作业2

任务零 环境配置

1. 安装panda3d(Windows/Linux/Mac)

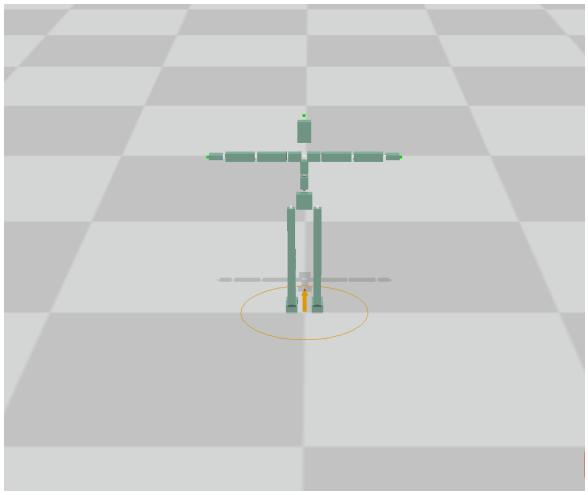
建议使用conda等建立新的虚拟环境

```
conda create -n games105 python=3.8
conda activate games105
conda install numpy scipy
pip install panda3d
```

如果下载过慢可使用清华镜像源安装 (https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/)

> 本作业只允许使用 `numpy` ， `scipy` ， `pytorch` (`tensor-flow` ， `jittor`)以及其依赖的库。评测时也以此为准。版本限定到2022年10月31日之前的发布版本。作业文件中请不要import除此之外的库。

完成后可以运行 `task0_build_and_run.py` ，你将会看到一个地面上的黄色圆圈。你可以通过键盘或左手柄来控制它的移动。可以通过鼠标左右中键或右手柄操作相机视角。(检测到手柄会无视键盘输入，具体情况请参考 `task_0_build_and_run.py` 的注释.)



任务一 简单动作处理

在之前的lab1中你完成了对BVH数据的读入和前向/逆向运动学。而在这个部分你需要对运动数据们进行进一步的处理，从而为搭建自己的交互角色动画做准备。

这部分你需要填写 `answer_task1.py` 的内容。 在这里我们实现一个 `BVHMotion` 对象，它预先实现了读取BVH和前向运动学的功能。

不太相同的是，我们这次记录了每个joint有多少个channel(end joint有0个， 普通joint3个， 根节点和特殊joint有6个，且前三个为position后三个为rotation)。

需要注意的是，BVH中如果该关节有position channel， 则offset会被忽略， 而是把position channel的数字当作offset使用，相当于offset是时不变的position channel。 故而我们记录了 `joint_position` 这一(帧数x关节x3)的数组， 对3自由度关节，其始终是offset，而对有position的关节，其会填入position。

故而前向运动学可以简单地写为

```
child_translation = parent_translation + parent_orientation.apply(child_position)
```

我们统一让 `position` ， `rotation` 为局部位置，旋转， `translation` , `orientation` 为全局位置旋转。

part1: 平移和面朝向

在此部分，你需要实现控制BVH运动的位置和面朝向。 比如我们在拼接动作时，总是希望新动作第一帧和上一个动作的最后一桢位置相同，朝向相同，这样才能比较平滑地将其连接起来。

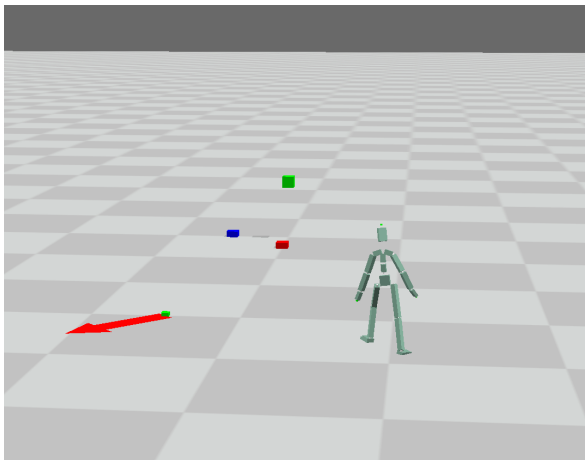
在这里你需要完成 `BVHMotion` 中的 `translation_and_rotation` 函数以及辅助函数 `decompose_rotation_with_yaxis` 函数。注意到类对象里只保留了局部信息 `joint_position` 和 `joint_rotation` ， 全局信息是在后期重新计算FK使用的，故而我们只需要调整根节点的水平position和rotation。

对position的水平调整比较直观，我们当作例子给出

```
res = self.raw_copy() # 拷贝一份，不要修改原始数据
offset = target_translation_xz - res.joint_position[frame_num, 0, [0,2]]
res.joint_position[:, 0, [0,2]] += offset
```

而对于rotation，回忆到课上讲过旋转 `R` 可以被分解为一个绕y轴的旋转 `Ry` 和一个绕xz平面上的轴的旋转 `Rxz` ， 并且 `R=RyRxz` 。调整水平转向实际上是调整 `Ry` 。

当然，调整完rotation后，position需要进一步调整，否则会出现本来是向前走，现在是斜着走的问题。



part2: 动作插值

有时候我们录了向前走路和跑步的动作，这些动作的运动速度是固定的。而现在我们需要新的速度的向前运动，可以通过对已有的动作进行插值得到。

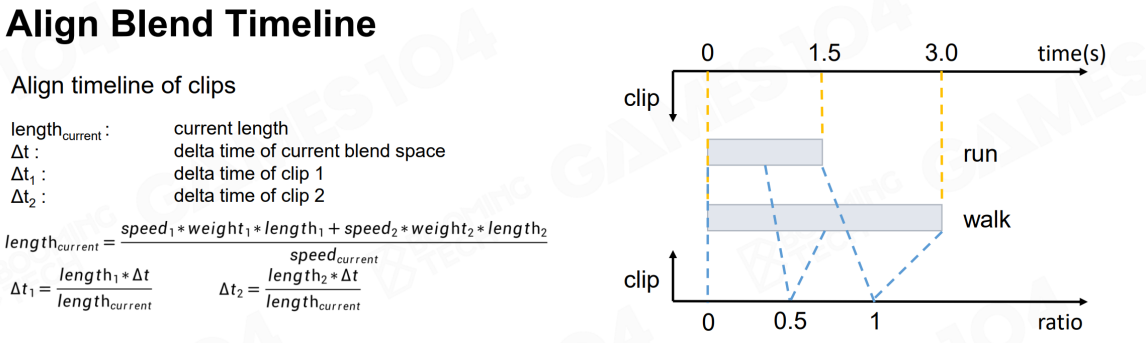
对于两个pose，我们可以用线性插值(lerp)处理局部偏移(joint_position)，用slerp或者nlerp处理局部旋转。

而插值动作时，我们会面临动作长度不一的问题。因为一般走路，跑步动作都是左右脚各迈一次，速度不同必然导致动作时间不同。假设走路，跑步的速度为v1,v2，动作帧数为n1,n2。那么如果想获得速度为v3的动作，左右脚各迈一次的时间为

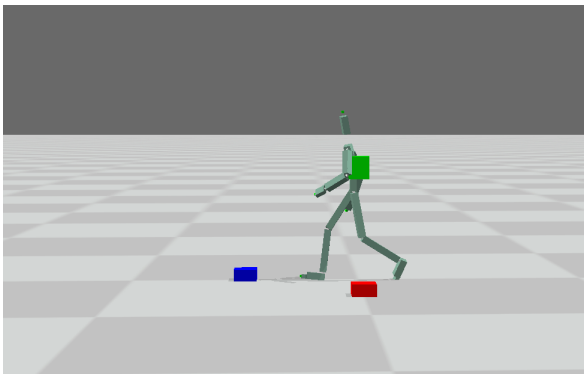
ParseError: KaTeX parse error: Undefined control sequence: * at position 14: n = \frac{w_1*v_1*n_1+w_2*v…

其中w为混合权重

那么我们做插值时，新动作的n/2帧处应该用走路的n1/2帧和跑步的n2/2帧进行混合。混合系数同样为上述w。详情可以参考GAMES104(如下图)



在知晓上面内容以后，请完成 `blend_two_motions` 函数。我们已经把 和混合系数为你算好并当作输入。

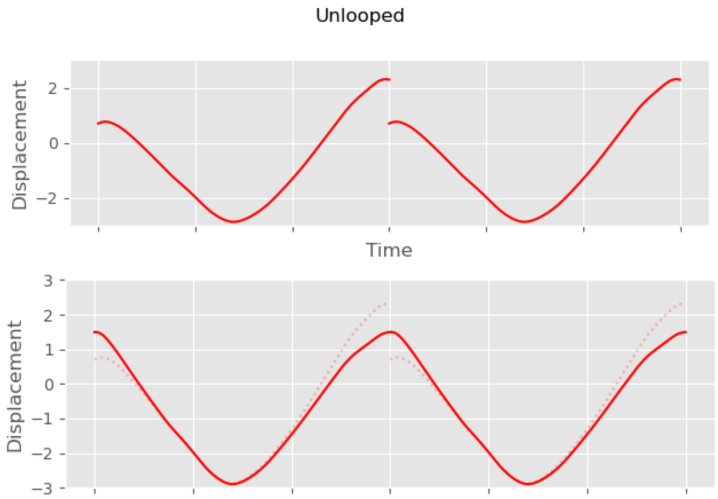


part3：制作循环动画

我们提供的走路/跑步动画为左右脚各迈一步。那么如何获取一直向前走的动画呢？很简单，把它复制一次然后把新的动画初始位置与它原本的末尾位置对齐即可(使用part1的函数)。

然而，这个动作只是从长动作截取的一段，并不是一个完美的循环动画，直接的拼接会导致内容的突变。

故而如何把一个差不多循环的动作变成循环动作？其实就是尽可能把原本的首尾差异抹平，并均摊到其他帧。可以阅读并参考Creating Looping Animations from Motion Capture。



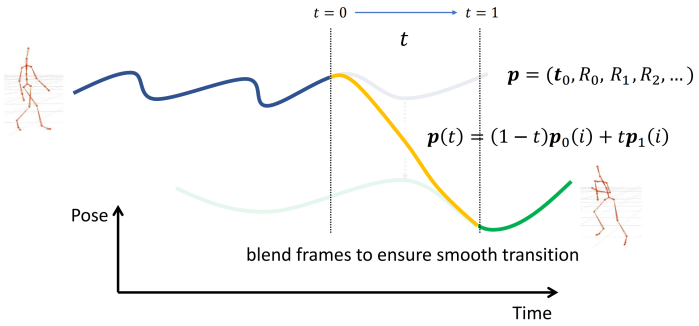
由于比较复杂又较为重要，我们为大家提前写好了一个版本 `build_loop_motion` 。作业中可以不用自己实现。但是如果感兴趣可以尝试一下。

你可以尝试注释 `part3_build_loop` 中的 `motion=build_loop_motion(motion)` 来观察不同。调节 `ShowBVHUpdate` 中的 `speed_inv` 可以调节播放速度。

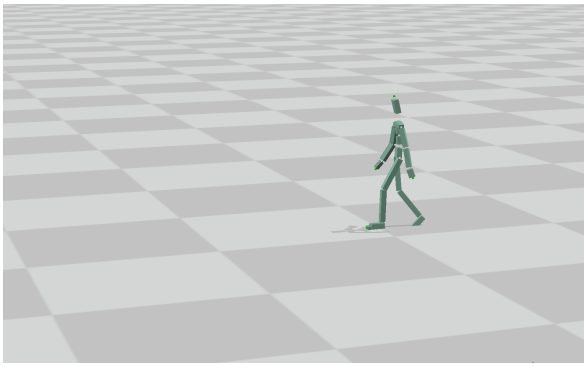
part4: 动作拼接

接下来一个问题是如何衔接两个动作，比如自然地从走路到跑步？

由于局部旋转不同，直接的拼接必然导致肉眼可见的突变。故而需要使用课上讲的blend方法从一个动作平滑地过渡到另一个动作。



你需要实现 `concatenate_two_motions` 函数。函数输入的 `mix_frame1` 表示动作1的开始转移的帧数，mix_time表示转移用了几帧。实际上除了linear blend， Inertialize等方法也可以用于动作的拼接，并且不需要mix_time参数，我们不限定方法，最终会根据连接的平滑性和动作质量评分。



任务二 可交互角色动画

这一部分将实现一个可交互的角色动画，即通过动作捕捉数据的插值、拼接和组合等方式，实现虚拟角色根据键盘/手柄的控制进行相应的动作。你需要完成的代码在 `answer_task2.py` ，实现后，运行 `task2_interactive_character.py` 查看效果。

在这一部分你需要实现一个可交互的角色动画控制器。通过我们包装好的控制器(如task0所示)，你会获得用户输入的希望的速度，面朝向等等信息，你的角色需要根据这些信息来动态调整自己的运动。

为了简单，我们只考虑角色的站立和走路。同时我们也实现了跑步的输入映射，你可以尝试实现跑步。

由于方法各异，评分标准按照动作连续性，响应速度与准确性，动作自然程度进行评分。

- 通过： 能够响应输入，但是上面评分标准中有一项或以上有较大缺陷
- 良好： 上述评分标准无较大缺陷，或缺陷不明显(如较小的脚底打滑等)
- 优秀： 上述评分标准无缺陷，并在某一方面效果十分良好

以下列出一些可能的思路:

简单的拼接/状态机

一种最简单的实现方式是根据用户输入在虚拟角色正在播放的动作捕捉数据上拼接新动作，例如当用户按下左转方向键时，程序将一段向左转的动作捕捉数据的初始位置和朝向与虚拟角色当前的状态对齐，再将这段处理过的动作数据拼接到当前播放的数据之后，从而实现通过用户输入控制虚拟角色行进方向。需要注意的是，拼接新动作时角色的姿态可能会有跳变，需要用插值等方式进行平滑。

进阶一些的方法是构建一个状态机，即一个动作图模型表示虚拟角色的动作和动作之间的切换，图中每个节点表示一种动作，例如站立、走路、跑步等，图中的每条边表示两个动作之间的切换。在有了动作捕捉数据后，我们可以把每一小段动作作为节点，并设计相邻节点之间的切换过程，例如用插值将走路动作平滑地切换为跑步动作。当动作图构建完成后，就可以用键盘/手柄控制虚拟角色从当前节点切换到相邻节点，从而实现可交互的角色动画生成。关于动作图模型，可以参考Motion Graphs[Kovar, Lucas, Michael Gleicher, and Frédéric Pighin. "Motion graphs." ACM SIGGRAPH 2008 classes. 2008. 1-10.]。

Motion Matching

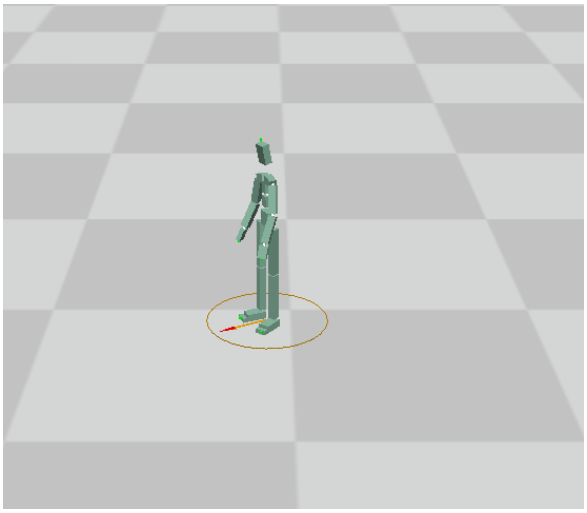
在有长motion的情况下，Motion Matching方法不需要对动作进行切割，也不需要自己定义动作转移，故而实现起来实际上更为简单。

一个非常好的Motion Matching实现Toturial和对应代码仓库。

一些小Tips:

- 为了实现博文里的效果，你可能需要提取 `Simulation Bone` 来当作根节点。但这个很容易通过对BVHMotion的修改实现。你可以让controller跟着Simulation Bone走来观看simulation bone提取的位置旋转是否准确。
- 速度和角速度是非常重要的动作特征，一般通过有限差分计算得到。在构建循环动作时我们在 `smooth_utils` 里给出了计算角速度的方法以供参考。
- Motion Matching需要手动让动作转移更加平滑，也即Inertialization的方法。其中需要的damped spring我们在 `smooth_utils` 里给出了一个实现。

一个简易的Motion Matching:



Learning-based 方法

如果在繁杂的手工定义和玄学的炼丹中你更能接受后者的话，那么learning-based方法可能是一个好的选择。

同课程第六讲所展示的那样，你需要学习合理的动作的 分布 ，并将其和用户输入对应起来。

能够实现交互角色动画的learnning方法有很多，我们简单列出一些：

- Phase-Functioned Neural Networks for Character Control
- Character Controllers using Motion VAEs
- Learned Motion Matching

动作数据

在 `motion_material` 文件夹我们提供一些基本的动作数据。包含一些短motion和两个长motion。你可以使用blender等软件查看长motion，并剪辑出需要的短motion。

短motion:

- idle: 站立不动
- walk_forward: 向前走两步，先右脚后左脚
- run_forward： 向前跑两步，先右脚后左脚
- walkF: 向前走四步，首尾的局部动作是loop的
- walk_and_turn_left： 走两步然后向左转
- walk_and_turn_right： 走两步然后向右转，由上一个动作镜像得到

长motion主要是walk,run和他们对应的镜像版本。我们提供kinematic和physics两个版本。前者由动捕数据重定向得到。后者是使用Supertrack方法在物理环境中重现一遍。后者的好处是不会有地面穿透，脚

底打滑等非物理问题。缺点是动作比较保守，且有时候会抖动(双臂平举时比较明显)。可以根据需要进行选取。

提交

需要提交的文件是 `answer_task1.py` 和 `answer_task2.py` 。 如果是learning-based需要提交一个预训练好的模型。如果自己切割了一些bvh需要上传可以一起上传。如果文件过大可以附加链接。但是尽量不要太大避免助教无法从网盘成功下载。。。

如有问题或Bug，可以在issue，QQ群和我们的课程讨论版内进行讨论。