What does it mean when we talk about cinematography?

- 当我们谈及电影镜头时，我们能想到什么？

  - 特写、近景、中景、远景……
  - 人物、风景……
  - 构图、One-Third Rule、Hitchcock Rule……
  - 镜头语言……

- 电影中的镜头都是在传递信息：
  - 心理活动
  - 剧情
  - 主题
  - ……

- 观众通过镜头**被动（passively）地**接受电影所传达的信息，但由于电影本身的非实时性交互，所以需要观众主动地再加工电影内容，形成自己的理解。
- 而游戏镜头却服务于游戏的实时性和高互动性，因此需要一个**更加稳定统一**的镜头语言，以便于玩家更加**主动（actively）地**收集处理信息。
  - 近年来有更多电影式游戏镜头，使得游戏更具艺术性。

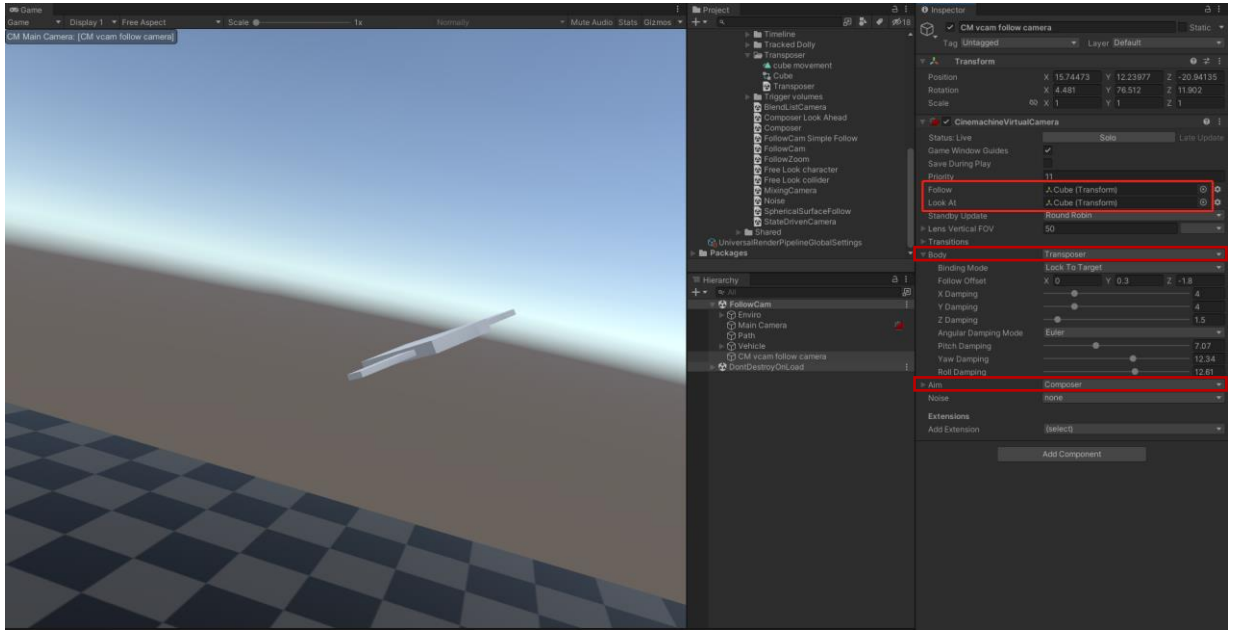
Cinematography in games

- 游戏镜头：交互式镜头+演出式镜头——互动性+观赏性

- 今天我们只介绍交互式常驻镜头;)

Interactive Camera

- 交互式镜头：以特定角色为中心，对交互指令做出响应的规则性镜头
  - 典型的指令包括：移动、朝向、FOV、远近……
  - 在Unity中，镜头如何移动（跟着谁走）和如何旋转（看向哪里）是最重要的两个因素，分别被称为Follow点和LookAt点：



  - Body选项决定了相机**如何跟随Follow点移动 (Position)**，而Aim选项决定了相机**如何看向LookAt点 (Rotation)**

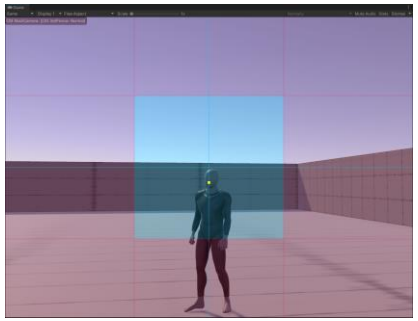
- **下面通过几个例子说明交互式镜头的常见要素，以及如何快速实现类似效果**

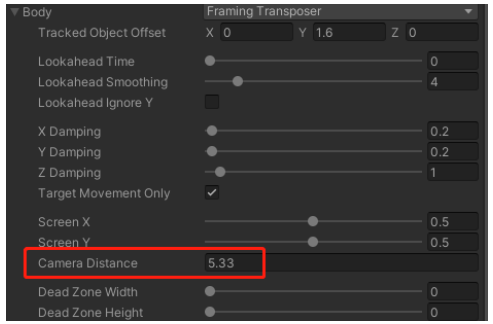
在Unity中下载Cinemachine Package，内含多个官方Demo，是很好的学习材料

原神

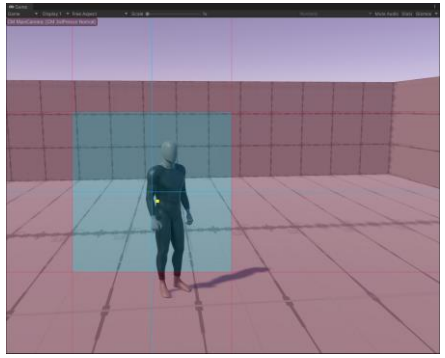
- 原神镜头跟随角色移动，同时可以自由拖动相机看向角色不同方位
  - 因此使用**Body——Framing Transposer + Aim——POV**的组合
    - Framing Transposer: 让Follow点始终保持在屏幕的某个位置
    - POV: 根据玩家输入指令旋转相机

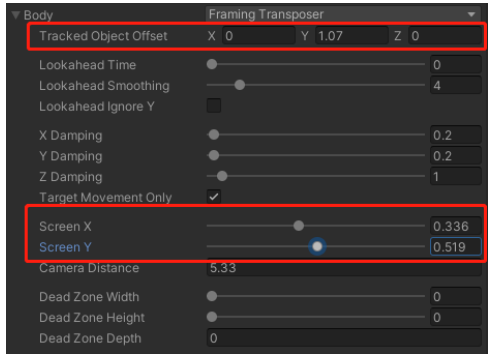
- 1）相机到**实际Follow点**的距离：Camera Distance



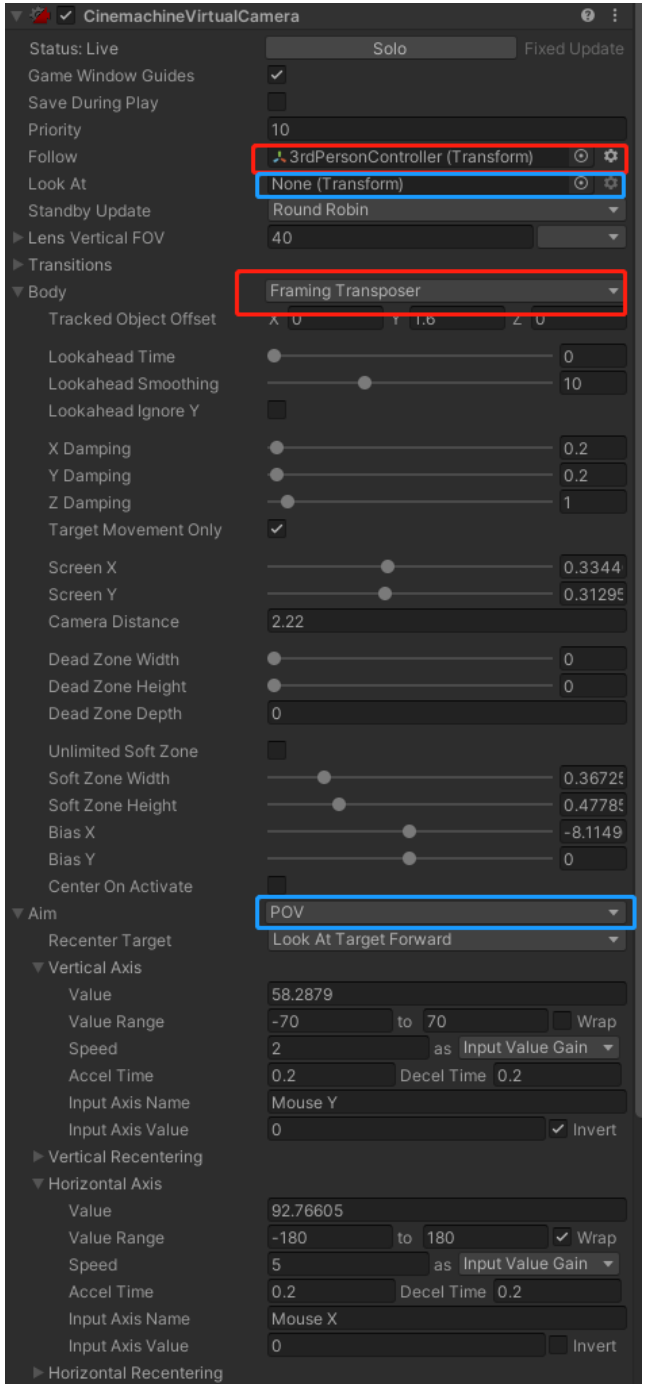


- 2）Follow点在屏幕中的位置：Tracked Object Offset + Screen X + Screen Y



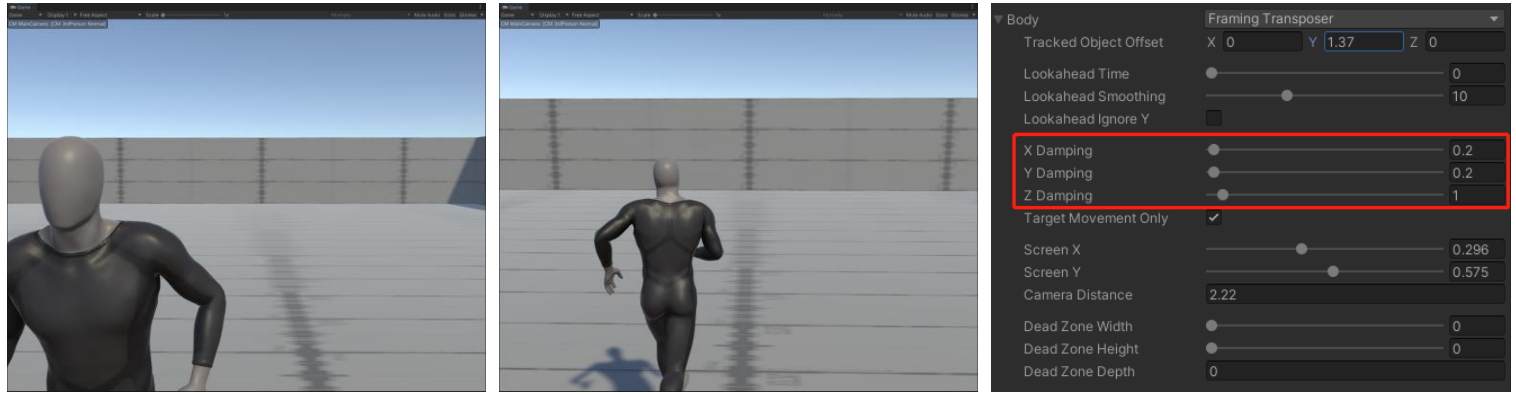


  - Tracked Object Offset不改变Follow点在屏幕中的位置，但是Screen X/Y改变了Follow点在屏幕中的位置，Offset修改的是**实际的Follow点位置**

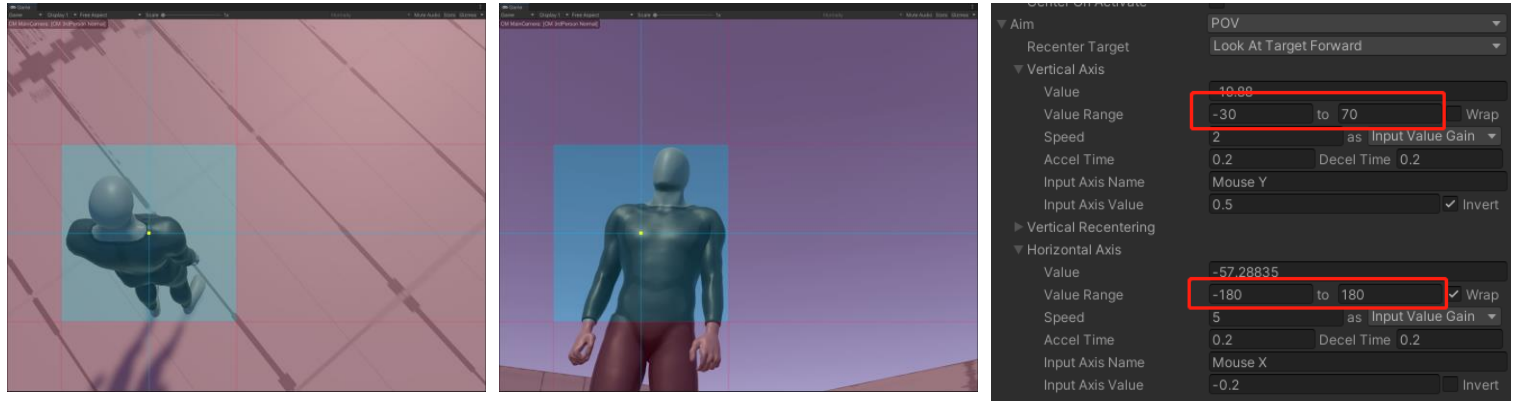


原神

- 原神镜头跟随角色移动，同时可以自由拖动相机看向角色不同方位
  - 因此使用**Body——Framing Transposer + Aim——POV**的组合
    - Framing Transposer: 让Follow点始终保持在屏幕的某个位置
    - POV: 根据玩家输入指令旋转相机
- 3）移动Damping：X/Y/Z Damping



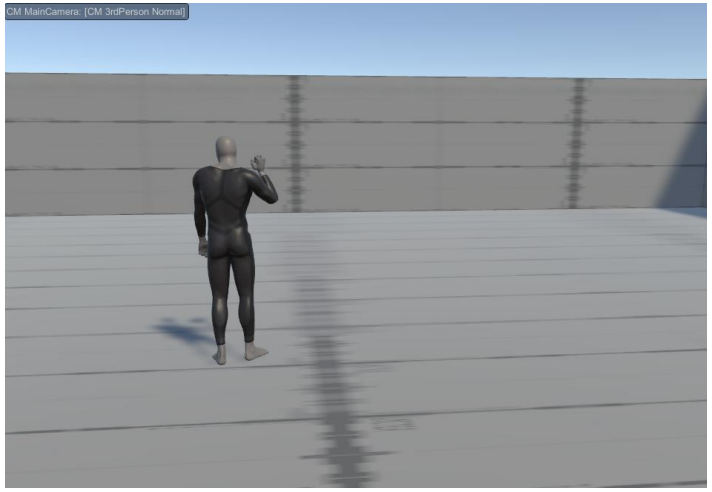
- 4）转动的最大角度：

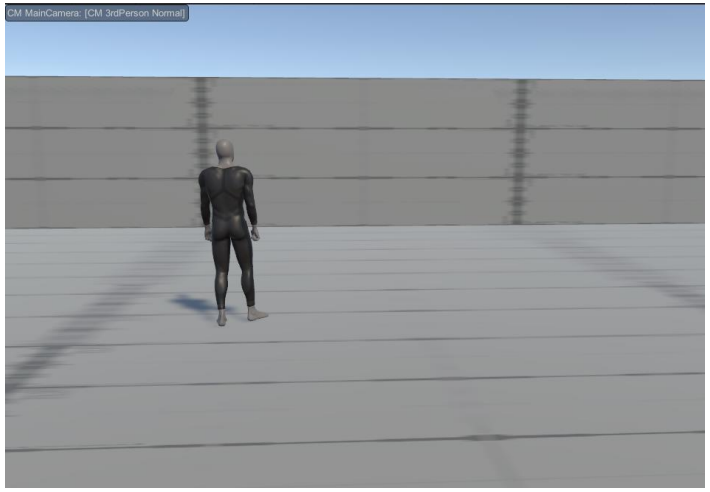


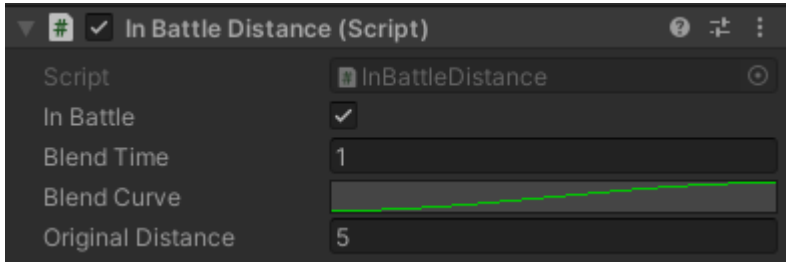
油管上一老哥复刻原神镜头：
https://www.youtube.com/watch?v=bdMAfZBoG4U&list=PL0yxB6cCkoWKuPoh_9dSvdltQENVx7YTW&index=8&t=179s

原神

- 原神镜头跟随角色移动，同时可以自由拖动相机看向角色不同方位
  - 因此使用**Body——Framing Transposer + Aim——POV**的组合
    - Framing Transposer: 让Follow点始终保持在屏幕的某个位置
    - POV: 根据玩家输入指令旋转相机
- 5）战斗时镜头距离增大：



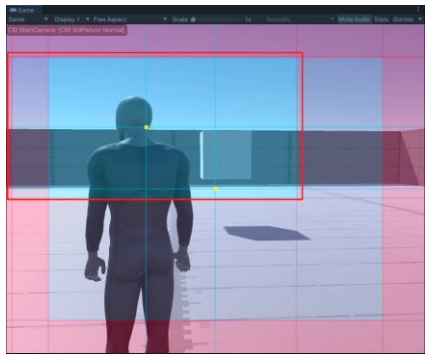


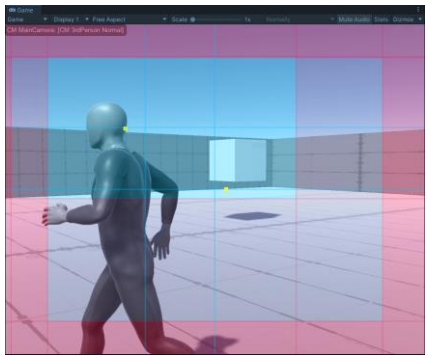


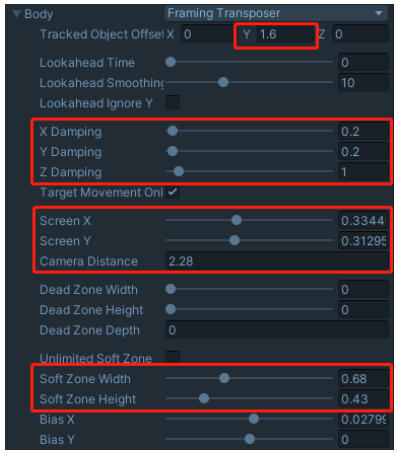

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Cinemachine;

/// <summary>
/// An add-on module for Framing Transposer camera that changes the camera distance parameter
/// in and out battle state.
/// </summary>

[ExecuteAlways]
[SaveDuringPlay]
[AddComponentMenu("")]
public class InBattleDistance : CinemachineExtension
{
    [Tooltip("Change camera distance when in and out battle")]
    public bool inBattle = false;

    [Tooltip("Blend in and blend out time")]
    public float blendTime = 1f;

    [Tooltip("Curve when blending")]
    public AnimationCurve blendCurve = AnimationCurve.EaseInOut (0, 1f, 1, 1.2f);

    [Tooltip("Original camera distance")]
    public float originalDistance = 3f;

    private float elapsedTime = 10f;
    private bool prevBattle = false;

    /// <summary>
    /// Applies distance blending when in and out battle
    /// </summary>
    /// <param name="vcam">The virtual camera being processed</param>
    /// <param name="stage">The current pipeline stage</param>
    /// <param name="state">The current vm state</param>
    /// <param name="deltaTime">The current applicable deltaTime</param>
    protected override void PostPipelineStageCallback (
        CinemachineVirtualCameraBase vcam,
        CinemachineCore.Stage stage,
        ref CameraState state,
        float deltaTime)
    {
        if (stage == CinemachineCore.Stage.Aim)
        {
            var framingTransposer = ((CinemachineVirtualCamera)vcam).GetCinemachineComponent<CinemachineFramingTransposer>();
            if (framingTransposer == null)
                return;

            if (inBattle != prevBattle)
            {
                prevBattle = inBattle;
                elapsedTime = 0f;
            }

            if (elapsedTime < blendTime)
            {
                if (inBattle == true)
                    framingTransposer.m_CameraDistance = originalDistance * blendCurve.Evaluate (elapsedTime);
                else
                    framingTransposer.m_CameraDistance = originalDistance * blendCurve.Evaluate (1f - elapsedTime);

                elapsedTime += deltaTime;
            }
        }
    }
}
```

对马岛之魂：决斗镜头

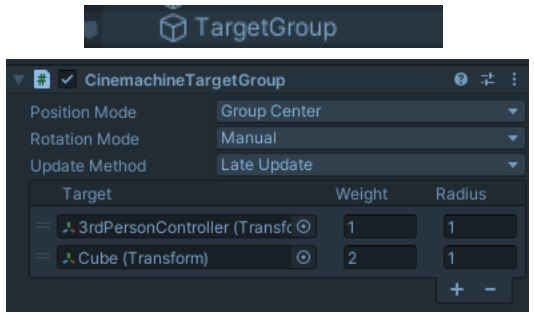
- 对马岛的决斗镜头在跟随主角的同时也看向了主角和敌人
  - 因此使用**Body——Framing Transposer + Aim——Composer**的组合
    - Framing Transposer: 让Follow点始终保持在屏幕的某个位置
    - Composer：相机始终看向LookAt点
- 1）Follow点在屏幕中的位置与Damping：

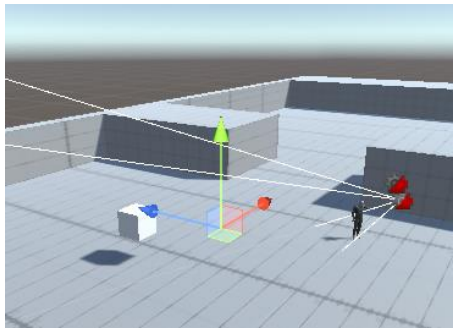


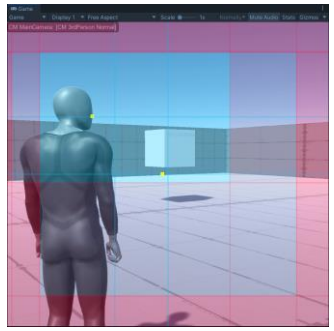


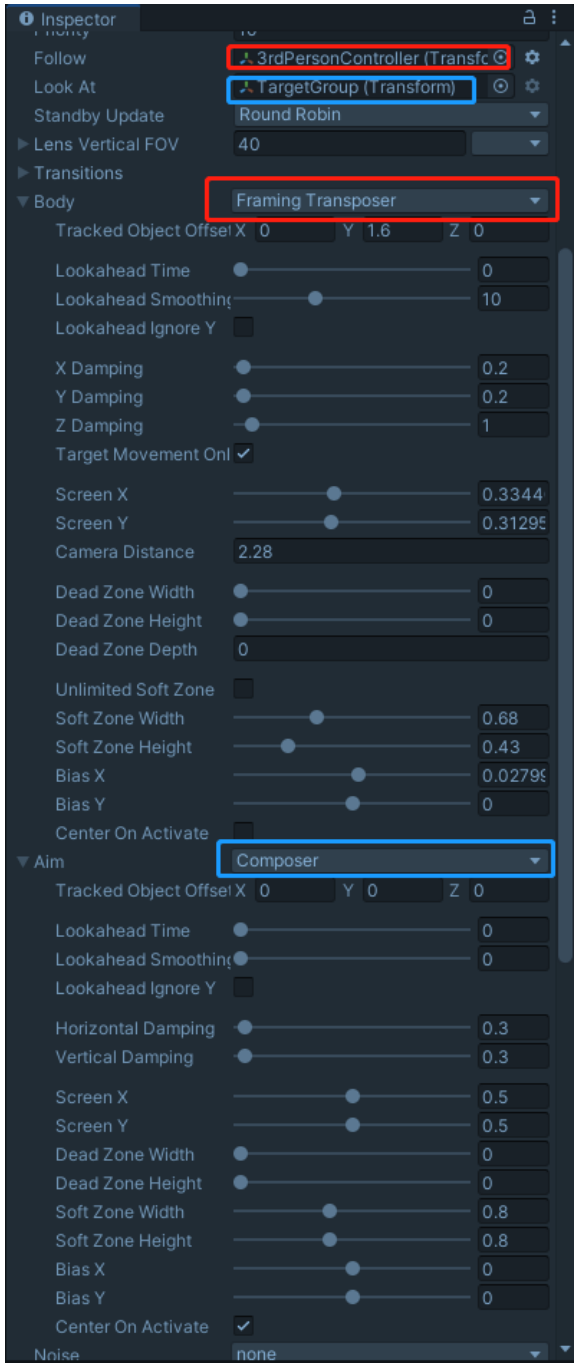


- 2）设置TargetGroup（多个物体形成的看向点）：
  - 新建空物体→挂CinemachineTargetGroup脚本→添加子物体并设置权重→将LookAt点设置为该GameObject



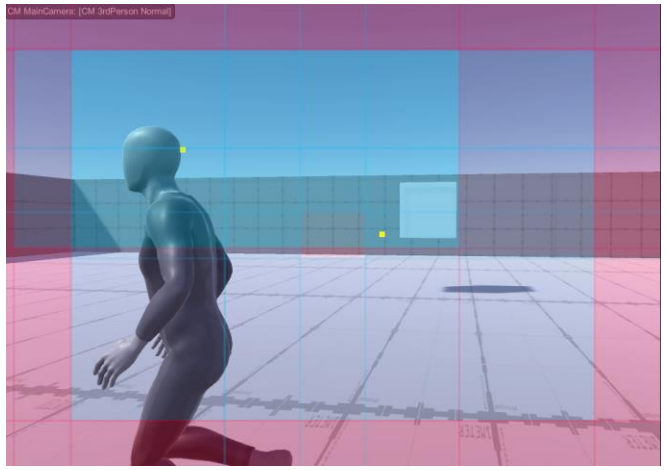






对马岛之魂：决斗镜头

- 对马岛的决斗镜头在跟随主角的同时也看向了主角和敌人
  - 因此使用**Body——Framing Transposer + Aim——Composer**的组合
    - Framing Transposer: 让Follow点始终保持在屏幕的某个位置
    - Composer：相机始终看向LookAt点
- 3）设置Composer参数：



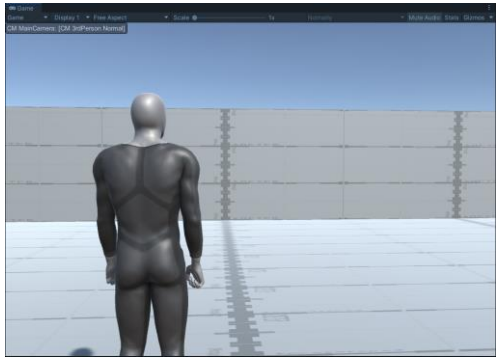
  - Tracked Object Offset：实际LookAt点的偏移；
  - Horizontal/Vertical Damping：水平和垂直方向的Damping；
  - Screen X/Y：看向点在屏幕空间中的位置；
  - Dead Zone Width/Height：看向点在该区域内相机不会改变；
  - Soft Zone Width/Height：看向点在该区域内相机会根据Damping转动。

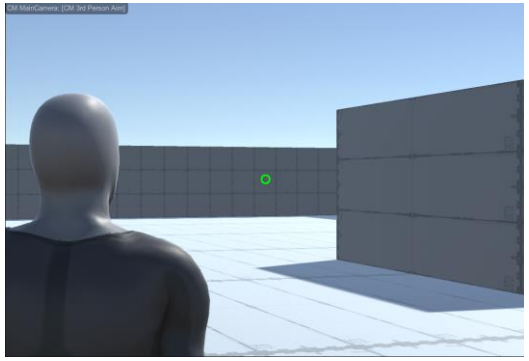
最后生还者2：射击镜头

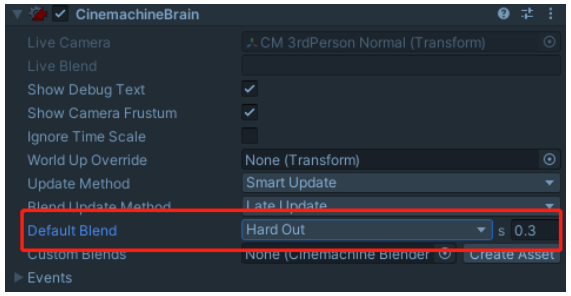
- TLOU的常规镜头仍然和之前介绍的一样，但射击镜头会跟着人物朝向转动
  - 因此采用**Body——3rd Person Follow + Aim——Do Nothing**的组合
    - 3rd Person Follow：与Follow保持固定的相对位置与朝向
    - Do Nothing：不使用LookAt点

TLOU.mp4

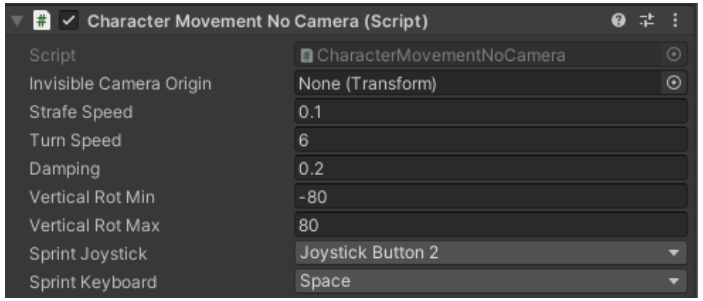
- 1）常规镜头与射击镜头拟合：

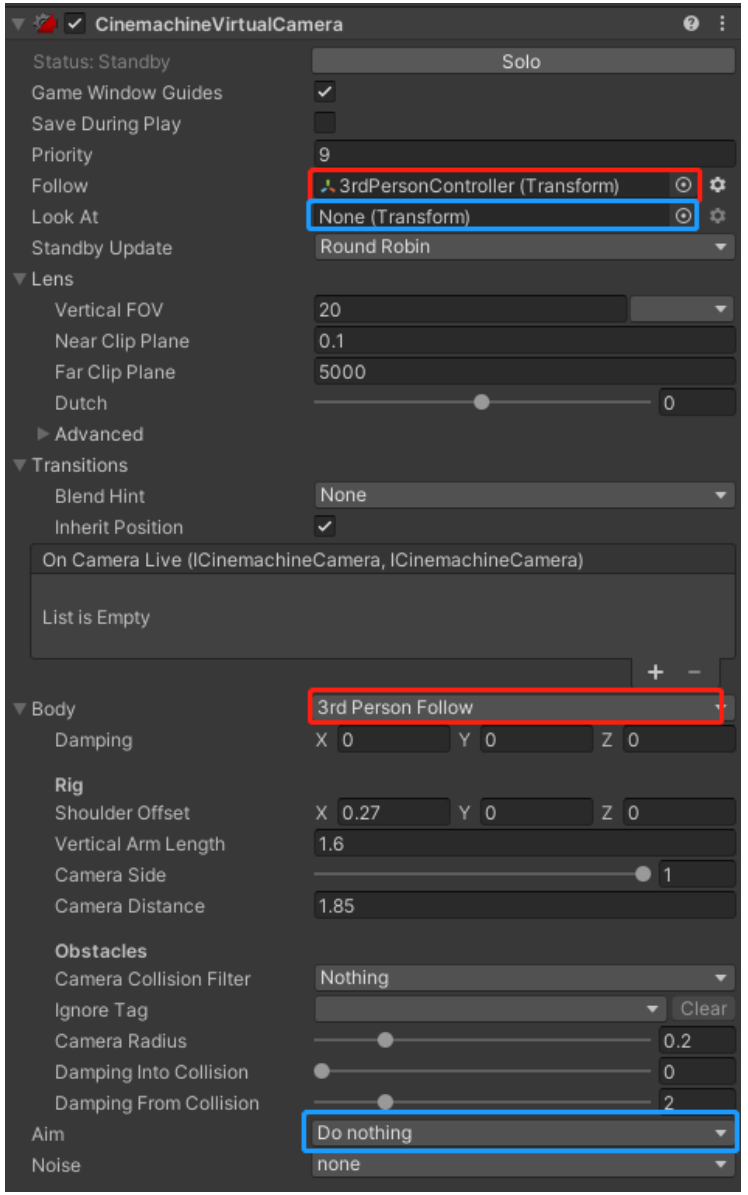






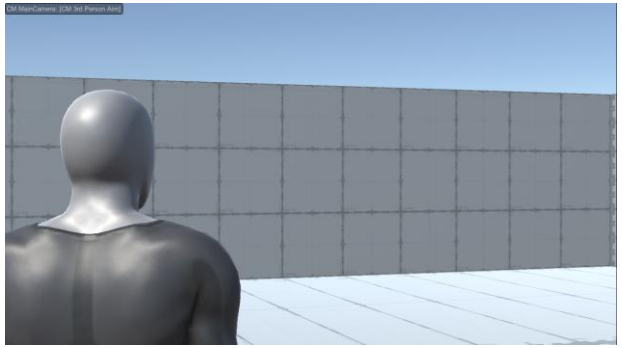
- 2）创建三人称相机：
  - 新建GameObject→添加CinemachineVirtualCamera脚本→添加Follow点→设置Body为3rd Person Follow，此外，还需要添加脚本控制角色转向

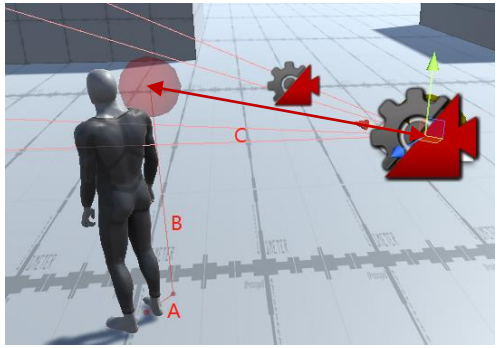


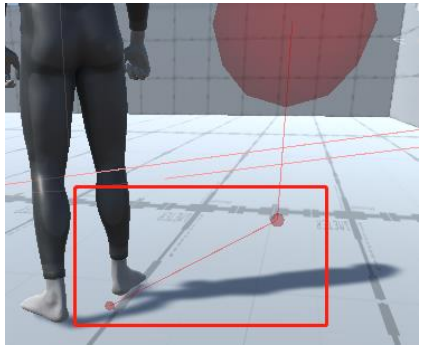


最后生还者2：射击镜头

- TLOU的常规镜头仍然和之前介绍的一样，但射击镜头会跟着人物朝向转动
  - 因此采用**Body——3rd Person Follow + Aim——Do Nothing**的组合
    - 3rd Person Follow：与Follow保持固定的相对位置与朝向
    - Do Nothing：不使用LookAt点
- 3）调整3rd Person Follow参数：







  - Damping：相机跟随的Damping
  - Shoulder Offset：距离Follow点的Offset量，如上图A段所示
  - Vertical Arm Length：在Offset之后的垂直高度偏移量，确定了**实际Follow点**，如上图B段所示
  - Camera Side：相机放在哪一侧
  - Camera Distance：相机到实际Follow点的距离，如上图C段所示
  - Obstacles：有关碰撞的参数

战神

POV.mp4　　Damping.mp4

- 观看战神的常规镜头表现，尝试回答下面的问题：
  - 1）你觉得可以用怎样的Body+Aim组合？
  - 2）水平旋转镜头时的速度变化如何？为什么要这么做？
  - 3）垂直旋转镜头的速度变化如何？
  - 4）站立和跑步状态切换时镜头表现是什么？如何实现？
  - 5）为什么现在很多RPG/ARPG游戏都把人物放在屏幕左侧？战神和他们有什么不同？为什么？








References

- Cinemachine Documentation, https://docs.unity3d.com/Packages/com.unity.cinemachine@2.9/manual/index.html
- Genshin Impact Movement in Unity | #7 – Adding the Player Camera, https://www.youtube.com/watch?v=bdMAfZBoG4U&list=PL0yxB6cCkoWKuPoh_9dSvdItQENVx7YTW&index=8&t=179s
- Creating a Deeper Emotional Connection: The Cinematography of God of War, https://www.youtube.com/watch?v=z0-ddTqz0XE
- Designing and Implementing a Dynamic Camera System, https://www.gdcvault.com/play/192/Designing-and-Implementing-a-Dynamic
- Keyframes and Cardboard Props: The Cinematic Process Behind 'God of War', https://www.youtube.com/watch?v=MNinZWlhprE
- Behind the Scenes of the Cinematic Dialogues in The Witcher 3: Wild Hunt, https://www.youtube.com/watch?v=chf3REzAjgI
- Procedural Generation of Cinematic Dialogues in Assassin‘s Creed Odyssey, https://www.youtube.com/watch?v=DFM5zbekZ7c