基于空间位置捕捉的

动画制作辅助方法

实验报告

组长：计 42 王　倩

组员：计 42 陈禹东

计 42 陈光斌

2017 年 1 月

一、选题

三维动画制作在当今的社会生活中有着广泛的应用。游戏业、广告业、影视业甚至是公众演讲场景下都常常需要 3 维动画。然而，虽然开发三维动画的软件种类繁多，但经过前期的调研，我们发现大部分软件学习成本非常高。即使有人手把手教学也常常需要一整天才能入门做出非常基本的动画。其中涉及的概念更是繁复抽象。究其原因，恐怕还是仅仅通过鼠标、键盘等设备来编辑三维动画的局限，这样就像用二维的工具来编辑三维动画。但是随着 Leap Motion 等设备的推出，我们发现在三维空间中编辑动画十分符合人的直观理解，并非天方夜谭，因此我们希望能够在这学期的课程通过 Unity 来开发一款能利用 Leap Motion 的空间移动特性来编辑三维动画的应用。

二、设计与分工

设计

经过讨论与分析，我们决定实现以下功能：

◆ IK 动画与基本控制

考虑到实际 3D 动画制作中最常用的就是人体/类人体的动画，因此我们学习了 IK 动画的基本实现，以及对物体的位置、朝向等的手势控制。

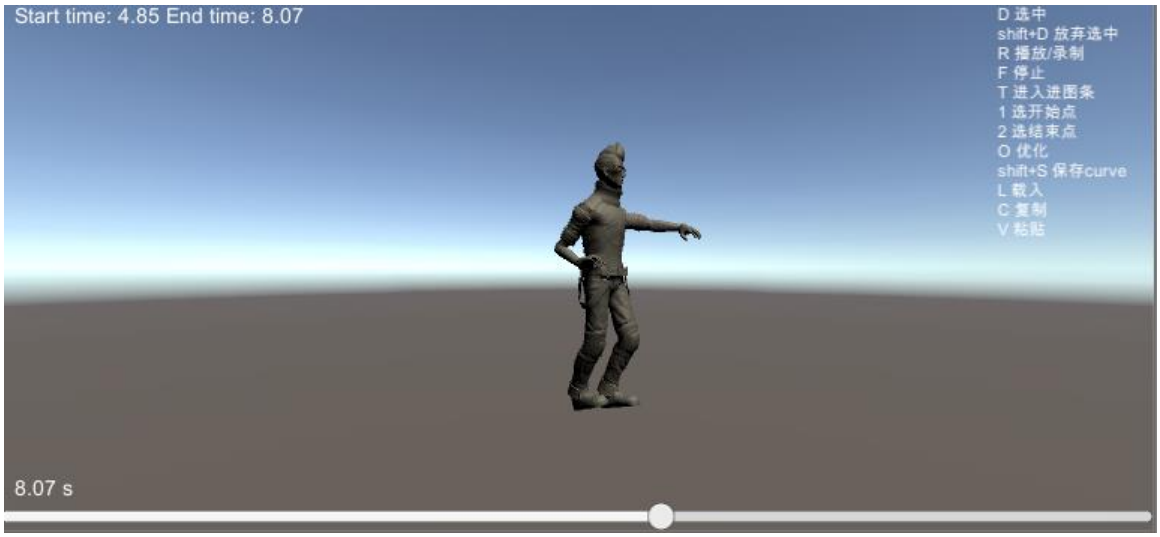
◆ 动画的录制、重放、保存、载入

我们将 log 文件记录的格式定为“ 时间间隔，x 位置，y 位置，z 位置，x 朝向，y 朝向，

z 朝向”。并通过此方法实现了轨迹的录制、重放、保存到文件和载入制定轨迹的功能。

◆ 轨迹的选择覆盖录制

如何编辑修改动画中的某一段呢？我们采取了时间段重新录制的方案。一开始也考虑过和目前市场上软件类似的关键帧方法，但由于我们的录制本身就不依赖于关键帧的概念，因此我们决定用统一的思路来解决问题。而且只要将时间设置得足够短，并且把优化参数（后文会介绍到）调整合适，重录本质上和关键帧是一样的，也就是说，关键帧本质上是重录的子集。扩展录制只需要将结束时间设置为-1 即可。



◆ 轨迹优化

由于重新录制必然会带来新录制的轨迹开始和结束两个点的位置与原本动画衔接不上，因此我们实现了自动轨迹优化功能来使轨迹的衔接更加流畅。

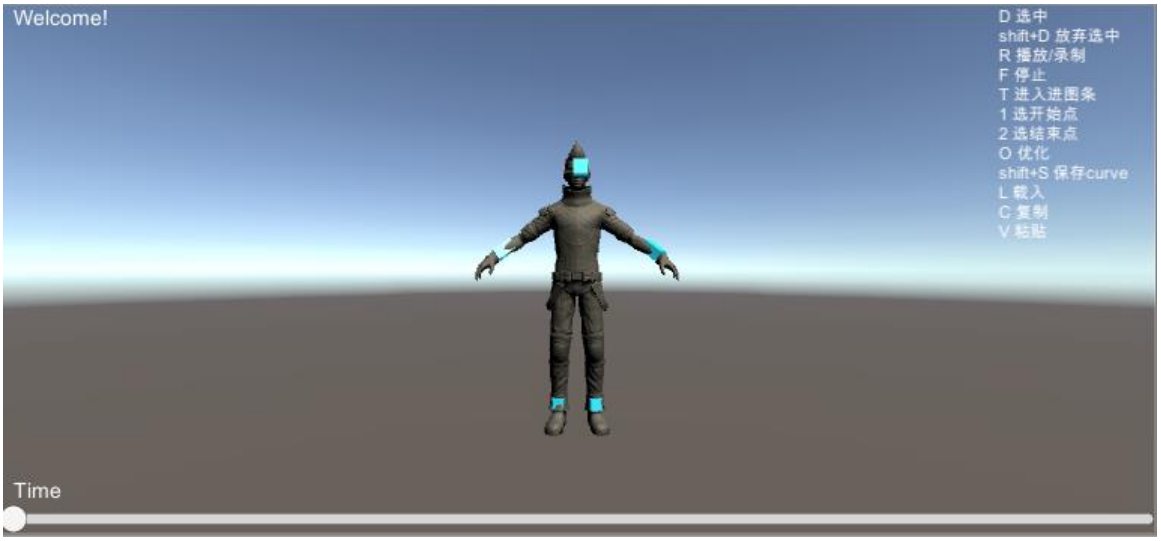
◆ 轨迹复制及调整

考虑到动画中经常需要相同或相似的轨迹（比如烟花爆炸中每一粒烟花都以相同的轨迹

在不同的方向运动），因此我们实现了复制轨迹到物体以及轨迹调整功能。比如我们的 demo 中可以将左手的轨迹翻转后复制到右手似的两手的动作对称。还可以通过设置参数改变轨迹的速度、平移、旋转和拉伸。

◆ UI 界面

为了能够选择覆盖录制的时间点，我们设计了滑动进度条和数字共同显示播放时间（类似视频的进度条），可以随意拉动。



(图为基本界面)

分工

由于一开始我们对 unity 和 leapmotion 都没有任何基础，因此我们首先分头用不同方法尝试最基本的录制小方块轨迹，并最终采用了陈禹东的方法。

前期设计由我们共同讨论得出。后期我们的分工大致是：王倩负责人物模型动画、基本控制、覆盖录制；陈光斌负责 UI 部分的设计实现以及覆盖录制时间点的选取；陈禹东负责轨迹的优化和时间空间维度的调整。

三、实现方法与原理

1、基本原理概述

我们的动画控制技术主要基于 Leap Motion 和 Unity3D 提供的部件实现，辅助以机械键盘的按键控制。

首先在动画模型的关键部位上放置"控制方块"，利用 Leap Motion 手检测碰撞后控制 IK 动画。辅以键盘操作，可录制动画过程并自动存储到文件，并利用 Unity 的 AnimationCurve 对象生成每个控制方块的动画轨迹。编辑动画时，使用 Unity 的进度条部件辅助编辑过程，定位要修改的区间并重新录制。编辑结束后，通过对 AnimationCurve 的数学变换，可对轨迹"脱轨"处进行平滑处理。此外，还可通过 Leap Motion 手辅助数学变换，实现动画的对称复制功能。

下面将依次介绍各模块的实现方法与原理。

2、IK 动画与控制方块

IK 动画模块是 Unity3D 自带的一个动画控制模块，IK 的全名是 Inverse Kinematics，即反

向动力学，表示子骨骼节点带动父骨骼节点运动。具体到我们的 Demo 里，就是通过控制小方块的运动带动小人模型的四肢和头部运动。

在实现上，我们选用了 Unity 自带的小人模型，它有 5 个 IK 小方块，分别控制小人的四肢和头部视线的运动。我们在 IKControl.cs 中具体实现了对于 IK 动画的控制。

首先，在 Start()函数中得到动画的控制对象。这里的 Animator 对象用于对 IK 动画的控制。

```
void Start () {
    animator = GetComponent<Animator>();
}
```

在 Unity 导航菜单栏中打开 Window->Animator 打开动画控制器窗口，并勾选 IK Pass 的前提下，通过回调方法进行对于 IK 动画进行控制。这里 ikActive 表示 IK 动画已开始，SetWeight()表示设置骨骼的权重。

```
void OnAnimatorIK(){
    if (animator) {
        if (ikActive) {
            if (lookObj != null) {
                animator.SetLookAtWeight (1);
                animator.SetLookAtPosition (lookObj.position);
            }//以下省去对其他4个方块的控制代码
```

最后，当取消 IK 动画时，重置骨骼的坐标。

```
animator.SetIKRotationWeight (AvatarIKGoal.RightHand, 0);
animator.SetIKPositionWeight (AvatarIKGoal.RightHand, 0);//以下省去
```

在 CubeControl.cs 中，我们检测五个小方块的 collider 是否是 Leap Motion 手，如果是，就在每次 Update()时把 Leap Motion 手的位置+offset 赋值给被选中的小方块的位置。

其中 offset 是刚检测到碰撞时，小方块相对 Leap Motion 手的偏移：

```
//在void OnTriggerEnter(Collider c)中:
offset = transform.position - c.transform.position;
//在void Update()中:
transform.position = ctrler.transform.position + offset;
```

这样，就可以实现 Leap Motion 对小人的 IK 动画控制。

需要注意的是，每次控制 IK 动画的过程中，需要保证按下 D 键且不松开，以表示选中某一方块进行控制。

3、动画录制与存储到文件

动画录制模块我们采用了相对简单的处理方法，即在每次录制动画时，把 5 个小方块的七维信息（时间、x、y、z、rotateX、rotateY、rotateZ）存入项目 Assets\UserFiles\Positions 目录下的一个 txt 文件中，其中时间维度存储每条记录相对上一条记录的时间差。

值得一提的是，我们在存储时只存储关键帧的信息，这里关键帧的定义是：与上一帧有显著位移的帧。具体来说，我们定义了一个阈值（PersonCtrlRecord.cs 中）：

```
public float recordPrecision = 0.02f;
```

当且仅当相邻帧的欧氏距离大于 recordPrecision 时，才进行文件记录：

```
if (isRecording) {
    if (Vector3.Distance (temp, recordFlagPos) > recordPrecision || (!isRecording) ) {
    //省略存储到文件的代码
    }
}
```

此外，我们采用键盘按键来控制"是否在录制动画"，即按一下 R 键代表进入录制模式。在录制结束时，再按一下 F 键（表示 Finish），可将所有录制结果保存至文件并生成动画曲线 curve。此时再按下 R 键，即可重放录制好的动画。

值得一提的是，录制和重放两个功能我们都采用按下 R 键来控制，这是因为在录制小人右手的动画时，往往需要同时播放已录好的左手的动画，以期达到更好的同步效果。

4、AnimationCurve 的使用原理

AnimationCurve（以下简称 curve）是 Unity3D 自带的一个类，用于存储一个连续时间的函数 y(t)，函数值可以应用到动画的任何一个维度。同时，curve 可以由一系列的离散数组中的值直接生成，从而达到了"记录离散函数，生成连续动画"的目的。

与之紧密相关的一个类是 KeyFrame，即关键帧。每个关键帧对象对应两个成员变量：函数值（value）与时间（time）。

具体实现上，我们在 PersonCtrlRecord.cs 中用六个全局的 KeyFrame 数组 KsX、KsY、KsZ、fsX、fsY、fsZ 存储一个小方块 x、y、z、rotateX、rotateY、rotateZ 六维的关键帧。在每次结束录制时（按下 F 键时），会调用 makeCurve()函数，先从文件将信息读入到 KsX 等数组中，再由它生成 animX 等三个 curve。在重放阶段，即可由根据 curve 来重放：

```
if (isReplaying)
{
    if (!isSliding) {
        float delta = Time.time - curveStartTime;
        //...
        transform.position = new Vector3 (animX.Evaluate (delta),
                                          animY.Evaluate (delta),
                                          animZ.Evaluate (delta));
    }
}
```

5、进度条与帧定位

进度条我们使用了 Unity 自带的 Slider UI 来实现。按数字 1 键记录下开始截取的时间点，按

数字 2 键记录结束截取的时间点。我们既支持在播放动画的同时进行截取，也支持暂停动画后通过移动圆形光标来选择指定时间点。在选定完两个时间点后，只要再录制一遍即可。

```
int getFrameIndex(float time)
{
    int cnt = 0;
    while (frameTime[cnt] < time)
        cnt++;
    return cnt-1;
}

void getTimeAndFrame(float value)
{
    if (Input.GetKeyDown(KeyCode.Alpha1))//数字1键获取截取开始时间点
    {
        startTime = value;
        begin_frame = getFrameIndex(value);
        Debug.Log("begin: " + "time: " + startTime.ToString() + " frame: " + begin_frame.ToString());
        infoText.text = "begin: " + "time: " + startTime.ToString() + " frame: " + begin_frame.ToString();
    }
    else if (Input.GetKeyDown(KeyCode.Alpha2))//数字2键获取截取结束时间点
    {
        endTime = value;

        if (getFrameIndex(value) > begin_frame)
            end_frame = getFrameIndex(value);
        infoText.text = "end: " + "time: " + endTime.ToString() + " frame: " + end_frame.ToString();

    }
}
```

6、“脱轨”现象与平滑处理

在重录动画之后，常常会出现“脱轨”现象，即重录区间的首帧与尾帧容易与原轨迹有较大偏移，导致原本流畅的轨迹出现一头一尾两个突变点。究其原因，主要是“接轨处”的位移太大，时间差太小。位移是用户决定的，不能改变；因此要想让轨迹更加平滑，就只能合理地增大时间差。

为此，我们采取了这样的优化策略：首先，在重录时自动获取其首帧和尾帧的位置，我们令其为第 s 帧和第 e 帧。对于重录后的完整轨迹，增大第 s-1 帧与 s 帧、第 e 帧与第 e+1 帧之间的时间差至原来的 10 倍，并按相应比例缩短其他帧之间的时间差，使得总时间不变。

具体实现比较长，在 PersonCtrlRecord.cs 的 Optimize()函数中。

操作方法为，每当结束重录后，可按下 O 键，再按下 R 键即可重放平滑化处理后的曲线。若对优化结果满意，则可按下 Shift+S 键，将其保存至文件中。

7、对称动画的复制

由于 AnimationCurve 实际上是保留关键帧的（value, time）对，因此，想要对已有 curve 进行空间变换并不困难。考虑到动画录制中常常要追求对称，我们实现了一个“复制并粘贴对称动画”的功能。

操作方法为，在按下 D 键选中左手的情况下按一下 C 键（表示 copy），松开 D 键，重新按下 D 键选中右手并按一下 V 键（表示 paste），即可将左手的轴对称动画“复制到右手”。

具体实现上，我们使用了跨文件全局变量的思想，在 ll.cs 中设置了几个“tmp”意义的变量，作为复制操作的中间传递者。在 copy 操作时，将左手相应的值赋给“tmp”；在 paste 操作时，将“tmp”的值做关于 YOZ 平面的对称变换后赋给右手。

8、其他空间变换

在实现过程中，我们还实现了对轨迹的平移、平面内旋转、延拓与压缩、加速与减速等多种变换。具体实现详见 PersonCtrlRecord.cs 的 changeCurveSpeed()、translateCurve()、extendCurve()、rotateInXOY()等函数。

利用空间变换，我们又实现了对小人的整体位置移动的控制和编辑。

四、项目小结

1、成果小结

经过漫长的设计和调试，我们的最终版 Demo 终于诞生。

- 从功能角度来看：在现有 Demo 中，动画制作者可以通过录制、重放、编辑、优化等一系列操作，控制模型小人的四肢和头部做出不同的动作，最终跳出一段简单的舞蹈。在整个制作过程中，用户可以通过进度条查看和定位动画细节，利用轨迹优化使编辑后的动画更加平滑，也可以便捷地复制对称动画，让整个过程轻松写意。

- 从人机交互的角度来看：我们利用 Leap Motion 捕捉用户双手在空间中的位置，结合 IK 动画技术，使得三维动画的制作变得十分自然，符合人本身的习惯。另一方面，我们又通过键盘按键辅助控制整个制作过程，解决了 Leap Motion 手势识别方法不自然也不准确的问题，既简单又实用。此外，进度条、轨迹优化等功能的实现，也提高了工程的可用性，提升了用户体验。

2、经验小结

在项目的设计和实现过程中，我们一起从零基础学习 Unity 和 Leap Motion，讨论和尝试过不同的实现方法；我们遇到过不少"坑"，也为此看到了凌晨两点清华的月亮。在这里总结一下开发过程中我们收获的一些小经验：

- 官方教程、文档、模型非常实用。一开始的时候，为了学习 Unity，我们一起看官方教程，做了一个非常简单的 Demo。尽管花费了一些时间，但对于零基础的我们来说，官方教程细致全面的讲解让我们很好地熟悉了 Unity 的使用方法。为了使用 Leap Motion，我们又查看了许多官方 Demo 和文档，其中有些 Demo 的实现也给了我们不少启发。而对于 Unity 文档的查阅更是贯穿整个开发过程的。

- C#读写文件需要特别注意。C#读写文件一般使用 FileStream 配合 StreamWriter 和 StreamReader 实现，这些类的 API 有一些比较方便的接口，但用起来容易出错。

容易出现的错误包括（以写文件为例）：两个 StreamWriter 不能同时写一个打开的文件；如果要一次性写入较多内容，需要在中途调用 StreamWriter 的 Flush()函数；写完文件后一定要及时关闭 StreamWriter，即调用 StreamWriter.close()等等。

- 减少用户的学习成本是一个难题。这是一个很大的话题，是人机交互设计中一个重要的命题。在我们的设计中，键盘的按键操作其实比较繁杂，这对于用户来说需要花一定的时间来学习和记忆，不够方便。但想要进一步减少用户成本，就需要更巧妙的设计思路，这可能需要花费更多的时间，甚至需要更换实现的工具等等。

五、不足与思考

最后，由于时间原因，一些计划没能来得及实现。而且在实现了以上功能后，我们发现我们的应用仍然需要改进的地方。因此请允许我们在此从"应用功能丰富"以及"用户体验优化"两个方面谈一谈我们的思考。

- 应用功能丰富

1.导入模型是我们计划实现但还未实现的部分。目前我们的应用仅仅支持导入 Unity 自带的模型来进行编辑，如果想让我们的应用为更多人使用，显然我们需要实现模型的导入以及动态创建。

2.轨迹显示也是我们计划实现但还未实现的部分。考虑到有时候用户希望观察模型运动的轨迹，从而更好地对模型运动轨迹进行修改，故我们希望实现在必要时让运动轨迹可视化的功能。

◆ 用户体验优化

1.操作优化。我们最初为了方便采用各种通过键盘实现的操作，但让同学体验时发现功能键的记忆是非常大的阻碍，因此我们认为功能键+button+手势的多种设计才能最大化制作效率。

2.leapMotion 的抖动消除。由于硬件原因，控制物体时手模型的抖动非常厉害，影响了动作的控制。因此应当使用算法来消除抖动。

3.视角切换。始终从一个方向观看动画对控制不是十分有利，加上 leapMotion 的控制范围实在太小，我们认为视角的移动十分必要。这一功能可以通过两种方式实现。一是通过摄像机移动以及 controller 移动的结合。而是通过将物体置于 VR/AR 场景中来让人的视角在三维方向上移动。