GIANT
GIANT NETWORK

Giant Group
Introduction

DOTS在角色换装动画上的实践

Laugh & Ricky

2023.09


20:18
15666
44
2阶
[活动] 列王战争
进攻至高公爵94%
剩余时间:00:21:38
[活动] 战争英雄榜
1 东方欧巴 3752
2 路上 3464
3 夜无影 2780
未上榜 闪之阿贾塔尔 249
1 传说中的约翰
践踏先锋
重盾守卫
71%
宫廷法师
女武神
94%
至高公爵
我方控制
王城
2 闪之阿贾塔尔
基卡尔
3 Moosen
4 夜无影
191,252
列王战争
21:38
3 男妙法
涅诺尔
1 尼尼莫
王国
王国 BoBo:别打车了
王国 晨曦:骑士们，跟我来！ <银翼王国
王城> [前往助战]
18

- 百个玩家PVP战斗动画性能
- DOTS暂时没有成熟的动画


多部位动画解构复现
IComponentData
Animator的解构和复现
Blob Asset & Mathematics
Animation的解构与复现
GPUSkin & Vertex Animation
动画驱动
ECS & Material Overrides
实现换装
SharedMesh & SharedMaterial
细节优化

0、多部位动画解构复现

GameObject

抽象结构数据

IComponentData

```
public partial struct AnimationHierachyD
{
    public bool Container;
    public bool Vehicle;
    public bool Horse;
    public bool Body;
    public boo    .;

}
```

下一页

动画挂点层次

容器层级抽象：顶级容器 < 交通工具 < 坐骑 < 身体 < 各个部位

```
public partial struct AnimationHierachyData:IComponentData
{
    public bool Container;/*flag:顶级容器。预留*/
    public bool Vehicle;/*flag:交通工具。可能具有动画效果，人骑马时可以站在船上，影响骑马动画*/
    public bool Horse;/*flag:坐骑。具有动画效果，影响身体动画）*/
    public bool Body;/*flag:身体。具有动画效果，影响部位挂点动画）*/
    public bool Part;/*flag:部位。可能具有动画效果，可以扩展下级挂点）*/
}
```

演示部分代码，不适用于生产环境

```
public struct ObjectPartItemData
{
    public int PartID;
    public Entity PartEntity;
}
```

演示部分代码，不适用于生产环境

部位列表

```
public partial struct ObjectPartListData : IComponentData
{
    public ObjectPartItemData Body; //身体，使用骨骼动画GPUSKIN
    public ObjectPartItemData BodyXRay;
    public ObjectPartItemData BodyShadow;//独立的简模阴影渲染，顶点数量较少时，可以使用顶点动画
    public ObjectPartItemData Hair;
    public ObjectPartItemData HairXRay; //头发阴影不好使用简模做动画

    public ObjectPartItemData WeaponRight; //右手武器，不带动画。（弓箭可能带动画，可以用顶点动画）
    public ObjectPartItemData WeaponRightXRay;
    public ObjectPartItemData WeaponRightShadow;//不需要简模做阴影，则在武器Shader中做使用阴影pass

    public ObjectPartItemData Horse;
    public ObjectPartItemData HorseXRay;
    public ObjectPartItemData HorseShadow;
    public ObjectPartItemData Swing;
    public ObjectPartItemData Helmet;
    public ObjectPartItemData Vehicle;
    public ObjectPartItemData Effect;
    ......
}
```

演示部分代码，不适用于生产环境

1．每个部位都是独立的Entity，拥有特定的Component。

2．每一个部位独立Animate、Transform、Render，数据驱动关联

3．阴影也可以设计成一个部位，方便用简模做阴影。

4．XRAY等效国也可以设计成一个部位。

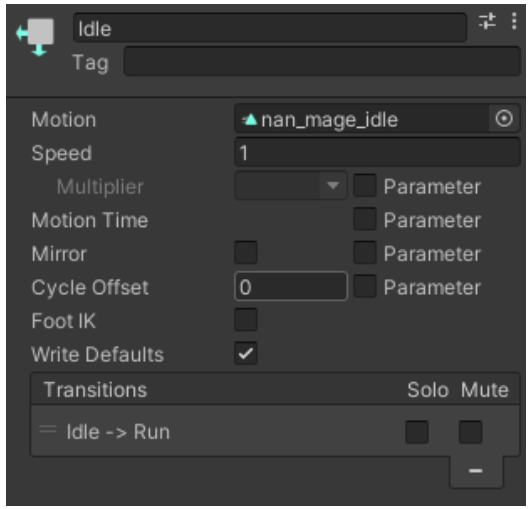
5．多Mesh的部位，需要分开

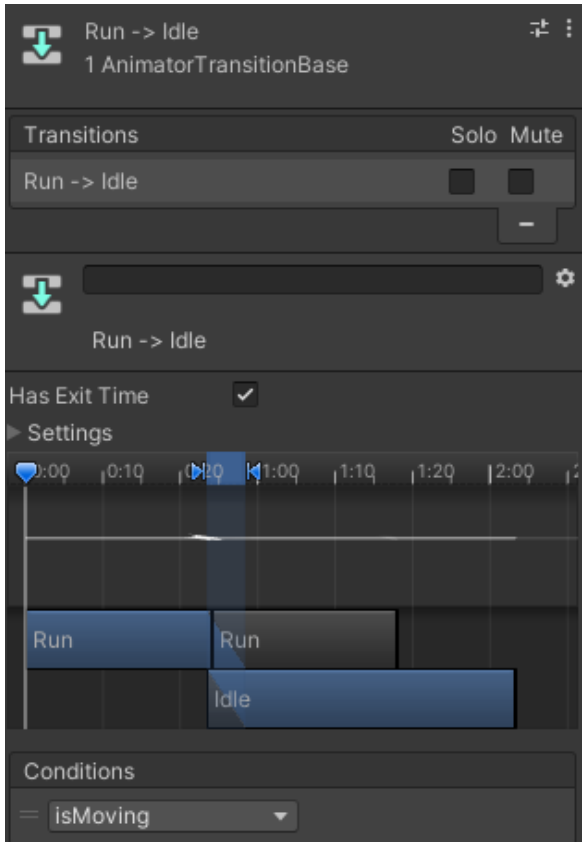
① 只能独立
② 方便多线程处理
③ 方便细致优化
④ 方便SRP Batch、Instancing

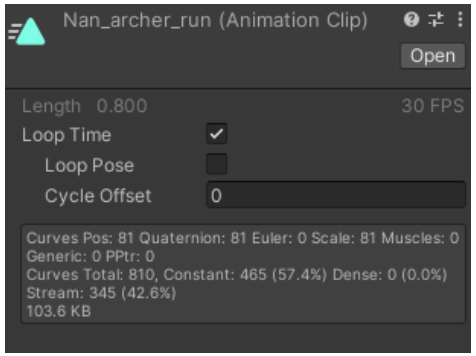
一、Animator的解构与复现

Animator解构

Animator → Blob Asset







动画状态、动画剪辑、 过度混合

转换工具

```
/*动画头信息*/
public partial struct AnimationMapHaderInfoBlob
            : IComponentData
{

    public BlobAssetReference<AnimationMapHeade
           AnimationMapHeaderInfoBlobRef;
}

public struct AnimationMapHeaderInfoBlobAsset
{
    public int VertexCount;
    public int BoneCount;
    public int FrameRate;
    public BlobArray<int3> HeaderInfoArray;
    /*动画过度相关数据*/
    public BlobArray<int> transitionFrames;
    public BlobArray<int> transitionNames;
}
```

演示部分代码，不适用于生产环境

下一页

Mathematics库进行数据计算

在System中，对动画数据进行检索和计算

```
p = parentTransform.Rot;
q = clipData.Point_Wing_Rot[curFrame];
parentTransform.Rot = new float4(
p.w * q.x + p.x * q.w + p.y * q.z - p.z * q.y,
p.w * q.y + p.y * q.w + p.z * q.x - p.x * q.z,
p.w * q.z + p.z * q.w + p.x * q.y - p.y * q.x,
p.w * q.w - p.x * q.x - p.y * q.y - p.z * q.z
);
m = new Quaternion(p.x, p.y, p.z, p.w);
localPos = clipData.Point_Wing_Pos[curFrame];
localPos = m * localPos;
parentTransform.Pos += localPos;
```

演示部分代码，不适用于生产环境

二、Animation的解构与复现

动画片段的实现方案

- 基于CPU的动画
  - JobSystem
  - Mathemtics
  - Burst
- 基于GPU的动画
  - GPUSkin
  - Vertex Animation

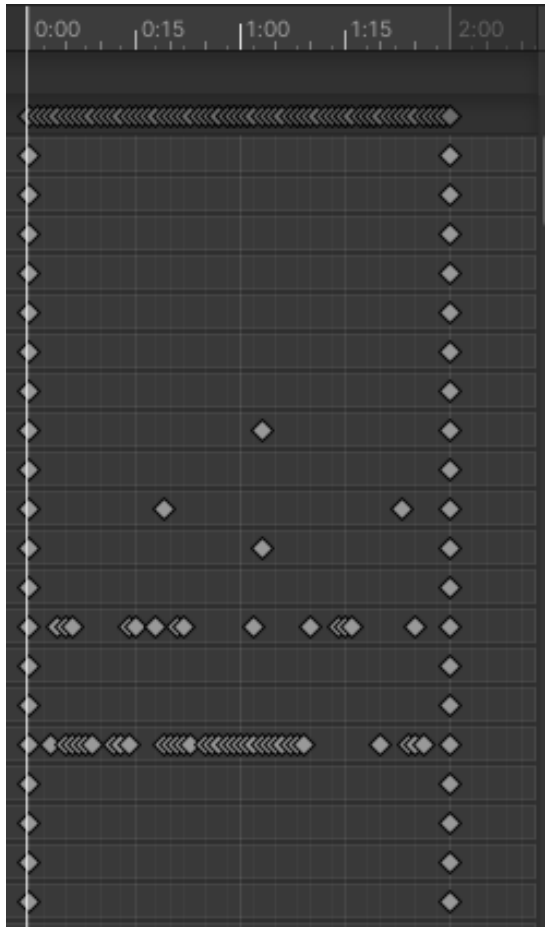
每帧都要更新Shader参数，带宽占用过大

Animation Clip

Bones

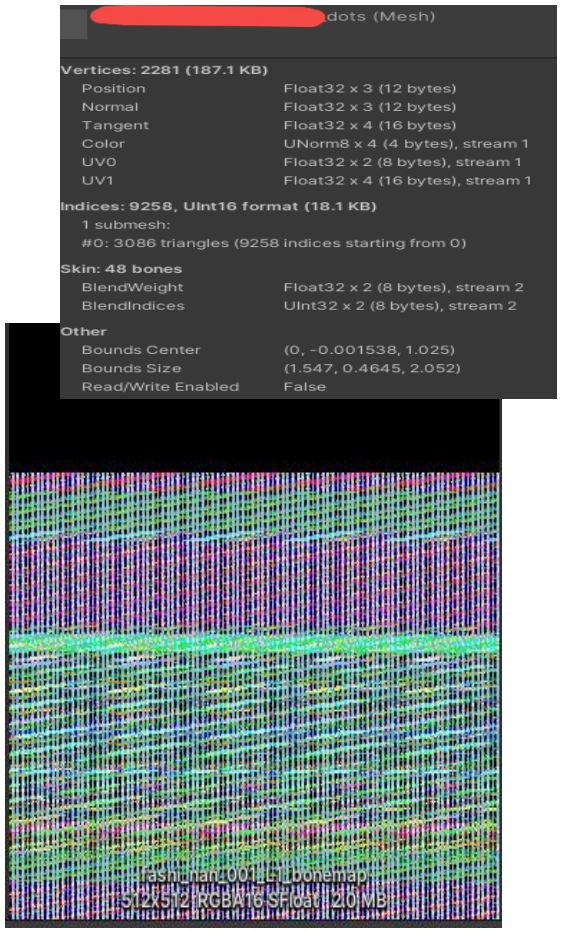
Bindpose

BoneWeight



Bake工具

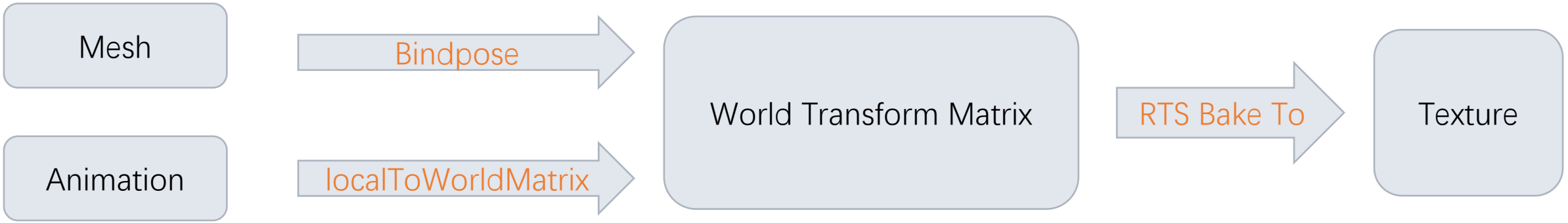
BoneMap和Mesh



UV2

BoneMap

骨骼动画Bake


Mesh
Animation
Bindpose
localToWorldMatrix
World Transform Matrix
RTS Bake To
Texture


Mesh
BoneWeight
BoneWeight
Mesh UV2

GPUSkin Shader实现

```
//位置&法线
struct VertexAni
{
    float3 vertex;
    float3 normal;
};

//四元数插值
inline float4 Slerp(float4 p0,float4 p1,float t)
{
    ......
}
```

演示部分代码，不适用于生产环境

```
//计算顶点最终的位置&法线
inline VertexAnimated AnimateVextex(
.....
    //先加权，后融合
    //当前动画帧采样
    float4x4 mat0 = GetMat**(aniMap, aniMapTexelSize, round((frameIndex + uv.y) * BoneCount),scale);
    float4x4 mat1 = GetMat**(aniMap, aniMapTexelSize, round((frameIndex + uv.w) * BoneCount),scale);
    VertexAni vertAni1;
    vertAni1.vertex = mul(mat0, positionOS).xyz * uv.x + mul(mat1, positionOS).xyz * uv.z;
    vertAni1.normal = mul(mat0, normalOS).xyz * uv.x+ mul(mat1, normalOS).xyz * uv.z;
    //前一动画帧采用
    float4x4 mat2 = GetMat**(aniMap, aniMapTexelSize, round((frameIndex2 + uv.y) * BoneCount),scale);
    float4x4 mat3 = GetMat**(aniMap, aniMapTexelSize, round((frameIndex2 + uv.w) * BoneCount),scale);
    VertexAni vertAni2;
    vertAni2.vertex = mul(mat2, positionOS).xyz * uv.x + mul(mat3, positionOS).xyz * uv.z;
    vertAni2.normal = mul(mat2, normalOS).xyz * uv.x + mul(mat3, normalOS).xyz * uv.z;
    //融合
    float t = pow(lastState.w,percent*100);
    vertAni2.vertex = lerp(vertAni1.vertex,vertAni2.vertex,t);
    vertAni2.normal = lerp(vertAni1.normal,vertAni2.normal,t);
.....
```

常规的GPUSkin。注意融合插值的计算，可参考Unity引擎源码

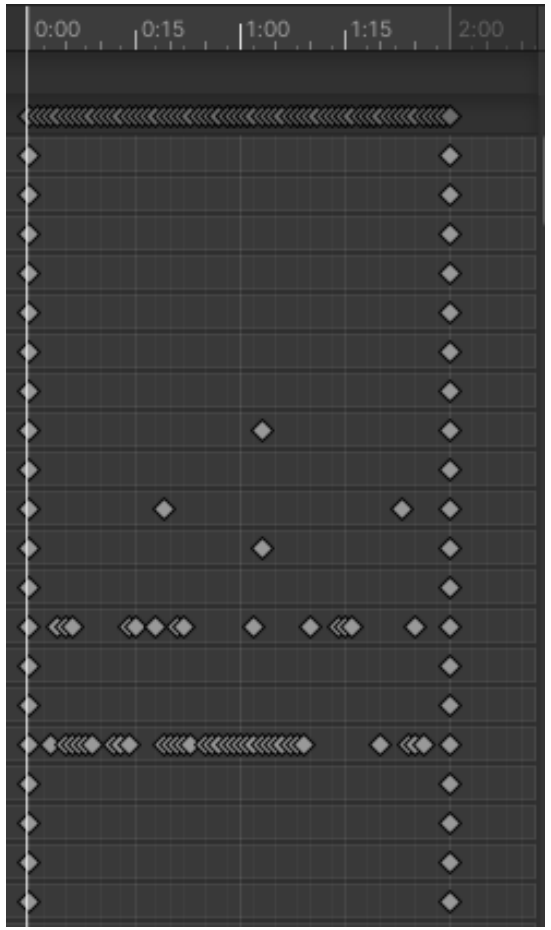
Vertex Animation

Animation Clip

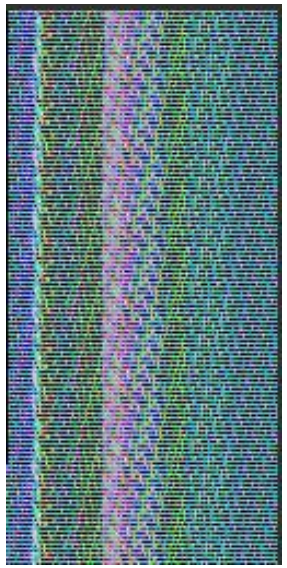
Animation Map 和 Mesh

Vertex

Normal



Bake工具



Vertex

Normal

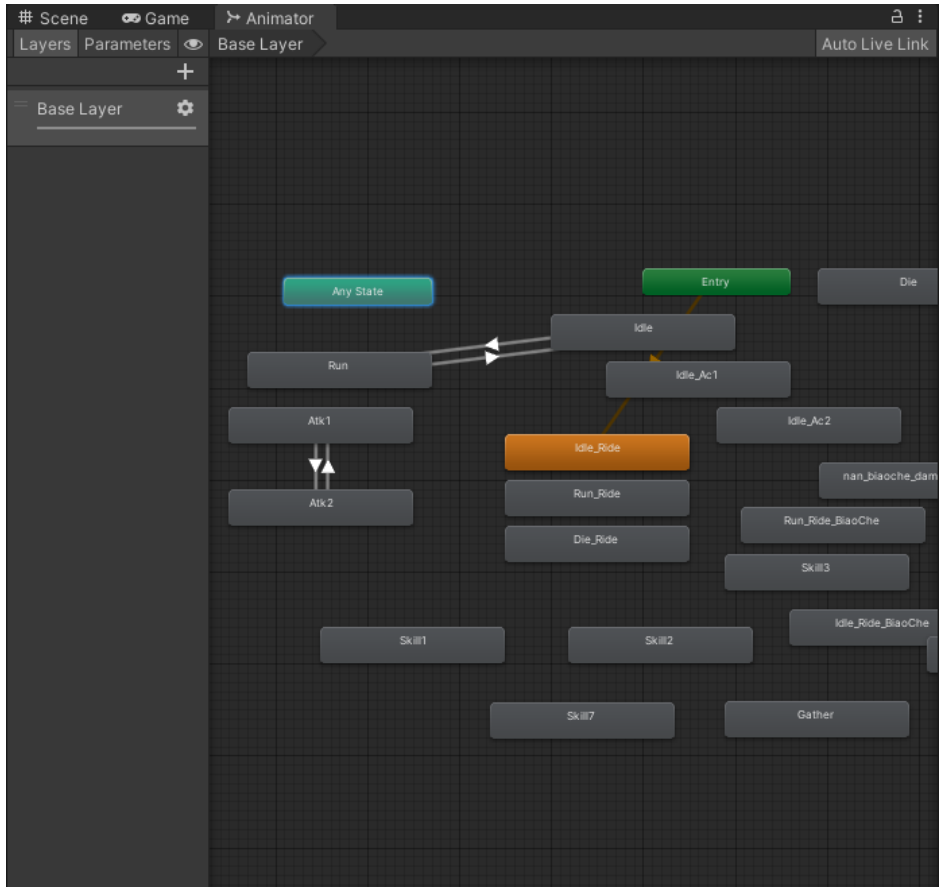
Vertex Animation Shader

```
VertexAnimated VertexAnimate(
sampler2D aniMap, float4 ...,
float ...,
float4 rot, float4 aniState,int vid)
{
...
float2 posUV = ....;
float2 normalUV = ....;
float4 vpos = tex2Dlod(aniMap, float4(posUV, 0,0));
...
float4x4 mat = float4x4(....
1,0,
0,0,0,1);
vpos = mul(mat,vpos);
float4 vnormal = tex2Dlod(aniMap, (float4(normalUV, 0,0)));
VertexAnimated vadata;
vadata.vertex = vpos;
vadata.normal =  mul(mat,vnormal);
return vadata;
}
```

演示部分代码，不适用于生产环境

三、ECS动画驱动

动画驱动方式转换



动作名	起始帧	总帧数	是否循环
string		Vector3	
Idle	0	60	1
Run	60	30	1
Die	90	20	0
Hit	110	40	0

Binary Data

Blob

IComponentData

动画控制

AniSystem

Component
_AniState("AniState",Vector)

初始化 → x=骨骼数

动作切换 → y=当前动画起始帧

动作切换 → z=当前动画总帧数

帧数自增 → w=当前动画帧

坐骑骨骼动画

叠加挂点变换

身体骨骼动画

叠加挂点变换

部位位移旋转

Material Override

ECS中设置属性：

```
[MaterialProperty("_AniState")]
public struct AniState : IComponentData
{
        public float4 Value;
}
```

Shader中声明：

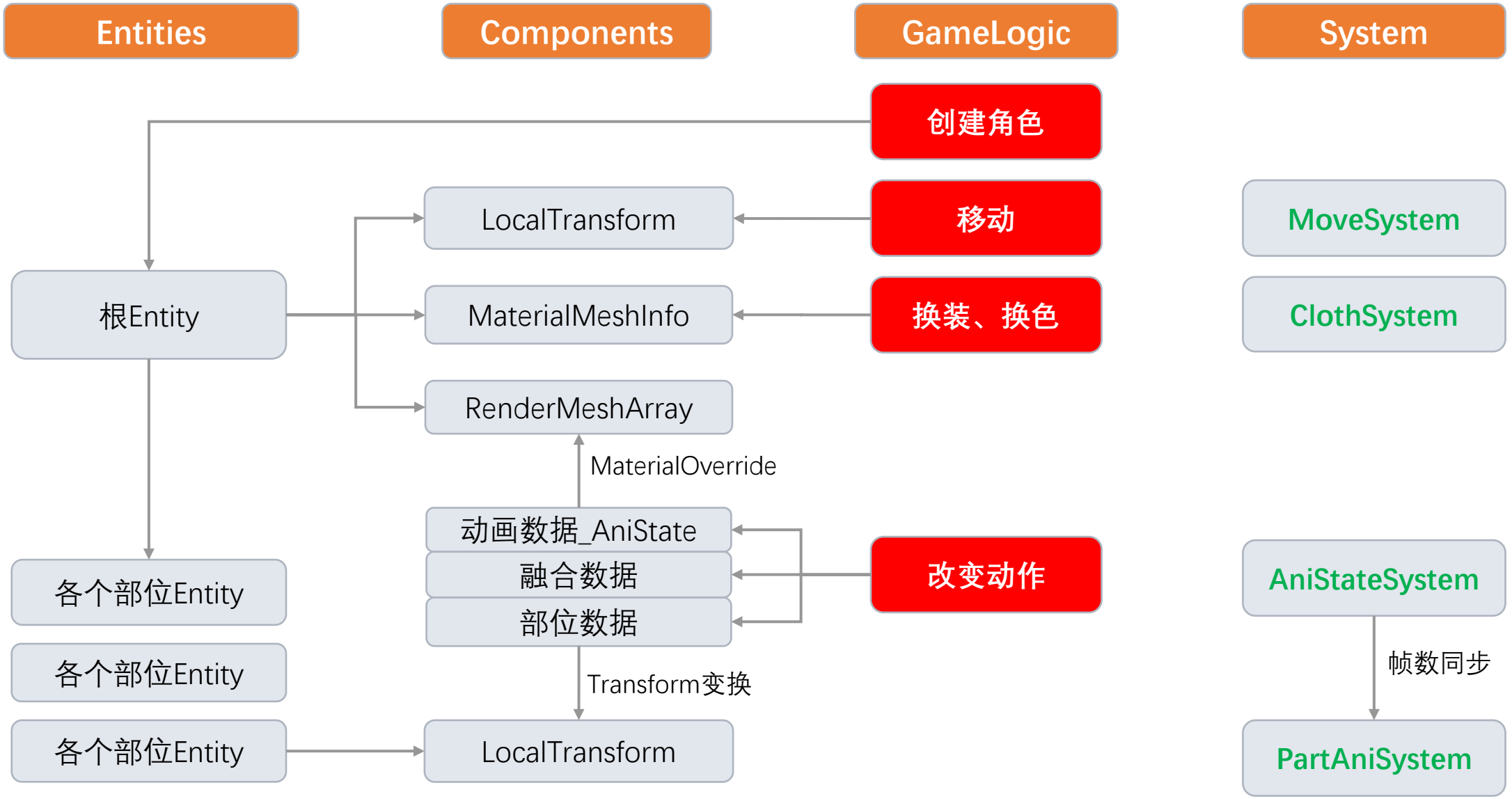
```
#ifdef UNITY_DOTS_INSTANCING_ENDABLED
        UNITY_DOTS_INSTANCING_START(MaterialPropertyMetadata)
        UNITY_DOTS_INSTANCED_PROP(float4,_AniData);
        UNITY_DOTS_INSTANCING_END(MaterialPropertyMetadata)
#endif
```

Shader获取：

```
#ifdef UNITY_DOTS_INSTANCING_ENDABLED
        float4 aniData =
        UNITY_ACCESS_DOTS_INSTANCED_PROP(float4,_AniData);
#else
    float4 aniData = _AniData;
#endif
```

演示部分代码，不适用于生产环境

ECS动画驱动


Entities
Components
GameLogic
System
创建角色
移动
换装、换色
改变动作
LocalTransform
MaterialMeshInfo
RenderMeshArray
MaterialOverride
动画数据_AniState
融合数据
部位数据
Transform变换
根Entity
各个部位Entity
各个部位Entity
各个部位Entity
LocalTransform
MoveSystem
ClothSystem
AniStateSystem
帧数同步
PartAniSystem

四、换装实现

更换部位流程

美术资产加载和转Entity的方式

GIANT
GIANT NETWORK

Baker in SubScene

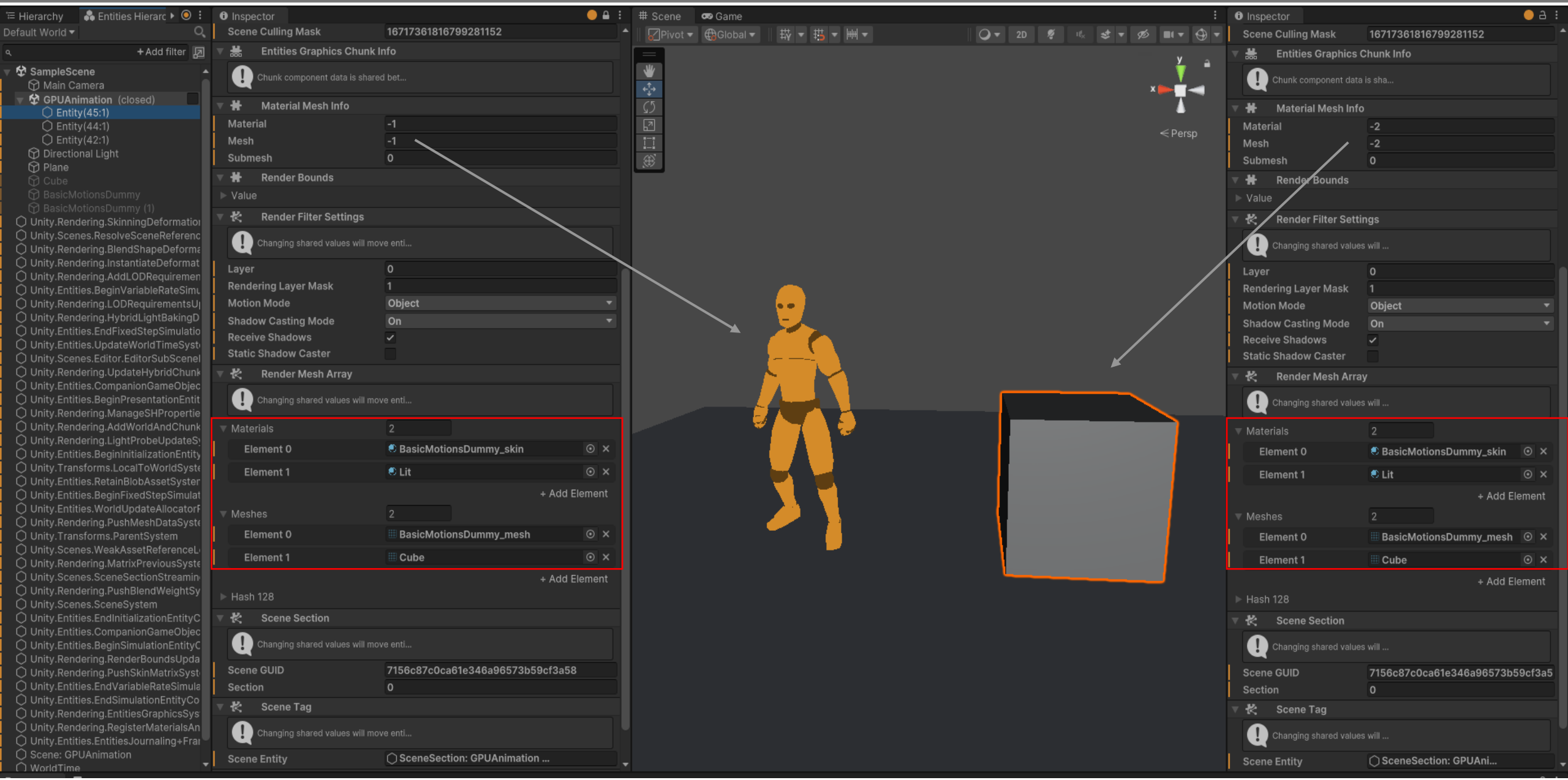
```
using Unity.Entities;
using Unity.Entities.Content;
using UnityEngine;

public class MeshRefSample : MonoBehaviour
{
    public WeakObjectReference<Mesh> mesh;
    class MeshRefSampleBaker : Baker<MeshRefSample>
    {
        public override void Bake(MeshRefSample authoring)
        {
            var entity = GetEntity(TransformUsageFlags.Dynamic);
            AddComponent(entity, new MeshComponentData { mesh = authoring.mesh });
        }
    }
}

public struct MeshComponentData : IComponentData
{
    public WeakObjectReference<Mesh> mesh;
}
```

动态转化

动态替换Mesh和Material



Ricky

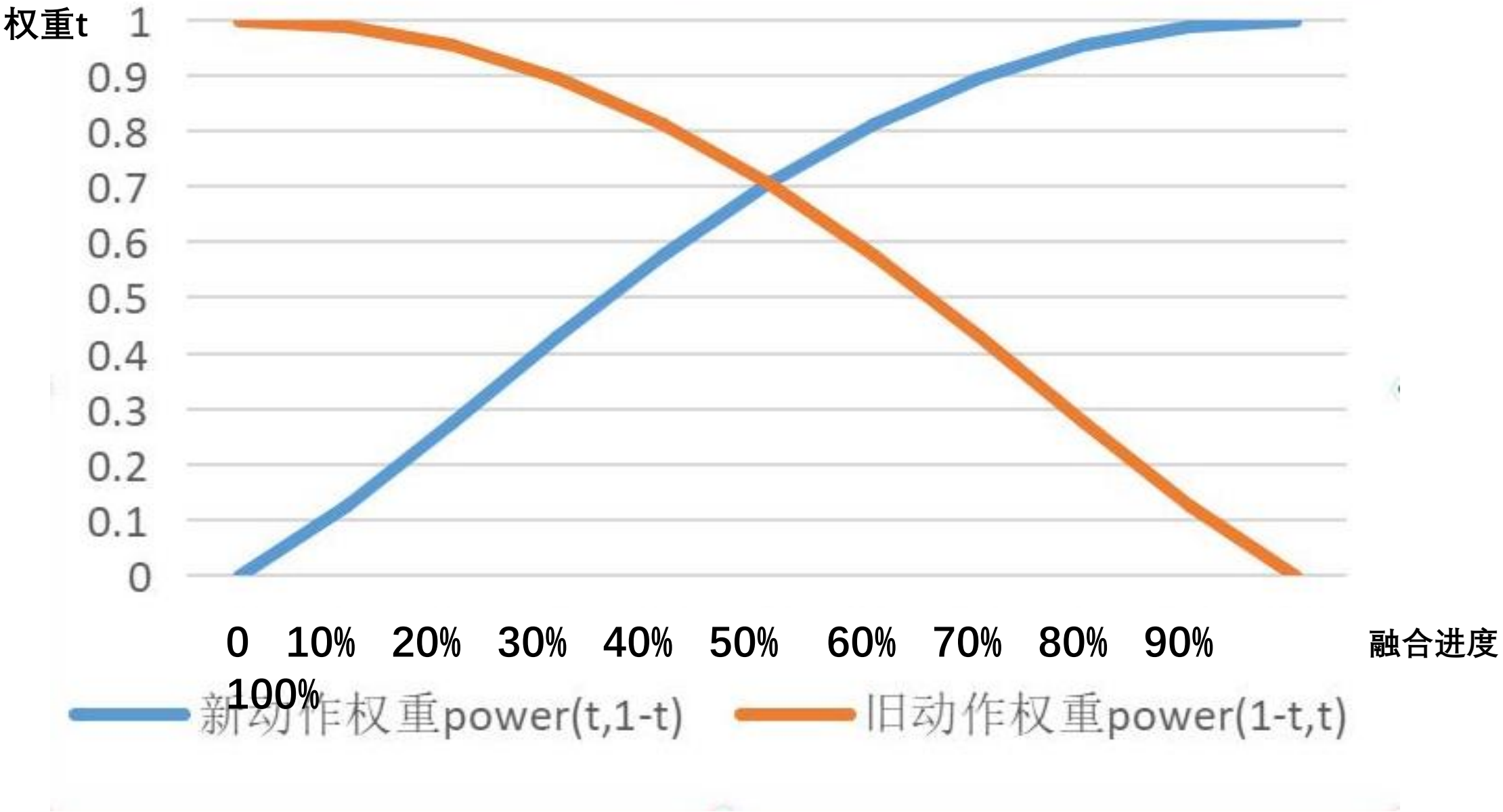
五、细节优化

- 骨骼数量设定为2
- Mesh只用一套UV，第二套用来存顶点依赖的骨骼ID&权重
- 各个部位挂点放到根节点下K帧
- 修改任一动作都需要重新Bake

- 动画融合，动画暂停，动画截断，动画事件音效，
- 简模与全模切换，高模与低模共用动画，
- 材质半透明，材质特效，
- 模型缩放，部位缩放，

……

融合平滑算法


权重t
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
融合进度
新动作权重power(t,1-t)
旧动作权重power(1-t,t)

谢谢

问题交流