**一、综述**

**二、研究成果**

**（一）成果总览**

1.Unity可视化美术资源性能分析工作流；

2.地下城堡3基于2D地图可视化编辑器；

3.地下城堡3基于Spine角色可视化编辑器；

4.局部Bloom的实现方式；

5.实现unity世界坐标中的Spine与屏幕坐标下的UI交互;

6.解决在unity同时使用Sprite与Spine产生的批次过大问题;

7. unity使用Spine结合TexturePacker的Z轴排序错位问题解决方案;

1.Unity可视化美术资源性能分析工作流  
研发时限:  
2019年06月~2019年06月  
  
2.地下城堡3基于2D地图可视化编辑器；  
研发时限:  
2019年05月~2019年11月  
  
3.地下城堡3基于Spine角色可视化编辑器；  
研发时限:  
未完成  
  
4. 局部Bloom的实现方式  
研发时限:  
2019年04月~2019年05月  
  
5.实现unity世界坐标中的Spine与屏幕坐标下的UI交互;  
研发时限:  
2019年6月~2019年10月  
  
6.解决在unity同时使用Sprite与Spine产生的批次过大问题;  
研发时限:  
2019年08月~2019年08月  
  
7. unity使用Spine结合TexturePacker的Z轴排序错位问题;  
研发时限:  
2019年08月~2019年08月

**（二）成果细项**

**1.Unity可视化美术资源性能分析工作流**

1. 研发时限

2019年04月~2019年05月

1. 问题

【名词解释】：OverDraw：在屏幕内同一坐标上，像素与像素的叠加层数；RunTime：代码在游戏运行时候的状态；AlphaBlend ：透明测试；Avg FPS：游戏运行时的平均帧数；moba：移动平台多人对战竞技类型游戏；Shader：unity内置可编程着色器；

为了让移动端运行时，具有检测美术资源OverDraw和帧数的功能。则必须制作一个RunTime模式下，可以渲染OverDraw和FPS的可视化窗口。而Unity提供一个可以在Scene窗口下查看OverDraw和FPS显示的功能，使用者只能在编辑器内查看。这种方式有如下几个缺点：

1. 在编辑器显示OverDraw如果非常多，其实对电脑下性能是毫不在乎的，反而手机上只要几层全屏OverDraw就受不了了这正是手机GPU的硬伤。那么这里显示的帧数就不具有代表性，而编辑器内想通过OverDraw确定性能和帧数确实不实用的。
2. 不能根据不同机型和不同平台进行测试。
3. Unity提供的帧数显示非常不精准，有时候编辑器里面帧数一会几百帧，一会就只有几十帧，跨度大不准确。
4. 解决方案

利用Unity强大的Shader、相机全局渲染、GUI开发功能，我们通过以下3点解决上述问题。

1.在图像渲染的时候其实不管是否透明的物体，进行叠加都会存在OverDraw。那么使用一种半透颜色，去做AlphaBlend检测判断先后顺序，然后进行不断叠加就会越来越趋近于白色，这种原理来表现OverDraw层数的原理就非常相近了。编写一个Shader来显示当叠加色彩越多的时候就越趋近白色，那么他的OverDraw就越高。移动端上面全屏幕无叠加层数是性能最理想状态，颜色上呈暗橙色；全屏幕叠加层数2层也是可以接受范围内，颜色呈微亮橙色如此趋近白色；如果出现叠加层数3层，希望能够尽量减少叠加区域，不要出现相机内大范围的3层叠加；如果出现4层以上叠加，这是错误，要修复他们。

SubShader

{

Tags { "RenderType" = "Transparent" "Queue" = "Transparent" }

LOD 100

Fog { Mode Off }

ZWrite Off

ZTest Always

Blend One One

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 = UnityObjectToClipPos(v.vertex);

return o;

}

fixed4 frag (v2f i) : SV\_Target

{

return fixed4(0.1, 0.04, 0.02, 0);

}

ENDCG

}

}

2. 通过unity的相机全局渲染，把上面完成的Shader渲染到当前游戏的主相机中。

m\_Camera.SetReplacementShader(m\_OverDrawShader, "");

3.通过unity提供GUI开发，将计算得到间隔时间内Avg FPS可视化显示。除了通过OverDraw模式下颜色方式来判断美术资源性能以外，还可以通过Avg FPS来判断来判断性能，并且通过Avg FPS来判断性能是最准确的方式之一。一般手游画面帧数维持在30帧无波动的状态是最佳的，也是普通手机能承受的状态；性能较好的手机可以开启60帧低延迟模式，一般这用于moba或多人射击类型游戏，但帧数高了意味着对手机硬件消耗大，要考虑到耗电和发热为题，需要更高要求的优化美术资源。

    void Update () {

        UpdateTick();

    }

    void OnGUI()

    {

        DrawFps();

    }

    private void DrawFps()

    {

        if (mLastFps > 50)

        {

            GUI.color = new Color(0, 1, 0);

        }

        else if (mLastFps > 40)

        {

            GUI.color = new Color(1, 1, 0);

        }

        else

        {

            GUI.color = new Color(1.0f, 0, 0);

        }

        GUI.Label(new Rect(50, 32, 64, 24), "fps: " + mLastFps);

    }

    private long mFrameCount = 0;

    private long mLastFrameTime = 0;

    static long mLastFps = 0;

    private void UpdateTick()

    {

        if (true)

        {

            mFrameCount++;

            long nCurTime = TickToMilliSec(System.DateTime.Now.Ticks);

            if (mLastFrameTime == 0)

            {

                mLastFrameTime = TickToMilliSec(System.DateTime.Now.Ticks);

            }

            if ((nCurTime - mLastFrameTime) >= 1000)

            {

                long fps = (long)(mFrameCount \* 1.0f / ((nCurTime - mLastFrameTime) / 1000.0f));

                mLastFps = fps;

                mFrameCount = 0;

                mLastFrameTime = nCurTime;

            }

        }

    }

    public static long TickToMilliSec(long tick)

    {

        return tick / (10 \* 1000);

    }

1. 效果展示

以下两张图片分别展现同一相机画面里的美术资源，在关闭性能分析工具，以及开启性能分析工具的画面；如图1，在没有性能分析工具的情况下只能通过运行是否流畅判断当前相机的美术资源OverDraw性能部分的消耗，但是这种方式不能严谨的表现OverDraw性能消耗部分；如图2，开启性能分析工具后，可以通过叠加颜色的亮度判断当前相机的美术资源是否可用，能相对严谨的表现OverDraw性能部分的消耗。



图1 图2

1. 成果

实现了高效的工作流，可以在IOS、Android、WebGL平台内，使用真机调试下可以根据OverDraw显示的颜色和帧数去判断相机区域内美术资源性能消耗程度，从而减少美术资源因性能问题的返工。

1. **局部Bloom的实现方式**
2. 研发时限

2019年05月~2019年06月

1. 问题

【名词解释】： Bloom：泛光效果；Avg FPS：游戏运行时的平均帧数； Shader：unity内置可编程着色器；

为了让美术可以只对某些物体制作Bloom效果，并且不影响场景内的其他物体。则必须制作一种可以标记某些物体，并采集这些物体的图像进行泛光处理。而传统的全屏幕泛光效果，使用者直接对相机上所有物体泛光。这种方式有如下几个缺点：

1. 全屏泛光消耗较大，一般的移动端机型上展现会出现掉帧卡顿。
2. 有些物体不像做泛光影响，只能通过调暗色调之后调整阈值，导致这些物体暗淡无光难以调到满意效果。
3. 解决方案

利用Unity强大的Rendertexture渲染、stencil功能，我们通过以下6点解决上述问题。

1. 创建一个临时的Rendertexture RT1，尺寸为屏幕尺寸，设置为主相机的target。用于保存包含stencil值的场景图。创建一次，且当分辨率更新时重新创建。
2. 创建一个临时的Rendertexture RT2,尺寸为屏幕尺寸，用于获取stencil过滤后的图像。创建一次，创建一次，且当分辨率更新时重新创建。
3. 一个临时Rendertexture RT3,尺寸初始为屏幕尺寸，用于迭代实现Bloom效果，可调手动调节尺寸。并将RT2的的信息通过Graphics.Blit传递到RT3中，如果RT3尺寸比RT2小，也就起到一个降采样的效果。
4. 一个临时的Rendertexture RT4,和RT3一样的尺寸。用于迭代实现Bloom效果。
5. 迭代完后，将RT3通过Graphics.Blit传递到RT2,在OnRenderImage使用RT2进行最后的混合。

        void OnRenderImage(RenderTexture source, RenderTexture destination)

        {

            if (CheckResources() == false)

            {

                Graphics.Blit(source, destination);

                return;

            }

            int divider = resolution == Resolution.Low ? 4 : 2;

            float widthMod = resolution == Resolution.Low ? 0.5f : 1.0f;

            fastBloomMaterial.SetVector("\_Parameter", new Vector4(blurSize \* widthMod, 0.0f, threshold, intensity));

            source.filterMode = FilterMode.Bilinear;

            var rtW = source.width / divider;

            var rtH = source.height / divider;

            // downsample

            RenderTexture rt = RenderTexture.GetTemporary(rtW, rtH, 0, source.format);

            rt.filterMode = FilterMode.Bilinear;

            Graphics.Blit(Buffer, rt);

            var passOffs = blurType == BlurType.Standard ? 0 : 2;

            for (int i = 0; i < blurIterations; i++)

            {

                fastBloomMaterial.SetVector("\_Parameter", new Vector4(blurSize \* widthMod + (i \* 1.0f), 0.0f, threshold, intensity));

                // vertical blur

                RenderTexture rt2 = RenderTexture.GetTemporary(rtW, rtH, 0, source.format);

                rt2.filterMode = FilterMode.Bilinear;

                Graphics.Blit(rt, rt2, fastBloomMaterial, 2 + passOffs);

                RenderTexture.ReleaseTemporary(rt);

                rt = rt2;

                // horizontal blur

                rt2 = RenderTexture.GetTemporary(rtW, rtH, 0, source.format);

                rt2.filterMode = FilterMode.Bilinear;

                Graphics.Blit(rt, rt2, fastBloomMaterial, 3 + passOffs);

                RenderTexture.ReleaseTemporary(rt);

                rt = rt2;

            }

            fastBloomMaterial.SetTexture("\_Bloom", rt);

            Graphics.Blit(source, destination, fastBloomMaterial, 0);

            RenderTexture.ReleaseTemporary(rt);

            Buffer.Release();

            if (GetComponent<Camera>().allowMSAA)

            {

                GetComponent<Camera>().targetTexture = null;

            }

        }

1. 修改目标对象的Shader文件，在其中加入Stencil命令如图所示。

    SubShader {

        Tags { "RenderType"="Opaque" }

        LOD 200

        Stencil{

            Ref 5

            comp always

            Pass Replace

        }

        CGPROGRAM

        // Physically based Standard lighting model, and enable shadows on all light types

        #pragma surface surf Standard fullforwardshadows

        // Use shader model 3.0 target, to get nicer looking lighting

        #pragma target 3.0

        sampler2D \_MainTex;

        struct Input {

            float2 uv\_MainTex;

        };

        half \_Glossiness;

        half \_Metallic;

        fixed4 \_Color;

        // Add instancing support for this shader. You need to check 'Enable Instancing' on materials that use the shader.

        // See https://docs.unity3d.com/Manual/GPUInstancing.html for more information about instancing.

        // #pragma instancing\_options assumeuniformscaling

        //UNITY\_INSTANCING\_BUFFER\_START(Props)

        //  // put more per-instance properties here

        //UNITY\_INSTANCING\_BUFFER\_END(Props)

        void surf (Input IN, inout SurfaceOutputStandard o) {

            // Albedo comes from a texture tinted by color

            fixed4 c = tex2D (\_MainTex, IN.uv\_MainTex) \* \_Color;

            o.Albedo = c.rgb;

            // Metallic and smoothness come from slider variables

            o.Metallic = \_Metallic;

            o.Smoothness = \_Glossiness;

            o.Alpha = c.a;

        }

        ENDCG

    }

1. 效果展示

以下两张图片分别展现同一相机画面里的美术资源，使用全屏Bloom，以及使用局部Bloom的画面；如图1，使用全屏Bloom不但使得美术难以控制Bloom物体与场景所有物体泛光度的协调，并且性能消耗较大；如图2，局部Bloom后，精准控制物体是否泛光，而且不会影响到场景内的其他物体，并且性能消耗较小；为了方便对比图3展现了没有Bloom效果的对比图。

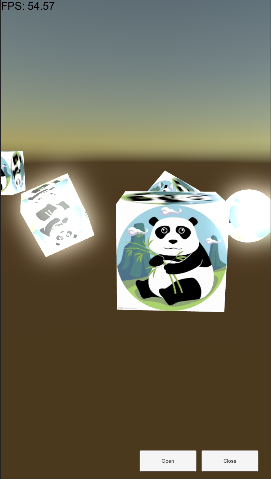
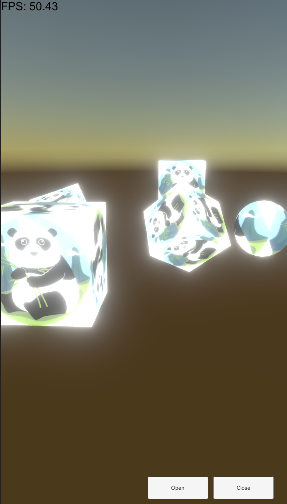


图1 图2

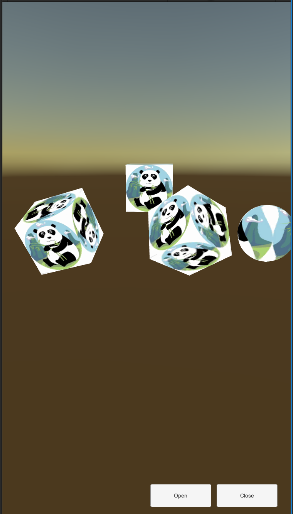


图3

1. 成果

实现了高性能的泛光效果，可以在IOS、Android、WebGL平台内，使得美术可以精准的控制场景内需要做泛光的物体，从而减少美术资源因性能和美观的问题。