GPU动画烘培工具

一.原理及优缺点

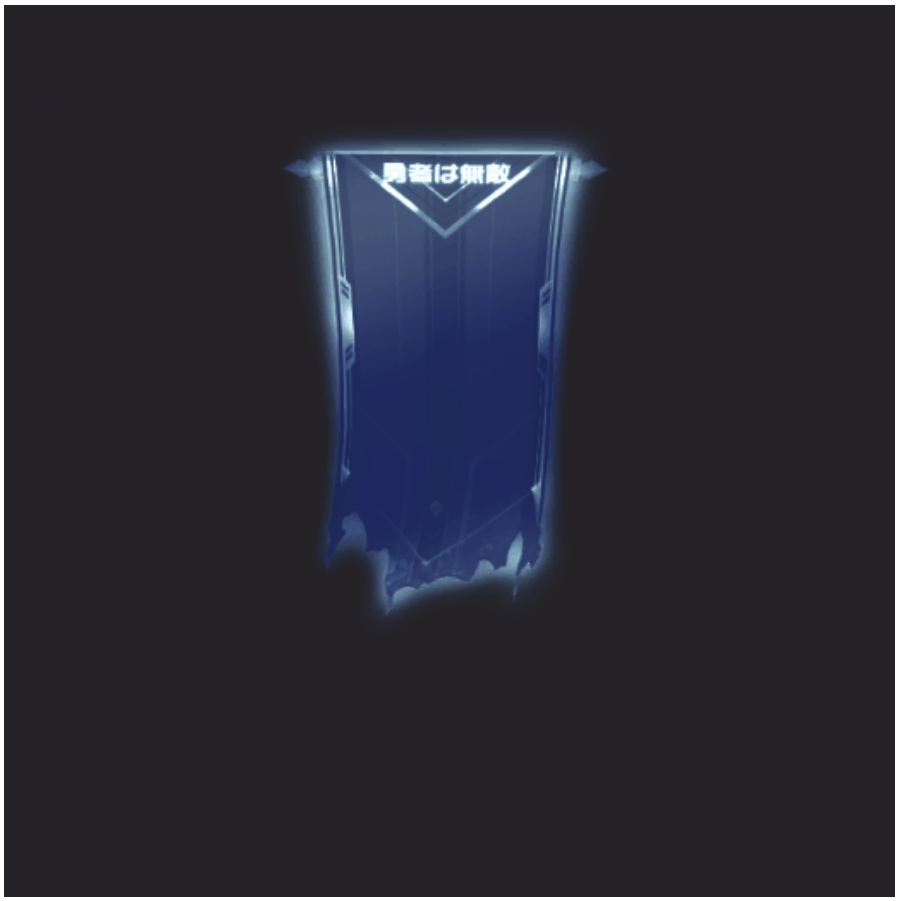
原理： 将模型的动画信息记录在贴图上，在顶点Shader阶段采样贴图还原动画效果。

优点：

1. 动画不再依赖Animation或Animator组件，优化了CPU和内存的占用，避免了播放时的性能峰值；
2. 可以使用GPU Instancing，适用于集群动画；
3. 播放方式灵活，适合做特效；

缺点：

1. 平台限制，SM3.0及以上可用；
2. 烘培精度，因为不是所有主流机型都支持RGBHalf或RGBFloat，目前安全的选择是RGB24Bit，即每个通道8Bit，动画精度上会有损失；
3. 动画融合，暂时没有完备的GPU动画融合的方案；



二.两种实现方式

按动画信息的不同记录还原方式，GPU动画分为两种解决方案：

1. 逐个顶点的，将动画过程中顶点的位置信息Bake到贴图上；
2. 逐根骨骼的，将动画过程中骨骼的Transform矩阵Bake到贴图上；

比较方案1和方案2：

1. 方案1，因为一般模型顶点远大于骨骼数，所以方案1占用的贴图大小一般大于方案2；如果模型顶点数很高，贴图可能大到无法接受；
2. 方案2，采样到一个完整的Transform矩阵，至少需要做3次采样；且一个点至少要收到2根以上的骨骼影响，所以一个顶点至少要做6次贴图采样，性能不佳；

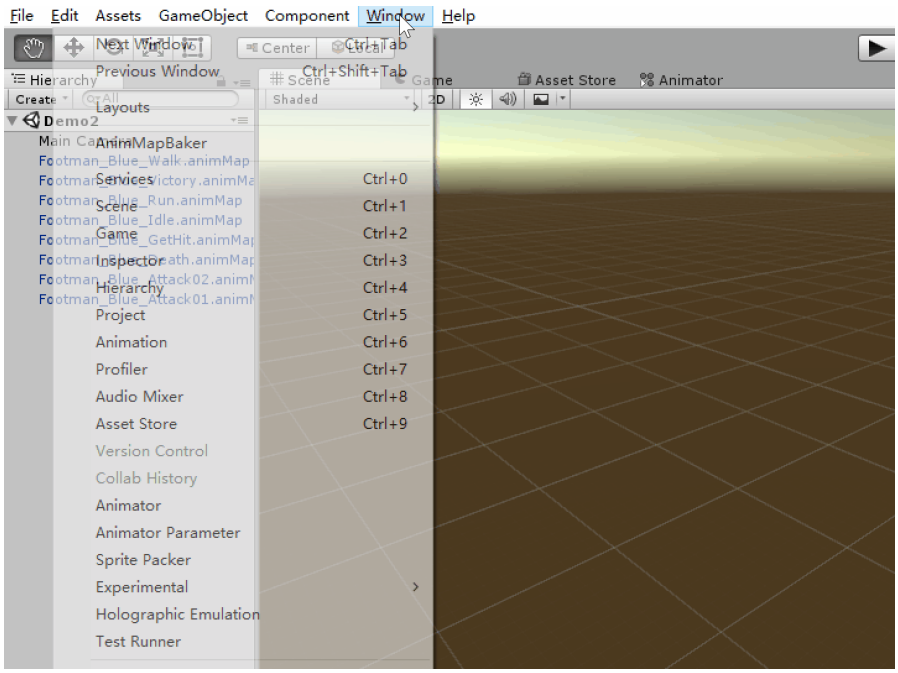
三.项目需求

特效希望在制作中加入模型的骨骼动画，但是Animation和Animator是不被允许的选择，于是就考虑使用GPU动画来实现。

因为特效用的模型，顶点数不高且有规范约束，所以我们采用上述的方案1来实现。

四.轮子

因为这块功能在2016和2017的Unity开发者大会上都有专题介绍，想必轮子肯定有人造好了。为了避免重复劳动，我找TA组罗勇那边要了一份Max脚本，也从Github上下到一份Unity的插件。因为我不会MaxScripts，所以选择了从后者入手。



上图为待改造的轮子AnimMapBaker，同时它也提供了一套Shader来配合GPU动画的实现，下载地址为 https://github.com/chenjd/Render-Crowd-Of-Animated-Characters 。

五.工具改进

最好的情况当然是轮子直接就可以跑，但往往不行，用AnimMapBaker跑了一遍项目需求，发现一些问题：

1. 兼容性方面：工具提供的Shader用到了某些特性，对移动端有兼容问题；AnimMap输出格式只能是RGBAHalf，目前很多主流手机都不支持；
2. 易用性方面：需要考虑美术实际需求做改进。

六.解决兼容问题

1.Shader的兼容性问题

```
v2f vert (appdata v, uint vid : SV_VertexID)
{
        UNITY_SETUP_INSTANCE_ID(v);

        float f = _Time.y / _AnimLen;

        fmod(f, 1.0);

        float animMap_x = (vid + 0.5) * _AnimMap_TexelSize.x;
        float animMap_y = f;

        float4 pos = tex2Dlod(_AnimMap, float4(animMap_x, animMap_y, 0, 0));

        v2f o;
        o.uv = TRANSFORM_TEX(v.uv, _MainTex);
        o.vertex = UnityObjectToClipPos(pos);
        return o;
}
```

AnimMapBaker提供的Shader，在顶点shader中声明了一个接收顶点ID的变量，这个特性只支持DX10 SM4.0 或 GLCore / OpenGL ES 3，对手游而言目前不可接受。我们可以把顶点ID存放在Mesh的某个UV通道里，借此避免对此特性的依赖，在我们的工具中选择的是UV3的x通道。

2.贴图格式的兼容性问题

> - *Graphics: Added support for BC4,BC5,BC6,BC7 compressed texture formats, and RGBAHalf format. These formats are supported on PC (DX11+, GL4, Metal) and consoles (PS4, XboxOne).*

上文引自Unity 5.5.0f3的Release Notes，AnimMapBaker一律采用RGBAHalf作为AnimMap的输出格式，移动端不可接受。但直接把输出模式从RGBAHalf改成RGB24Bit也不可行，因为RGB24Bit每个通道的值域为（0.0，1.0），此值域外的位置信息都会丢失。

所以，可行的办法是，预先估算模型所有顶点在动画过程中，在空间内移动的范围，用一组缩放位移值将此范围映射到有效值域（0.0，1.0）。再将缩放位移值记录在Material中，在顶点Shader中将其还原。

Bake脚本计算采样范围代码：

```
        // 计算采样范围
        void GetSamplerParams(ref Vector3 offset, ref
float scale)
        {
            switch (sampleMode)
            {
                case SampleMode.Normal:
                    offset = Vector3.zero;
                    scale = 1.0f;
                    return;
                case SampleMode.Optimize:
                    // 计算模型边界
                    var xMin = 0.0f;
                    var yMin = 0.0f;
                    var zMin = 0.0f;
                    var xMax = 0.0f;
                    var yMax = 0.0f;
                    var zMax = 0.0f;
                    var vertexCount =
skinedMesh.vertices.Length;
                    foreach (var state in
animationStateList)
                    {
                        // 获得帧数量和帧间隔
                        var frameCount =
Mathf.ClosestPowerOfTwo((int)(state.clip.frameRate *
state.length));
                        var frameDuration = state.length
/ frameCount;
                        // 烘培动画纹理
                        animation.Play(state.name);
                        var sampleTimer = 0.0f;
                        for (int i = 0; i < frameCount;
i++)
                        {
                            state.time = sampleTimer;
                            animation.Sample();
```

```
 skinMeshRenderer.BakeMesh(meshBuffer);
                            for (int j = 0; j <
vertexCount; j++)
                            {
                                var vertexPos =
meshBuffer.vertices[j];
                                xMin = Mathf.Min(xMin,
vertexPos.x);
                                yMin = Mathf.Min(yMin,
vertexPos.y);
                                zMin = Mathf.Min(zMin,
vertexPos.z);
                                xMax = Mathf.Max(xMax,
vertexPos.x);
                                yMax = Mathf.Max(yMax,
vertexPos.y);
                                zMax = Mathf.Max(zMax,
vertexPos.z);
                            }
                            sampleTimer += frameDuration;
                        }
                    }
                    // 计算模型bound
                    var size = new Vector3(xMax - xMin,
yMax - yMin, zMax - zMin);
                    var center = new Vector3(xMax + xMin,
yMax + yMin, zMax + zMin) * 0.5f;
                    // 计算平移缩放值
                    offset = center;
                    scale = Mathf.Max(size.x,
Mathf.Max(size.y, size.z));
                    offset -= Vector3.one * 0.5f * scale;
      // 平移0.5单元至正值区间
                    return;
                }
            }
```

Shader还原位移缩放代码：

```
        localPos.xyz = localPos.xyz * _SamplerParams.w +
_SamplerParams.xyz;
```

弊端：8Bit数的精度为1/256，考虑缩放，精度等于 1/256 * 缩放值；放缩值和动作范围正相关，精度和放缩值负相关；动画制作时的活动范围受限。

七.易用性提升

1. 导入器的一键规范化

因为动画的Bake操作要用到Animation的API，所以动画导入类型需设置为Legacy；再有动画导入后是供Bake用，而非最终资源，压缩优化无意义，应当设置为无损无压缩；所有这些导入器的设置，不能期望和美术口头约定就能保证正确，能落实到代码上的最好不BB。

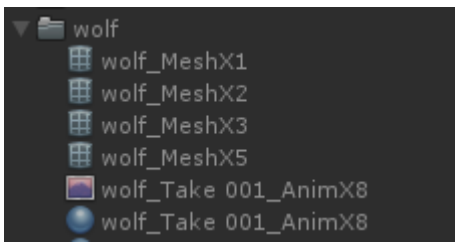
2.Bake时可选保留或删除某些Mesh信息

考虑优化，非GPU动画功能所需的Mesh信息一律默认删除；亦可主动勾选保留某信息，以供特殊需求之用。

3.GUI界预览烘培结果

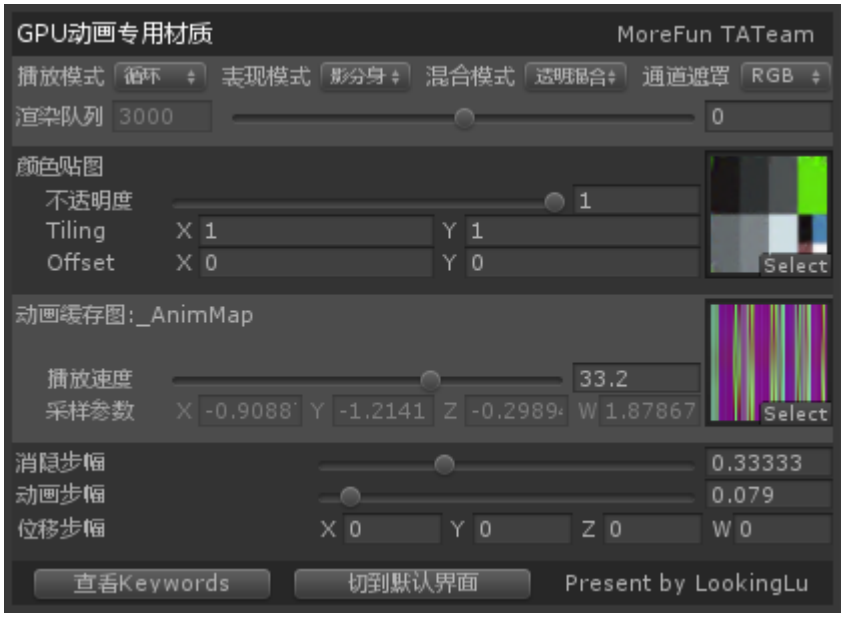
Bake后在GUI上自动部署预览场景，快速验证，提高美术自查效率和资源出错的可能性；

4.规范化资源命名和导出路径



一键输出所有资源到原始资源同目录的同名文件夹下，按资源用途和输出设置给以尾缀标识；自动生成Material，设置采样缩放位移值，美术可以做到对这个参数无感知。

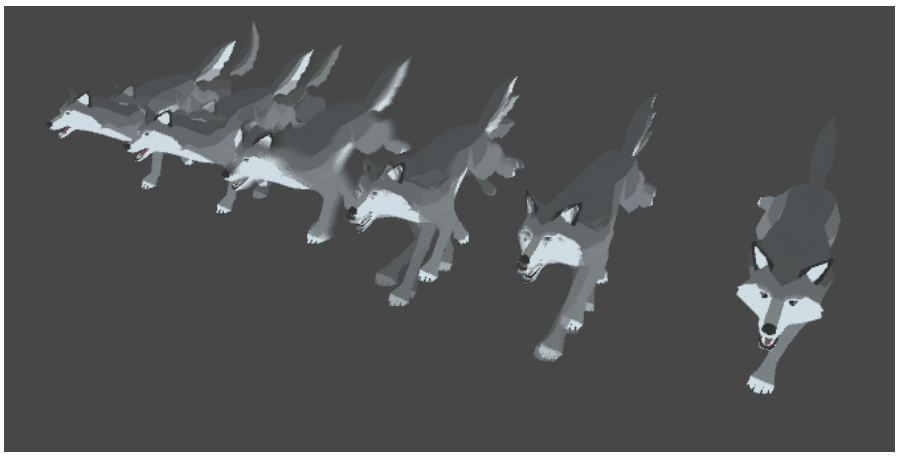
5.专用的多功能Shader及编辑面板



为避免美术记忆各种shader的用途，命名和设置方式，我对所有GPU动画所需的Shader做了整合；美术可以在中文化面板下将同个Shader配置成不同用途分支；GUI会根据美术的设置正确配置好Material的Keywords。

八.美术效果创新

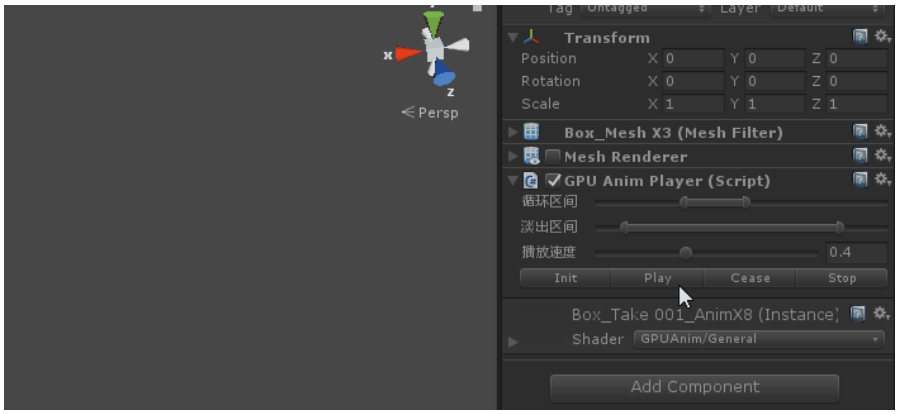
1.残影效果



因为可以在贴图上采样到任意动画时刻的模型状态，那么如果我们在BakeMesh的时候多存储几个备份（SubMesh）就可以实现残影效果。上面提到过，我们将顶点序号存在了UV3的x通道，物尽其用，我们再将SubMesh序号存在UV3的y通道里，这样Shader就可以识别模型中的各个SubMesh，再对它们设置不同的动画延迟，透明度，位移缩放既可实现如上图所示的各种残影效果。

从右到左：1.普通动画；2.连续动作残影；3.抽帧动作残影；4.动态模糊效果；5.连续动作残影漂移；6，抽帧动作残影漂移。

2.简单状态机制



为了实现Next项目中特效的通用状态机制（淡入 > 循环 > 得到结束命令 > 淡出），我写了一个组件GPUAnimPlayer，效果如上图。

谢谢！