GpuInstancing Animation

High-performance mobile platform animation

USER GUIDE

Release 1.1

Oct 2018

Email:yxriyin@163.com

Contents

Overview.....3
Quick Start.....4
Advantage.....5
Insufficient.....5
Advanced Usage.....5
GpuInstancing animation shader parameter settings.....5
Script Parameters.....7
Custom Usage：Add gpuinstancing animation to your shader.....7
Future Content.....9

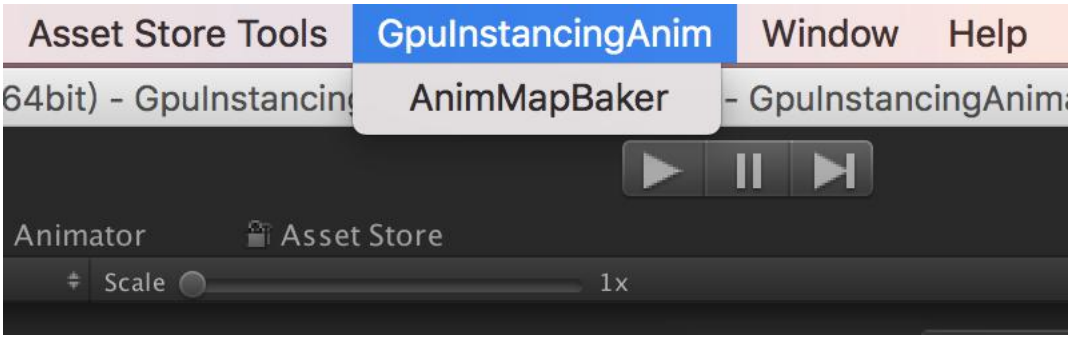
Overview

With the development of mobile platforms, mobile phones have gradually entered the era of Opengl3.0, and a new technology has entered our field of vision: Gpu Instancing. This technology has a qualitative improvement for large-scale object rendering. By default, skeletal animation can't use this technique, but we can save the matrix information of the skeletal animation into a texture and finish the skin in the shader to meet the usage conditions of Gpu Instancing and achieve a significant performance improvement.

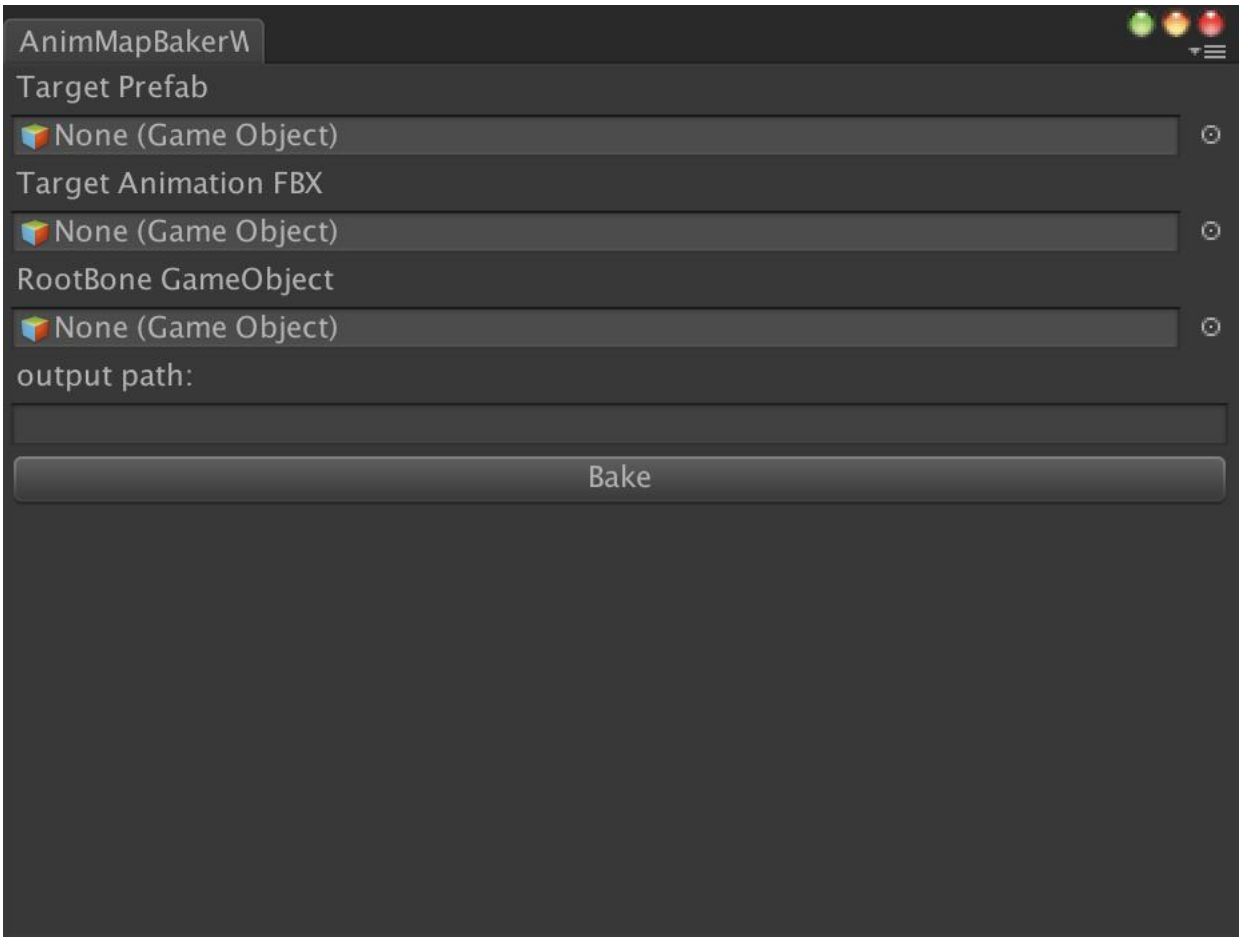
This document describes the basic usage of the high-performance mobile platform's animation and its parameter configuration instructions.

Quick Start

Click “GpuInstancingAnim->AnimMapBaker” to open the window for baking animation map.



Configure the desired parameter value in the window. Including:



Target Prefab – The prefab we need bake.

Target Animation FBX – The animation file to be baked (if the animation file and the model are not separated, it is the model file).

RootBone GameObject – The root skeleton of the model. If the model has multiple root bones, then just put the root of the model here.

Output path – File output path, default and prefab a directory.

After the configuration is complete, click bake and we can generate an animated map.

Advantage

1. Compared to the real-time operation of skeletal animation, this method is skinned in the gpu, and the technology of gpu instancing is used to greatly optimize the animation performance.

2. The animation information is recorded in the texture by baking the bone transformation matrix, adopting the format of RGBAHalf, which guarantees the precision and controls the size of the texture.

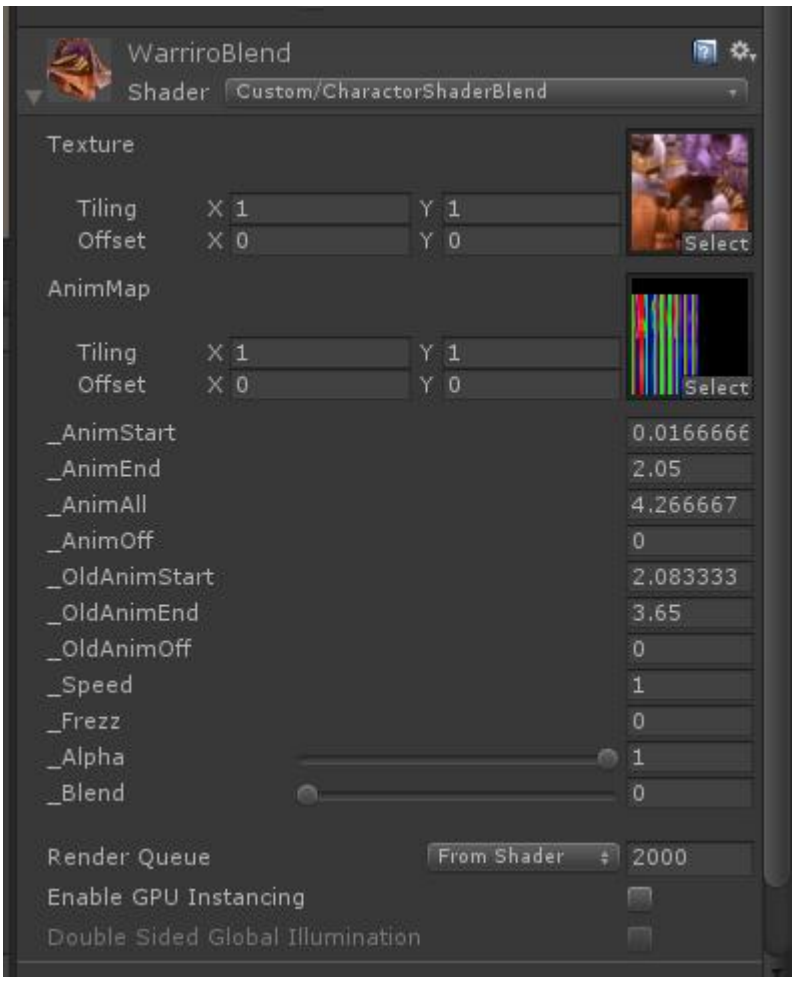
Insufficient

1. In performance considerations, currently only supports weight calculation for 2 bones.

2. The blend between actions is not supported yet, but will be added later.

Advanced Usage

GpuInstancing animation shader parameter settings



Texture - The main texture of the model.

AnimMap – Animated texture file with all bone information

_AnimStart – Animation start time

_AnimEnd – Animation end time

_AnimAll – All animation duration

_AnimOff – Animation Off

_OldAnimStart – Animation start time For Blend

_OldAnimEnd – Animation end time For Blend

_OldAnimOff – Animation Off For Blend

_Speed – Animation playback speed

_Frezz – Freeze animation

_Alpha – Alpha of model

Script Parameters

BaseAnimation – The controller of animation

API Reference:

Frezz: Frezz the animation

Resume: Resume the animation

changeSpeed: Change the speed of animation

changeAnimation: Start the animation

Custom Usage： Add gpuinstancing animation to your shader

Projects often have their own shaders, and then how to add gpuinstancing animation related code to their shaders.

Add the following variables to the property:

_AnimMap("AnimMap", 2D) = "white" {}

_AnimStart("_AnimStart", Float) = 0

_AnimEnd("_AnimEnd", Float) = 0

_AnimAll("_AnimAll", Float) = 0

_AnimOff("_AnimOff", Float) = 0

_Speed("_Speed", Float) = 1

_Frezz("_Frezz", Float) = 0

Also add gpu instancing

#pragma multi_compile_instancing

Note that we put the bone information in uv2. If you use uv2, please change the uv2 in the code to uv3.

Under normal circumstances, you only need to do the following:

```
struct appdata
{
    float2 uv : TEXCOORD0;
    float4 uv2 : TEXCOORD1;
    float3 normal : NORMAL;
    float4 vertex : POSITION;
```

```
        UNITY_VERTEX_INPUT_INSTANCE_ID

    };
```

Note the use of gpu instancing to define variables and access variables:

```
    UNITY_INSTANCING_BUFFER_START(Props)
        UNITY_DEFINE_INSTANCED_PROP(float, _AnimStart)
        UNITY_DEFINE_INSTANCED_PROP(float, _AnimEnd)
        UNITY_DEFINE_INSTANCED_PROP(float, _AnimOff)
        UNITY_DEFINE_INSTANCED_PROP(float, _Frezz)
        UNITY_DEFINE_INSTANCED_PROP(float, _Alpha)
        UNITY_DEFINE_INSTANCED_PROP(float, _Speed)
    UNITY_INSTANCING_BUFFER_END(Props)
```

Here are the most important parts of the vertex shader:

```
    float start = UNITY_ACCESS_INSTANCED_PROP(Props, _AnimStart);
    float end = UNITY_ACCESS_INSTANCED_PROP(Props, _AnimEnd);
    float off = UNITY_ACCESS_INSTANCED_PROP(Props, _AnimOff);
    float speed = UNITY_ACCESS_INSTANCED_PROP(Props, _Speed);
    float speed = UNITY_ACCESS_INSTANCED_PROP(Props, _Speed);
    float _AnimLen = (end - start);
    float f = (off + _Time.y * speed) / _AnimLen;
    f = fmod(f, 1.0);


        float animMap_x1 = (v.uv2.x * 3 + 0.5) * _AnimMap_TexelSize.x;
        float animMap_x2 = (v.uv2.x * 3 + 1.5) * _AnimMap_TexelSize.x;
        float animMap_x3 = (v.uv2.x * 3 + 2.5) * _AnimMap_TexelSize.x;
        float animMap_y = (f * _AnimLen + start) / _AnimAll;
        float4 row0 = tex2Dlod(_AnimMap, float4(animMap_x1, animMap_y, 0, 0));
        float4 row1 = tex2Dlod(_AnimMap, float4(animMap_x2, animMap_y, 0, 0));
        float4 row2 = tex2Dlod(_AnimMap, float4(animMap_x3, animMap_y, 0, 0));
```

```
float4 row3 = float4(0, 0, 0, 1);
float4x4 mat = float4x4(row0, row1, row2, row3);
float4 pos = mul(mat, v.vertex);
float3 normal = mul(mat, float4(v.normal, 0)).xyz;
```

More details can be found in the Shader file, haractorShader.shader

Future Content

Increase action event system.

Welcome to contact us.