**Unity3D教程：动画融合、动画层、动画混合、附加动画、程序动画、动画重放和取样**

Posted on 2013年05月17日 by U3d / [Unity3D 基础教程](http://www.unitymanual.com/category/manual/unity3d-%e5%9f%ba%e7%a1%80%e6%95%99%e7%a8%8b)/被围观 12 次

**动画脚本 Animation Scripting**

Unity's 动画系统允许你创建一个漂亮的动画蒙皮角色，动画系统支持动画融合,混合,添加动画,步调周期时间同步.动画层，控制动画回放的所有方面(时间,速度,混合权重) 每个顶点有1.2.4个骨骼影响的mesh，基于物理系统的布娃娃系统，另外还有程序动画.为了获得最佳效果推荐您在制作模型和动画绑定前阅读一下 Modeling Optimized Characters 章节。

制作一个动画角色主要包括两个方面; 在世界中移动 和 由此产生的动画。如果你想了解角色移动相关的更多内容, 请参阅 Character Controller page. 实际上角色动画是由Unity's 脚本界面完成的。

你可以下载 example demos 中预设置好的动画角色. 当你学完本页的基础部分你还可以看一看 animation script interface.

**Animation Blending 动画融合**

在现今的游戏中Animation Blending是一项保证游戏动画顺畅过渡的基本的特性.动画师创建的动画例如： walk 循环, run 循环, idle原地空闲动画 或射击动画.在游戏的任何时间点你都有可能从空闲站立转换到走动，反之亦然. 当然你不希望两个不同的动作之间突然跳转, 你需要动画平滑过渡.

而这个问题的解决就依赖动画融合技术. 在Unity中你可以让同一个角色拥有任意数量的动画.所有这些动画融合添加成为一个总的动画.

首先我们来为一个角色添加两个动画原地空闲站立和走动并平滑的使这两个动画过渡. 为了使我们在写脚本时简单些, 首先我们设置动画的 Wrap Mode为 Loop. 然后关闭 Play Automatically来让我们的脚本来独占动画的播放.

我们第一个动画脚本很简单; 我们需要一些方法来探查角色移动的有多快, 然后在走和站立之间淡入淡出. 在这个简单的测试中我们使用 pre-setup input axes.

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | function Update () { |
| 02 |  |
| 03 | **if** (Input.GetAxis(“Vertical”) > 0.2) |
| 04 |  |
| 05 | animation.CrossFade (“walk”); |
| 06 |  |
| 07 | **else** |
| 08 |  |
| 09 | animation.CrossFade (“idle”); |
| 10 |  |
| 11 | } |

下面我们来让这个脚本运行:

1.创建一个js脚本 Assets->Create Other->Javascript.

2.把代码贴进去。

3.把脚本拖拽给角色 character (It needs to be the same GameObject as the animation)

点击Play 按钮, 当你按上下键时角色会走动，松开上下键时角色站立不动、

**动画层Animation Layers**

层是一个非常有用的概念它可以让你将动画片段任意成组并且区分优先顺序。在Unity's动画系统中， 你可以混合任意数量的动画片段。你可以手工分配权重或者直接使用animation.CrossFade(),来自动分配权重。

混合权重混合权重总是在应用前被规格化 normalized。

比如说我们现在有一个 walk cycle 和一个run cycle，权重都是1 (100%).当unity计算最终动画时会规格化权重，这意味着 walk占50% 权重, run cycle占50% 权重。

这在大多数情况下都是不错的, 但当两个动画片段同时运行而其中一个权重明显大于另外一个 。那么你需要手动调整权重值,但如果你使用动画层来解决这个问题过程会容易得多。

**制作动画层的范例Layering Example**

例如现在你有一个射击动画, 一个空闲站立，一个走动循环。 你需要在走和站两个动作间持续的淡入淡出(在玩家走动速度的基础上) 但当玩家射击时我们只想展示射击动画。因而射击动画此时的优先度最高。

为了达到这一目的最简单的方法是在射击时简单的保持 walk 和 idle动画。接下来需要确定shoot animation在一个比idle 和 walk更高的层. 这意味着shoot animation 将首先收到混合权重. walk 和idle 只有在 shoot animation不使用 100% 混合权重的情况下接收权重. 所以当 CrossFading the shoot animation in, 权重将从0开始很短时间内到达 100%. 在开始阶段 walk 和 idle 层将依然可以收到混合权重 但当 shoot animation 完全切入时, 他们就收不到权重了。 这才是我们需要的!

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | function Start () { |
| 02 |  |
| 03 | *// Set all animations to loop 设置所有动画为循环* |
| 04 |  |
| 05 | animation.wrapMode = WrapMode.Loop; |
| 06 |  |
| 07 | *// except shooting 除了射击（不循环）* |
| 08 |  |
| 09 | animation[“shoot”].wrapMode = WrapMode.Once; |
| 10 |  |
| 11 | *//放置idle 和 walk 进低一级别的 layers (默认 layer 总是 0)* |
| 12 |  |
| 13 | *// This will do two things这将作两件事情* |
| 14 |  |
| 15 | *// - 当 calling CrossFade时，由于shoot 和 idle/walk 在不同的layers 中* |
| 16 |  |
| 17 | *// 它们将不会影响互相之间的重放.* |
| 18 |  |
| 19 | *// - 由于 shoot 在高一级的 layer, 当faded in 时shoot动画将替换* |
| 20 |  |
| 21 | *// idle/walk 动画 .* |
| 22 |  |
| 23 | animation[“shoot”].layer = 1; |
| 24 |  |
| 25 | *// Stop animations that are already playing停止已经播放的动画* |
| 26 |  |
| 27 | *//(万一 user 忘记的话，自动disable播放)* |
| 28 |  |
| 29 | animation.Stop(); |
| 30 |  |
| 31 | } |
| 32 |  |
| 33 | function Update () { |
| 34 |  |
| 35 | *// Based on the key that is pressed,基于按下的键* |
| 36 |  |
| 37 | *// play the walk animation or the idle animation播放走，站动画* |
| 38 |  |
| 39 | **if** (Mathf.Abs(Input.GetAxis(“Vertical”)) > 0.1) |
| 40 |  |
| 41 | animation.CrossFade(“walk”); |
| 42 |  |
| 43 | **else** |
| 44 |  |
| 45 | animation.CrossFade(“idle”); |
| 46 |  |
| 47 | *// Shoot射击* |
| 48 |  |
| 49 | **if** (Input.GetButtonDown (“Fire1”)) |
| 50 |  |
| 51 | animation.CrossFade(“shoot”); |
| 52 |  |
| 53 | } |

默认情况下 animation.Play() 和 animation.CrossFade() 将停止或淡出在同一层里面的动画. 这是我们在绝大多数情况下需要的. 在我们shoot, idle, run 范例中, 播放 idle 和 run 将不会影响到 shoot动画 反之亦然 (you can change this behavior(行为) with an optional parameter(任意参数) to animation.CrossFade if you like)。

**动画混合Animation Mixing**

动画混合可以让你缩减你必须为游戏制作的动画片断数量 ，方法是制作只对身体某个部分起作用的动画。这意味着这些动画可以和其他动画合并起来一起使用。

如果你想给一个动画添加 animation mixing transform to an animation by calling AddMixingTransform() on the given AnimationState.

**混合范例Mixing Example**

例如你可能有一个挥手(hand-waving)动画. 你可能需要让一个空闲站立(idle)角色或者一个走动(walking)角色 来挥手. 如果没有动画混合你可能需要制作两个挥手hand-waving动画 : 一个给 idle, 一个给walking. 可是, 如果你将挥手(hand-waving)动画作为一个mixing transform 添加到shoulder transform,挥手动画将只控制肩膀. 身体余下部位不受其影响, 下半身会继续播放 idle 或者 walk 动画. 因而你只需要一个挥手(hand-waving)动画。

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | */// Adds a mixing transform using a Transform variable* |
| 02 |  |
| 03 | **var** shoulder : Transform; |
| 04 |  |
| 05 | animation[“wave\_hand”].AddMixingTransform(shoulder); |
| 06 |  |
| 07 | Another example **using** a path. |
| 08 |  |
| 09 | function Start () { |
| 10 |  |
| 11 | *// Adds a mixing transform using a path instead* |
| 12 |  |
| 13 | **var** mixTransform : Transform = transform.Find(“root/upper\_body/left\_shoulder”); |
| 14 |  |
| 15 | animation[“wave\_h和”].AddMixingTransform(mixTransform); |
| 16 |  |
| 17 | } |

**附加动画 Additive Animations**

附加动画和动画混合可以让你缩减为游戏制作的动画片断的数量,并且对面部动画(facial animations)来说非常重要，我们来看看如果创建一个在跑和转身时身体可以自动倾斜的角色。

你已经制作好了一个 walk 和 run循环, 现在你还要制作一个走动左倾( walk-lean-left), 走动右倾(walk-lean-right), 跑左倾(run-lean-left), 跑右倾(run-lean-right)动画.

这意味着你需要多做4个动画片断! 制作这么多数量的动画会累死人的. 而附加动画(Additive animations) 和混合(Mixing) 可以大大减少这些工作量!

**附加动画范例 Additive Animation Example**

附加动画允许你在顶层覆盖其他所有可能播放的动画的效果( allow you to overlay the effects of animation on top of any others that may be playing). 当你制作一个附加动画时, Unity将计算动画片断里的第一帧 (first frame)和当前帧(current frame)的差异. 然后它将在所有其他播放的动画之上应用这个差异(Then it will apply this difference on top of all other playing animations).

现在你只需要制作一个左倾( lean-left) 和右倾( lean-right)动画. Unity将为此倾斜动画新建一个层并置于walk, idle 或 run循环的层级之上.

下面是代码:

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |  |
| --- | --- |
| 01 | **private** **var** leanLeft : AnimationState; |
| 02 |  |
| 03 | **private** **var** leanRight : AnimationState; |
| 04 |  |
| 05 | function Start () { |
| 06 |  |
| 07 | leanLeft = animation[“leanLeft”]; |
| 08 |  |
| 09 | leanRight = animation[“leanRight”]; |
| 10 |  |
| 11 | *// Put the leaning animation in a separate layer* |
| 12 |  |
| 13 | *// So that other calls to CrossFade won't affect it.* |
| 14 |  |
| 15 | leanLeft.layer = 10; |
| 16 |  |
| 17 | leanRight.layer = 10; |
| 18 |  |
| 19 | *// Set the lean animation to be additive 混合模式为附加* |
| 20 |  |
| 21 | leanLeft.blendMode = AnimationBlendMode.Additive; |
| 22 |  |
| 23 | leanRight.blendMode = AnimationBlendMode.Additive; |
| 24 |  |
| 25 | *// Set the lean animation ClampForever* |
| 26 |  |
| 27 | *// With ClampForever animations will not stop* |
| 28 |  |
| 29 | *// automatically when reaching the end of the clip* |
| 30 |  |
| 31 | leanLeft.wrapMode = WrapMode.ClampForever; |
| 32 |  |
| 33 | leanRight.wrapMode = WrapMode.ClampForever; |
| 34 |  |
| 35 | *// Enable the animation 和 fade it in completely* |
| 36 |  |
| 37 | *// We don't use animation.Play here because we manually adjust the time* |
| 38 |  |
| 39 | *// in the Update function.* |
| 40 |  |
| 41 | *// Instead we just enable the animation 和 set it to full weight* |
| 42 |  |
| 43 | leanRight.enabled = **true**; |
| 44 |  |
| 45 | leanLeft.enabled = **true**; |
| 46 |  |
| 47 | leanRight.weight = 1.0; |
| 48 |  |
| 49 | leanLeft.weight = 1.0; |
| 50 |  |
| 51 | *// For testing just play “walk” animation 和 loop it* |
| 52 |  |
| 53 | animation[“walk”].wrapMode = WrapMode.Loop; |
| 54 |  |
| 55 | animation.Play(“walk”); |
| 56 |  |
| 57 | } |
| 58 |  |
| 59 | *// Every frame just set the normalized time* |
| 60 |  |
| 61 | *// based on how much lean we want to apply* |
| 62 |  |
| 63 | function Update () { |
| 64 |  |
| 65 | **var** lean = Input.GetAxis(“Horizontal”); |
| 66 |  |
| 67 | *// normalizedTime is 0 at the first frame 和 1 at the last frame in the clip* |
| 68 |  |
| 69 | leanLeft.normalizedTime = -lean; |
| 70 |  |
| 71 | leanRight.normalizedTime = lean; |
| 72 |  |
| 73 | } |

**提示Tip:**

当使用附加动画时它会判断你同时也在播放一些其他的使用了附加动画的非附加动画, 否则动画将添加到最后一帧结果的顶部，这通常不是你所需要的 。

**程序动画角色Procedurally Animating Characters**

有时你需要程序化的驱动你的角色骨骼. 例如你可能需要你的角色的头注视3d空间的某个点. 这个活最好让脚本来干. 幸运的是, Unity做这个很容易. 在Unity 中所有骨骼来驱动蒙皮网格(skinned mesh)的变换(Transforms). 因而你可以给角色的骨骼写脚本,就和其他GameObject一样.

很重要的一点是动画系统updates the Transforms 是在Update() function调用之后 ，LateUpdate() function 调用之前. 因而如果你要调用 LookAt() function 你应该do that in LateUpdate() to make sure that you are really overriding the animation.

布娃娃系统Ragdolls 也是用同样的方法制作出来的. 你可以简单的把刚性物体(Rigidbodies), 角色关节(Character Joints) 和 胶囊碰撞体(Capsule Colliders)连接给不同的骨骼. 这样物理系统就可以作用于蒙皮角色(skinned character). (什么是布娃娃系统，当你在射击类游戏中打死对手时可以注意到当角色快接近地面时，他的四肢开始瘫软在地面上，这个不是动画师调出来的，而是布娃娃系统自动计算出来的。)

**动画重放和取样Animation Playback 和 Sampling**

这一部分 将说明引擎如何在动画重放时取样.

动画片断制作时总是有一个特定的速率. 举例来说, 你可能在Max 或Maya at 创建了一个帧速为 60 frames 每秒(fps)的动画. 当导入 Unity后, 输入模块将读取帧速, 所以导入的动画帧速还是60fps.

可是, 游戏运行时的速率是不断变化的. 有的电脑帧速快有的电脑帧速慢, 即使是同一台电脑前一秒和后一秒因为视角的不同帧速也不一样. 基本上当游戏开始运行时我们无法确定一个精确的帧速. 这意味着即使我们的动画片断制作时是 60 fps, 它重放时也许用的是另外一个速率, 例如 56.72 fps, 或 83.14 fps. 它可以变成任何一个速率.

Unity 对这些变化的速率取样, 不在于其制作时的速率. 幸运的是，3d电脑图形动画不是由分散的动画组成, 确切地说是由连续的曲线构成的. 这些曲线可以让我们在任何时间点取样; 而不是适配某一个原始帧的时间点. I这也意味着如果游戏运行速率高于原始制作速率, 动作事实上看起来会更平滑流畅.

对绝大多数应用场合, Unity对变化帧速的采样我们无需对其进行干预. 可是, 如果你的某个游戏逻辑所依赖的动画变化或道具(transforms or properties)结构十分特殊, 那你必须知道这一点. 举例说, 如果你有一个动画是把一个物体30帧内从 0旋转到180度, 你想从代码中得知什么时候动画完成一半, 你不能写一段条件语言来检查现在旋转值是不是90度. 因为 Unity 依照游戏的变化速率来对动画采样, 它可能在旋转快到90度时进行采样, 或者是刚好过90度的时候采样. 如果你需要通报动画中一个特殊点到达时, 你可以使用 AnimationEvent 来替代.

同样需要注意的是变化的帧速采样结果, 一个使用WrapMode.Once 模式重放的动画的采样不一定是精确的最后一帧( last frame). 在游戏中很有可能是刚好结束前的某一帧, 在下一帧时间可能超过动画的长度, so it is disabled 和 not sampled further. 如果你需要动画的最后一帧采样精确，你可以使用WrapMode.ClampForever. 如果是那样的话动画将不停的对最后一帧进行采样直到你自己停止动画.