<http://www.unitymanual.com/4709.html>

**在Unity引擎中使用Easing Function制作动画**

Posted on 2013年05月04日 by U3d / [Unity3D脚本/插件](http://www.unitymanual.com/category/script)/被围观 223 次

在制作游戏的时候，动画是不可或缺的一部份，即使是与游戏核心无关的 GUI 部份，若少了动画就显得粗制滥造。然而大多数的 GUI 效果：像是视窗飞进画面中央、按下按钮时放大效果之类的动画等等，其内容都有很高的相似性，也就是「在某段时间内，把物件的某些状态转移到另一个状态」。若要使用 3D Max 之类的软件一一制作，将显得麻烦而没有效率。

在这篇文章中，以 Unity 作为范例，介绍如何实作一个简单的 easing function 元件。

**范例需求**

想像一下我们正在制作游戏过程中的暂停按钮。当玩家按下暂停钮时，画面上会出现暂停的面板以及三个按钮。当然，直接让它们出现是很粗糙的作法，因此我们希望面板及按钮可以从画面外飞进来。

**使用线性内插**

最简单的呈现方式是使用线性内插法，指定好某个物件的起始状态（位置、大小、顏色等）与结束状态，再指定动画时间。只要有了这些资讯，我们可以用简单的数学运算内插出动画播放时每一格 frame 的物件状态。

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

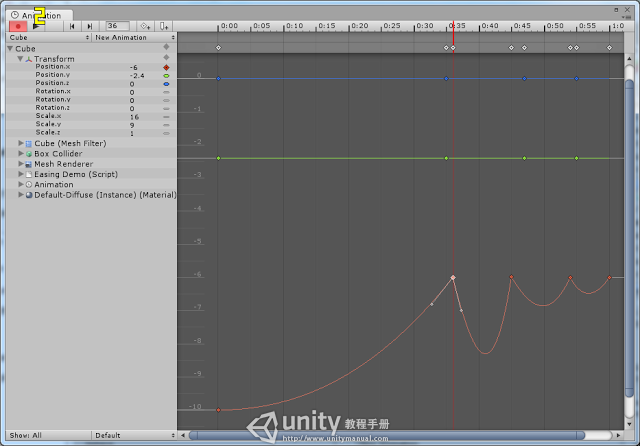
|  |  |
| --- | --- |
| 01 | **using** UnityEngine; |
| 02 | **using** System; |
| 03 |  |
| 04 | **public** **class** EasingDemo : MonoBehaviour { |
| 05 | **public** Vector3 destination; |
| 06 | **public** **float** duration = 1; |
| 07 | **public** **float** delay = 0; |
| 08 |  |
| 09 | **private** Vector3 source; |
| 10 | **private** Vector3 delta; |
| 11 | **private** **float** elapsed = 0; |
| 12 |  |
| 13 | **void** Start() |
| 14 | { |
| 15 | source = transform.position; |
| 16 | delta = destination &minus; source; |
| 17 | } |
| 18 |  |
| 19 | **void** Update() |
| 20 | { |
| 21 | elapsed += Time.deltaTime; |
| 22 | **if**(elapsed > delay && elapsed < (delay+duration)){ |
| 23 | **float** ratio = Math.Min(1, (elapsed&minus;delay)/duration); |
| 24 | transform.position = source + ratio\*delta; |
| 25 | } |
| 26 | } |
| 27 | } |

这个简单的元件让我们可以在 Unity 编辑器中直接设定物件的起始位置、结束位置及动画播放的时间。接下来，我们把面板与按钮放在镜头外面，然后把它的结束位置设定在画面内，按下 play 后马上可以看到它的效果……

这个动画之所以看起来很无聊，大部份的原因在于我们使用了很无聊的线性内插法。想像一下如果这个按钮是用以下的方式飞进画面：

1. 很快地冲进来，然后紧急剎车停在目的地。
2. 冲过头然后拉回目的地。
3. 撞到目的地后像皮球般反弹，最后再落回目的地。

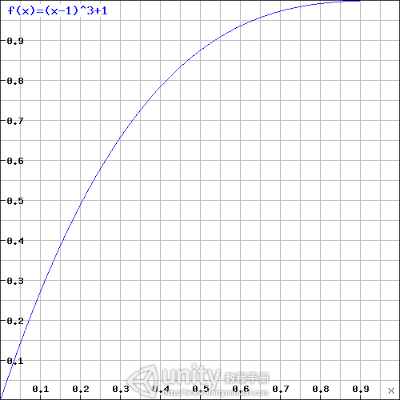
这样的动画，显然会比死板的等速度飞入更活泼、更有动感。然而制作这种动画效果就没办法用线性内插了。当然，若使用 3D Max 就可以借由贝兹曲线 (Bézier curve) 来编辑这种较复杂的动画，或是直接用 Unity 的动画编辑器也能办到。然而编辑贝玆曲线的各个控制点却是一件很花时间的工作，尤其像第三个例子，为了达到反弹的效果，我们得额外加入两三个 key frame 才行。过程并不是很难，但就是很花时间。

[](http://www.unitymanual.com/wp-content/uploads/2013/05/Unity-Animation-Editor.png)

Unity 使用 Easing Function 制作动画

**使用非线性内插法**

我们不希望动画是平板的线性内插法，但又觉得编辑贝兹曲线的控制点太麻烦，那么答案就呼之欲出了：使用其它的数学曲线来代替直线。比如说如下的曲线：

[](http://www.unitymanual.com/wp-content/uploads/2013/05/outCubic.png)

在Unity引擎中使用Easing Function制作动画

尽管物理上并不正确，但它看起来就像是「很快地冲进来，然后紧急剎车」的样子。因此我们可以把它套用到内插计算式中：

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

|  |  |
| --- | --- |
| 1 | **void** Update() |
| 2 | { |
| 3 | elapsed += Time.deltaTime; |
| 4 | **if**(elapsed > delay && elapsed < (delay+duration)){ |
| 5 | **float** x = Math.Min(1, (elapsed&minus;delay)/duration); |
| 6 | **float** ratio = (**float**)Math.Pow(x&minus;1, 3) + 1; |
| 7 | transform.position = source + ratio\*delta; |
| 8 | } |
| 9 | } |

结果如下，同样都是一秒钟的动画，感觉起来是不是有微妙的差异呢？

**Easing Function 惯例格式**

Easing function 具有一项优势：只要抽换不同的 easing function，不需要另外编辑 key frame 或是曲线控制点，就马上可以得到另一种动画。但为了达到这个「可抽换」的目的，我们得要使用大家所惯用的写法：

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

|  |  |
| --- | --- |
| 01 | **public** **class** Easing { |
| 02 | **public** **static** **float** Linear(**float** t, **float** b, **float** c, **float** d) |
| 03 | { |
| 04 | **return** c\*(t/d) + b; |
| 05 | } |
| 06 |  |
| 07 | **public** **static** **float** OutCubic(**float** t, **float** b, **float** c, **float** d) |
| 08 | { |
| 09 | t /= d; |
| 10 | t&minus;&minus;; |
| 11 | **return** c\*(t\*t\*t+1)+b; |
| 12 | } |
| 13 | } |

一般化的 easing function 有四个参数：

1. t (time): 代表动画开始播放到现在所经过的时间。
2. b (beginning): 代表某项属性的初始值。
3. c (change): 代表到动画结束时，该属性的变化值。亦即动画结束后，这项属性的值应為 b+c。
4. d (duration): 代表整段动画的播放时间。

因此我们可以改写原本的 Update()：

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

|  |  |
| --- | --- |
| 01 | **public** **delegate** **float** EasingFunction(**float** t, **float** b, **float** c, **float** d); |
| 02 | EasingFunction easing = new EasingFunction(Easing.OutCubic); |
| 03 |  |
| 04 | **void** Update() |
| 05 | { |
| 06 | elapsed += Time.deltaTime; |
| 07 | **if**(elapsed > delay && elapsed < (delay+duration)){ |
| 08 | Vector3 p = new Vector3(); |
| 09 | **float** t = elapsed &minus; delay; |
| 10 | p.x = easing(t, source.x, delta.x, duration); |
| 11 | p.y = easing(t, source.y, delta.y, duration); |
| 12 | p.z = easing(t, source.z, delta.z, duration); |
| 13 | transform.position = p; |
| 14 | } |
| 15 | } |

这么一来，我们就可以利用别人写好的 easing function 套用到我们的动画元件中。事不迟疑，马上就来看看范例。

**弹跳动画**

上述「像是皮球般地弹跳」显然是一种很困难的曲线，但早就有人写好了它的 easing function，只要 google 一下马上就能找到如下的程式码：

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

|  |  |
| --- | --- |
| 01 | **public** **static** **float** Bounce(**float** t, **float** b, **float** c, **float** d) |
| 02 | { |
| 03 | **if** ((t /= d) < (1.0f / 2.75)) { |
| 04 | **return** c \* (7.5625f \* t \* t) + b; |
| 05 | } |
| 06 | **else** **if** (t < (2.0f / 2.75)) { |
| 07 | **return** c \* (7.5625f \* (t&minus;=(1.5f/2.75f)) \* t + 0.75f) + b; |
| 08 | } |
| 09 | **else** **if** (t < (2.5f / 2.75)) { |
| 10 | **return** c \* (7.5625f \* (t&minus;=(2.25f/2.75f)) \* t + 0.9375f) + b; |
| 11 | } |
| 12 | **else** { |
| 13 | **return** c \* (7.5625f \* (t&minus;=(2.625f/2.75f)) \* t + 0.984375f) + b; |
| 14 | } |
| 15 | } |

然后我们把 EasingDemo 中的 easing function 设定为 Easing.Bounce，马上就能看到弹跳的面板与按钮：

这样的效果是不是比前面慢慢飞进来的动画要生动许多呢？

只要指定起始状态、结束状态、播放时间及 easing function 的种类，即可组合出千变万化的各种效果，因此 easing function 是相当广泛的 2D 动画技术。知名的 JavaScript 函式库 jQuery 也内建了许多不同种类的 easing function。

Unity 范例场景可以在 这边 [下载](http://vdisk.weibo.com/s/zPG93)。可导入Unity看到范例程式码及实际的动画效果。