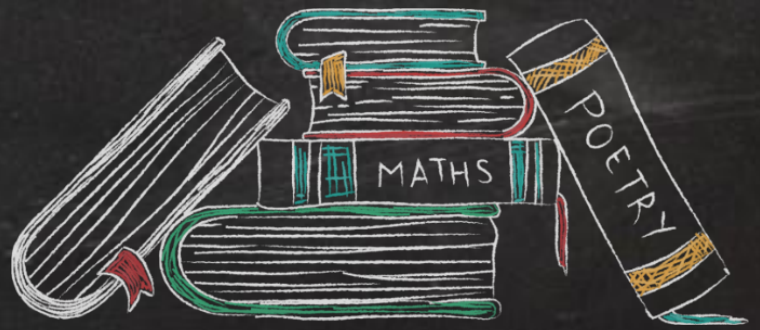
今日内容

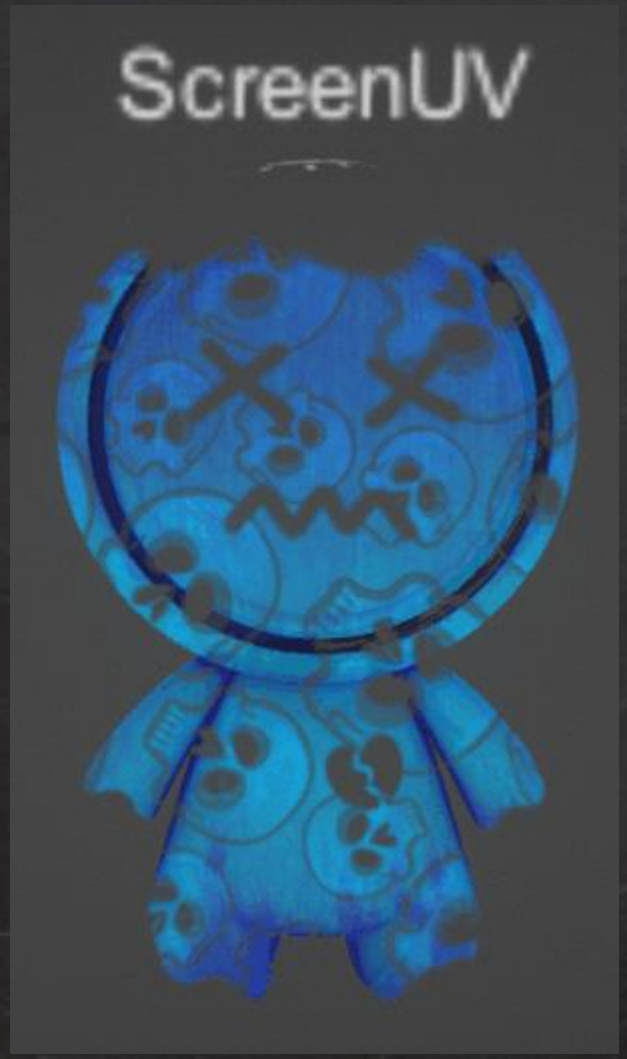
- 符文·化神·屏幕UV
- 符文·化神·屏幕扰动
- 任务委托

$\frac{24 \times 5}{33}$

1 符文·化神·屏幕UV



01 案例



- AB基础上追加一个屏幕采样并流动的Alpha；
- 屏幕UV在卡通渲染的时候讲过连连看的实现方法，今天介绍代码的实现方法，并解决一些细节问题；

02 代码实现

1. 以L13_AB为模板，CtrlCV；
2. 修改路径名；
3. 定义材质面板：
    1. _MainTex：RGB：颜色 A：透贴
    2. _Opacity：总体不透明度
    3. _ScreenTex：屏幕纹理
4. SubShaderTags，不用改；
5. 混合模式：Blend One OneMinusSrcAlpha；
6. 对应声明输入参数；

```
Shader "AP01/L17/ScreenUV" {
    Properties {
        _MainTex ("RGB: 颜色 A: 透贴", 2d) = "gray"{}
        _Opacity ("透明度", range(0, 1)) = 0.5
        _ScreenTex ("屏幕纹理", 2d) = "black" {}
    }
    SubShader {
        Tags {
            "Queue"="Transparent"                // 调整渲染顺序
            "RenderType"="Transparent"           // 对应改为Cutout
            "ForceNoShadowCasting"="True"        // 关闭阴影投射
            "IgnoreProjector"="True"             // 不响应投射器
        }
        Pass {
            Name "FORWARD"
            Tags {
                "LightMode"="ForwardBase"
            }
            Blend One OneMinusSrcAlpha           // 混合方式

            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"
            #pragma multi_compile_fwdbase_fullshadows
            #pragma target 3.0
            // 输入参数
            uniform sampler2D _MainTex;
            uniform half _Opacity;
            uniform sampler2D _ScreenTex;   uniform float4 _ScreenTex_ST;
```

03 代码实现

7. 输入结构，不用改；

8. 输出结构，追加screenUV用于屏幕纹理采样；

9. 顶点Shader，对应追加screenUV 的计算方法：

10. 像素Shader：

    1. 分别对MainTex，screenTex采样；
    2. 计算finalRGB，opacity；
    3. 返回值；

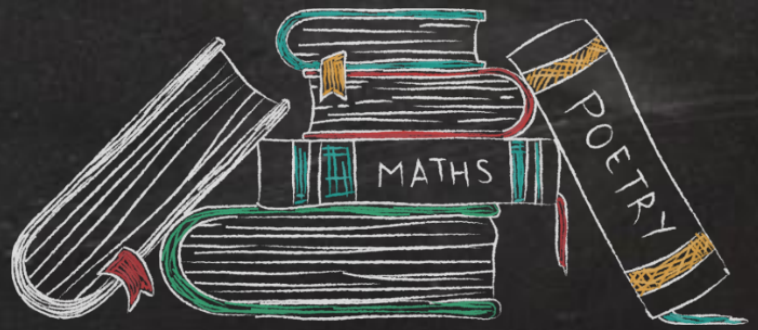
```
// 输入结构
struct VertexInput {
    float4 vertex : POSITION;       // 顶点位置 OS
    float2 uv : TEXCOORD0;          // UV信息
};
// 输出结构
struct VertexOutput {
    float4 pos : SV_POSITION;       // 顶点位置 CS
    float2 uv : TEXCOORD0;          // UV信息
    float2 screenUV : TEXCOORD1;    // 屏幕UV
};
// 输入结构>>>顶点Shader>>>输出结构
VertexOutput vert (VertexInput v) {
    VertexOutput o = (VertexOutput)0;
        o.pos = UnityObjectToClipPos(v.vertex);     // 顶点位置 OS>CS
        o.uv = v.uv;                                // UV信息
        float3 posVS = UnityObjectToViewPos(v.vertex).xyz;                  // 顶点位置 OS>VS
        float originDist = UnityObjectToViewPos(float3(0.0, 0.0, 0.0)).z;   // 原点位置 OS>VS
        o.screenUV = posVS.xy / posVS.z;            // VS空间畸变校正
        o.screenUV *= originDist;                   // 纹理大小按距离锁定
        o.screenUV = o.screenUV * _ScreenTex_ST.xy - frac(_Time.x * _ScreenTex_ST.zw);  // 启用屏幕纹理ST

    return o;
}
// 输出结构>>>像素
half4 frag(VertexOutput i) : COLOR {
    half4 var_MainTex = tex2D(_MainTex, i.uv);              // 采样 基本纹理 RGB颜色 A透贴
    half var_ScreenTex = tex2D(_ScreenTex, i.screenUV).r;   // 采样 屏幕纹理
    // FinalRGB 不透明度
    half3 finalRGB = var_MainTex.rgb;
    half opacity = var_MainTex.a * _Opacity * var_ScreenTex;
    // 返回值
    return half4(finalRGB * opacity, opacity);
}
ENDCG
        }
    }
}
```

04 核心代码分析

```
float3 posVS = UnityObjectToViewPos(v.vertex).xyz;                    // 顶点位置 OS>VS
float originDist = UnityObjectToViewPos(float3(0.0, 0.0, 0.0)).z;     // 原点位置 OS>VS
o.screenUV = posVS.xy / posVS.z;                  // VS空间畸变校正
o.screenUV *= originDist;                         // 纹理大小按距离锁定
o.screenUV = o.screenUV * _ScreenTex_ST.xy - frac(_Time.x * _ScreenTex_ST.zw);   // 启用屏幕纹理ST
```

1. 计算顶点在ViewSpace位置，其xy轴即大概对应屏幕UV坐标；
2. 除以屏幕深度校正畸变；
3. 计算模型原点在ViewSpace位置，取Z轴深度，即距离；
4. 乘以screenUV以固定屏幕纹理Tiling尺寸；
5. 启用_ScreenTex的ST功能，乘以ScreenTex_ST.xy以支持缩放；
6. 加强Offset功能为Flow动画，参照L15课内容；

2 符文·化神·屏幕扰动



01 案例

- 彩色玻璃正片叠底效果，支持透贴；
- 半透明部分，按颜色纹理对背景做折射扭曲；

02 代码实现

1. 以L13_AB为模板，CtrlCV；
2. 修改路径名；
3. 定义材质面板：
    1. _MainTex：RGB：颜色 A：透贴
    2. _Opacity：总体不透明度
    3. _WarpMidVal：扭曲中值
    4. _WarpInt：扭曲强度
4. SubShaderTags，不用改；
5. 追加GrabPass，以获取扭曲背景_BGTex；
6. 混合模式：Blend One OneMinusSrcAlpha；
7. 对应声明输入参数，并声明获取扭曲背景_BGTex；

```
Shader "AP01/L17/ScreenWarp" {
    Properties {
        _MainTex    ("RGB: 颜色 A: 透贴", 2d) = "gray"{}
        _Opacity    ("不透明度", range(0, 1)) = 0.5
        _WarpMidVal ("扰动中间值", range(0, 1)) = 0.5
        _WarpInt    ("扰动强度", range(0, 5)) = 1
    }
    SubShader {
        Tags {
            "Queue"="Transparent"               // 调整渲染顺序
            "RenderType"="Transparent"          // 对应改为Cutout
            "ForceNoShadowCasting"="True"       // 关闭阴影投射
            "IgnoreProjector"="True"            // 不响应投射器
        }
        // 获取背景纹理
        GrabPass {
            "_BGTex"
        }
        // Forward Pass
        Pass {
            Name "FORWARD"
            Tags {
                "LightMode"="ForwardBase"
            }
            Blend One OneMinusSrcAlpha          // 混合方式
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"
            #pragma multi_compile_fwdbase_fullshadows
            #pragma target 3.0
            // 输入参数
            uniform sampler2D _MainTex;
            uniform half _Opacity;
            uniform half _WarpMidVal;
            uniform half _WarpInt;
            uniform sampler2D _BGTex;   // 拿到背景纹理
```

03 代码实现

8. 输入结构，不用改；
9. 输出结构，追加grabPos用于背景纹理采样；
10. 顶点Shader，对应追加grabPos的计算方法：
10. 像素Shader：
    1. 对MainTex采样；
    2. 将MainTex某通道或者灰度作为扰动背景纹理UV的源，扰动背景纹理UV；
    3. 用扰动后的背景纹理UV对背景采样；
    4. 计算finalRGB，opacity；
    5. 返回值；

```
        // 输入结构
        struct VertexInput {
            float4 vertex : POSITION;       // 顶点位置 总是必要
            float2 uv : TEXCOORD0;          // UV信息 采样贴图用
        };
        // 输出结构
        struct VertexOutput {
            float4 pos : SV_POSITION;       // 顶点位置 总是必要
            float2 uv : TEXCOORD0;          // UV信息 采样贴图用
            float4 grabPos : TEXCOORD1;     // 背景纹理采样坐标
        };
        // 输入结构>>>顶点Shader>>>输出结构
        VertexOutput vert (VertexInput v) {
            VertexOutput o = (VertexOutput)0;
                o.pos = UnityObjectToClipPos( v.vertex);    // 顶点位置 OS>CS
                o.uv = v.uv;                                // UV信息
                o.grabPos = ComputeGrabScreenPos(o.pos);    // 背景纹理采样坐标
            return o;
        }
        // 输出结构>>>像素
        half4 frag(VertexOutput i) : COLOR {
            // 采样 基本纹理 RGB颜色 A透贴
            half4 var_MainTex = tex2D(_MainTex, i.uv);
            // 扰动背景纹理采样UV
            i.grabPos.xy += (var_MainTex.r - _WarpMidVal) * _WarpInt;
            // 采样背景
            half3 var_BGTex = tex2Dproj(_BGTex, i.grabPos).rgb;
            // FinalRGB 不透明度
            half3 finalRGB = lerp(1.0, var_MainTex.rgb, _Opacity) * var_BGTex;
            half opacity = var_MainTex.a;
            // 返回值
            return half4(finalRGB * opacity, opacity);
        }
        ENDCG
    }
  }
}
```

04 核心代码分析

1. 获取背景纹理方法：
    1. GrabPass：方便，但贵；
    2. CommandBuffer：前Srp时代管线自定义方法；
    3. Lwrp/Urp：后Srp时代管线自定义方法（推荐）；
2. 拿到背景纹理方法：输入参数中做对应声明；
3. 输出结构追加采样坐标：grabPos；（黑盒）
4. 顶点shader追加grabPos计算方法；（黑盒）
5. 扰动grabPos.xy，类似L15_GhostWarp；
6. 采样背景纹理：tex2Dproj(_BGTex, i.grabPos)；（黑盒）

1
```
// 获取背景纹理
GrabPass {
    "_BGTex"
}
```

2
```
uniform sampler2D _BGTex;    // 拿到背景纹理
```

3
```
float4 grabPos : TEXCOORD1;      // 背景纹理采样坐标
```

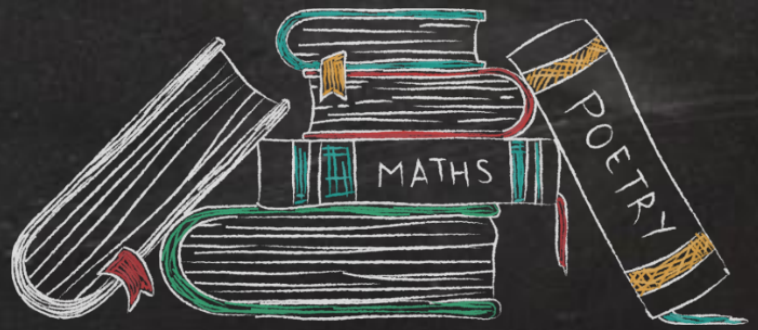
4
```
o.grabPos = ComputeGrabScreenPos(o.pos);     // 背景纹理采样坐标
```

5
```
// 扰动背景纹理采样UV
i.grabPos.xy += (var_MainTex.r - _WarpMidVal) * _WarpInt;
```

6
```
// 采样背景
half3 var_BGTex = tex2Dproj(_BGTex, i.grabPos).rgb;
```

$\frac{24 \times 5}{33}$

3 任务委托



01 作业

必做作业：

- 代码：ScreenUV，ScreenWarp；

创意作业：

- 课内知识自由发挥；