**【Unity3d FootIK】写一个最简单的IK（1）**

[](https://www.zhihu.com/people/li-heng-76-59)

[**李恒**](https://www.zhihu.com/people/li-heng-76-59)

只辩经，不对线。

34 人赞同了该文章

​

目录

收起

前言：

一、 预备设置

1. 设置Humanoid Avatar

2. 创建AnimatorController资源，设置IKPass

3. 创建IK设置脚本，声明【OnAnimatorIK方法】

二、 创建FootIK脚本

1. 创建脚本，声明变量

2. 在FixedUpdate中获取骨骼信息，解算IK位置。

3. AdjustFeetTarget 方法

4. FootPositionSolver 方法

5. OnAnimatorIK

6. MovePelvisHeight

7. MoveFeetToIkPoint

三、动画曲线设置

四、实践

五、原理与算法

1. 时序

2. Lerp、Speed与yVar

3. 揭开Unity IK 的本质

小结

**前言：**

踩了一万个坑，总算是有了一个在Unity中实现FootIK的基本方案了，做得这么费劲其实是我太菜了呜呜呜。

顺手推一下基本的CCD，TwoBoneIK的算法详解[[1]](https://zhuanlan.zhihu.com/p/529556406#ref_1)，真是宝藏。

本文基于Unity的Mecanim动画系统内部IK，实现一个轻量级FootIK。

FootIK系列目录：

[【Unity3d FootIK】写一个最简单的IK（1）](https://zhuanlan.zhihu.com/p/529556406)

[【Unity3d FootIK】痛斥AnimationRigging（2）](https://zhuanlan.zhihu.com/p/532376690)

[【Unity3d FootIK】With Final IK（3）](https://zhuanlan.zhihu.com/p/529786996)

[【Unity3d FootIK】Better Hybrid IK（4）](https://zhuanlan.zhihu.com/p/535264371)

[【Unity3d FootIK】FinalIK's Grounder Algorithm（5）](https://zhuanlan.zhihu.com/p/539287515)

[【Unity3d FootIK】Ju Foot Placement Algorithm（6）](https://zhuanlan.zhihu.com/p/541887646)

**一、 预备设置**

想要使用Unity内部的IK系统，需要进行以下设置。

**1. 设置Humanoid Avatar**

在某个fbx的Inspector面板，设置其【AnimationType】为【Humanoid】，【Avatar Definition】为【Create From This Model】

如果有复数个模型，骨骼一样，只需要一个Avatar即可，其他模型的设置就是“Copy From Other Avatar”，这里只是从0开始设置而已。

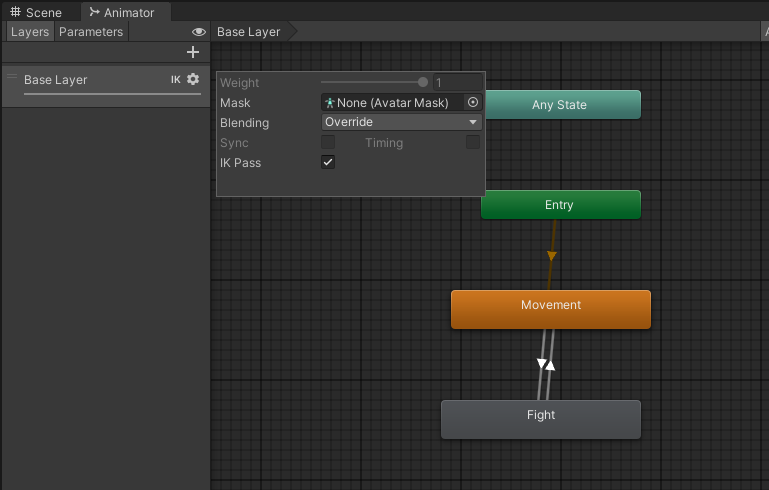
设置如图，模型来自Mixamo[[2]](https://zhuanlan.zhihu.com/p/529556406#ref_2)

如果某些模型出现Avatar匹配错误，请点击Config进行调整

自行绑定关键节点

**2. 创建AnimatorController资源，设置IKPass**

在具体的AnimatorController视图中，在想要进行IK的层级设置中，勾选【IK Pass】，这样，在这个层中的动画渲染时，会启用IK进行解算。



**3. 创建IK设置脚本，声明【OnAnimatorIK方法】**

创建一个Mono脚本，在类中声明OnAnimatorIK方法，Like This

**public** **class** **EmptyIK** : MonoBehaviour

{

**private** **void** OnAnimatorIK(**int** layerIndex)

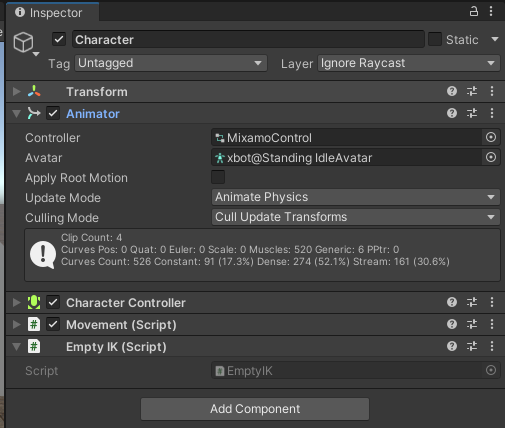
{

Debug.Log("IKing....");

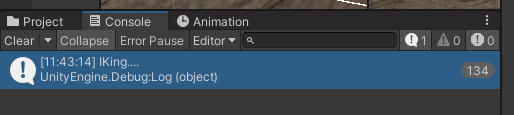
}

}

将其挂载在角色上，面板类似这样：



点击运行，如果设置无误，你应该会看见打印信息：



成功打印的话，说明基本设置完毕

**二、 创建FootIK脚本**

**1. 创建脚本，声明变量**

[RequireComponent(typeof(Animator))]

**public** **class** **IKSetting** : MonoBehaviour

{

**public** **bool** enableFeetIk = **true**; *//是否开启ik*

[Range(0, 2)] [SerializeField] **private** **float** heightFromGroundRaycast = 1.2f; *//从地面向上的cast距离*

[Range(0, 2)] [SerializeField] **private** **float** raycastDownDistance = 1.5f; *//向下cast 距离*

[SerializeField] **private** LayerMask environmentLayer; *//检测layer*

[SerializeField] **private** **float** pelvisOffset = 0f; *//盆骨offset*

[Range(0, 1)] [SerializeField] **private** **float** pelvisUpAndDownSpeed = 0.28f; *//盆骨赋值速度*

[Range(0, 1)] [SerializeField] **private** **float** feetToIkPositionSpeed = 0.5f; *//足IK赋值速度*

**public** **string** leftFootAnimCurveName = "LeftFoot"; *//权重曲线*

**public** **string** rightFootAnimCurveName = "RightFoot"; *//权重曲线*

[Range(0, 100)] **public** **float** leftFootAngleOffset; *//旋转偏移*

[Range(0, 100)] **public** **float** rightFootAngleOffset; *//旋转偏移*

**public** **bool** useIkFeature = **false**; *//是否使用IK旋转*

**public** **bool** showSolverDebug = **true**;*// Debug绘制*

**private** Animator m\_animator; *//动画机*

**private** Vector3 \_rightFootPosition, \_leftFootPosition; *//足部骨骼posiition*

**private** Vector3 \_rightFootIkPosition, \_leftFootIkPosition; *//足部IK position*

**private** Quaternion \_leftFootIkRotation, \_rightFootIkRotation; *//足部IK rotation*

**private** **float** \_lastPelvisPositionY, \_lastRightFootPositionY, \_lastLeftFootPositionY; *//上帧信息，用于lerp动画*

**private** **void** Start()

{

m\_animator = GetComponent<Animator>();

}

}

百八十个变量，看不懂没关系，这些都是为了后面的算法逻辑而声明的，接下来会一步步解释这些变量的用途。

**2. 在FixedUpdate中获取骨骼信息，解算IK位置。**

**private** **void** FixedUpdate()

{

**if** (!enableFeetIk) **return**;

**if** (!m\_animator) **return**;

AdjustFeetTarget(**ref** \_rightFootPosition, HumanBodyBones.RightFoot); *//设置 右足骨骼射线的pos*

AdjustFeetTarget(**ref** \_leftFootPosition, HumanBodyBones.LeftFoot); *// 设置 左足骨骼射线的pos*

*//IK 射线解算*

FootPositionSolver(\_rightFootPosition, **ref** \_rightFootIkPosition, **ref** \_rightFootIkRotation, rightFootAngleOffset);

FootPositionSolver(\_leftFootPosition, **ref** \_leftFootIkPosition, **ref** \_leftFootIkRotation, leftFootAngleOffset);

}

**3. AdjustFeetTarget 方法**

【AdjustFeetTarget】获取了左脚/右脚的【Transform】的pos，这个Transform就是之前在Avatar设置界面中绑定的骨骼。

然后对这个pos加上一个上方向的distance，以这个点为射线检测的起点（防止卡模型）。

**void** AdjustFeetTarget(**ref** Vector3 feetPosition, HumanBodyBones foot)

{

feetPosition = m\_animator.GetBoneTransform(foot).position; *//获取人形足部的transform position*

feetPosition.y = transform.position.y + heightFromGroundRaycast; *//y的值会加上【向上检测的距离】，主要是防止卡模型。*

}

**4. FootPositionSolver 方法**

这里使用Raycast进行简单的射线检测。

当存在hit时，记录其hitPoint的y值，计算Vector3.up与hitPoint法线的夹角。

**void** FootPositionSolver(Vector3 fromSkyPosition, **ref** Vector3 feetIkPosition, **ref** Quaternion feetIkRotation, **float** angleOffset)

{

**if** (showSolverDebug)

Debug.DrawLine(fromSkyPosition, fromSkyPosition + Vector3.down \* (raycastDownDistance + heightFromGroundRaycast), Color.green);

**if** (Physics.Raycast(fromSkyPosition, Vector3.down, **out** **var** feetOutHit, raycastDownDistance + heightFromGroundRaycast, environmentLayer))

{

feetIkPosition = fromSkyPosition; *//保存x,z值。*

feetIkPosition.y = feetOutHit.point.y + pelvisOffset; *//hit pos 的 Y 赋值*

feetIkRotation = Quaternion.FromToRotation(Vector3.up, feetOutHit.normal) \* transform.rotation; *//计算法向偏移*

feetIkRotation = Quaternion.AngleAxis(angleOffset, Vector3.up) \* feetIkRotation; *//计算额外的偏移*

**return**;

}

feetIkPosition = Vector3.zero; *//没有hit，归零*

}

**5. OnAnimatorIK**

在此方法中，先进行骨盆偏移，然后设置权重（根据动画曲线），然后执行IK Goal 坐标赋值。

**private** **void** OnAnimatorIK(**int** layerIndex)

{

**if** (!enableFeetIk) **return**;

**if** (!m\_animator) **return**;

MovePelvisHeight(); *//骨盆偏移*

m\_animator.SetIKPositionWeight(AvatarIKGoal.RightFoot, m\_animator.GetFloat(rightFootAnimCurveName)); *//设置pos 权重*

**if** (useIkFeature)

{

m\_animator.SetIKRotationWeight(AvatarIKGoal.RightFoot, m\_animator.GetFloat(rightFootAnimCurveName)); *//设置 rot 权重*

}

MoveFeetToIkPoint(AvatarIKGoal.RightFoot, \_rightFootIkPosition, \_rightFootIkRotation, **ref** \_lastRightFootPositionY); *//设置ik goal坐标*

m\_animator.SetIKPositionWeight(AvatarIKGoal.LeftFoot, m\_animator.GetFloat(leftFootAnimCurveName));

**if** (useIkFeature)

{

m\_animator.SetIKRotationWeight(AvatarIKGoal.LeftFoot, m\_animator.GetFloat(leftFootAnimCurveName));

}

MoveFeetToIkPoint(AvatarIKGoal.LeftFoot, \_leftFootIkPosition, \_leftFootIkRotation, **ref** \_lastLeftFootPositionY);

}

**6. MovePelvisHeight**

这个方法很重要，他是保证IK能达到的前提。如果不进行偏移，则可能会出现这种情况。[[3]](https://zhuanlan.zhihu.com/p/529556406#ref_3)

**void** MovePelvisHeight() *//调整pelvis，保证IK 能达到（比如左右脚高度差那种）*

{

**if** (\_rightFootIkPosition == Vector3.zero || \_leftFootIkPosition == Vector3.zero || \_lastPelvisPositionY == 0f)

{

\_lastPelvisPositionY = m\_animator.bodyPosition.y;

**return**;

}

**float** lOffsetPosition = \_leftFootIkPosition.y - transform.position.y; *//左脚ik pos与当前transform的高度差*

**float** rOffsetPosition = \_rightFootIkPosition.y - transform.position.y; *//右脚ik pos 与当前transform的高度差*

*//选择较小值（在以vector3.up为正轴的情况下）*

*//如果是正值，则向上偏移距离较小的。*

*//如果是负值，则向下偏移距离较大的。*

**float** totalOffset = (lOffsetPosition < rOffsetPosition) ? lOffsetPosition : rOffsetPosition;

Vector3 newPelvisPosition = m\_animator.bodyPosition + Vector3.up \* totalOffset; *//新的骨盆位置计算： 原位置+ up方向 \* offset。*

newPelvisPosition.y = Mathf.Lerp(\_lastPelvisPositionY, newPelvisPosition.y, pelvisUpAndDownSpeed); *//插值动画*

m\_animator.bodyPosition = newPelvisPosition; *//赋值*

\_lastPelvisPositionY = m\_animator.bodyPosition.y; *//记录信息*

}

**7. MoveFeetToIkPoint**

在此方法中，获取在FixedUpdate中计算出的位置、旋转信息，进行二次处理后赋值给IK Goal。

**void** MoveFeetToIkPoint(AvatarIKGoal foot, Vector3 positionIkHolder, Quaternion rotationIkHolder, **ref** **float** lastFootPositionY)

{

Vector3 targetIkPosition = m\_animator.GetIKPosition(foot); *//获取animator IK Goal 的 原本 pos*

**if** (positionIkHolder != Vector3.zero) *//如果新的IK pos 不为 0*

{

targetIkPosition = transform.InverseTransformPoint(targetIkPosition); *//把原本的ik goal 的pos转到本地坐标系*

positionIkHolder = transform.InverseTransformPoint(positionIkHolder); *//把现在的ik goal 的pos转到本地坐标系*

**float** yVar = Mathf.Lerp(lastFootPositionY, positionIkHolder.y, feetToIkPositionSpeed); *//进行插值*

targetIkPosition.y += yVar;

lastFootPositionY = yVar;

targetIkPosition = transform.TransformPoint(targetIkPosition); *//把新的ik goal pos转到世界坐标系*

m\_animator.SetIKRotation(foot, rotationIkHolder); *//旋转赋予*

}

m\_animator.SetIKPosition(foot, targetIkPosition); *//位置赋予*

}

**三、动画曲线设置**

还记得在代码中的这几句话吗？

**public** **string** leftFootAnimCurveName = "LeftFoot"; *//权重曲线*

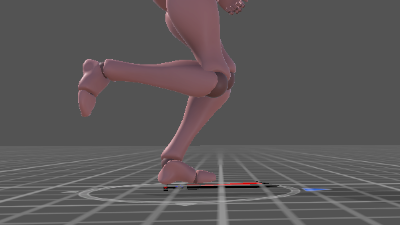
**public** **string** rightFootAnimCurveName = "RightFoot"; *//权重曲线*

...

m\_animator.SetIKPositionWeight(AvatarIKGoal.RightFoot, m\_animator.GetFloat(rightFootAnimCurveName)); *//设置pos 权重*

m\_animator.SetIKPositionWeight(AvatarIKGoal.LeftFoot, m\_animator.GetFloat(leftFootAnimCurveName));

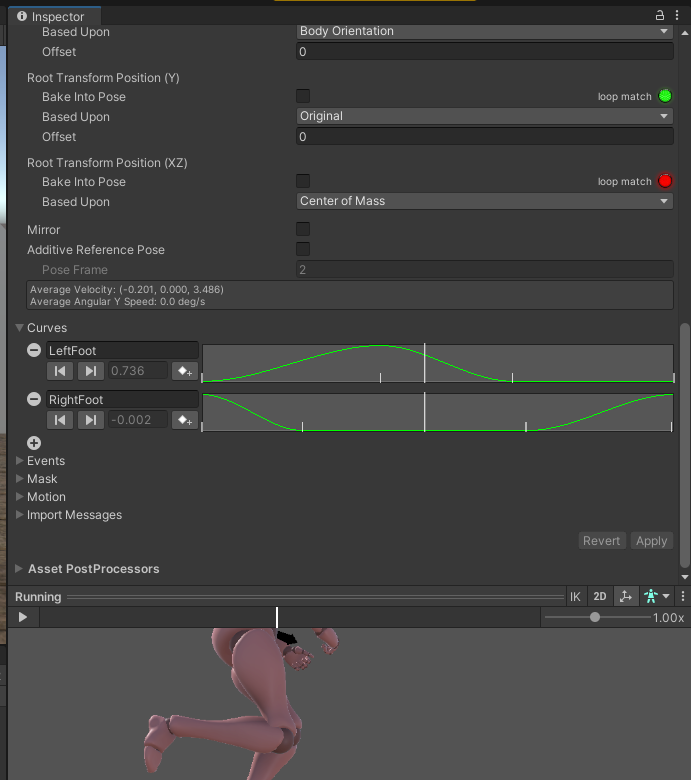
比如一段跑步的动画，在整个动画周期，是存在脚抬起来的时候。



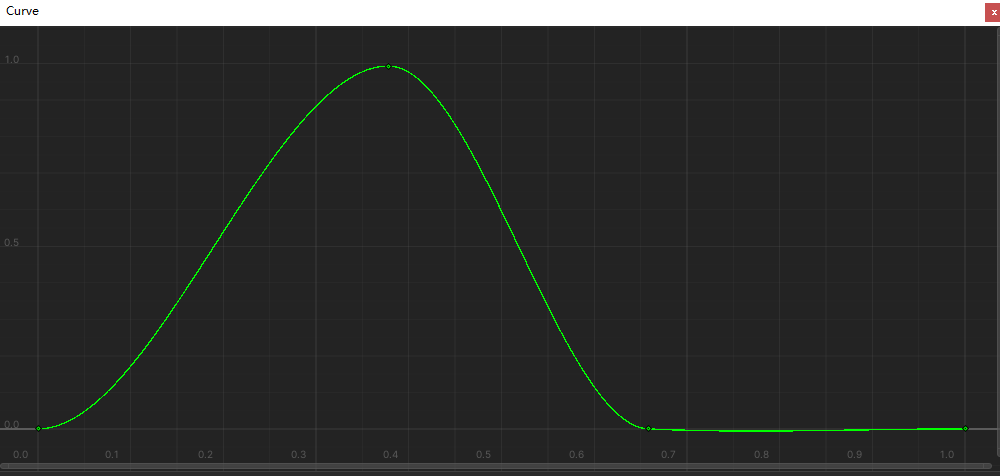
这个时候，IK Goal的权重应该为0，不然的话这个抬起的脚也会被牢牢钉在地面上。

那么不同时候的权重该如何设置？这就是【动画曲线】的设置。

在fbx的Inspector设置中，选择【Animation】，点开【Curves】，点击【+】添加动画曲线。



一个曲线大概长这样：

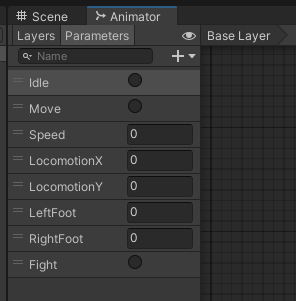


Y轴的最大值为1，最小值为0.

X轴则是这一段动画片段的【normalizedTime】，0代表第一帧，1代表最后一帧。

我们可以在下面的预览窗口，拖动白色滑块，逐帧播放、查看当前时间的运动情况（哪只脚在空中，哪只脚在地面），然后在对应的曲线X值处，设置合适的Y值（也就是Weight值）。

在创建好动画曲线之后，且正确命名后（在这里是“LeftFoot”与“RightFoot”），在【Animator】面板【Parameters】中，添加两个Float参数“LeftFoot”与“RightFoot”。



很明显，参数名必须和动画曲线名一致

运行游戏试试，当Animator播放含有动画曲线的clip时，我们应该能看见参数值发生变化：

0.261与0.3652的值，来自于动画曲线的读取

至此，动画曲线设置完毕。

**四、实践**

在整理好上述代码后，把脚本挂载到角色GameObject上，大致会是这样：

选择【EnvironmentLayer】的目标层级，这里选择【Default】。

然后勾选【Use IK Feature】，激活后会对IK的旋转进行赋值。

然后运行起来，找个阶梯试试（当然，阶梯的GameObject的Layer是“Default”）

在这里的“Idle”动画Clip中，我设置了“LeftFoot”“RightFoot”曲线恒为1

好耶！基本的IK设置就完成啦！看起来还行！

**五、原理与算法**

**1. 时序**

知其然不知其所以然怎么行呢，所以这节我们讲原理与算法。

本文代码实际参考[[4]](https://zhuanlan.zhihu.com/p/529556406#ref_4)，其代码来自视频参考[[5]](https://zhuanlan.zhihu.com/p/529556406#ref_5)。

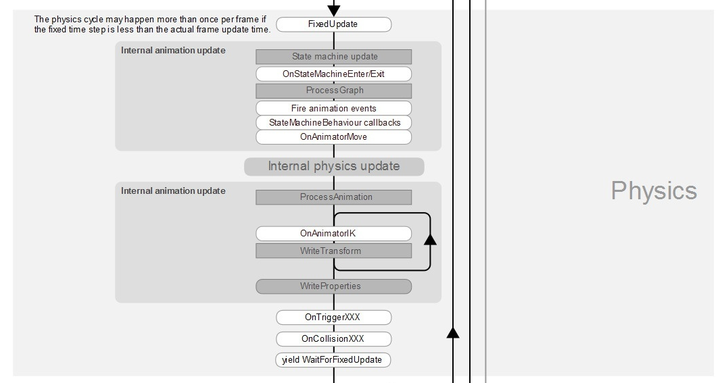
整体代码来看，它在【FixedUpdate】进行骨骼位置的坐标读取，然后进行IK Solver解算。

算法就是简单地在空中向下打射线，检查HitPoint的法线信息与位置，从而计算IK Goal的旋转与位置。

很简单，容易理解。

然后就是在【OnAnimatorIK】中，把权重、数据写入到IK Goal中，从而影响骨骼位置。

这两个方法的执行顺序如图：



**2. Lerp、Speed与yVar**

在变量声明中，有这些参数：

[Range(0, 1)] [SerializeField] **private** **float** pelvisUpAndDownSpeed = 0.28f;

[Range(0, 1)] [SerializeField] **private** **float** feetToIkPositionSpeed = 0.5f;

**private** **float** \_lastPelvisPositionY, \_lastRightFootPositionY, \_lastLeftFootPositionY;

在使用时，他们是这种写法：

----In MovePelvisHeight----

Vector3 newPelvisPosition = m\_animator.bodyPosition + Vector3.up \* totalOffset;

newPelvisPosition.y = Mathf.Lerp(\_lastPelvisPositionY, newPelvisPosition.y, pelvisUpAndDownSpeed);

m\_animator.bodyPosition = newPelvisPosition;

\_lastPelvisPositionY = m\_animator.bodyPosition.y;

...

---In MoveFeetToIkPoint----

**float** yVar = Mathf.Lerp(lastFootPositionY, positionIkHolder.y, feetToIkPositionSpeed);

targetIkPosition.y += yVar;

lastFootPositionY = yVar;

骨盆偏移还比较好理解，使用speed控制插值速度，每一帧只赋值Lerp过后的值，把Pelvis的Y值慢慢地向目标坐标靠拢。

但是在【MoveFeetToIkPoint】中，最开始我是完全没看懂的，为什么插值后使用【+=】进行增量赋予啊？！难道不是【=】赋值吗？

这种写法，使用增量赋值怎么想都不会收敛吧！！他怎么就Work了？

尝试把【+=】改为【=】，把Speed设为1试试？我们会得到：

脚踝的IK Goal贴在地面上，脚步穿模

WTF？为什么一个【+=】增量写法能控制与地面的Offset，而且还是正常运行的？！

**3. 揭开Unity IK 的本质**

我们再认真的详细看看【MoveFeetToIkPoint】方法到底做了什么。

**void** MoveFeetToIkPoint(AvatarIKGoal foot, Vector3 positionIkHolder, Quaternion rotationIkHolder, **ref** **float** lastFootPositionY)

{

Vector3 targetIkPosition = m\_animator.GetIKPosition(foot); *//获取animator IK Goal 的 原本 pos*

*//入参： positionIkHolder 就是计算出的Hitpoint位置*

*//入参： rotationIkHolder 就是计算出的Hitpoint法线信息得到的旋转量*

**if** (positionIkHolder != Vector3.zero)

{

targetIkPosition = transform.InverseTransformPoint(targetIkPosition); *//把原本的IK Goal 的pos转到本地坐标系*

positionIkHolder = transform.InverseTransformPoint(positionIkHolder); *//把期望的IK Goal 的pos转到本地坐标系*

**float** yVar = Mathf.Lerp(lastFootPositionY, positionIkHolder.y, feetToIkPositionSpeed); *//进行插值*

targetIkPosition.y += yVar;

lastFootPositionY = yVar;

targetIkPosition = transform.TransformPoint(targetIkPosition); *//把新的IK goal pos转到世界坐标系*

m\_animator.SetIKRotation(foot, rotationIkHolder); *//旋转赋予*

}

m\_animator.SetIKPosition(foot, targetIkPosition); *//位置赋予*

}

这个方法中，涉及到了transform的局部、世界坐标变换，与m\_animator的Getter、Setter，他们究竟是怎么起作用的？

在我逐步对这些计算的中间变量进行打印Debug后，我不得不郑重地宣布：

IK Goal会重置！会重置！会重置！

【yVar】才是实际上的地面y值！

【targetIkPosition.y】才是那个脚踝与地面的Offset偏移量！

这个方法实际上，是先获取了Unity IK系统中，在处理了AnimationClip之后，内部的IK Goal的值。而值得注意的是，内部脚部的IK Goal，他的Y值，在这个实体Transform局部坐标下，就是与脚底板（地面）的差值！

脚部骨骼Transform位置

模型Transform位置

也就是说，如图所示，正常状态下【targetIkPosition = transform.InverseTransformPoint(targetIkPosition)】经过本地坐标变换后，Y值就是离地的Y值！

而这句话【float yVar = Mathf.Lerp(lastFootPositionY, positionIkHolder.y, feetToIkPositionSpeed)】

positionIkHolder.y就是进行射线检测后的HitPoint的y值，Lerp之后，yVar 才是那个真正的地面Y值。

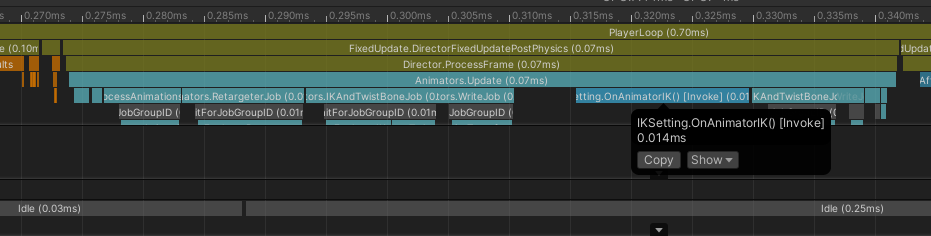
所以：地面Y值（yVar）+地面偏移值（targetIkPosition.y）得到的结果，自然是脚部应该在的Y值，防止穿模。

哦，原来如此，那么为什么【+=】能收敛呢？

那是因为：最开始这个【m\_animator.GetIKPosition(foot)】根本不是上一帧【m\_animator.SetIKPosition(foot, targetIkPosition)】Set后的值！

读者可自行在这两个变量处Debug，看看他们值是否一样。

而打开Profiler，查看Timeline，我们会发现：



AnimationUpdate有泾渭分明的两段计算

在进行Animation计算时，流程首先是：

【ProcessingAnimation】->【Retargeter】->【IKAndTwistBone】->【Write】

然后才是咱们的【OnAniamtorIK】->【IKAndTwistBone】->【Write】...

为什么两帧的【IKPosition】不一样，看图说话，我猜是在Unity Animatior 前一个【IKAndTwistBone】环节，根据实际的AnimationClip计算出了原本的IK Position。

然后才是我们的【OnAniamtorIK】进行IK Position赋值后， 再计算一次【IKAndTwistBone】与【Write】。

这样，最后渲染在屏幕上时，看起来我们的IK生效了，坐标也是对的。

而实际上在下一帧后，就如上所述，IK Position会重置！重置！重置！

重置成什么值？重置成原本的地面偏移值！

所以【targetIkPosition.y += yVar】当然收敛啦！targetIkPosition一直都是原本AniamtionClip计算出的的IK Position！完全没有累加上去！

那么问题来了，我们该如何保存上一帧的Y值，来实现Lerp动画呢？

【lastFootPositionY】这不就来了吗？为什么我们要声明这个变量，全都是因为IK Position会重置的锅啊。

同理，之前那个骨盆偏移【\_lastPelvisPositionY】是怎么运行的，故技重施啊。

为什么我想要强调这个信息？因为在扩展包【AnimationRigging】中，它的IK Apply又是另外一种流程！现在这种代码算法是完全不匹配的！

同时，【OnAniamtorIK】是在处理Animation之后再调用，完全可以看做一种“后处理”。那么【Final IK】的是怎么实现的？我猜也是一种在【LateUpdate】中的“后处理”。

同时，这个插件不用动画曲线怎么修改权重的？我猜是根据两帧间的骨骼速度，速度越大，权重越小。

**小结**

把Unity内置的IK给破解了之后，面对这些功能插件的原理终于有了眉目，泪目。

希望接下来有机会的话把【Final IK】和【AnimationRigging】给干翻。