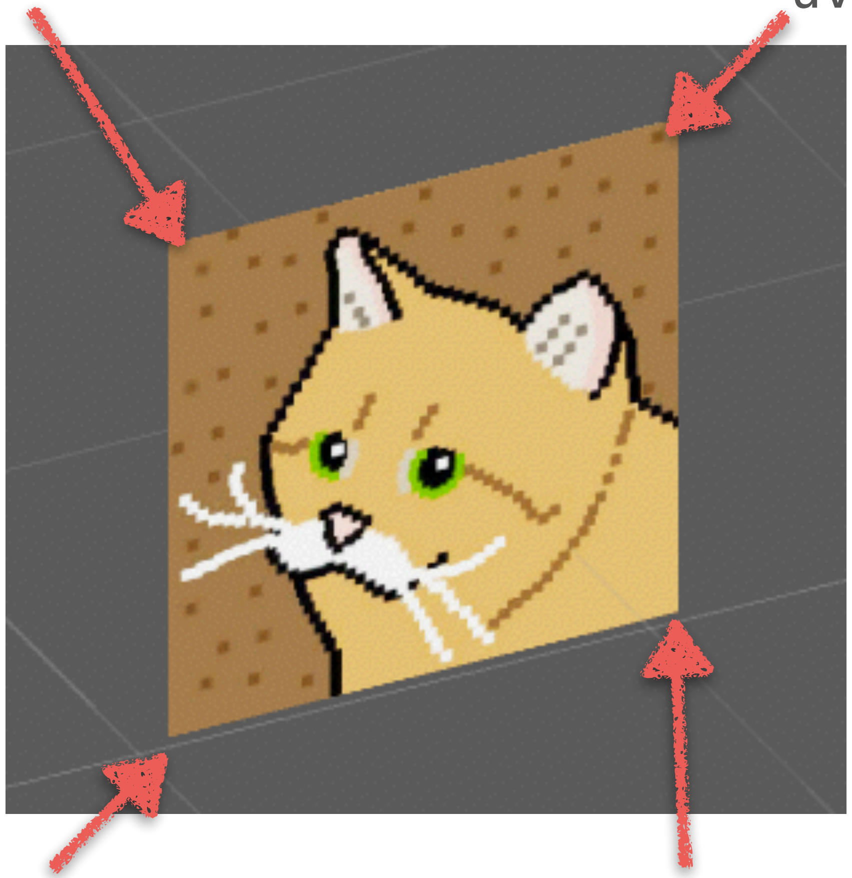
vertex: <0,0,0>
uv: <0,0>

What changes need to be made to get this full square?

```
m.vertices = new Vector3[] {
   new Vector3(0.0f,0.0f,0.0f),
   new Vector3(1.0f,0.0f,0.0f),
   new Vector3(0.0f,1.0f,0.0f),
} ;
```

```
m.uv = new Vector2[] {
   new Vector2 (0, 0),
   new Vector2 (1, 0),
   new Vector2(0, 1),
} ;
```

Just checking.



vertex: <1,1,0>
uv: <1,1>

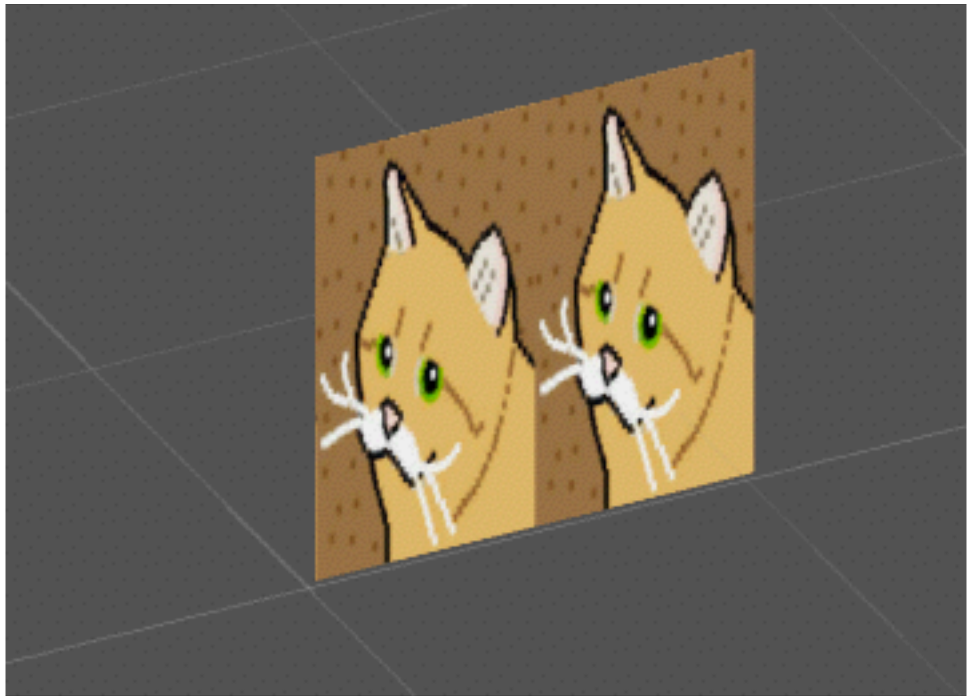
vertex: <0,1,0>
uv: <0,1>

vertex: <1,0,0>
uv: <1,0>

vertex: <0,0,0>
uv: <0,0>

```
m.vertices = new Vector3[] {
   new Vector3(0.0f,0.0f,0.0f),
   new Vector3(1.0f,0.0f,0.0f),
   new Vector3(0.0f,1.0f,0.0f),
   new Vector3(1.0f,1.0f,0.0f)
} ;
m.uv = new Vector2[] {
   new Vector2 (0, 0),
   new Vector2 (1, 0),
   new Vector2(0, 1),
   new Vector2(1,1)
} ;
```

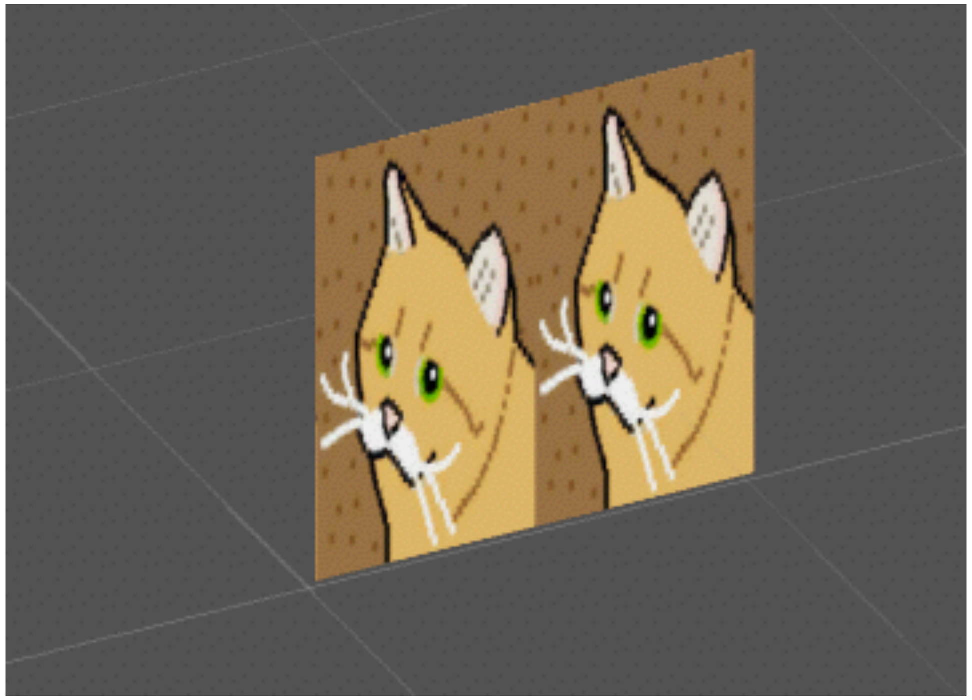
Just checking.



What changes need to get two cats?

```
m.vertices = new Vector3[] {
   new Vector3(0.0f,0.0f,0.0f),
   new Vector3(1.0f,0.0f,0.0f),
   new Vector3(0.0f,1.0f,0.0f),
   new Vector3(1.0f,1.0f,0.0f)
} ;
m.uv = new Vector2[] {
   new Vector2 (0, 0),
   new Vector2 (1, 0),
   new Vector2(0, 1),
   new Vector2(1,1)
} ;
```

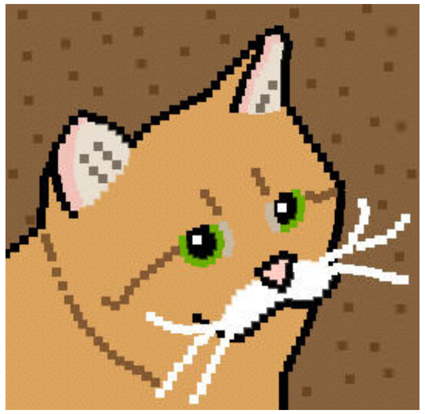
Just checking.

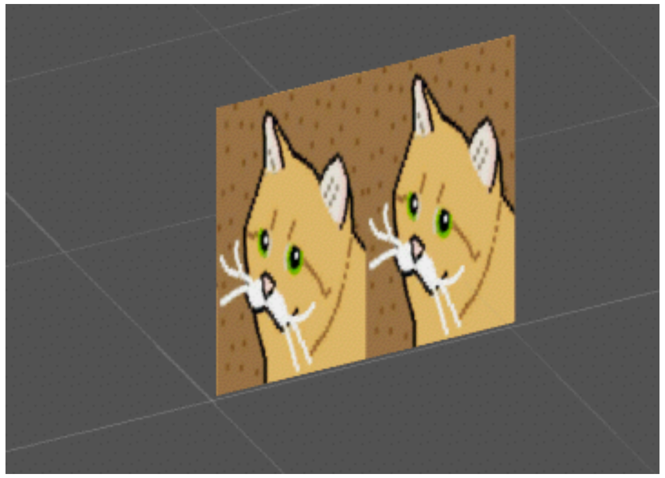


```
m.vertices = new Vector3[] {
   new Vector3(0.0f,0.0f,0.0f),
   new Vector3(1.0f,0.0f,0.0f),
   new Vector3(0.0f,1.0f,0.0f),
   new Vector3(1.0f,1.0f,0.0f)
} ;
m.uv = new Vector2[] {
   new Vector2 (0, 0),
   new Vector2 (2, 0),
   new Vector2(0, 1),
   new Vector2(2,1)
} ;
```

Back to shaders

```
m.vertices = new Vector3[] {
     new Vector3(0.0f,0.0f,0.0f),
     new Vector3(1.0f,0.0f,0.0f),
     new Vector3(0.0f,1.0f,0.0f),
     new Vector3(1.0f,1.0f,0.0f)
} ;
m.uv = new Vector2[] {
     new Vector2 (0, 0),
     new Vector2 (2, 0),
     new Vector2(0, 1),
     new Vector2(2,1)
} ;
```





Shaders take in textures, vertex, uv (and other per-vertex values).

Shaders output a rendered image.

Default Unity Vert/Frag Shader

```
Shader "Custom/testmesh_shader" {
    Properties {
        _MainTex ("Base (RGB)", 2D) = "white" {}
    }
    SubShader {
        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            uniform sampler2D _MainTex;

            struct vertexInput {
                float4 vertex : POSITION;
                float2 texcoord : TEXCOORD0;
            } ;
            struct vertexOutput {
                float4 pos : POSITION;
                float2 tex : TEXCOORD0;
            } ;

            vertexOutput vert(vertexInput input)
            {
                vertexOutput output;
                output.tex = input.texcoord;
                output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
                return output;
            }
            float4 frag(vertexOutput input) : COLOR
            {
                return tex2D(_MainTex, input.tex.xy);
            }

            ENDCG
        }
    }
}
```

We've been using a shader this whole time, this is it.

(Unity doesn't have the most beginner-friendly shader system)

Default Unity Vert/Frag Shader Explained

```
Shader "Custom/testmesh_shader" {
```

Define shader name as « Custom/testmesh_shader » (will show up in the inspector)

```
    Properties {
        _MainTex ("Base (RGB)", 2D) = "white" {}
    }
```

Define shader inputs as one texture (2D) with a default color of white and a name of « _MainTex ».

```
    SubShader {
```

All shader contents go within a pass within a subshader.

```
        Pass {
            CGPROGRAM
```

Begin vertex-fragment shader program code (CGPROGRAM).

```
            #pragma vertex vert
            #pragma fragment frag
```

Define vertex shader as function of name « vert », fragment shader as function of name « frag ».

```
            #include "UnityCG.cginc"
```

Just include me

```
            uniform sampler2D _MainTex;
```

Access input texture parameter within this pass.

```
            struct vertexInput {
                float4 vertex : POSITION;
                float2 texcoord : TEXCOORD0;
            } ;
```

Define vertex shader input object, POSITION and TEXCOORD0 are special tags used by Unity.

```
            struct vertexOutput {
                float4 pos : POSITION;
                float2 tex : TEXCOORD0;
            } ;
```

Define vertex shader output/fragment shader input object.

```
            vertexOutput vert(vertexInput input)
            {
                vertexOutput output;
                output.tex = input.texcoord;
                output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
                return output;
            }
```

Vertex Shader

```
            float4 frag(vertexOutput input) : COLOR
            {
                return tex2D(_MainTex, input.tex.xy);
            }
```

Fragment Shader

```

            ENDCG
        }
    }
}
```

Vert/Frag Explained

They are two « phases », which can be thought of as being run sequentially.

For every vertex in the geometry…

```
vertexOutput vert(vertexInput input)
{
 ...
}
```

For every pixel (fragment) within triangles formed by verticies…

```
float4 frag(vertexOutput input) : COLOR
{
 ...
}
```

Fragment Shaders

Fragment shaders are run per-pixel, and determine the final output color.

-Fragment shaders output a vector of 4 numbers (floats).

-These numbers represent (red, green, blue, alpha).

-These numbers are normalized (between 0 and 1).

```
float4 frag(vertexOutput input) : COLOR
{
    return float4(1.0,0.0,0.0,1.0);
}
```

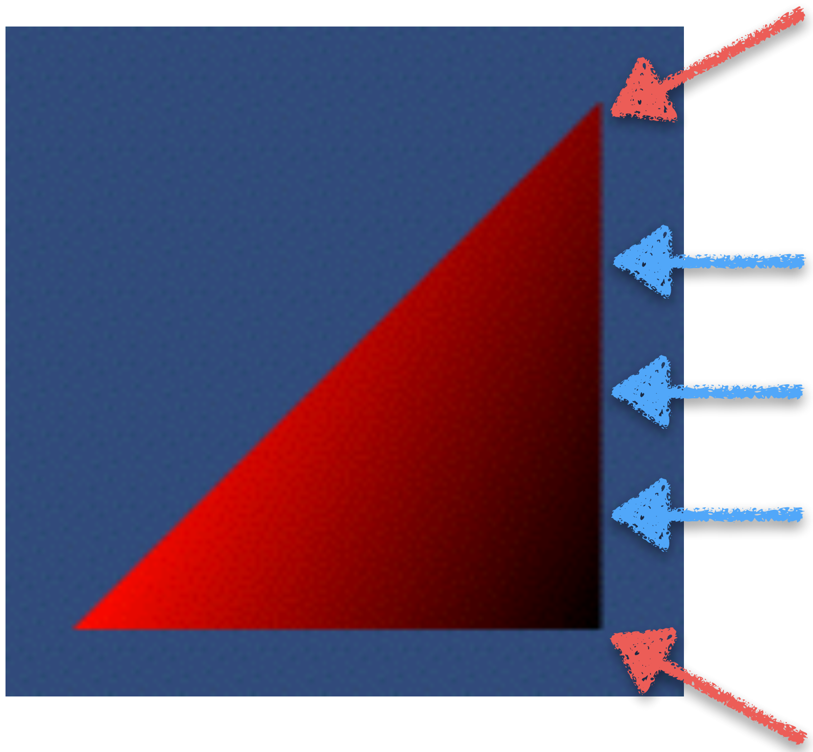
What will the triangle rendered with this shader always look like?

Vertex Shaders

Vertex shaders are run per-vertex.

-Their input and output is determined by the struct definitions.

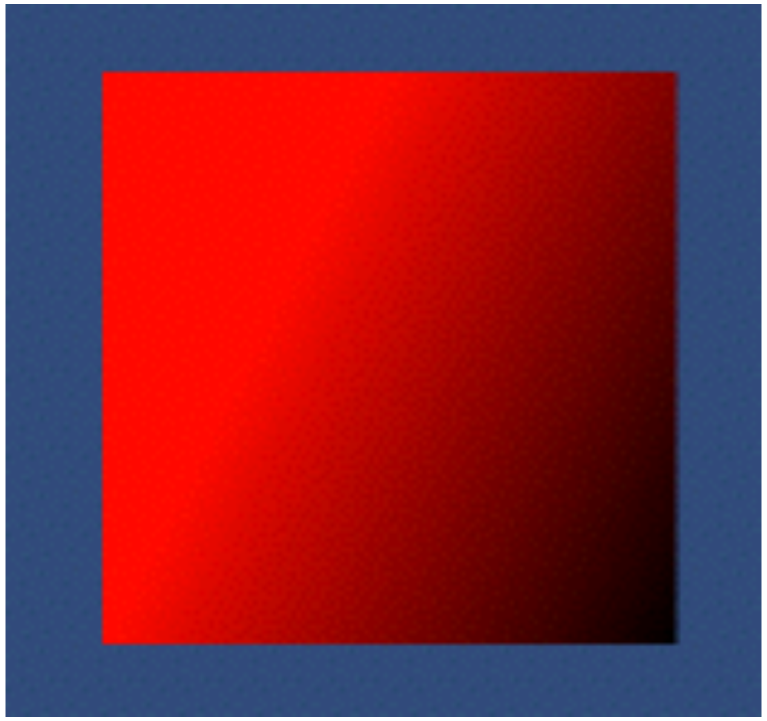
-Output values are interpolated to the fragments that are rendered.



```
vertex value: 1
fragment value: 0.75
fragment value: 0.5
fragment value: 0.25
vertex value: 0
```

Just checking.

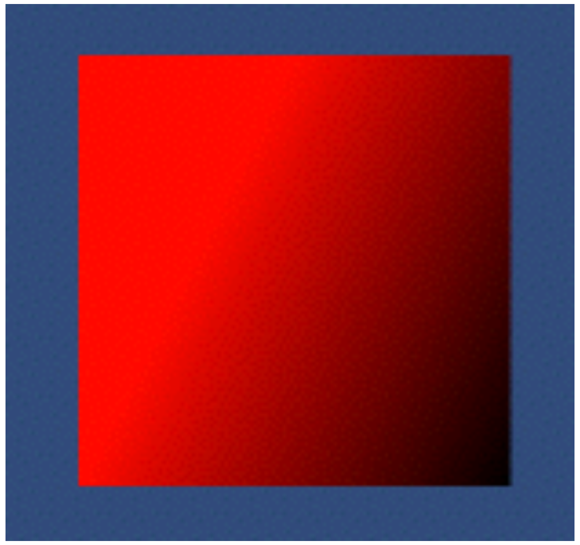
How to get a gradient box like this?



```
struct vertexInput {
    float4 vertex : POSITION;
    float2 texcoord : TEXCOORD0;
    float test_val;
} ;
struct vertexOutput {
    float4 pos : POSITION;
    float test_val;
} ;

vertexOutput vert(vertexInput input)
{
    vertexOutput output;
    output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
    return output;
}
float4 frag(vertexOutput input) : COLOR
{
    return float4(1.0,0.0,0.0,1.0);
}
```

Just checking.



```
struct vertexInput {
    float4 vertex : POSITION;
    float2 texcoord : TEXCOORD0;
} ;
struct vertexOutput {
    float4 pos : POSITION;
    float test_val;
} ;

vertexOutput vert(vertexInput input)
{
    vertexOutput output;
    output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
    output.test_val = (input.texcoord.x + input.texcoord.y) / 2.0;
    return output;
}
float4 frag(vertexOutput input) : COLOR
{
    return float4(input.test_val,0.0,0.0,1.0);
}
```

Define a new vertex shader output field that is interpolated.

Texture Inputs

Textures can be passed in as parameters.

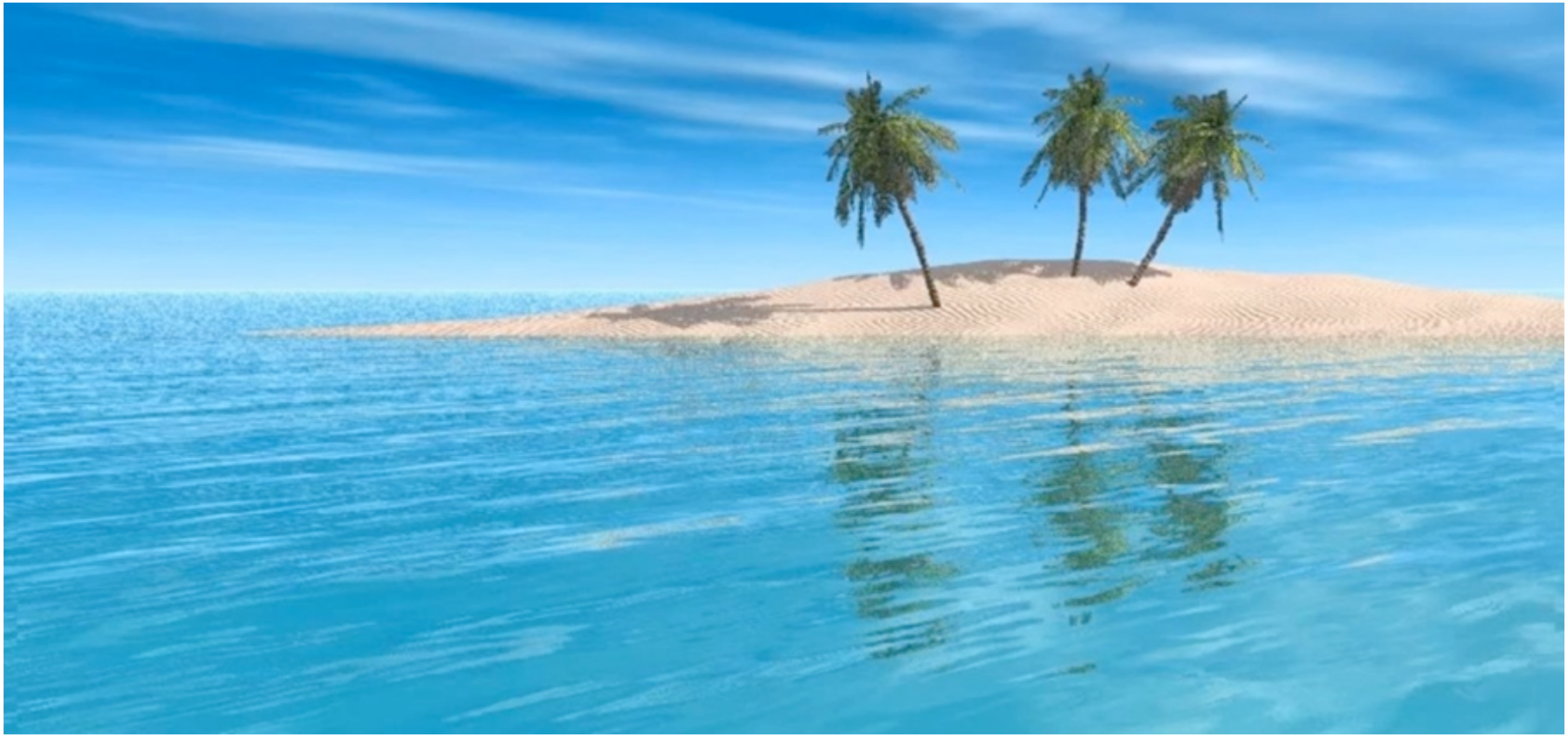
```
Properties {
   _MainTex ("Base (RGB)", 2D) = "white" {}
}


 ...

vertexOutput vert(vertexInput input)
{
   vertexOutput output;
   output.tex = input.texcoord;
   output.pos = mul(UNITY_MATRIX_MVP, input.vertex);
   return output;
}
float4 frag(vertexOutput input) : COLOR
{
   return tex2D(_MainTex, input.tex.xy);
}
```

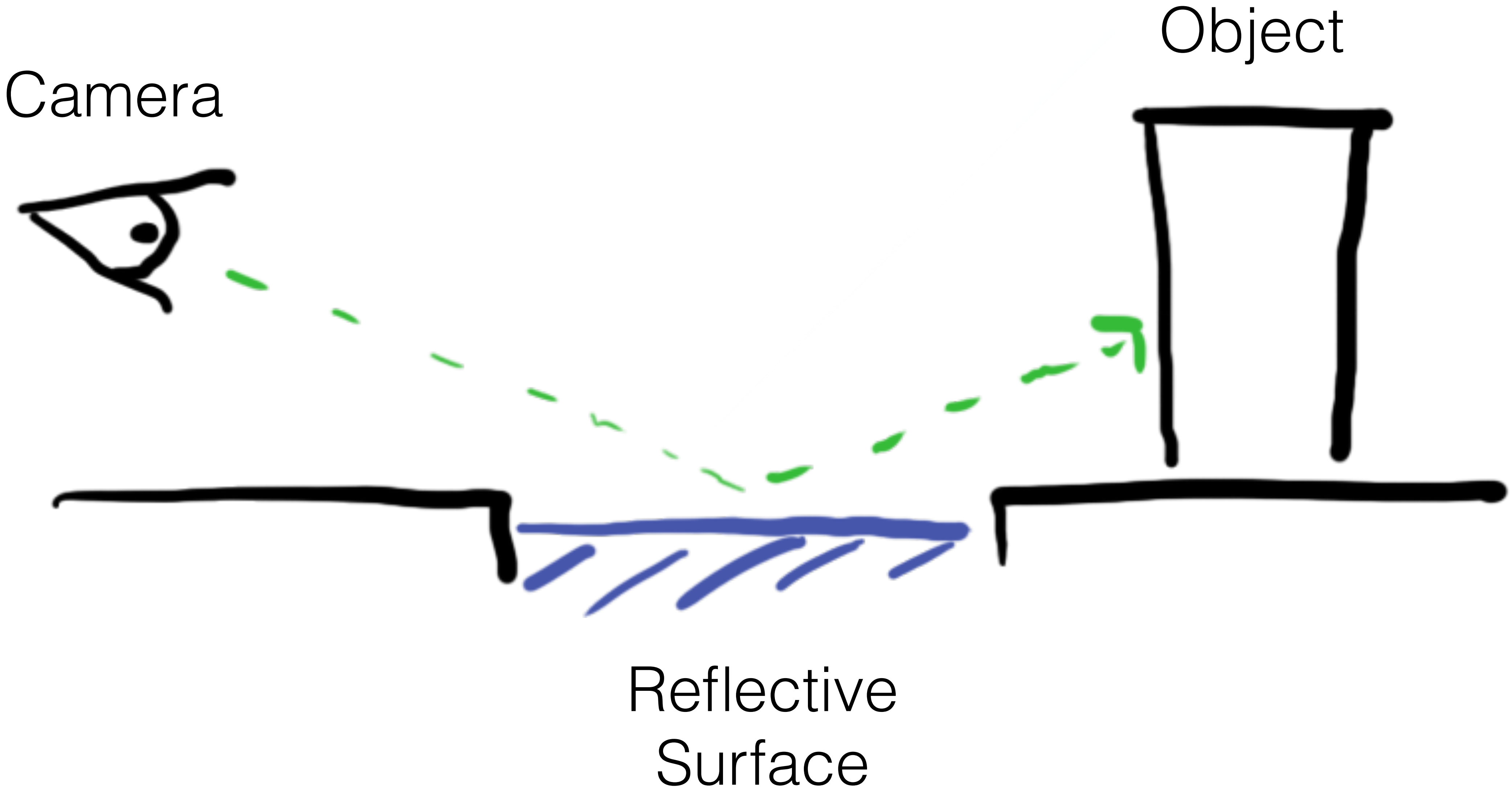
Given (x,y) coordinate in UV space, this line reads out the pixel data for the texture.

How to do a water reflection + ripple effect



How to do a water reflection + ripple effect

The idea



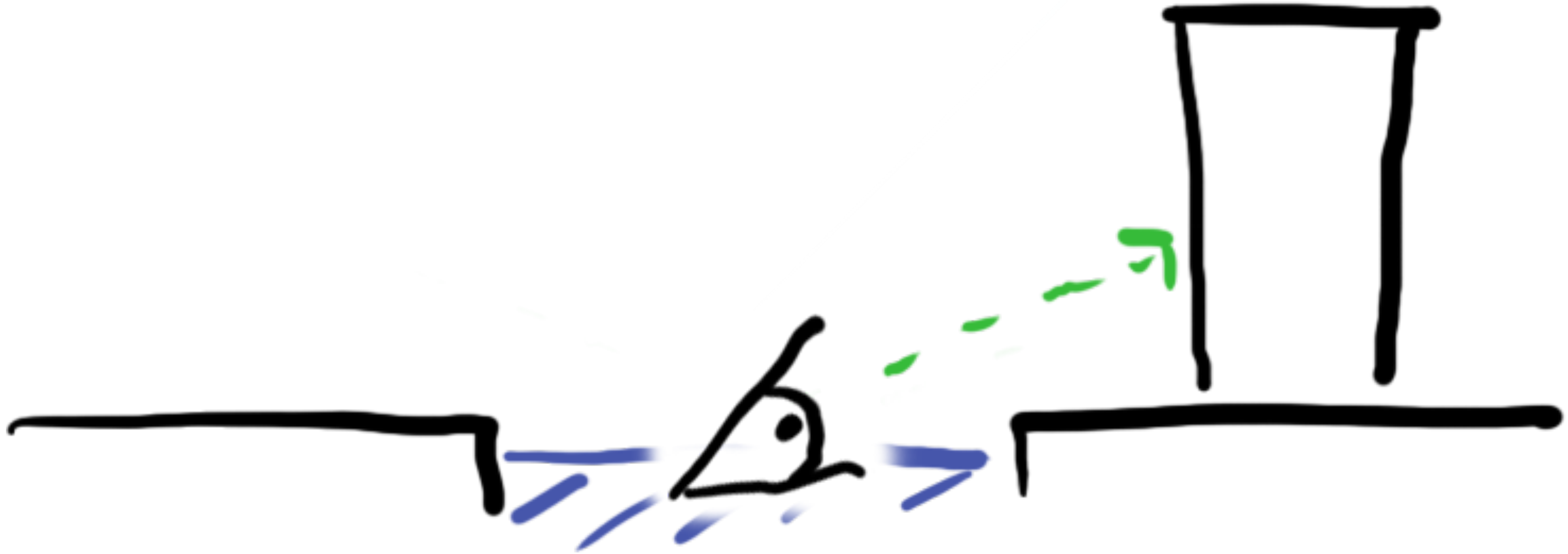
Camera

Object

Reflective Surface

How to do a water reflection + ripple effect

The trick



Put a camera at water level and have it render in the direction of the « reflection ».

Render this camera’s output texture on the reflective surface.

Then, apply some simple shader effects like sinusoidal ripples, color tint and alpha gradient.

Unity Shaders

```
Shader "Example/Diffuse Simple" {
  SubShader {
    Tags { "RenderType" = "Opaque" }
    CGPROGRAM
    #pragma surface surf Lambert
    struct Input {
        float4 color : COLOR;
    };
    void surf (Input IN, inout SurfaceOutput o) {
        o.Albedo = 1;
    }
    ENDCG
  }
  Fallback "Diffuse"
}
```

If you find an example shader online that looks like this, it is a « Surface Shader ».

It is a different, higher level (based off the Phong/Gouraud lighting model).

Similar ideas apply, but the code won’t.

Next Steps…

What you just saw was the basics of all real-time modern computer graphics. With this, you can start tackling all sorts of more advanced topics such as…

-3D Lighting (Phong/Gouraud model)

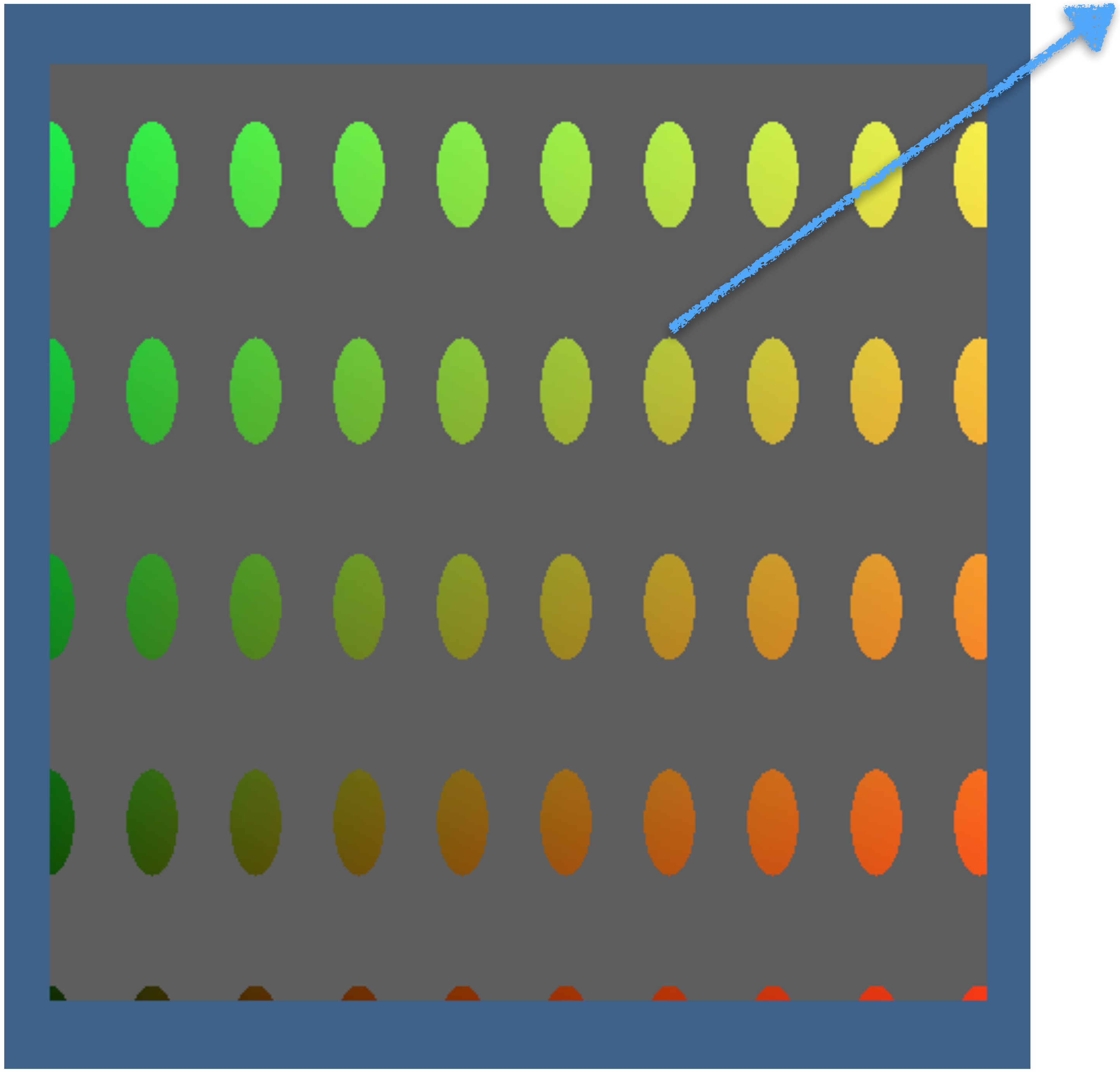
-Shadows (Shadow mapping, shadow volumes).

-Bump mapping (and other textures-as-data techniques)

-Raytracing

Activity

They move in this direction.



Make an animated, repeating textured shader in Unity.

Use any random image you want.

You should animate the color tinting too!

Helpful Unity Shader Links

http://docs.unity3d.com/462/Documentation/Manual/SL-BuiltinValues.html

http://wiki.unity3d.com/index.php/Shader_Code

http://docs.unity3d.com/Manual/SL-Reference.html

http://docs.unity3d.com/Manual/SL-VertexFragmentShaderExamples.html

http://docs.unity3d.com/Manual/SL-SurfaceShaderExamples.html

Unity example project:

https://github.com/spotco/gamedevclub_shadertest