Unity Shader

By 簡單黎講 C Plus Plus
Youtube:
https://www.youtube.com/channel/UCWW9g_90Ik1U8ULNG5PvoYQ

Rendering API / Shading Language

- DirectX – Windows / XBox1
  - High-Level Shader Language (HLSL)
  - Shader Assembly Language
- OpenGL/ES – Windows / MacOSX / Linux / iOS / Android
  - GL Shader Language (GLSL)
  - ARB Assembly language
- Apple Metal – MacOSX / iOS
  - Metal Shading Language
- PlayStation
  - PS4 PSSL (PlayStation Shader Language)
- Vulkan (next generation OpenGL)
  - SPIR-V

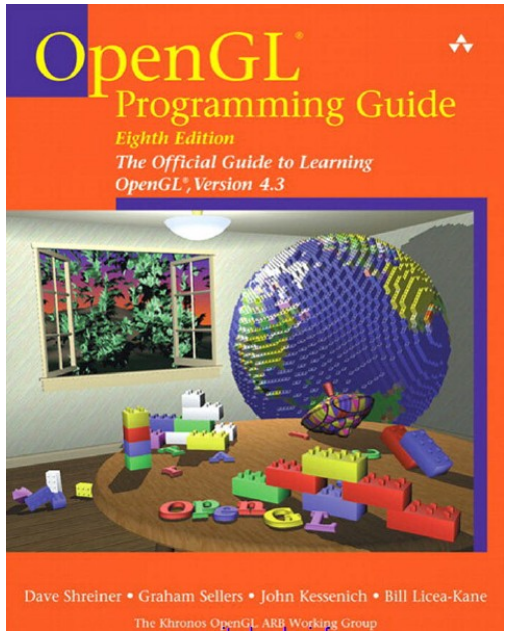
- Offline Renderers
  - Pixar RenderMan
    - RenderMan Shading Language (RSL)
  - Solid Arnold
    - Open Shading Language

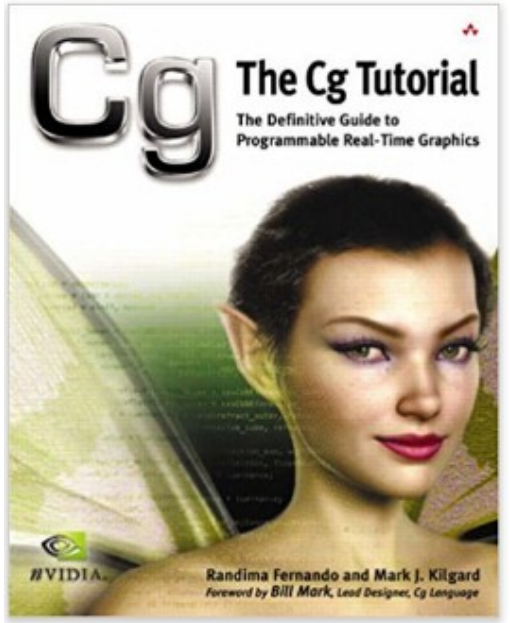
Rendering API / Shading Language

- nVidia Cg programming Language
- Output profiles
  - DirectX HLSL
  - OpenGL GLSL

- OpenGL Programming Gudie
- The CG Tutorial: The Definitive Guide to Programmable Real-Time Graphics

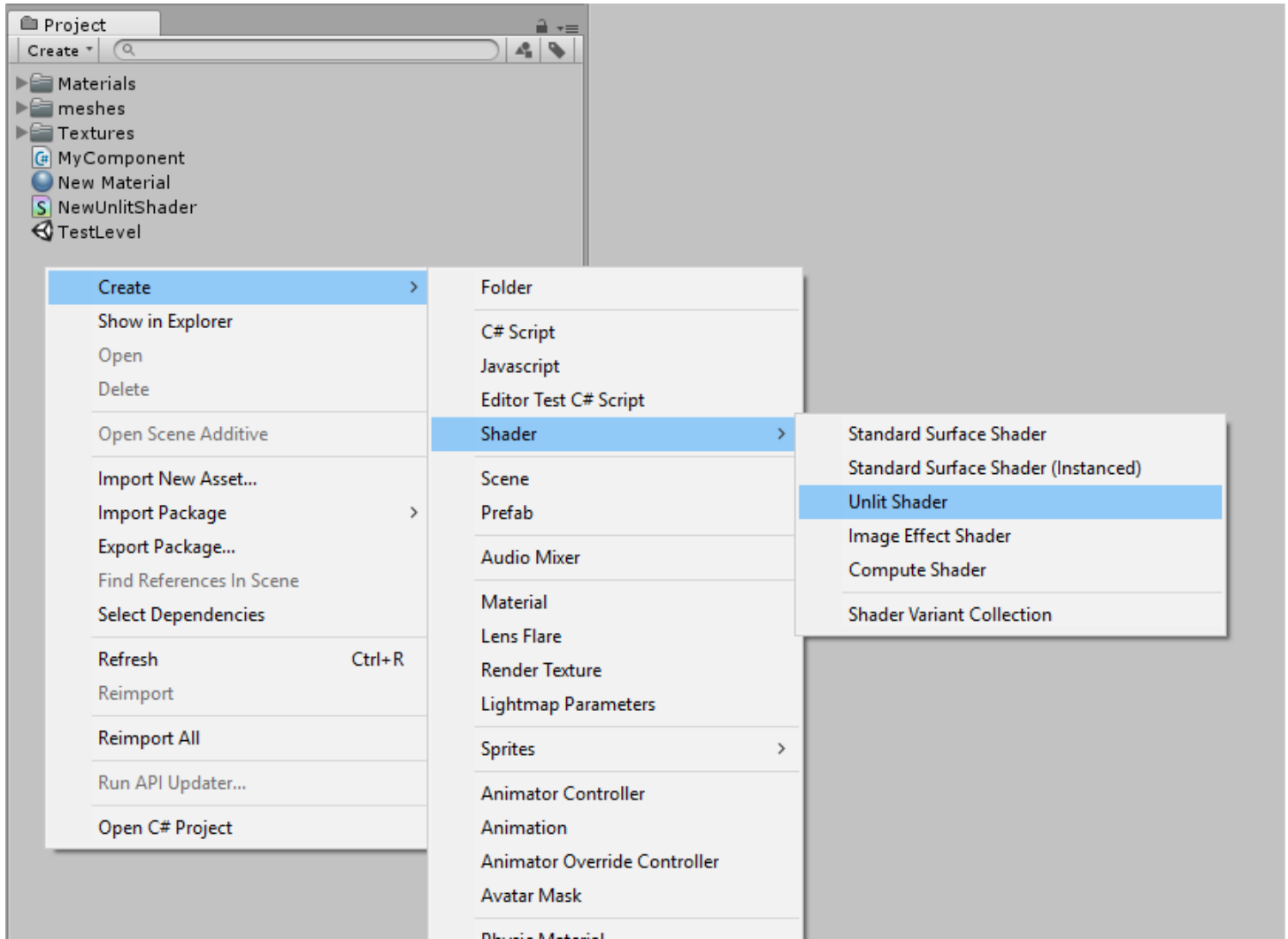
- \\upr-jchan1\Share\Shading Language Introduction





Shader in Unity

- Unity Shader Lab
- supports
  - Cg Program
  - HLSL
  - GLSL
- Demo
  - Create Shader
  - Create Material
  - Assign to object



Unity ShaderLab Syntax

- https://docs.unity3d.com/530/Documentation/Manual/SL-Shader.html
- SubShader, Tags, Pass

- CGPROGRAM / ENDCG
- Vertex Shader function
  - #pragma vertex vert
- Pixel Shader function
  - #pragma fragment frag

- Demo - output red from Pixel Shader
  - return float4(1,0,0,1);

```
Shader "Unlit/NewUnlitShader 1"
{
    SubShader
    {
        Tags { "RenderType"="Opaque" }
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"

            struct appdata
            {
                float4 vertex : POSITION;
            };

            struct v2f
            {
                float4 vertex : SV_POSITION;
            };

            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
                return o;
            }

            fixed4 frag (v2f i) : SV_Target
            {
                return float4(1,0,0,1);
            }
            ENDCG
        }
    }
}
```

OpenGL Pipeline

- Vertex Shader
- Fragment Shader

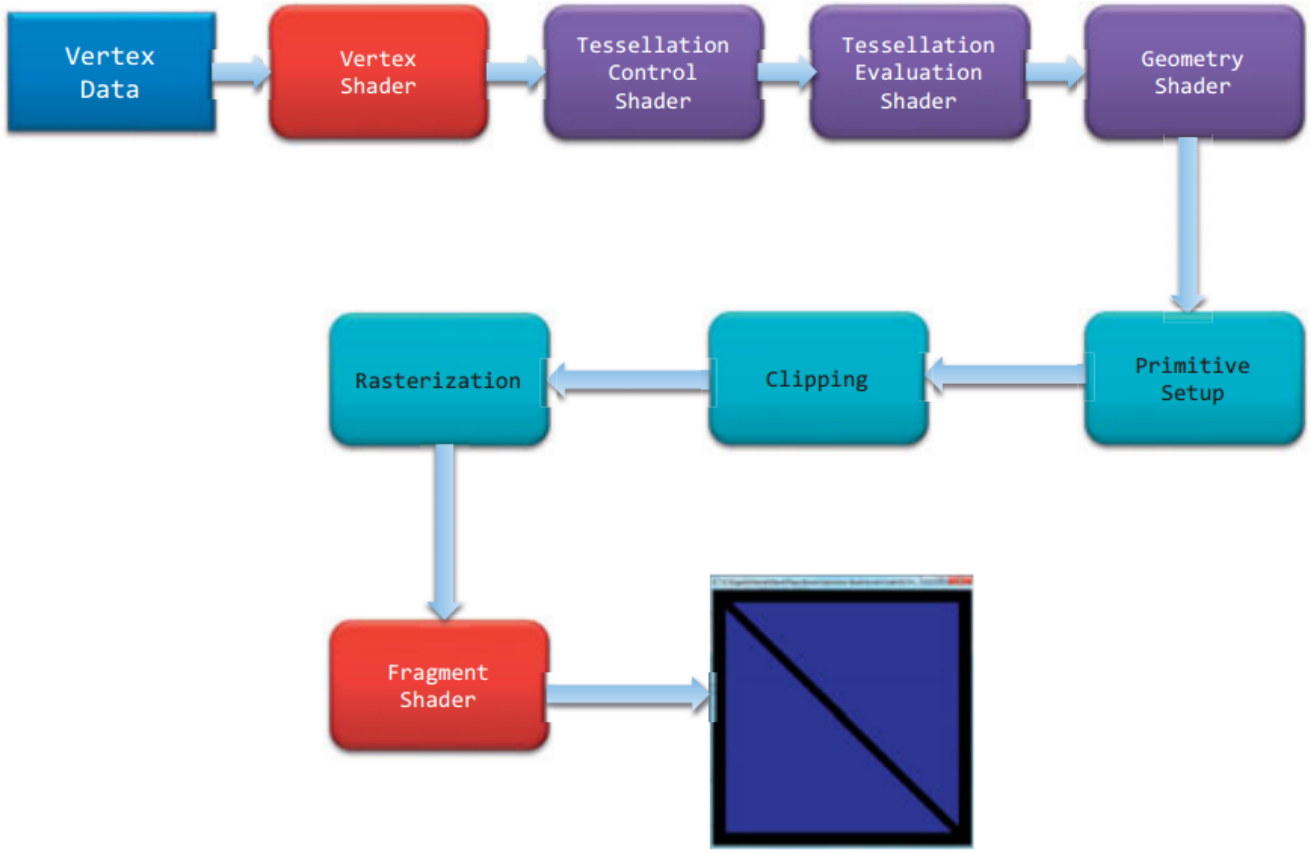
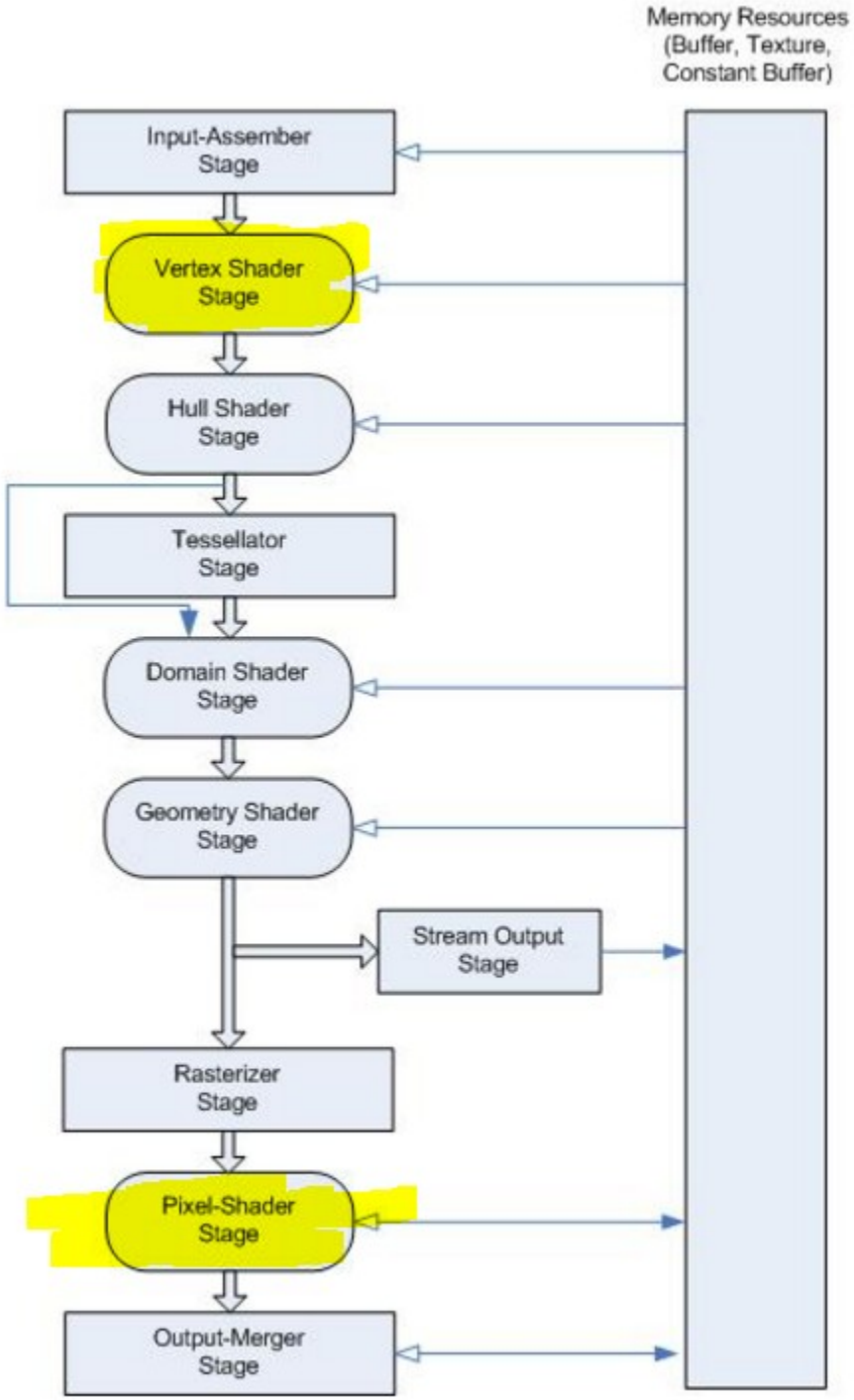


Figure 1.2 The OpenGL pipeline

DirectX Pipeline

- Vertex Shader Stage
- Pixel-Shader Stage



Adding variable

- In Cg program
  - float my_var;

Pass value from Application

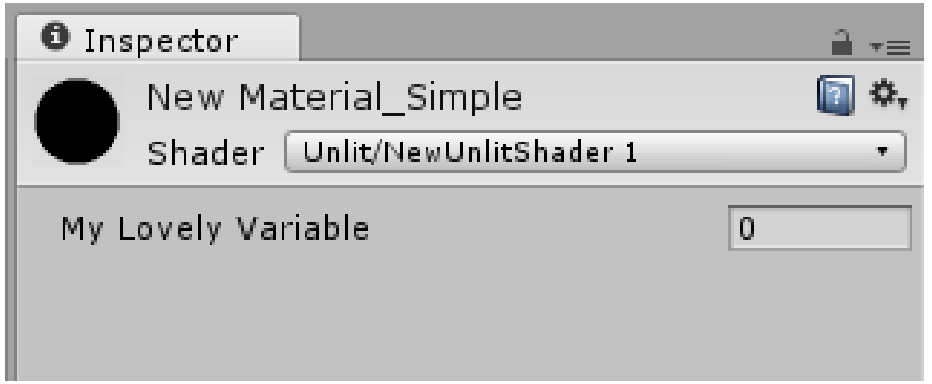
- OpenGL C/C++
  - int loc = glGetUniformLocation(program, “myVar”);
  - glUniform1f(loc, 0.5);

- **Unity C#**
  - **material.SetFloat(”myVar”, 0.5);**

Unity ShaderLab Property

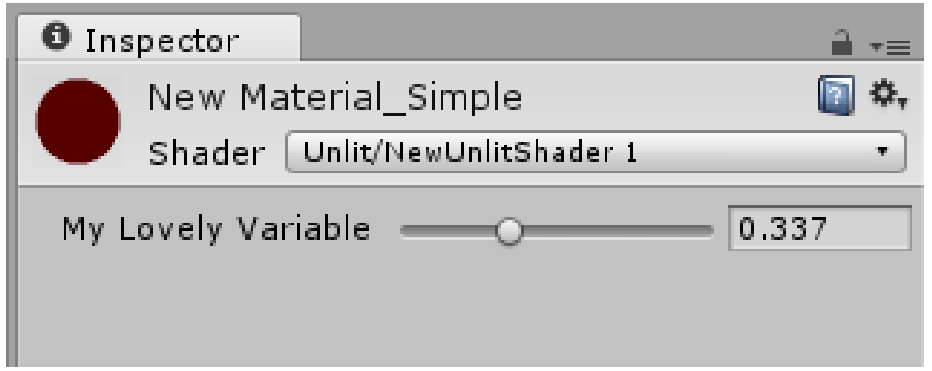
- float property

```
Properties
{
    my_var("My Lovely Variable", float) = 0
}
```



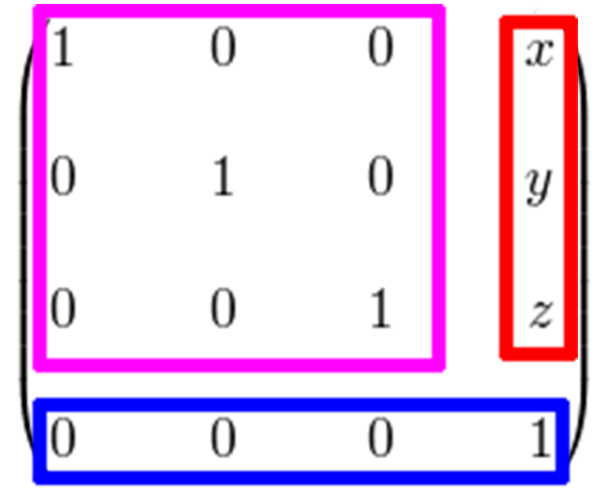
- Slider in Inspector

```
Properties
{
    my_var("My Lovely Variable", range(0,1)) = 0
}
```



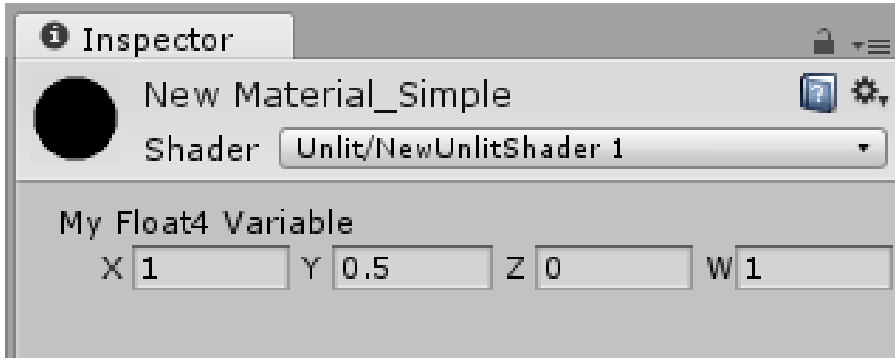
Data Type

- Scalar Types
  - **float – floating point (32bits)**
  - int – integer
  - bool – true/false
  - half – floating point (16bits)
  - fixed – fixed point (11bits)
- Vector Types (also for color)
  - float4 – xyzw, **rgba**
  - float3 – xyz, **rgb**
  - float2 – xy, **rg**
  - Int2/3/4, half2/3/4 ….etc
- Matrix Types
  - **float4x4 – translate, rotate, scale, projection**
  - float4x3 – without projection
  - float3x3 - without projection, translate
  - …

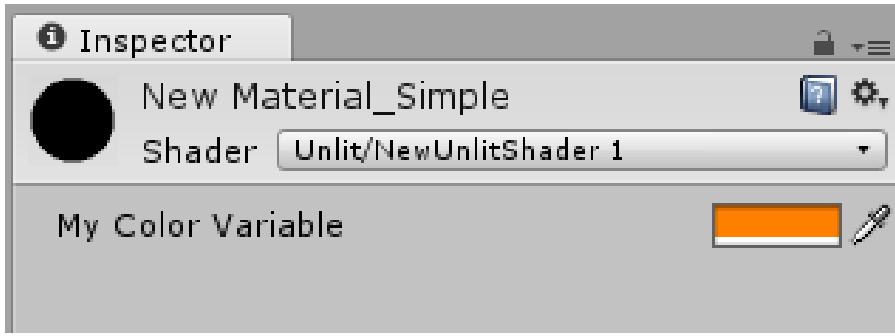


Float4 Property (Vector / Color)

- Float4 property
  - my_var("My vector Variable", vector) = (1,0.5,1,1)



- Color property
  - my_var("My color Variable", color) = (1,0.5,1,1)

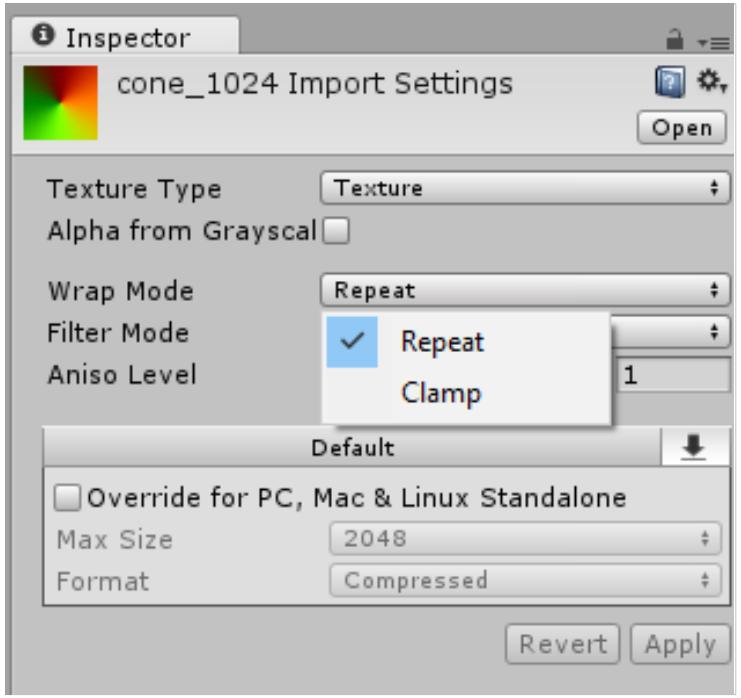


Pixel / Fragment Shader
& Texture Sampling

Texture Sampling

- Data Type
  - sampler2D mainTex;
- ShaderLab Property
  - mainTex(“Main Texture”, 2D) = “white” {}
- Sampling color from texture
  - float4 color = tex2D(mainTex, i.uv);
- Texture Coordinate (UV)
  - UVW for 3D just like XYZ
  - S,T (OpenGL, RenderMan)
- Demo - show uv in color
  - Color is the only way to debug (0..1)
- Using uvChecker texture to debug

UV Transform



- Offset / Translate
  - float2 uv = i.uv + float2(0.5, 0);
- Scale
  - float2 uv = i.uv * 2;
- Wrap mode
  - Repeat / Clamp / (Mirror)
  - Unity – Texture property
  - OpenGL
    - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
- Rotate
  - float2x2 R = float2x2(cos(th), -sin(th)
    sin(th), cos(th));
    float2 uv = mul(R, i.uv.xy);

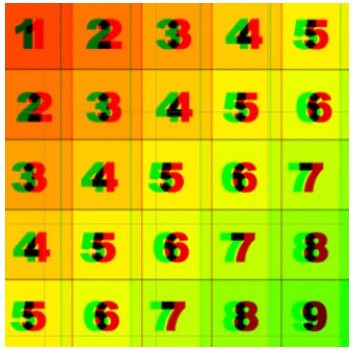
$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

- Translate, Rotate, Scale (TRS)

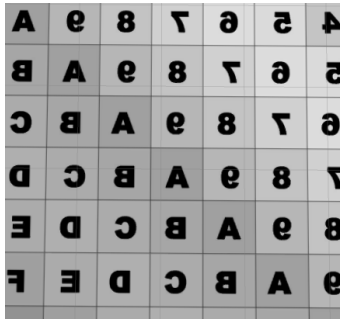
Example Texture Effects

- UV offset per channel
  - Bad printing effect / Color Scattering

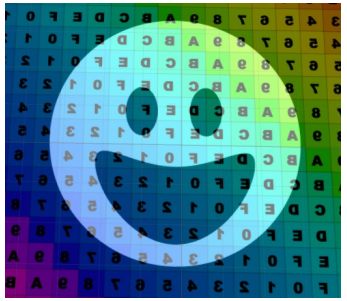
```
float4 c;
float2 offset = float2(0.01, 0);
c.r = tex2D(mainTex, i.uv + offset).r;
c.g = tex2D(mainTex, i.uv).g;
c.b = tex2D(mainTex, i.uv - offset).b;
c.a = 1
```



- Gray Scale
  - (R+G+B) / 3 (not the best!)
  - C = R * 0.299 + G * 0.587 + B * 0.114
  - Vector dot product
    - C = dot(color, float3(0.299, 0.587, 0.114));



- Blending 2 textures
  - C = A*w + B*(1-w)
- Linear interpolation (Lerp)
  - C = lerp(A,B,w);



Texture Mask

- Using 3rd texture to control w (weight)
  - float w = tex2D(maskTex, i.uv).r;
  - Now it's mask for texture A and B

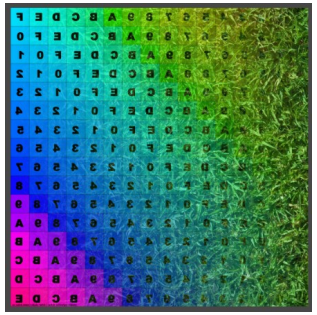
- Mask Transition

  w = tex2D(maskTex, i.uv).r;

  w += **transition** * 2 – 1;

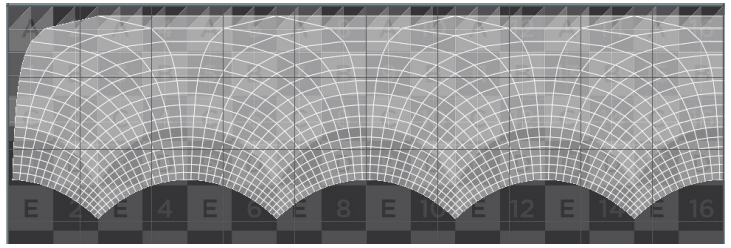
  w = clamp(w, 0, 1);

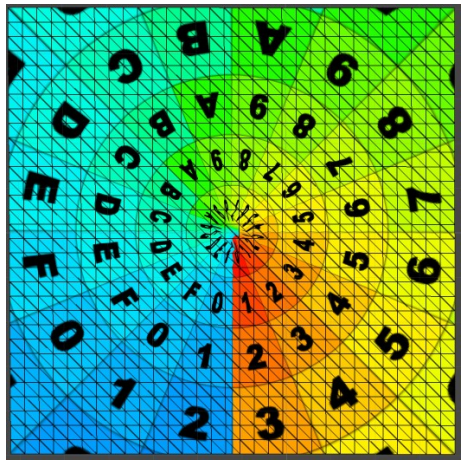


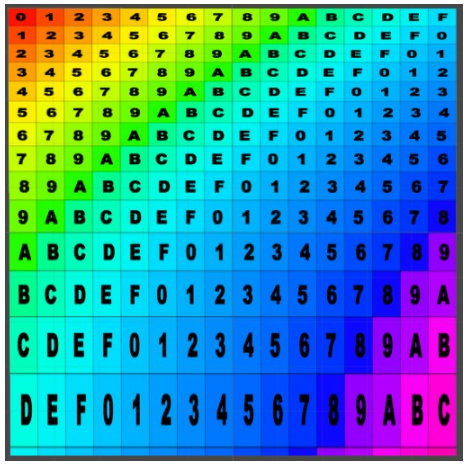



UV Layout / Mesh Topology

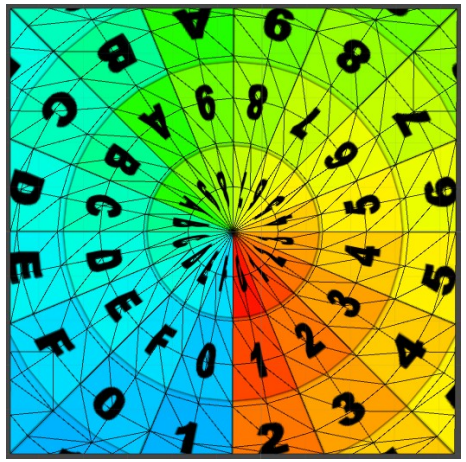
- UV Layout







- Mesh Topology

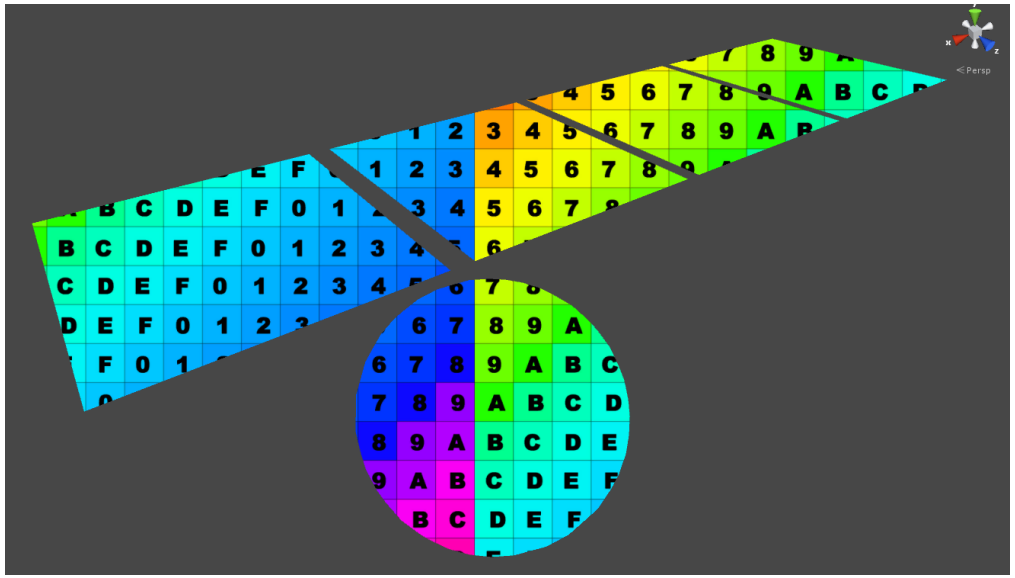


Screen Space Coordinate

- Screen Space to UV
  - i.vertex is screen space in pixels
  - Screen Buffer Size
    - Does not provide by render API
    - Pass in by user application
  - Unity Built-in shader variables
  - _ScreenParams

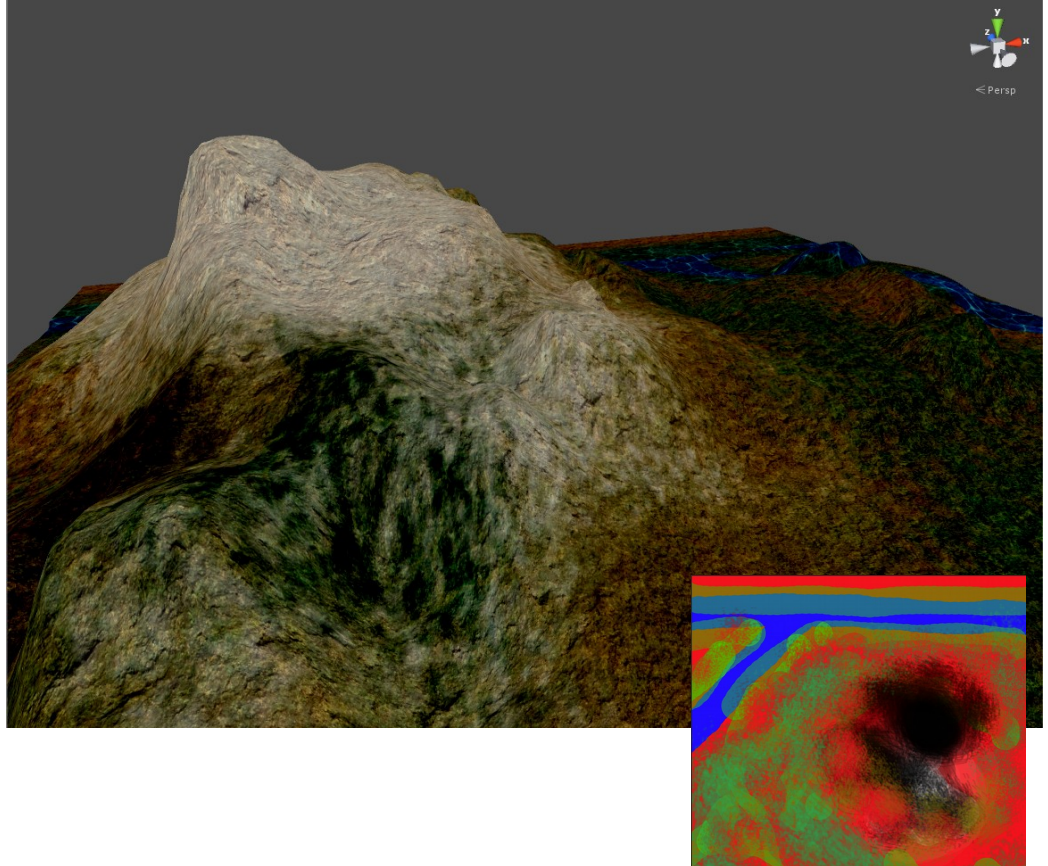
```
float2 screenpos = i.vertex.xy / _ScreenParams.y;
return tex2D(mainTex, screenpos.xy);
```

https://docs.unity3d.com/Manual/SL-UnityShaderVariables.html



Detail Texture

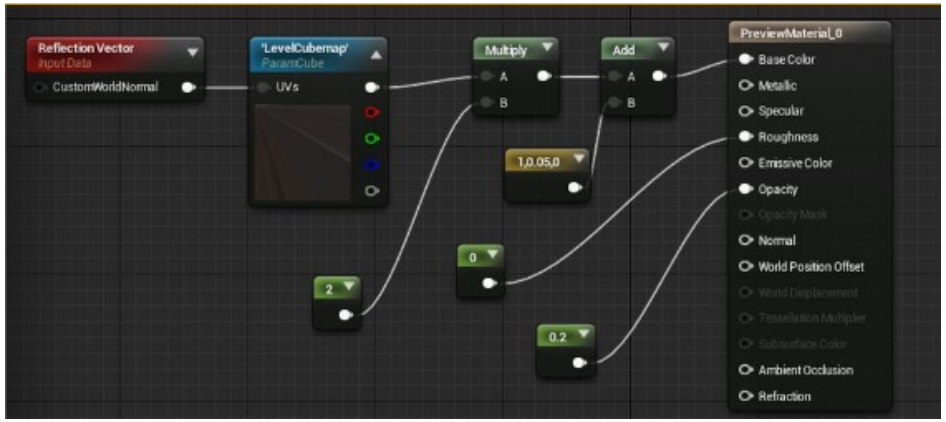
- Terrain use case
  - Cover huge landscape by small texture
  - Repeat texture (Too repeat and boxy)
  - Multi-layer of textures + Low Resolution Mask
  - varying translate, rotate, scale in each layer
- Texture has 4 Channels for mask
  - color = R * DetailTex0 + G * DetailTex1 + B * DetailTex2 + A * DetailTex3;
- 5th Channel
  - R+G+B+A = 1
  - Parity = 1 – (R+G+B+A)
  - Mask Texture + 5 Detail Textures
- Height map blend bias
  - Height in Detail Texture Alpha Channel

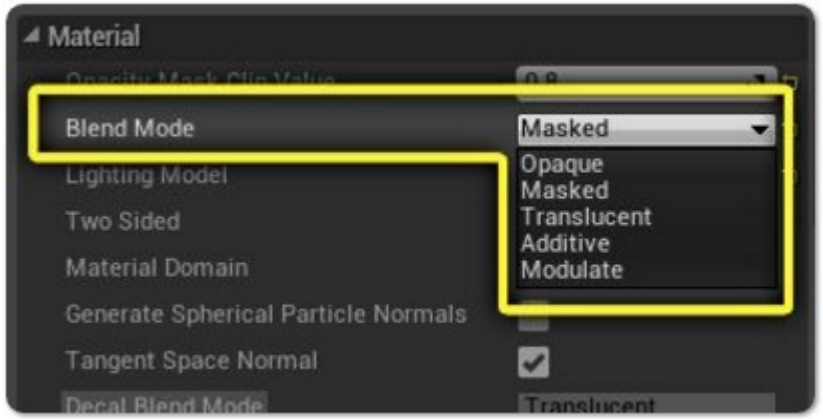


Render States

Render State, Shader Parameters

	Shading Program	Render State	Shader Parameter
Unity	.shader CgProgram section	.shader file ShaderLab Pass options - Blend Off	Material
OpenGL	GLSL	glDisable(GL_BLEND);	glUniform1f()
DirectX	HLSL	d3d->SetRenderState(D3DRS_ALPHABLENDENABLE, FALSE);	Constant Buffer
Unreal	Material Material Network	Material Material Property – Blend Mode	Material Instance

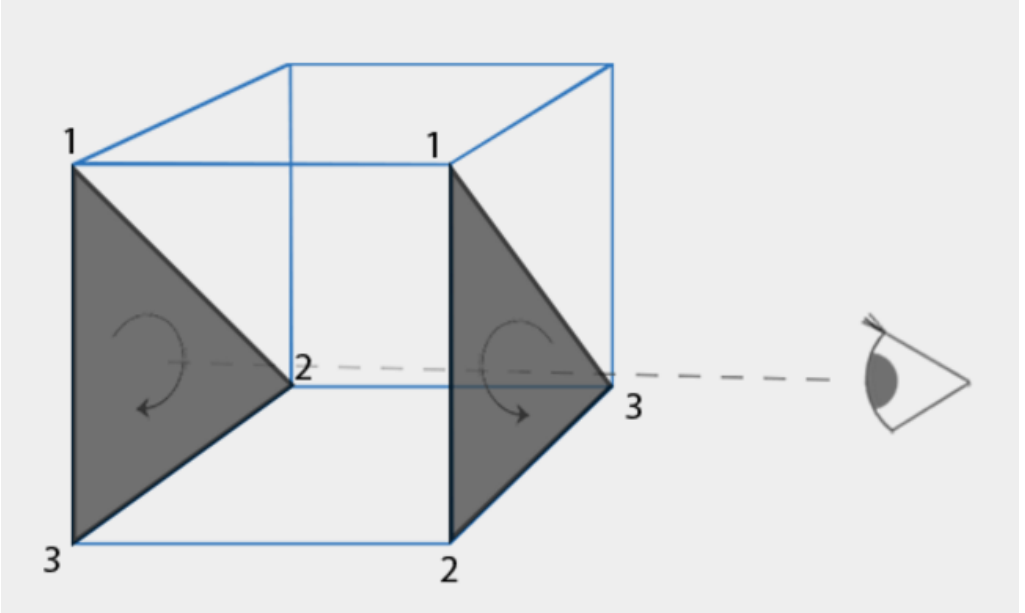


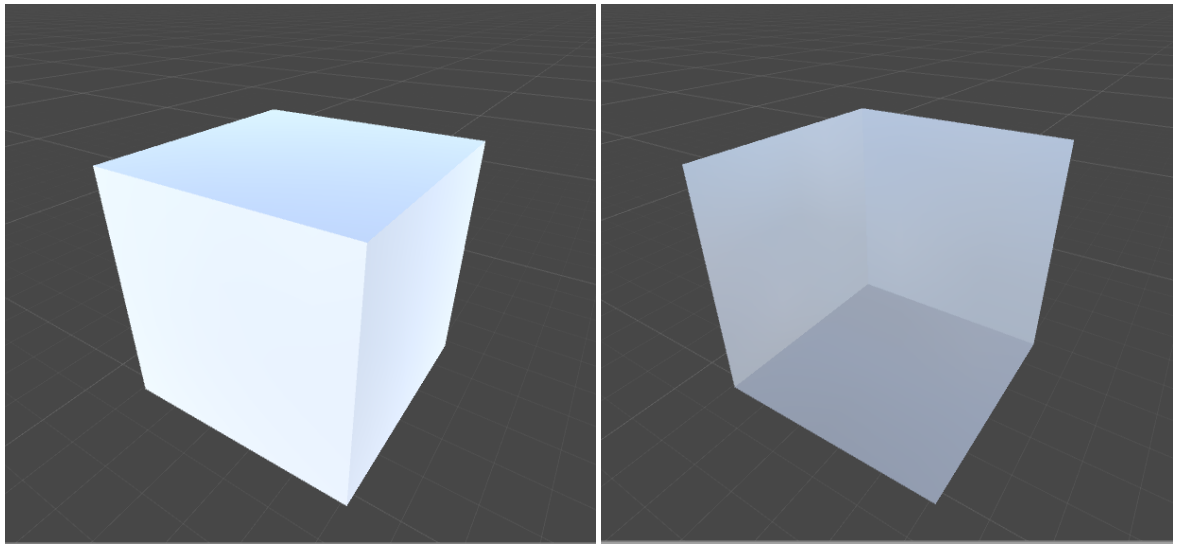


Face Culling

- Face direction
  - Winding order in clockwise

- Cull (Back | Front | Off)
  - Default: Cull Back
  - Disable: Cull Off
    - Double sided (Maya / 3DS Max)
    - Demo in plane
  - Back Face: Cull Front
    - Demo in cube





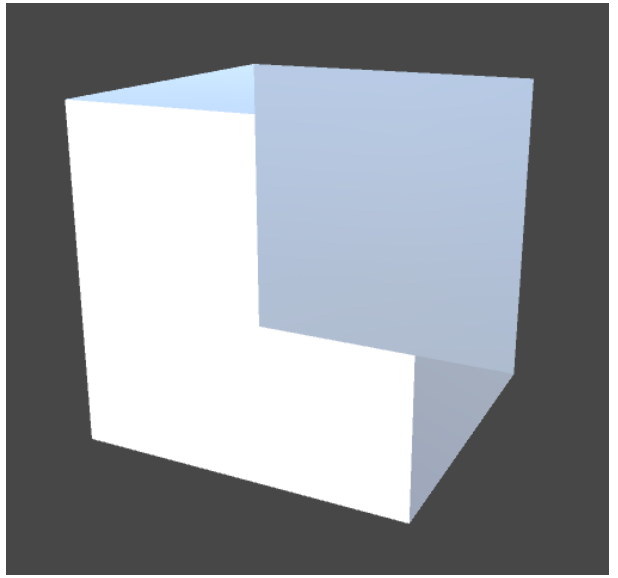
Depth Test / Buffer

- Depth Test op
  - ZTest (Less | Greater | LEqual | GEqual | Equal | NotEqual | Always)
  - Default: ZTest Less
  - Disable: ZTest Always

- Depth Buffer update
  - Default: ZWrite On
  - Don’t update: ZWrite Off

- Depth Offset
  - Offset OffsetFactor, OffsetUnits

- Further: OIT (Order Independent Transparency)



Blending to Screen Buffer

- Output color with alpha
  - Blending is disable by default: Blend Off

- Enable blending
  - Blend SrcAlpha OneMinusSrcAlpha - remember A*w + B*(1-w)
- Render Type
  - Default: Tags { "RenderType"="Opaque" }
  - Transparent: Tags {"RenderType"="Transparent" }

- Transparency objects rendering order
  - All Opaque objects, Transparency objects without depth test
  - Queue: Background=1000, Geometry=2000, Transparent=3000, Overlay=4000

  Tags
  {
  "RenderType"="Transparent"
  "Queue"="Transparent"
  }
  - Or "Queue"=”Geometry+10”

Blend operations, Factors

- No way to retrieve screen buffer color into shader
  - Exception: iOS GPU by PowerVR
  - Otherwise can be done in shader expression directly

- Blend function (Is fixed, only items in red are changeable)
  - RGB: DstColor * DstFactor [op] SrcColor * SrcFactor
  - Alpha: DstAlpha * DstFactorA [op] SrcAlpha * SrcFactorA
  - Operations (Op) - Add, Sub, Min, Max …

  - Unity ShaderLab - Blend SrcFactor DstFactor, SrcFactorA DstFactorA
  - OpenGL – glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

- Photoshop Blend Mode:
  - Multiply: A * B
    - Blend Zero DstColor
  - Screen:
    - 1 - (1 - DstColor) * (1 - SrcColor)
    - 1 - 1 + D + S - D * S
    - D + S - D * S
    - D + S * (1-D)
    - D * 1 + S * (1-D)
    - Blend One OneMinusDstColor //!< using Color instead of Alpha !

More Pixel Shader Examples

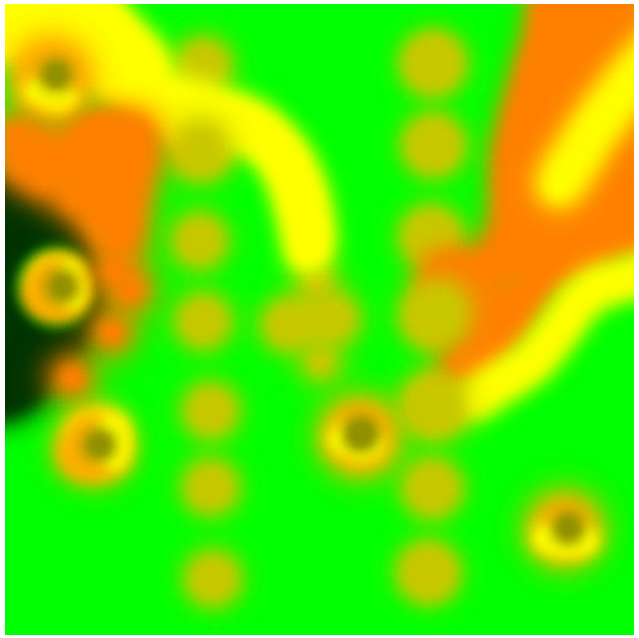
Flow Map

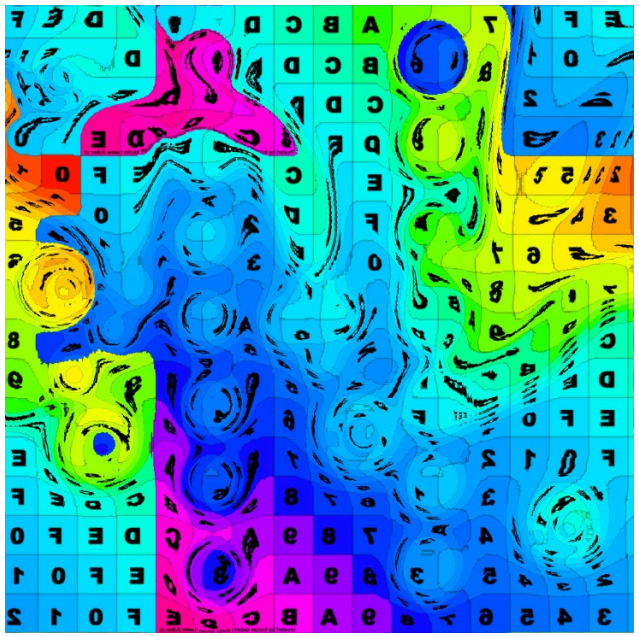
- Flow texture as UV offset

- Color (0..1) -> vector(-1..1)

  v = c * 2 – 1

- Animation by offset either flow map or color texture



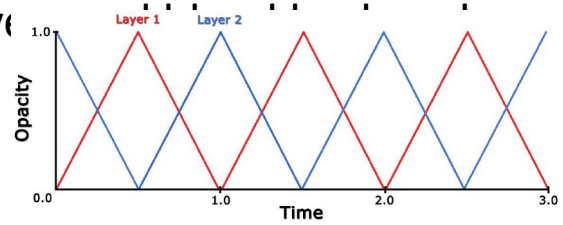


Flow Map – All direction

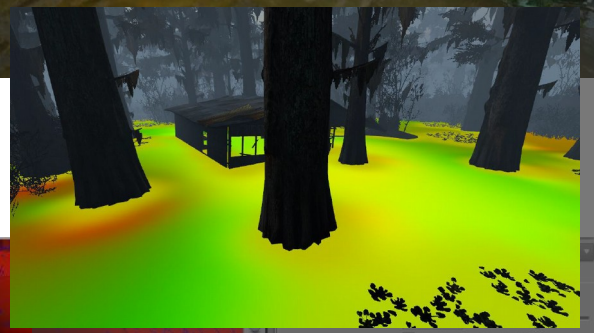
- Offset UV only provide animation in one direction
- SIGGRAPH 2010 – Water Flow in Portal 2

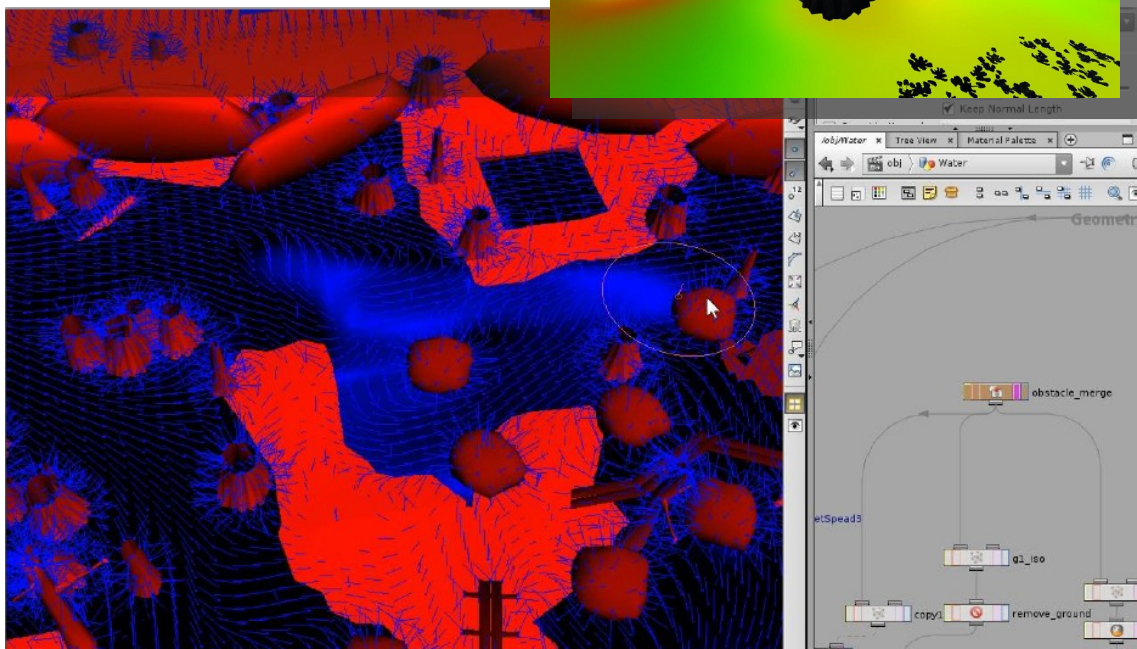
  http://www.valvesoftware.com/publications/2010/siggraph2010_vlachos_waterflow.pdf

  - Offset along flow map direction
  - Looks bad when going to far, because of distortion
  - Solution: using double laye… forward



- Other Application
  - https://plotagraphpro.com/home

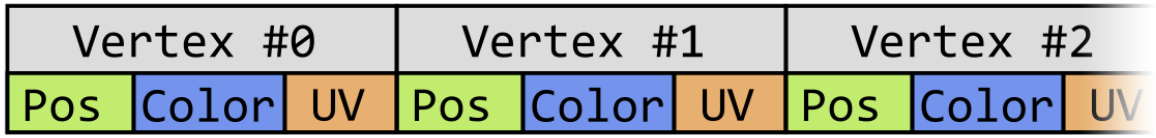




Vertex Shader
& Camera Transformation

Vertex Shader Stage

- Run Vertex by vertex
- No information for other vertices
- Vertex Input Layout (appdata)
  - Position / Color / UV ...
  - Semantics
    - POSITION[N]
    - COLOR[N]
    - TEXCOORD[N]
    - TEXCOORD can be float4 or other type
- Output (v2f)
  - Position in screen space
  - Other varying variable (color, uv …etc)
  - Semantics
    - SV_POSITION (not POSITION)
    - COLOR / TEXCOORD are the same



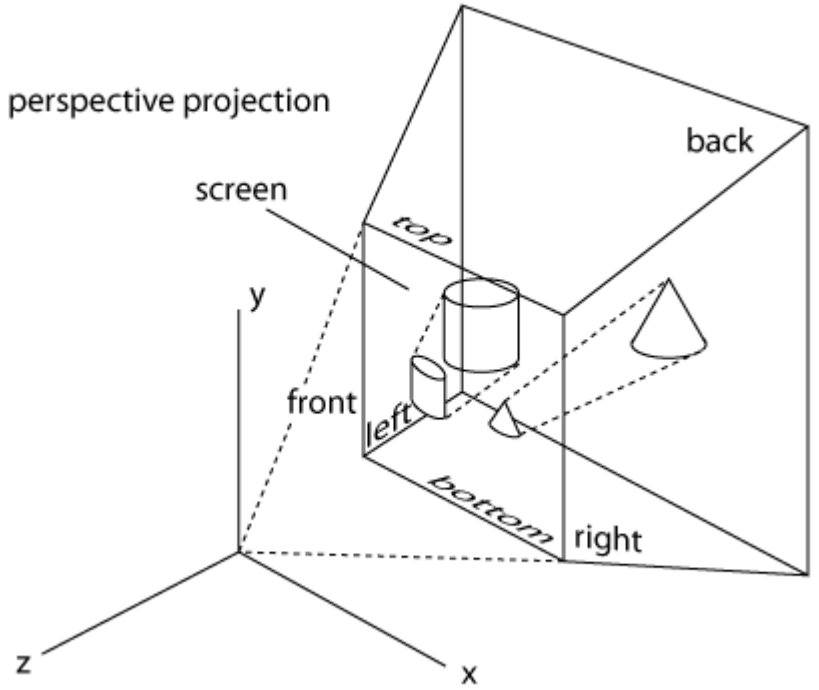

```
struct appdata
{
    float4 pos : POSITION;
    float4 color: COLOR;
    float2 uv : TEXCOORD0;
};

struct v2f
{
    float4 pos : SV_POSITION;
    float4 color : COLOR;
    float2 uv : TEXCOORD0;
};

v2f vert (appdata v)
{
    v2f o;
    o.pos = mul(UNITY_MATRIX_MVP, v.pos);
    o.color
    o.uv = v.uv;
    return o;
}
```

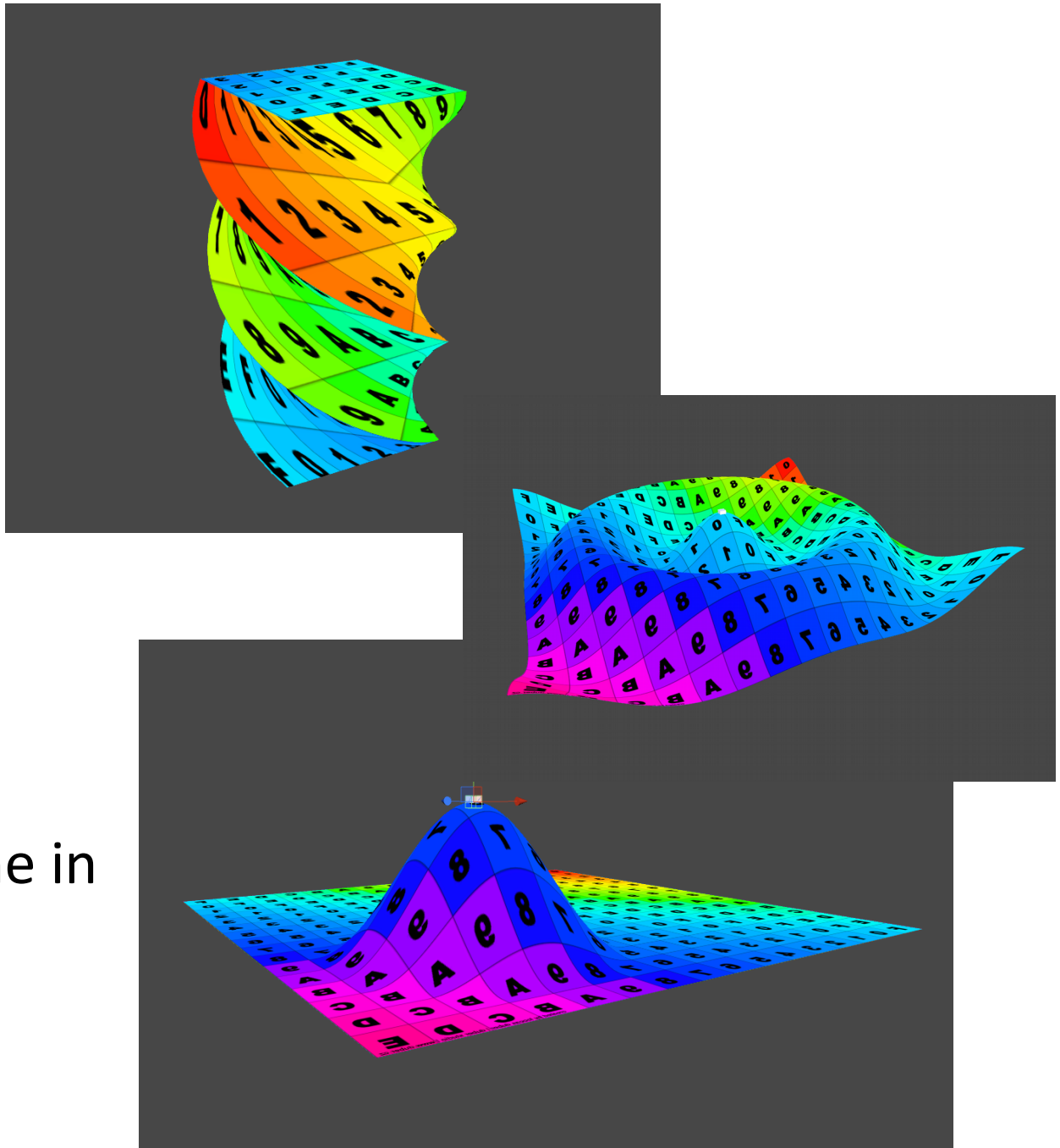
Viewing Transformation

- Projection Matrix (UNITY_MATRIX_P)
  - Perspective
  - Orthogonal
- Viewing (UNITY_MATRIX_V)
  - Camera Angle
  - No camera in GPU, move the whole world instead
- Model Matrix (_Object2World)
  - Object to World Transformation
  - Combine draw call requires world space vertices
- MVP (UNITY_MATRIX_MVP)
- https://docs.unity3d.com/Manual/SL-UnityShaderVariables.html



Deformation

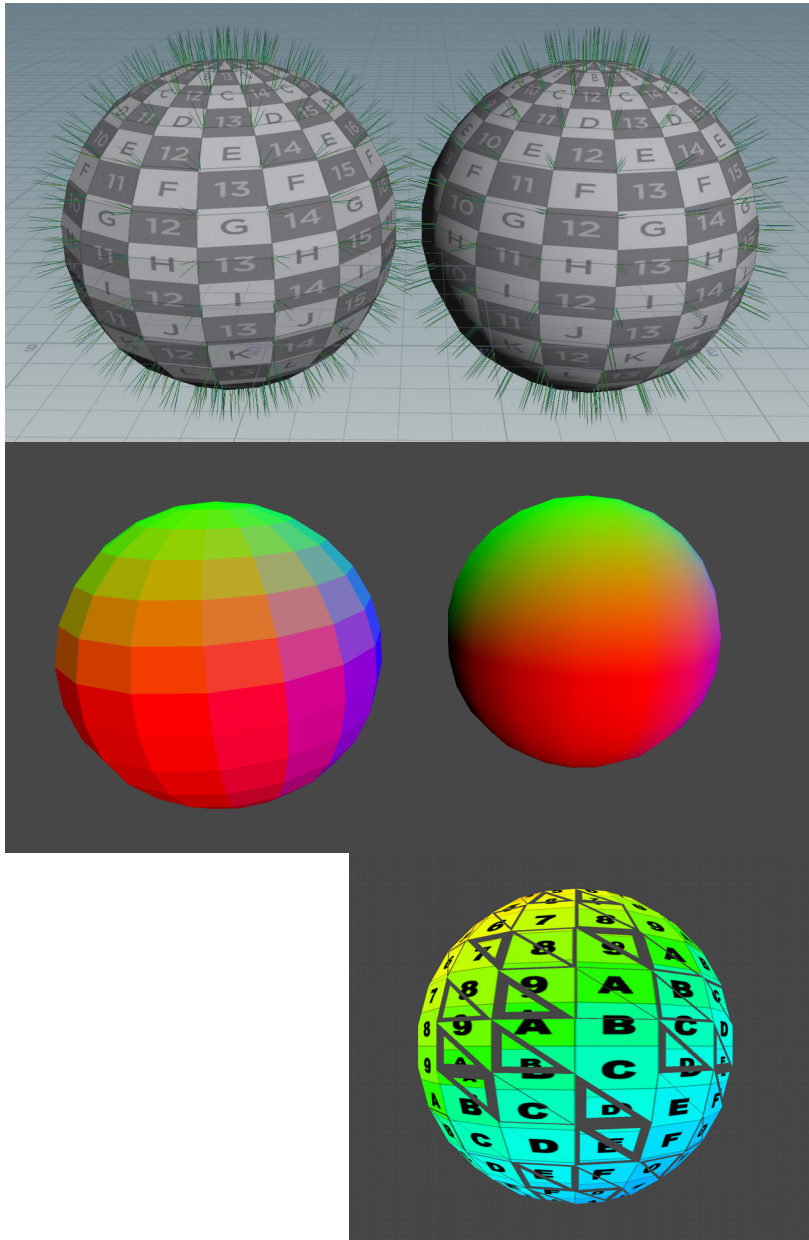
- Twist
  - world / local space
- Sin Wave
  - World space locator
    - Unity Component
      - Assign world position to material
      - [ExecuteInEditMode]
- Texture Transformation can be done in vertex shader instead



Vertex Normal

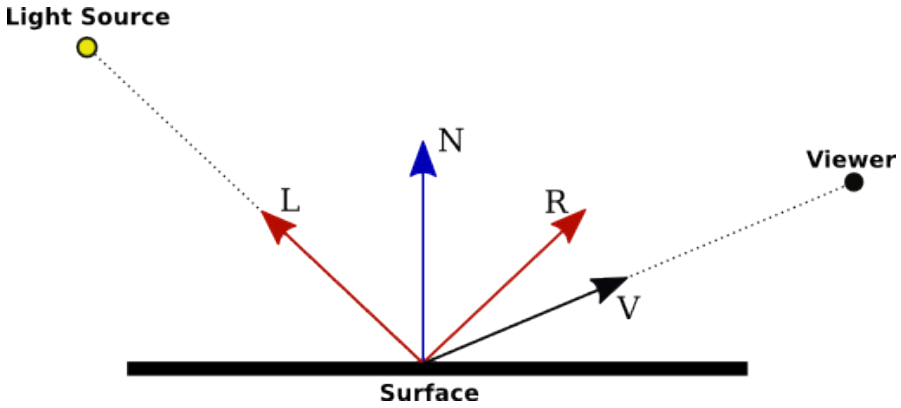
- Vertex Normal vs Face Normal

- Effects
  - Offset vertex by normal direction
    - Debug vertex normal by color
  - Scale individual triangles
    - Pre-compute center of triangle in CPU and store in vertex

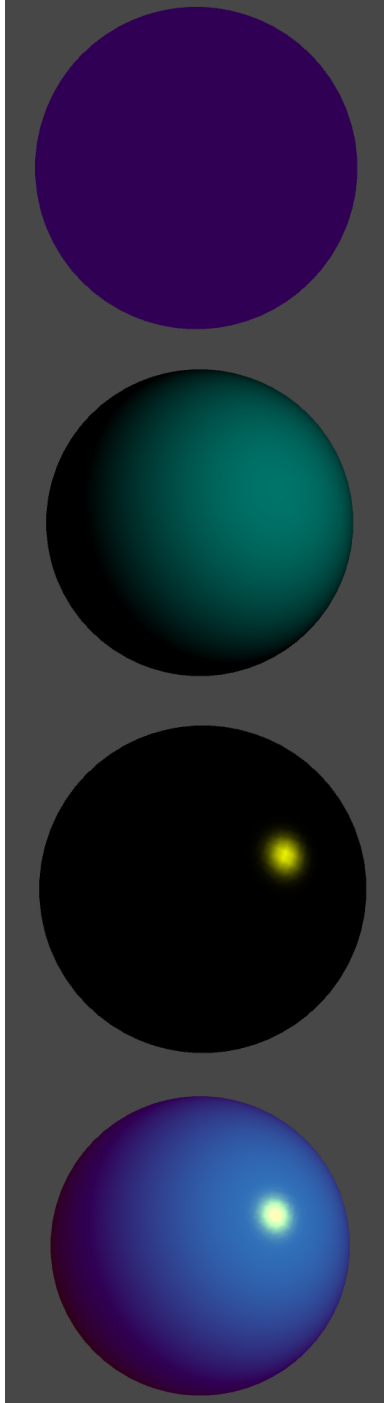


Basic Lighting

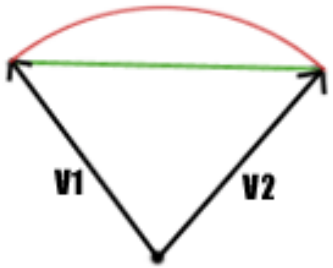
Lighting (Simple Local Illumination)

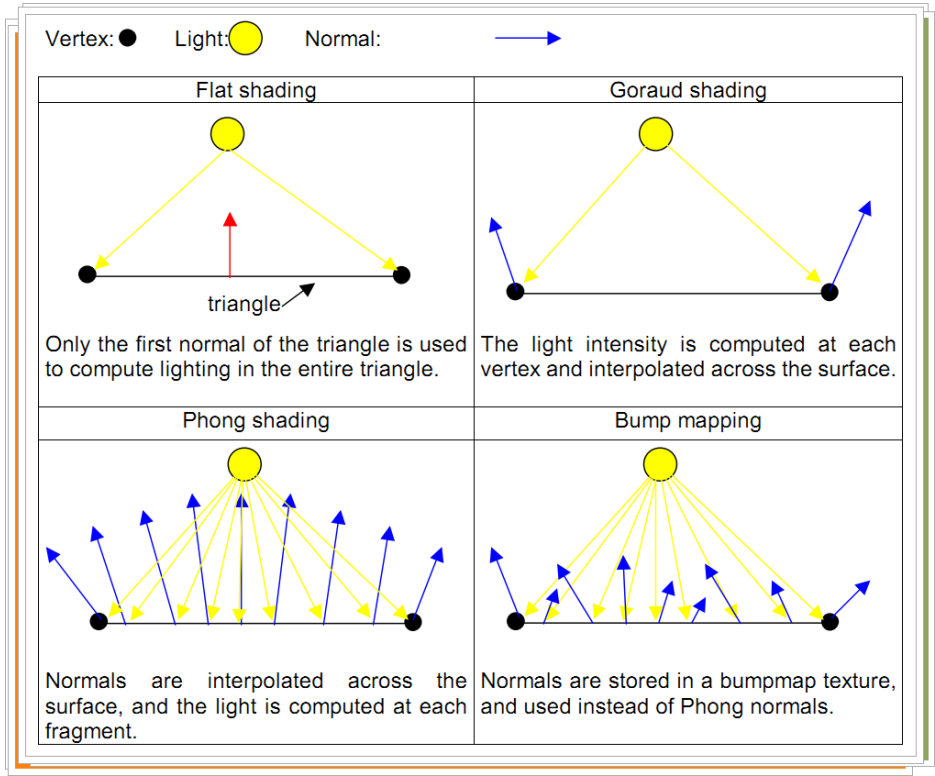


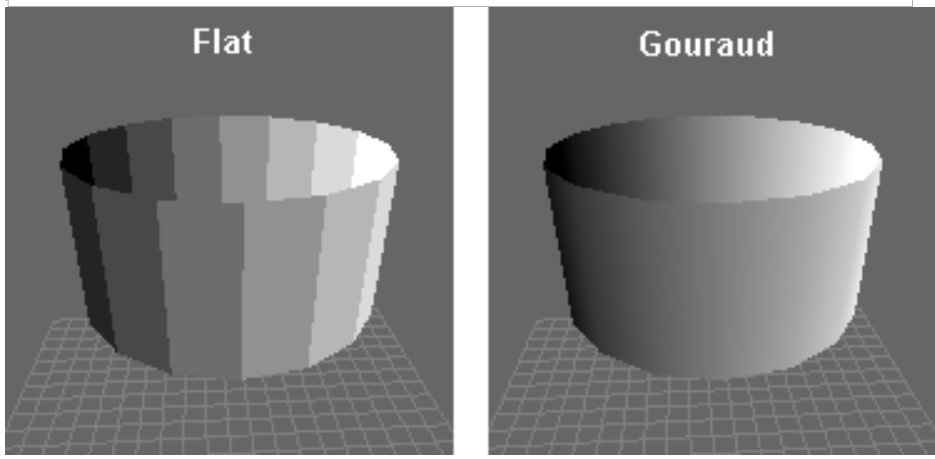
- Ambient
  - `Ambient = MaterialAmbientColor`
- Diffuse
  - Depends on light source direction
  - `L = normalize(LightPos – surfacePos)`
  - `Diffuse = MaterialDiffuseColor * max(0,dot(N,L))`
- Specular
  - Also depends on the view angle
  - `R = dot(N,L) * 2 * N - L`
    - `Or R = -reflect(L,N)`
  - `V = normalize(eyePos - surfacePos)`
  - `Angle = max(0, dot(R,V))`
  - `Specular = pow(Angle, MaterialSpecularShininess) * MaterialSpecularColor`
- Output color
- Output = Ambient + Diffuse + Specular



- Flat shading
  - Face Normal
- Smooth/Gouraud shading
  - Vertex Normal and Face Normal
  - Vertex color interpolation (varying variable)
- Per Vertex vs Per pixel lighting
- Per Pixel Normalization
- Light intensity decay from distanc
  - LightColor = LightColor * distance^2
- https://classroom.udacity.com/courses/cs291







Toon / Cel Shading

- Determine Color base on light direction
  - If (angle > 15) return color0
  - If (angle > 45) return color1
  - If (angle > 80) return color2
  -
- Texture Lookup
  - More Flexible for Artist
- Outline
  - Back face with offset
  - Cull Front Faces
- Better Outline
  - Edge detection in 2D
  - Ref: Frei-Chen edge detector

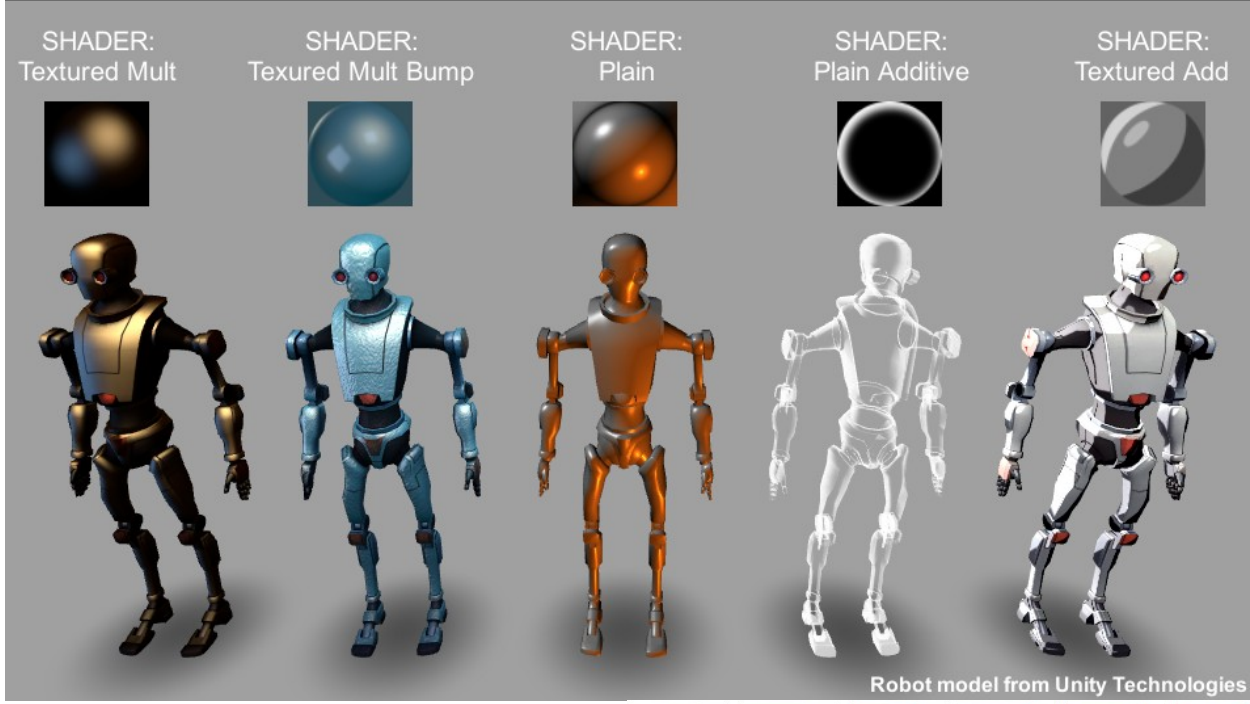
Image Based Lighting

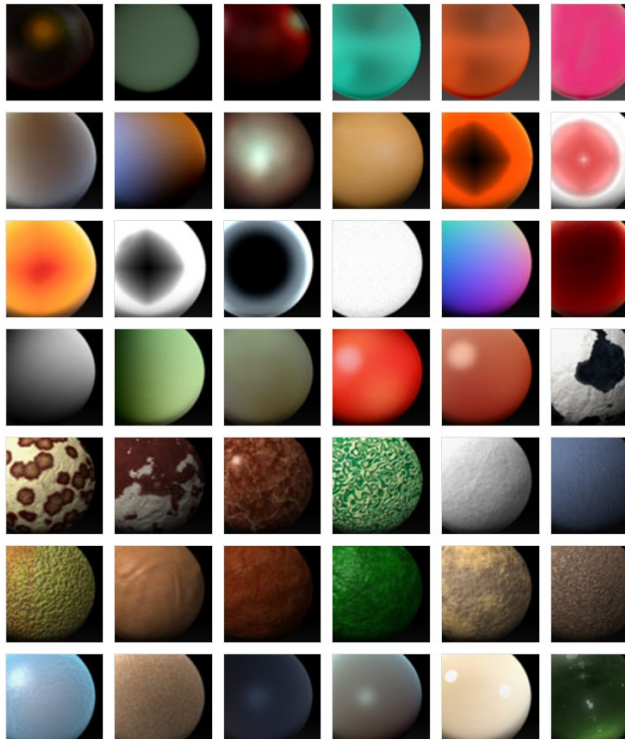
MatCap

Mapping view space normal to spherical image

- All in one
  - Ambient + Diffuse + Specular
  - Reflection (Environment Map)
- How to get MatCap texture
  - Capture by camera with Reflection ball
  - Render Sphere in 3D Software
  - Paint by Artist in Photoshop
- Pros - Very cheap, only 1 texture sampling
- Cons - May looks weird from different angle

- What about MatCap + Normal Map



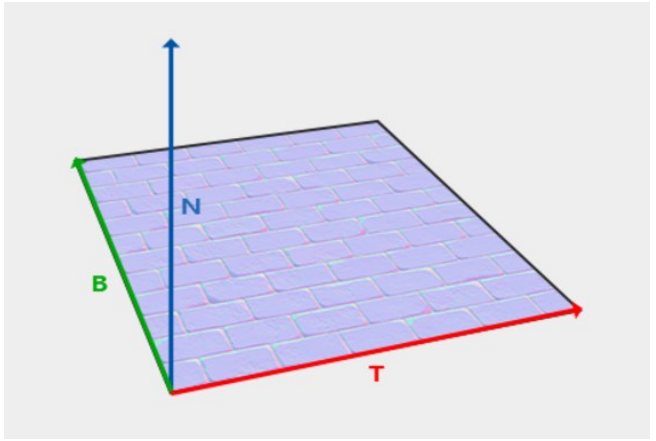


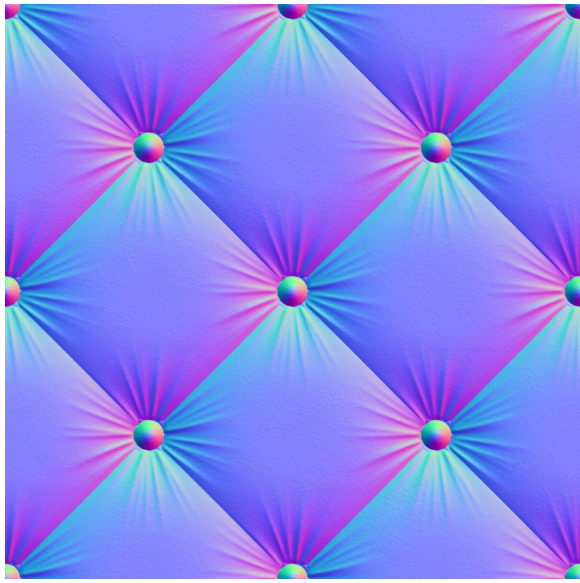
Specular Map and Normal Map

- Specular Color by texture

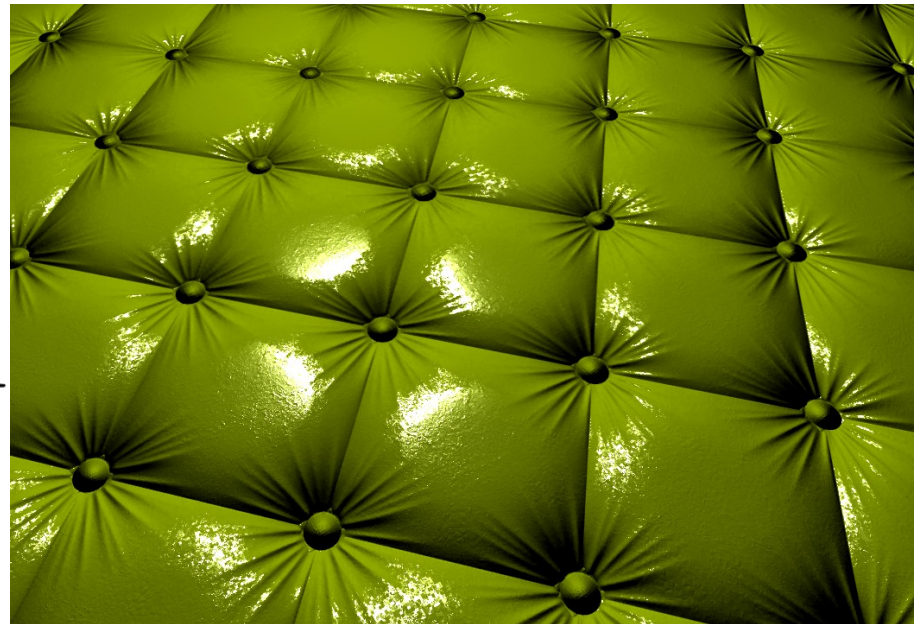
- Normal Map
  - Tangent Space
  - Vertex Tangent
  - Binormal = cross(Normal, Tangent)
  -

- Pack into one texture
  - RGB – Normal
  - or RG - Normal is unit vector
    - $X^2 + Y^2 + Z^2 = 1$
    - $Z = \text{sqrt}(1 - X^2 + y^2)$
  - A – Specular (intensity only)

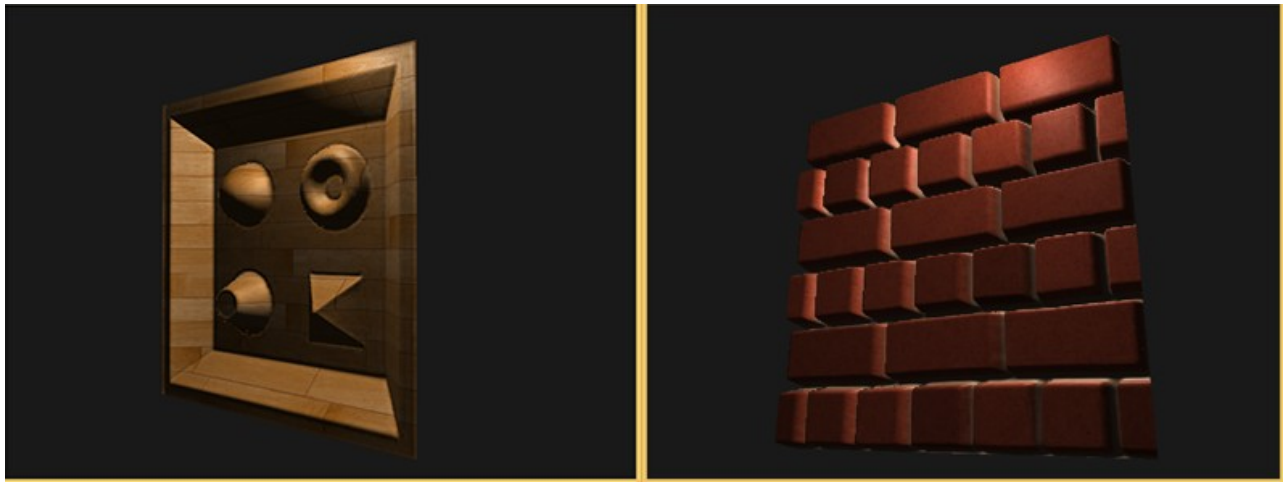


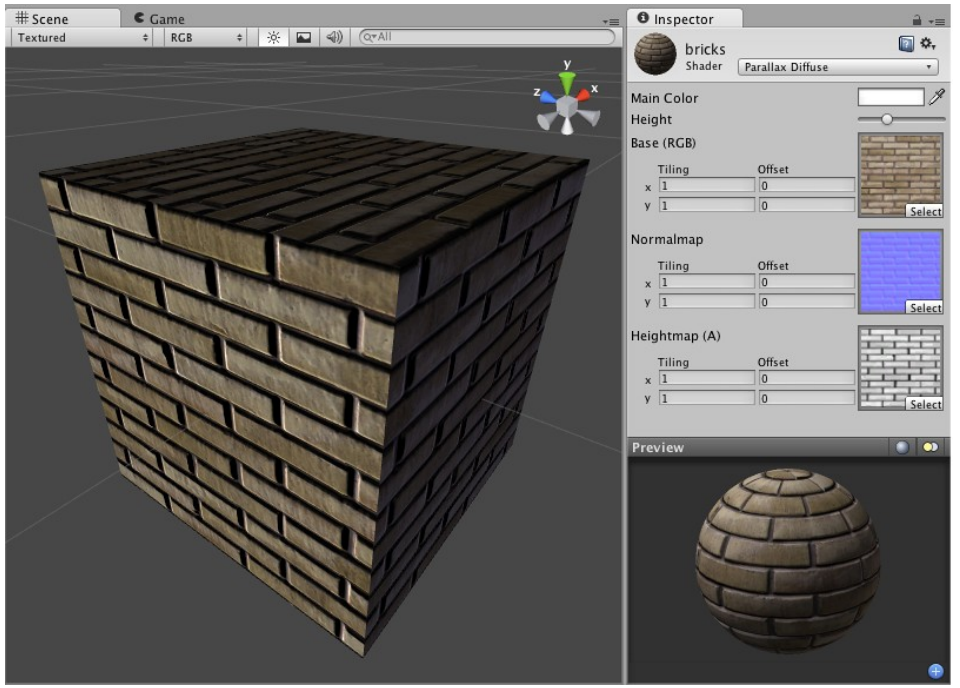


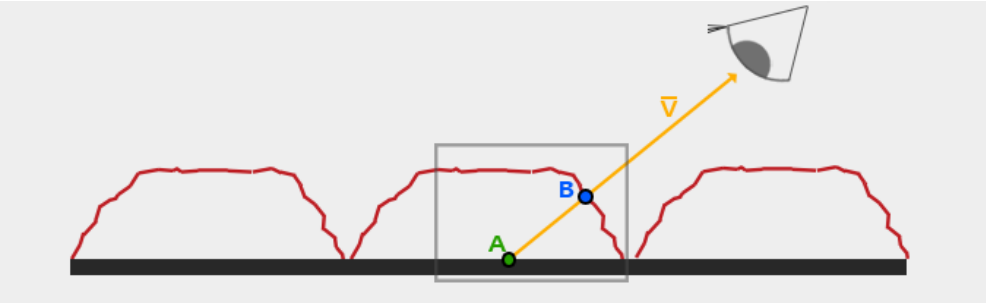
$$\text{mat3 TBN} = \begin{pmatrix} Tx & Ty & Tz \\ Bx & By & Bz \\ Nx & Ny & Nz \end{pmatrix}$$



Parallax Mapping







https://learnopengl.com/Advanced-Lighting/Parallax-Mapping

Environment Map

- CubeMap

Shader Performance & Optimization

Pixel Overdraw

- Drawing the same pixel multiple times in single frame
- Per pixel operation is not cheap
  - Per pixel lighting
  - 100+ light sources (multiple pass might needed)
  - Normal/Parallax map
- Solution:
  - Drawing objects from front-to-back
    - Depth buffer can help by early out
    - Sorting at object level instead of triangles

Deferred Shading

- Geometry Buffer (G-Buffer)
  - Lighting calculation do only once per pixel
  - Information needed by lighting in Screen space
  - G-Buffer Information
    - Diffuse Color
    - Z-Buffer
    - Surface Normal
  - Multiple Render Target
    - Multiple output from single pixel shader function

- Disadvantages
  - Unable to handle transparency
  - Hardware Anti-aliasing may broken

Render Target / Multi-pass rendering

- Blur
  - 3x3, 5x5
  - Separate Horizontal / Vertical pass
  - Extra round for H/V passes
- DOF
  - Depth buffer
  - Screen space - Toys camera effect

Shadow

Physically based rendering