第 5 章

CHAPTER 5

主 体 运 动

5.1 连续移动

VR 中的移动主要有两种模式，即连续移动和传送移动。本节先介绍连续移动的实现方式。

5.1.1 连续移动与传送移动的比较

VR 开发中的连续移动和传送移动的主要不同如下。

(1) 连续移动(Continuous Locomotion)：在连续移动模式中，玩家可以通过持续操控控制器或控制器输入来平滑地在虚拟世界中移动。这种模式通常使用控制器的摇杆或触摸板来控制移动方向和速度。

连续移动模式通常需要玩家适应虚拟现实的晕眩感，因为玩家身体没有实际移动，但视觉上感觉自己在移动，因此一些用户可能会感到晕眩或不适，这种不适感也被称为晕动。

(2) 传送移动(Teleportation)：传送移动是一种更舒适的虚拟现实移动模式，它通过点对点跳跃的方式允许玩家在虚拟环境中移动。

首先玩家使用控制器来选择目标位置，然后按下按钮进行传送移动。这种方式避免了连续移动可能引发的晕眩感，因为玩家不会经历平滑的移动过程，而是立即出现在新的位置，因此传送移动有助于解决虚拟现实中的晕动问题，因为它减少了身体与视觉之间的不一致感觉。

选择使用哪种移动模式取决于项目的需求和目标受众。连续移动提供了更自然的移动体验，但可能会导致晕眩感；而传送移动则更舒适，适用于更广泛的受众，特别是那些容易晕动的用户。在某些游戏中会提供两种移动模式的选择，以让玩家根据他们的个人偏好选择移动模式。

5.1.2 配置连续移动

为了实现连续移动，需要在 Unity 中进行如下设置：

在 Hierarchy 中选中 XR Origin，在 Inspector 面板单击 Add Component 按钮，在弹出的输入框中搜索 Locomotion System。这个新追加的 Locomotion System 组件的 XR Origin 属性需要关联一个 XR Origin 对象，因此需要将相同 Inspector 面板上方的 XR Origin 组件直接拖曳到 XR Origin 属性框中。这个 Locomotion System 组件的作用是接收移动信号。

下面系统讲解关于 Unity VR 开发中的 Locomotion System。

在 Unity 中进行 VR 开发时，Locomotion System（运动系统）用于控制玩家在虚拟世界中的移动。这个系统允许玩家在虚拟现实环境中自由移动，而不仅是杵在一个静态的位置上。

Unity VR 开发中应用 Locomotion System 可以实现的移动主要有下面几种类型。

（1）Teleportation（传送移动）：传送移动是 VR 中常见的一种移动方式。玩家可以使用 VR 控制器选择一个目标点，然后瞬间传送到该位置。这种方式可以减少晕动感，并且容易控制。

（2）Continuous Locomotion（连续移动）：连续移动是模拟现实中的步行或奔跑。玩家可以使用 VR 控制器或者其他输入设备来控制角色在虚拟环境中连续自由移动。这种方式的移动更仿真自然但也更容易引起晕动。这里从玩家角度介绍一个使用连续移动但降低晕动的办法：前、后、左、右连续移动时可以使用控制器，转向时则不要使用控制器，而是依靠自己在现实中的物理转向来实现虚拟空间中的同步转向。这样可以大大地降低虚拟空间与现实空间的变化不一致，从而减轻使用 VR 头盔时的晕动。

（3）Blink Locomotion（眨眼移动）：严格来讲，眨眼移动是一种优化过的传送移动。本身也是采用将玩家瞬间传送到新位置的方式，但与纯粹的传送移动不同，这种方式通常在传送过程中显示一个短暂的黑屏或过渡效果，其作用也是减少晕动引起的不适感。

（4）Node-based Locomotion（基于节点的移动）：基于节点的移动也属于传送移动的特例。限制玩家在虚拟世界中向着预定的路径或点移动。这种方式适用于需要限制玩家移动的场景，例如解谜游戏或观光导览应用。

（5）Hand-based Locomotion（基于手的移动）：这是一种更为仿真的运动，玩家可以使用 VR 控制器来模拟行走或飞行的手势，例如挥动双臂以进行移动。这种方式可以增加沉浸感，但需要更多的交互设计和学习成本。

在以上 5 种常见运动系统中，传送移动和连续移动是其他几种运动系统的基础，应用最广泛，本书也将着重介绍这两种基本运动系统的构建方法。

保持选中 XR Origin，在 Inspector 面板中单击 Add Component 按钮追加组件，在弹出的输入框中搜索 ContinuousMoveProvider（Action-based）并单击完成追加。这个组件用于实现连续移动。

保持选中 XR Origin，在 Inspector 面板中单击 Add Component 按钮追加一个组件，在弹出的输入框中搜索 CharacterController，这个组件用于控制 XR Rig 响应移动指令的方式，例如该组件下的 SlopeLimit 的默认值为 45，意味着限制 XR Rig 无法登上斜度超过 45°

的地面。StepOffset 属性则定义了玩家可以跨过的最高台阶。这里先把所有 CharacterController 组件下的属性保留默认值即可。只需把 Radius 设置为 0.2，这个合理的半径值可以避免因为与其他物体碰撞而阻碍 XR Rig 的移动，让移动体验更精细。

回到 Inspector 面板的 ContinuousMoveProvider(Action-based)组件部分，将 Inspector 面板上方的 LocalmotionSystem 组件拖曳到 ContinuousMoveProvider(Action-based)组件的 System 属性中。EnableStrafe 属性用于控制是否能侧面移动，默认勾选。Use Gravity 属性决定是否让重力生效，默认勾选。Gravity Application Mode 属性有两个选项：Attempting Move 表示仅在移动时让重力生效，Immediately 表示总是让重力生效，此处改选为 Immediately。Forward Source 属性则决定以什么为参照来确定哪个方向是前方。一般符合经验的是将视线方向作为前方，因此在 Hierarchy 中展开 XR Origin→Camera Offset，将 Main Camera 拖曳到 Forward Source 属性框中。这样移动时就会以 Main Camera(也就是以头盔的正面方向)作为移动的 Forward 方向。

接下来设置 ContinuousMoveProvider(Action-based)组件下的控制器按键属性，也就是用哪个按键触发连续移动。由于习惯上连续移动在游戏中使用左控制器的摇杆完成，所以这里对 Left Hand Move Action 部分下的属性进行设置。如果希望用右手控制器摇杆触发连续移动，则需要用相同方法对 Right Hand Move Action 部分下的属性进行设置。

左手控制器动作的设置和之前设置 Pinch 和 Grip 动画动作时类似，先勾选 Left Hand Move Action 部分下的 Use Reference 属性选框，单击 Reference 属性右边的小圆点按钮，在弹出的对话框列表中搜索 XRI Lefthand Locomotion/Move，选中该对象完成关联设置。

5.1.3 避免主体坠落

此时还有一点设置工作没有完成，由于 XR Origin 对象在 Inspector 的 CharacterController 中应用了 Gravity 属性，并且目前 XR Origin 的 Transform Position 属性使 XR Origin 的初始位置穿透了地面，如图 5-1 所示。

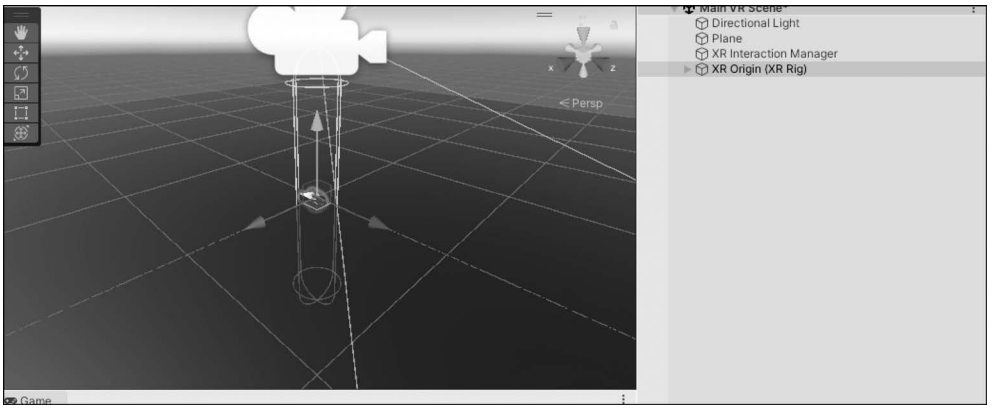

图 5-1 玩家碰撞体的初始位置一半在地面以下

目前如果运行 Unity 项目，玩家会在启动时就穿过地面落下无尽深渊。为了解决这个问题，只需把 XR Origin 对象 Transform Position 的 y 轴参数位置提升到地面上方。

在 Hierarchy 中选中 XR Origin，在 Inspector 面板中将 Character Controller 组件下 Center 的 y 轴属性值设置为 1 即可。设置完成后，代表玩家的碰撞体提升到了地面以上，如图 5-2 所示，这样运行测试后就不会坠落了。

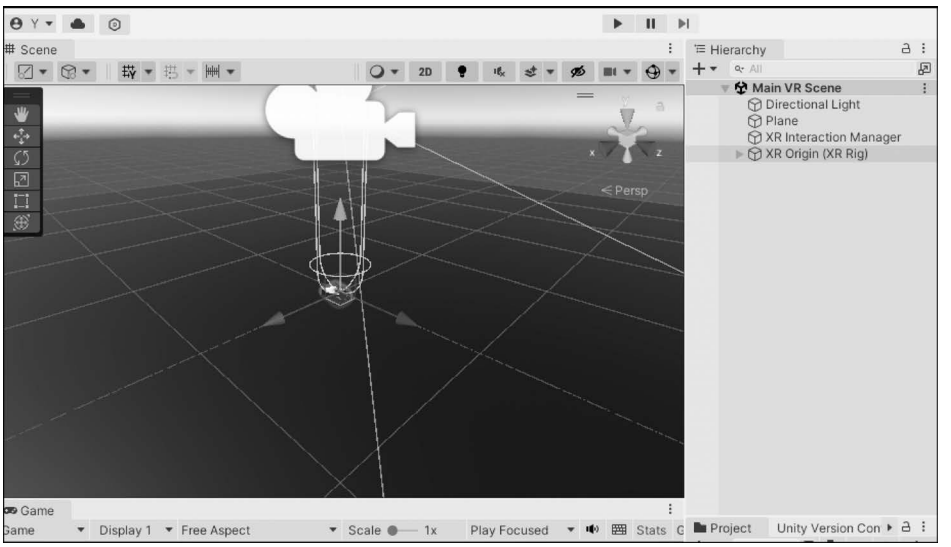

图 5-2　玩家碰撞体提升到了地面以上

XR Origin 的 Inspector 面板目前完整的设置如图 5-3 所示。

5.1.4　测试和总结

单击 Unity 编辑器的 Play 按钮，戴上 VR 头盔观察游戏场景，发现扳动左手控制器摇杆时，玩家在游戏场景中也相应地实现了连续移动。如果由于此时没有参照物而无法在头盔视野中判断自身是否移动，则可以摘下头盔，在 Unity 编辑器中选中 Hierarchy 的 XR Origin，然后观察操作左手控制器的摇杆时，XR Origin 对象的 Transform 的 Position 属性中 x 轴和 z 轴的数值是否发生连续变化，如图 5-4 所示。

本节的主要操作步骤如下：

在 Hierarchy 中选中 XR Origin，在 Inspector 面板中追加组件 LocamotionSystem，将 XR Origin 拖曳到 LocamotionSystem 组件的 XR Origin 属性框中。继续在 Inspector 面板中追加组件 ContinuousMoveProvider（Action-based），将相同的 Inspector 面板下的 LocalmotionSystem 组件拖曳到 ContinuousMoveProvider(Action-based)组件的 system 属性框中。勾选 EnableStrafe 属性框，将 Gravity Appliction Mode 属性设置为 Immediately，保持 Use Gravity 属性框默认勾选，从 Hierarchy 中将 XR Origin→Camera Offset→MainCamera 拖曳到 Forward Source 属性框中。

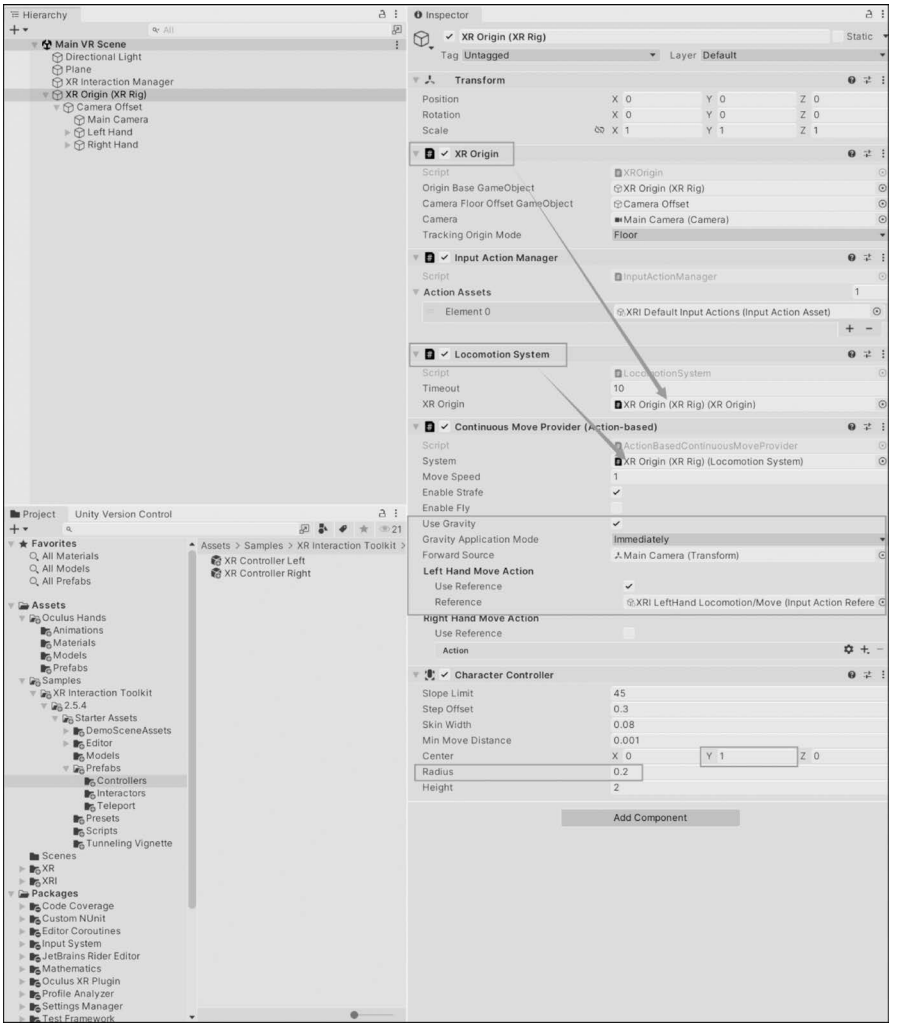

图 5-3　XR Origin 的完整设置

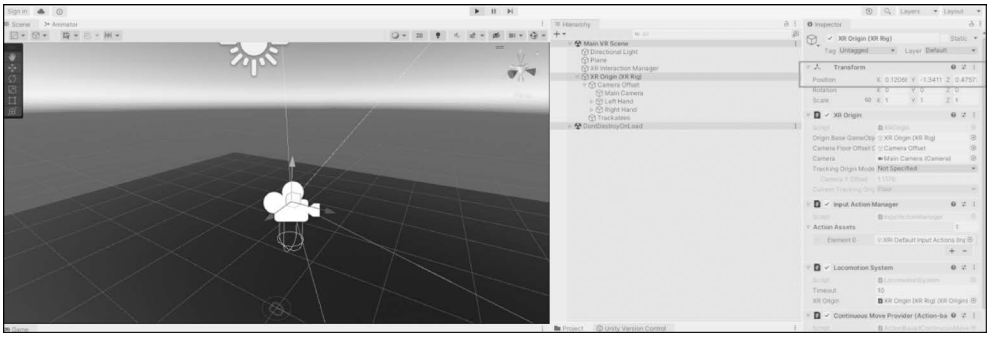

图 5-4　通过观察 Unity 编辑器中 XR Origin 的 Position 数据的变化判断是否成功实现连续移动

继续在 ContinuousMoveProvider(Action-based)面板中找到 Left Hand Move Action 部分,勾选 UseReference,Action 选择 XRI Lefthand Locomotion/Move。

在 Inspector 面板中继续追加组件 CharacterController,将 Radius 设置为 0.2,将 Center 设置为 1,其他设定保持原样。

5.2　转向运动

目前成功实现了玩家主体 Rig 通过左手控制器摇杆进行连续移动,接下来继续研究如何在 VR 中实现平滑转向。

5.2.1　VR 游戏中常见的转向方式

和移动一样,在 VR 中传统上有两种基本的转向方式,即连续(平滑)转向和传送(分段)转向。本节将同时实现并测试这两种转向运动。

平滑转向就是符合现实经验的平滑角度的转向,分段转向则是固定角度(例如 45°)为最小单位的单步转向。除此之外 VR 开发中还有其他形式的转向方式,但最基本和最常用的仍然是平滑转向与分段转向。游戏设计者需要根据需求场景结合合理的互动设计决定采用哪种转向方式,或者提供玩家在两种转向方式间切换的选择。

下面具体介绍 Unity 中总共有多少种较为常见的转向方式。

在 Unity VR 开发中,有不同的方式实现玩家在虚拟现实环境中进行转向,以便更好地适应不同的应用场景和用户体验需求。一些常见的转向方式如下。

(1) 基于控制器的旋转(Controller-based Rotation):这种方式通过 VR 控制器上的摇杆来实现玩家的旋转。玩家可以操作摇杆来左右旋转角色或摄像机的朝向。这种方式通常适用于连续移动或传送运动方式,以提供更自然的转向体验。

(2) 基于头部的旋转(Head-based Rotation):头部旋转方式是根据玩家的头部转动来旋转视角。当玩家转动头部时,摄像机或角色的朝向也相应旋转,模仿了真实世界中的头部转动。这种方式可以提供一种更沉浸的转向感觉,但也可能会导致晕动感。因为现实中只是头动了但身体没动。

(3) 基于定点的旋转(Point-and-Click Rotation):这种方式通过玩家使用 VR 控制器指向某个目标点加方向,传送后实现旋转。玩家可以先用控制器指向他们希望朝向的目标点,然后继续用摇杆来旋转确定传送后的方向。这种方式在简单游戏中比较常见,可以提供精确的控制。

(4) 基于传送的旋转(Teleportation-based Rotation):传送旋转方式与传送运动相似,玩家选择一个目标点并传送到那里,但同时也进行了旋转以适应新的方向。这种方式可以在传送过程中调整朝向,使玩家更容易适应新的位置和方向。

(5) 自动旋转(Automatic Rotation):自动旋转方式不需要玩家手动干预,而是根据场

景和目标点自动调整视角和朝向。这种方式通常用于简化操作,但可能会降低一些控制性。

可以发现,无论旋转的方式有多少种,本质上都是传送旋转和连续旋转的衍生。选择哪种转向方式取决于 VR 游戏或应用的目标场景及用户体验需求。在实际开发中,通常会根据应用的情景,或者允许玩家在设置中选择他们喜欢的方式来实现转向。同时,也需要特别关注晕动感和用户的舒适度,确保选择的方式不容易导致晕动感。

5.2.2 配置平滑转向和分段转向

在 XR Origin 上追加需要的组件。在 Hierarchy 中选中 XR Origin,在 Inspector 面板中单击 Add Component 按钮,在弹出对话框中搜索 ContinuousTurnProvider(Action-based),选中该条目完成追加。

继续追加第 2 个组件。在 Hierarchy 中保持选中 XR Origin,在 Inspector 面板中单击 Add Component 按钮,在弹出的对话框中搜索 SnapTurnProvider(Action-based),选中该条目完成追加。

在 Inspector 中观察这两个组件,发现这两个组件下都有一个 System 属性,该属性需要关联 LocalMotionSystem 对象。找到 Inspector 面板上方的 Locomotion System 组件,分别拖曳到 ContinuousTurnProvider(Action-Based)和 SnapTurnProvider(Action-based)下的 System 属性框中。

继续配置相关的控制器输入。在 Inspector 面板同时勾选 ContinuousTurnProvider(Action-based)组件下的 Right Hand Turn Action 和 SnapTurnProvider(Action-based)组件下的 Right Hand Snap Turn Action 的 UserReference 属性框。分别在两个 UserReference 属性框下的 Reference 属性框单击右侧圆点按钮,在弹出的对话框中分别搜索并选中 XRI RightHand LocoMotion/Turn 和 XRI RightHand LocoMotion/Snap Turn 完成关联。这样右手控制器的摇杆输入就分配给了平滑转向和分段转向动作。设置完成后的 Inspector 面板如图 5-5 所示。

在 Inspector 面板中继续观察 ContinuousTurnProvider(Action-based)和 SnapTurnProvider(Action-based)这两个组件的参数,了解这两个组件下各个属性的作用。

ContinuousTurnProvider(Action-based)下的属性如下。

Turn Speed:控制旋转速度,单位是度/秒,方向是顺时针。

SnapTurnProvider(Action-Based)下的属性如下。

(1) Turn Amount:指定分段转向的最小单位。

(2) Debounce Time:两次转向间的最小等待时间。

(3) Enable Turn Left Right:这个属性决定了是否允许玩家使用 SnapTurnProvider 进行左右方向的旋转。如果启用了此选项,则玩家可以使用 VR 控制器的输入(例如摇杆)来左右旋转角色或摄像机的朝向。如果禁用了此选项,则玩家将无法进行左右旋转,只能在其他方向上(例如垂直方向)旋转。

(4) Enable Turn Around:这个属性决定了是否允许玩家使用 SnapTurnProvider 进行瞬时后转。如果启用了此选项,则在向后扳动 VR 控制器摇杆时玩家可以瞬时实现 180°的

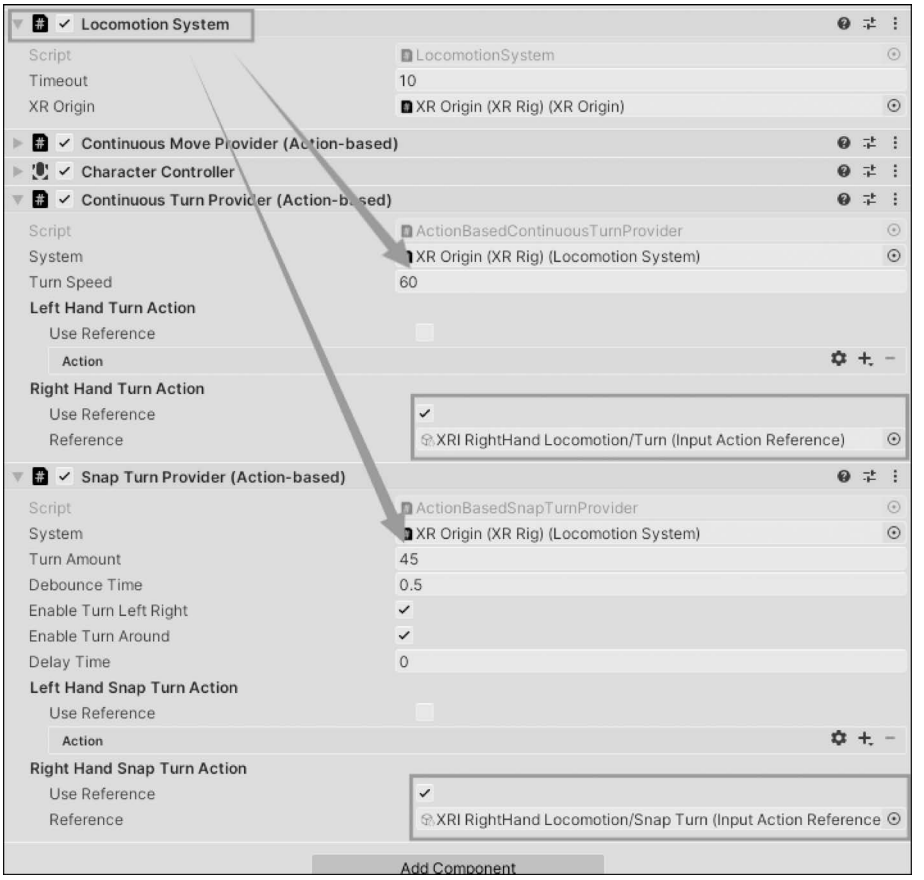

图 5-5　转向运动的相关设置

方向改变，也就是瞬间向后转。如果禁用了此选项，则玩家将只能进行逐渐旋转，而不能分段转向 180°面向身后。

(5) Delay Time：从接收到转向指令到做出转向动作间的延迟反应时间。

接下来分别测试平滑转向和分段转向的效果。

首先测试平滑转向，如果要使平滑转向设置生效，则需要在 Inspector 面板中勾选 ContinuousTurnProvider(Action-based)组件使该组件激活，同时取消勾选 SnapTurnProvider(Action-based)组件使该组件隐藏。

5.2.3　测试和总结

在 Unity 编辑器中单击 Play 按钮，戴上 VR 头盔观察游戏场景，可以观察到，通过操作遥感能够平滑转向了。

接着测试分段转向。如果要使分段转向设置生效，则需要在 Inspector 面板中取消勾选 ContinuousTurnProvider(Action-based)组件使该组件隐藏，勾选 SnapTurnProvider

(Action-based)组件使该组件激活。

再次在 Unity 编辑器中单击 Play 按钮,戴上 VR 头盔观察游戏场景,可以观察到变成了以 45°为单位进行分段转向。

可以继续修改本节介绍的两种不同旋转形式组件 ContinuousTurnProvider(Action-based)组件和 SnapTurnProvider(Action-based)组件下的属性,反复测试以体会效果。

本节的主要操作步骤如下:

在 Hierarchy 中选中 XR Origin 对象,在 Inspector 面板中追加 ContinuesTurnProvider(Action-based)组件,将相同的 Inspector 面板上的 LocalMotionSystem 组件拖曳到 ContinuesTurnProvider(Action-based)组件的 System 属性框中。勾选 ContinuesTurnProvider(Action-based)组件下 Right Hand Turn Action 部分下的 Use Reference 属性框,并将 Reference 属性设置为 XRI RightHand LocalMotion/Turn。

在 Inspector 面板中继续追加 SnapTurnProvider(Action-based)组件,将相同的 Inspector 面板上的 LocalMotionSystem 拖曳到 SnapTurnProvider(Action-based)组件的 System 属性框中。勾选 SnapTurnProvider(Action-based)组件下 Right Hand Snap Turn Action 部分下的 Use Reference 属性框,并将 Reference 属性设置为 XRI RightHand LocalMotion/Snap Turn。

如果希望测试平滑转向,则勾选 ContinuesTurnProvider(Action-based)组件使其生效,取消勾选 SnapTurnProvider(Action-Based)组件使其失效。

如果希望测试分段转向,则取消勾选 ContinuesTurnProvider(Action-based)组件使其失效,勾选 SnapTurnProvider(Action-based)组件使其生效。

5.3 使 Character Controller 跟随 XR Rig 移动

目前如果保持选中 XR Origin,在 Unity 编辑器中单击 Play 按钮,则移动头盔位置时可以发现即使 XR Origin 跟随头盔成功移动了,XR Origin 下 Character Controller 组件的胶囊状 Collider 仍然保持原地不动。这将导致 XR Origin 与 Character Controller 的位置之间产生偏差,游戏中代表玩家的 XR Origin 在运动和与环境互动时就会出现问题,如图 5-6 所示。

5.3.1 认识角色控制器

下面介绍 Unity VR 开发中的 Character Controller(角色控制器)组件的作用。

在 Unity VR 开发中,Character Controller(角色控制器)组件是用于控制角色在虚拟现实环境中移动和环境交互的一个常见组件。它允许开发者轻松地实现角色的移动、跳跃、碰撞检测等功能,是构建虚拟现实体验的重要组成部分之一。

Character Controller 组件的主要功能如下。

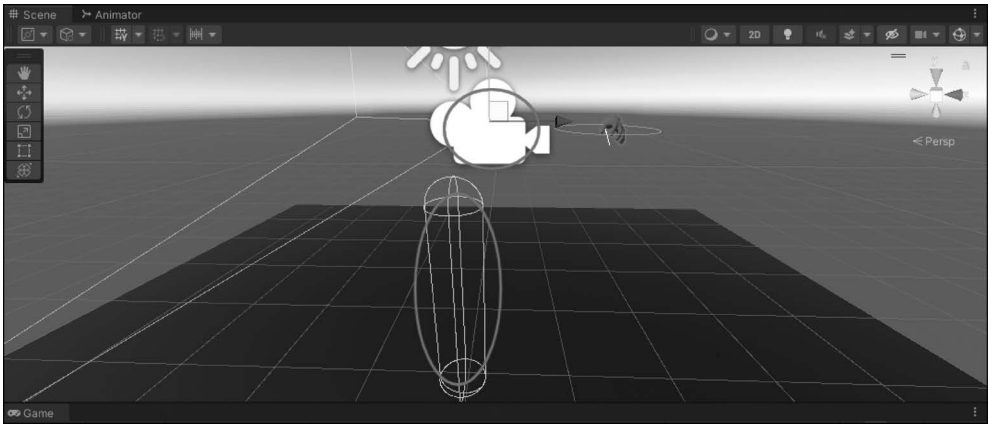

图 5-6　XR Origin 与 Character Controller 位置产生偏差

(1) 移动控制：Character Controller 组件允许开发者方便地控制角色在虚拟现实环境中进行移动。通过编程控制角色的位置和方向，开发者可以实现角色的行走、奔跑、蹲伏等动作。

(2) 碰撞检测：Character Controller 可以检测角色与环境之间的碰撞，包括与地面、墙壁、障碍物等的碰撞。这使角色能够在遇到障碍物时停止移动或进行相应的反应，从而增强了游戏或应用的真实感和交互性。

(3) 重力和跳跃：Character Controller 组件通常会考虑让重力生效，使角色能够在地面上受到重力影响，并能够进行跳跃等动作。开发者既可以通过编程来控制角色的跳跃高度和跳跃力度等参数，也可以通过属性直接设定可攀爬斜度 Slope Limit，以及可跨越高度 Step Offset 等属性。

(4) 动作响应：Character Controller 允许开发者根据角色的输入(例如玩家的控制器操作)来触发不同的动作响应，例如移动、转向、跳跃等。这使角色能够根据玩家的操作进行实时响应，并且增强了游戏或应用的交互性。

5.3.2　使 Character Controller 跟随主体移动

如果要解决 Character Controller 不跟随 XR Origin 主体移动问题，则需要追加一个 Character Controller Driver 组件。

在 Unity VR 开发中，Character Controller Driver 组件又起什么作用呢？

Character Controller Driver 组件是用于连接 VR 头盔的头部跟踪器和 Unity 的 Character Controller 组件的桥梁。它允许开发者利用头盔的位置和方向来控制角色在虚拟现实环境中的移动和旋转，以增强玩家的沉浸感和交互性。

在 Hierarchy 中选中 XR Origin，在 Inspector 面板上单击 Add Component 按钮，在弹出的对话框中搜索 Character Controller Driver，选中该条目完成追加操作。

观察 Character Controller Driver 组件的属性，可以发现存在一个 Locomotion Provider

属性。从 Inspector 面板的上方，将 Continues Movement Provider 组件拖曳到 Locomotion Provider 属性框中。Min Height 属性和 Max Height 属性用于设定代表玩家的胶囊 Collider 的有效最小高度和最大高度，这里留为默认值，Max Height 的默认值 infinity 的意思是无上限，Character Controller Driver 组件的完整设定如图 5-7 所示。

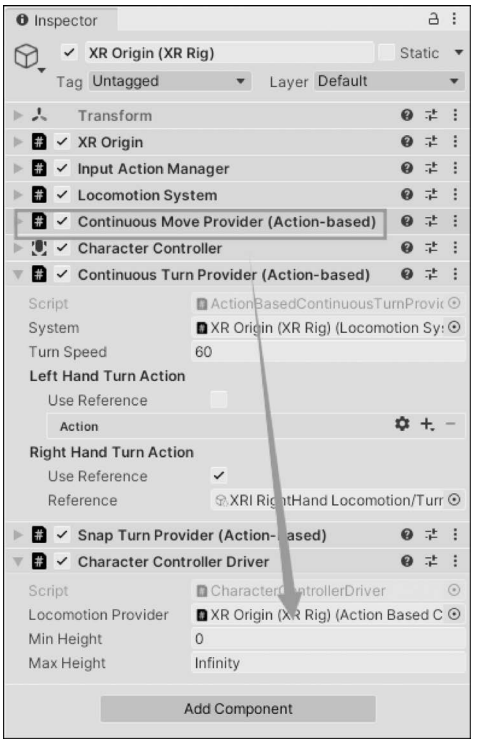

图 5-7　Character Controller Driver 组件设置

5.3.3　测试和总结

在 Unity 编辑器中单击 Play 按钮，戴上 VR 头盔观察编辑器中的游戏场景，可以观察到通过左手控制器摇杆移动玩家 XR Origin 位置后，Character Controller 组件可以正常跟随 XR Origin 移动，胶囊 Collider 的高度也随头盔的位置而变化，如图 5-8 所示。

在 Hierarchy 中选中 XR Origin 对象，追加 CharacterControllerDriver 组件，将 Inspector 上方的 ContinuesMovementProvider 组件拖入 Locomotion Provider 属性框中。

在 Unity 编辑器中运行测试，可以观察到 Character Controller 对象正常随 XR Origin 对象同步移动，高度也随头盔的高度位置而相应变化。

本节涉及的主要操作步骤如下：

XR Origin 新增 Character Controller Driver 组件。在 Inspector 面板中将 Continues Movement Provider 组件拖曳到 Locomotion Provider 属性框中。

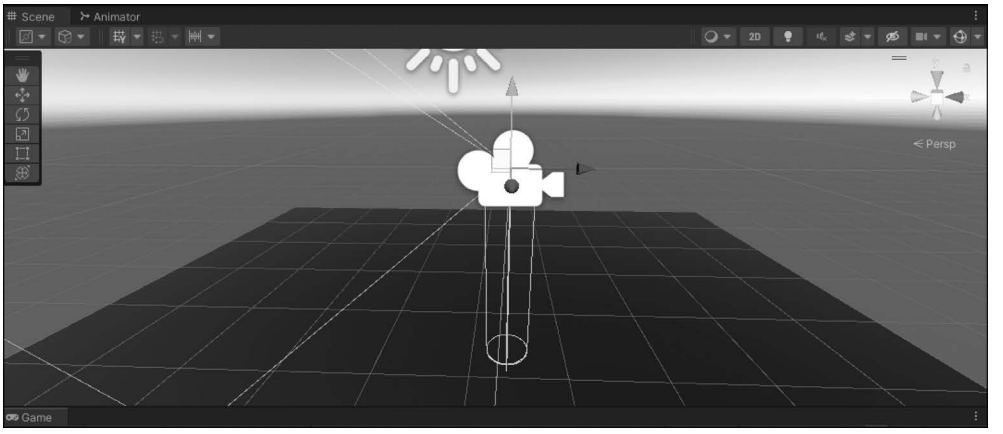

图 5-8　XR Origin 位置与 Character Controller 同步

5.4　传送移动时指示射线的呈现

5.1 节已经介绍了如何进行连续移动，接下来继续介绍如何进行传送移动。

5.4.1　认识指示射线

所谓的传送移动，也就是通过一条指示射线 Ray，选定目标地点，再使玩家的位置改变到所选目标地点的动作。

为了实现传送移动，首先需要一条指示射线 Ray。

接下来介绍传送移动时指示射线 Ray 的具体作用。

在 VR 开发中，传送移动时的指示射线 Ray 是一种可视化的工具，用于帮助玩家选择目标位置，并指示玩家在虚拟世界中将要传送到该位置。

传送移动时指示射线 Ray 的主要作用如下。

(1) 目标位置选择：传送移动通常需要玩家选择一个目标位置进行传送。指示射线 Ray 可以帮助玩家准确定位和选择目标位置。玩家通过操控 VR 控制器使指示射线 Ray 准确地瞄准他们希望传送到的位置，从而进行传送。

(2) 可视化传送路径：指示射线 Ray 通常在虚拟环境中以一条直线或曲线的形式呈现，可以清晰地展示传送的路径和目标位置。这种可视化有助于玩家了解他们将要传送到的位置，提供了更直观的传送体验。

(3) 避免碰撞：指示射线 Ray 通常会在玩家传送目标位置的周围进行碰撞检测，以确保目标位置是安全的并且不会与障碍物相交。这有助于避免玩家在传送过程中出现意外碰撞或穿过物体的情况。

(4) 提供反馈：指示射线 Ray 通常会在玩家进行目标位置选择时提供反馈，例如改变

颜色、放大或震动等。这种反馈有助于增强玩家的交互体验，并提供对传送操作的视觉和触觉确认。

在 Hierarchy 中展开 XR Origin，首先在 Camera Offset 对象上右击，选择 XR 菜单，然后在二级菜单选择 RayInteractor（Action-based）。将新追加的对象命名为 RightTeleportationRay。观察 Inspector 面板，可以发现目前该对象下包含 XR Controller 和 XR Ray Interactor 两个组件。

先介绍 XR Controller 组件的作用。

XR Controller 组件是一种用于管理和控制虚拟现实控制器的重要组件，它提供了一种统一的方式来与不同的虚拟现实设备（例如 Oculus Rift、HTC Vive、Windows Mixed Reality 等）进行交互。XR Controller 组件的主要作用如下。

（1）控制器跟踪：XR Controller 组件负责追踪虚拟现实控制器的位置和方向。它可以通过传感器和追踪设备来获取控制器的准确位置和姿态，从而实现真实的手部交互。

（2）用户输入：XR Controller 组件接收来自控制器的用户输入，包括按键、触摸板、扳机等的互动输入。开发者可以根据用户的输入来触发不同的动作、事件或交互功能。

（3）手部交互：XR Controller 组件使开发者能够轻松地实现手部交互功能，例如抓取、放置、移动、旋转目标对象等。通过追踪控制器的位置和手势，开发者可以模拟真实世界中的手部动作，并与虚拟环境中的对象进行互动。

（4）虚拟物体生成：XR Controller 组件可以用于生成虚拟物体，例如光束、射线、箭头等，用于指示玩家的目标位置、交互范围或提示信息。这些虚拟物体可以帮助玩家更直观地理解虚拟环境中的交互和场景。XR Ray Interactor 组件必须依赖于 XR Controller 组件发挥作用。

（5）交互反馈：XR Controller 组件通常会提供交互反馈，例如震动、声音、光效等，以增强用户体验和沉浸感。这种反馈可以在用户操作中提供视觉、听觉和触觉上的确认，使用户更加投入到虚拟环境中。

XR Ray Interactor 组件的作用如下。

RightTeleportationRay 对象下还有一个 XR Ray Interactor 组件，这个组件定义了指示射线与场景的互动形式。

在 Unity VR 开发中，XR Ray Interactor 组件主要用于控制射线投射，检测交互对象，并响应用户的手势和操作。XR Ray Interactor 组件的主要作用如下。

（1）射线投射：XR Ray Interactor 组件可以根据用户的手势或控制器的位置和方向发射射线，以模拟用户的触摸或指向操作。这使用户可以通过简单的手势或控制器移动来与虚拟环境中的对象进行交互。

（2）交互检测：XR Ray Interactor 组件负责检测射线与虚拟环境中的交互对象之间的碰撞和交互。它可以检测到射线与虚拟物体的交叉点、碰撞物体的信息及触发的事件等，从而实现对交互对象的准确识别和响应。

（3）交互响应：XR Ray Interactor 组件可以根据射线与交互对象的碰撞情况，触发相

应的交互事件或动作，例如，当射线与按钮、物体或 UI 元素相交时，可以触发按钮的单击、物体的抓取、UI 的选中等交互行为，从而实现丰富的用户交互体验。

(4) 物体抓取和放置：XR Ray Interactor 组件通常与物体抓取和放置功能配合使用，使用户能够通过手势或控制器来抓取、移动和放置虚拟物体。它可以检测手势动作并实时更新虚拟物体的位置和姿态，从而实现自然、直观的物体操作。

(5) UI 交互：XR Ray Interactor 组件还可以用于与虚拟现实环境中的用户界面进行交互。用户可以通过射线投射来选择、单击和操作虚拟界面上的按钮、滑块、文本框等 UI 元素，从而实现对应的功能和操作。

XR Ray Interactor 组件的相关属性分为几部分，其中 LineRenderer 部分的属性用来控制如何渲染指示射线 Ray，XR Interactor Line Visual 组件下的属性则用来控制 Ray 的样式。

XR Ray Interactor 组件下有一个 Interaction Manager 属性，此属性需要关联 Hierarchy 面板中的 XR Interaction Manager 对象。

5.4.2　配置指示射线对象

在 Hierarchy 中展开 XR Origin→Camera Offset，选中子对象 RightTeleportationRay。在 Inspector 面板找到 XR Controller(Action-based)组件，单击组件 Title 右侧的“设置”按钮，这里假定需求是通过右手控制器控制 XR Ray Interactor，所以在弹出的列表中双击选择 XRI Default Right Controller。右手控制器的相关设置会自动关联到当前的 XR Ray Interactor 组件，如图 5-9 所示。

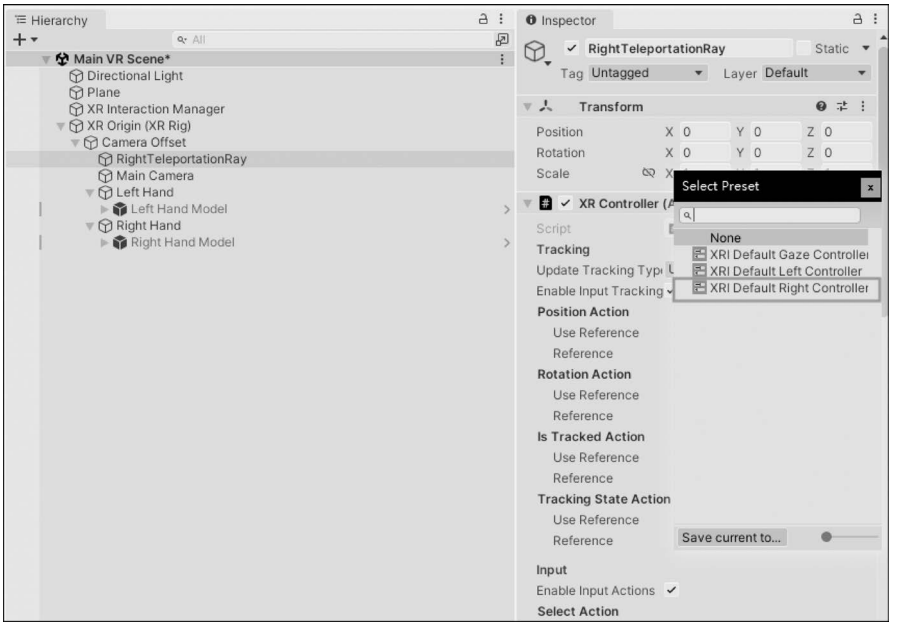

图 5-9　对 XR Controller(Action-based)应用右手控制器预设

如果希望左手控制器也能够控制 XR Ray Interactor，则只需在 Hierarchy 中选中 RightTeleportationRay 对象，按快捷键 Ctrl+D 完成复制，复制后在 RightTeleportationRay（1）对象上右击，在弹出的快捷菜单后选择 Rename，改名为 LeftTeleportationRay。

在 Hierarchy 中继续选中 XR Origin→Camera Offset 下的 LeftTeleportationRay。在 Inspector 面板中找到 XR Controller 组件，单击组件 Title 右侧的“设置”按钮，在弹出的列表中双击选择 XRI Default Left Controller。左手控制器的相关设置会自动关联到当前的 XR Controller 组件，如图 5-10 所示。

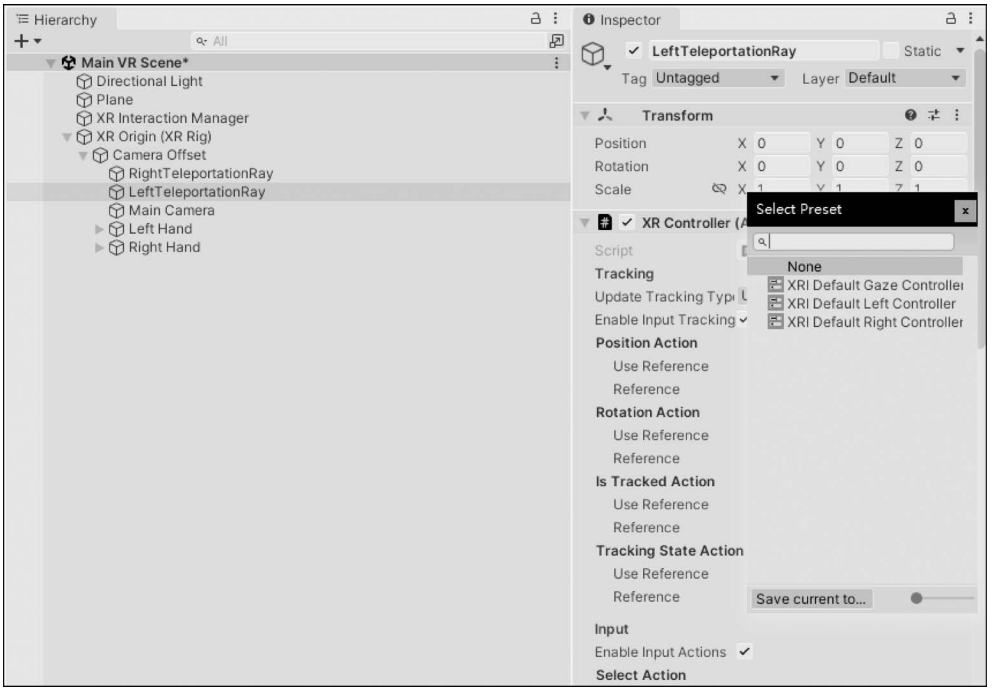

图 5-10　LeftTeleportationRay 下 XR Controller 组件套用左手控制器默认设置

5.4.3　测试和总结

在 Unity 编辑器中单击 Play 按钮开始测试，戴上 VR 头盔后观察游戏场景中的左右手，可以发现跟随左右手对象的指示射线都出现了，如图 5-11 所示。

本节实现了指示射线 Ray 的呈现，但是目前既无法通过射线产生互动，也无法实现传送移动，这将是 5.5 节介绍的内容。

本节的主要操作步骤如下：

在 Hierarchy 中展开 XR Origin→Camera Offset，在 Camera Offset 下新建子对象 RayInteractor（Action-based），将 RayInteractor（Action-based）对象重命名为 Right Teleportation Ray。在 Inspector 面板中找到 XR Controller 组件，应用预设设置 XRI

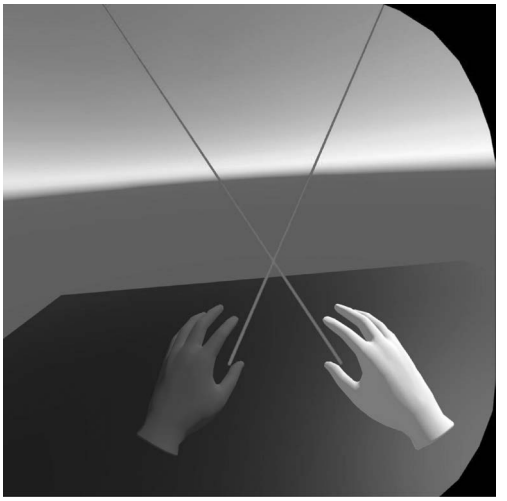

图 5-11　跟随左右手对象的指示射线

Default Right Controller。

在 Hierarchy 中复制 Right Teleportation Ray，重命名为 Left Teleportation Ray，在 Inspector 面板中找到 XR Controller 组件，应用预设 XRI Default Left Controller。

5.5　实现指定区域内的传送移动

目前已经实现了指示射线 Ray 的展现和初步设置，这些设置是实现传送移动的前提条件，本节将继续介绍如何实现传送移动。

5.5.1　配置 Teleportation Provider 组件

在 Hierarchy 中选中 XR Origin，在 Inspector 面板中单击 Add Component 按钮，在弹出的对话框中搜索 Teleportation Provider，在搜索结果中单击该条目完成添加。

观察 Inspector 面板，发现 Teleportation Provider 组件下有一个 System 属性，由于该属性需要关联一个 Locomotion System 对象，所以需要把 Inspector 面板上方的 Locomotion System 组件拖曳到 Teleportation Provider 组件下的 System 属性框中。

Teleportation Provider 组件下的另一个属性 Delay Time 表示用户发出传送指令后，传送动作实际生效之前的等待时间。在传送动作开始之前的这段延迟时间内，通常会显示一些视觉效果或提示，以告知玩家传送即将发生，并帮助准确定位目标位置。

设置较短的延迟时间可以使传送更加即时和流畅，但可能会导致玩家在传送过程中感觉突兀或不舒服，而设置较长的延迟时间则可以给玩家更多的准备时间，减少突然传送带来的不适感，但可能会降低传送的实时性和响应性。

因此，在设置延迟时间时，需要综合考虑玩家的舒适度和传送体验，通常需要通过测试

和调整来找到最佳的延迟时间设置，以确保传送操作的流畅性和舒适性。

这里将 Delay Time 属性保持默认值 0。

Teleportation Provider 组件的完整设置如图 5-12 所示。

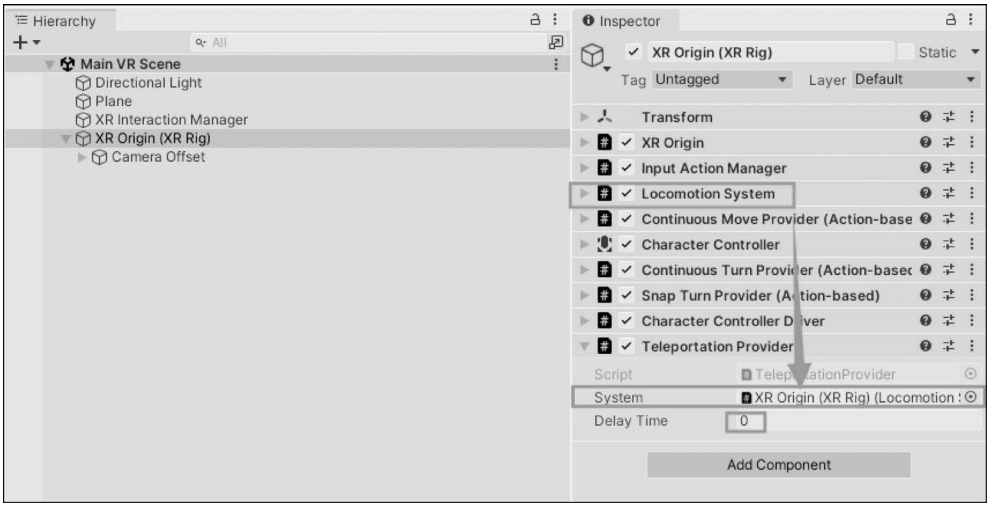

图 5-12　Teleportation Provider 组件的完整设置

5.5.2　两种传送移动

下一步需要决定传送移动的落点。在 Unity 的 XR Interacter Toolkit 中有两种类型的传送区域指定模式，即 TeleportationArea 和 TeleportationAnchor。

在 Unity VR 开发中，TeleportationArea 和 TeleportationAnchor 是用于实现指向传送功能的组件，它们通常与 VR 交互系统（例如 XR Interaction Toolkit）一起使用。

TeleportationArea（传送区域）是一个用于定义可传送的区域的组件。它通常被放置在场景中的一个区域内，表示玩家可以往该区域的任意位置传送。传送区域通常由一个或多个碰撞体定义，以确保传送范围的准确性和可靠性。

当玩家触发传送操作时，系统会检查玩家所处的位置是否位于任何 TeleportationArea 内部，如果是，则可以执行传送操作，将玩家传送到指定的目标位置。

TeleportationAnchor（传送锚点）是用于定义传送目标位置的组件。它通常是被放置在场景中的一个点，表示玩家在进行传送操作时将要传送到的目标点位。

当玩家触发传送操作时，系统会将玩家传送到 TeleportationAnchor 的位置，从而实现定点传送功能。

TeleportationAnchor 通常被设置在场景中的重要关键点，例如目标点、安全区域等。

5.5.3　配置传送区域

本节先实现第 1 种传送模式，即区域传送。

在 Hierarchy 中选中 Plane，在 Inspector 面板中单击 Add Component 按钮，在弹出的对话框中搜索 Teleportation Area，单击此条目完成添加。

Teleportation Area 组件允许开发者在场景中定义一个或多个可以进行传送的区域。这些区域可以是任意形状和大小的几何体，例如立方体、球体、圆柱体等，以满足不同场景的需求。

观察 Teleportation Area 组件下的属性，发现存在 Colliders 属性，通过 Colliders 属性关联的一个或多个 Collider 与 Right Teleportation Ray 或 Left Teleportation Ray 产生碰撞可以决定传送移动的目标地点。

在 Hierarchy 中保持选中 Plane，在 Inspector 面板中将上方的 Mesh Collider 组件拖曳到 Teleportation Area 组件的 Collider 属性 Title 上，发现 Colliders 数组下多了一个 Element 0，此 Element 0 关联了 Mesh Collider 组件，表示 Plane 所在范围已被设置为可以进行传送移动的区域，此时的 Teleportation Area 组件设置如图 5-13 所示。

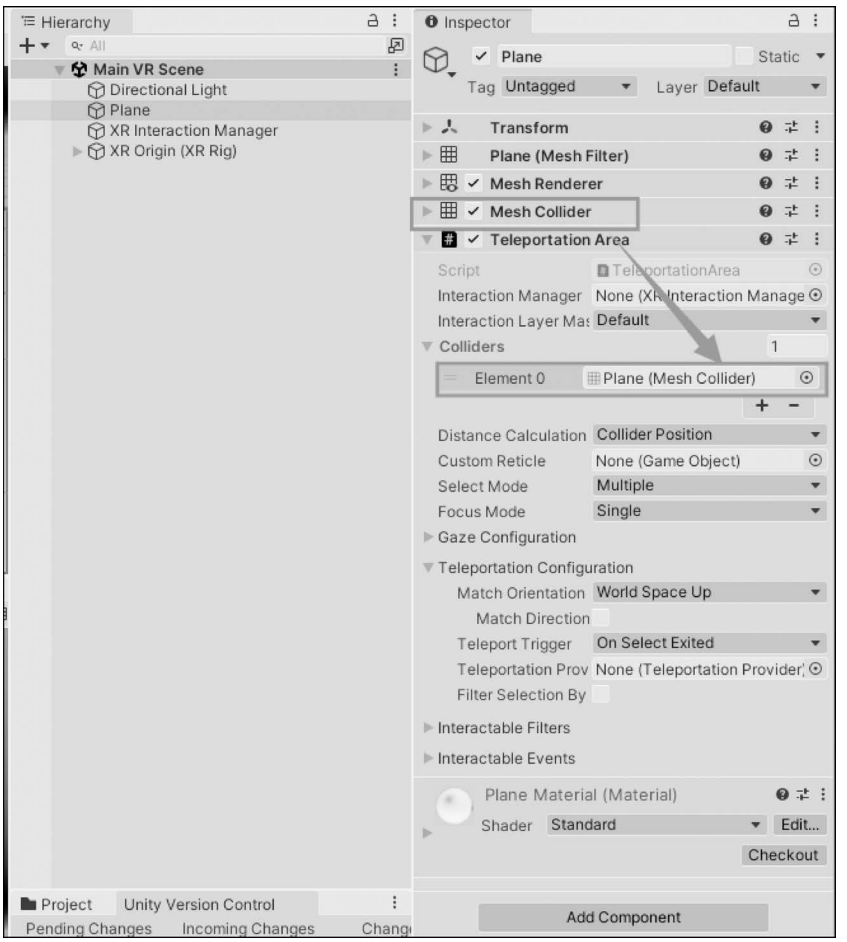

图 5-13　Teleportation Area 组件设置

在 Unity 编辑器中单击 Play 按钮，戴上 VR 头盔，按下左右任一控制器的 Grip 按键，将 Ray 指向平面区域后松开，可以观察到自己被传送移动到了指示射线 Ray 指向的位置，如图 5-14 所示。

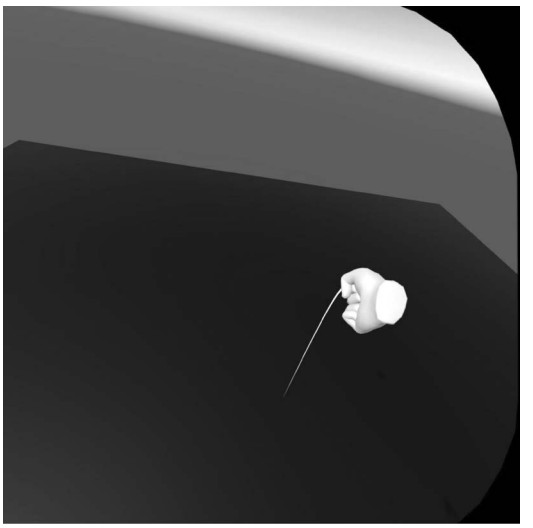

图 5-14　按下 Grip 按键触发传送移动

目前的操作本质上是把整个平面设置为传送区域，在传送区域内，只要单击 Ray 指向的地点，玩家就可以实现传送。

5.5.4　消除按键冲突

目前由于传送默认的按钮是 Grip 按键，这和抓握动作产生按键冲突。需要通过针对性的设置把传送的默认按键修改成更合理的其他按键。

在 Hierarchy 中展开 XR Origin→Camera Offset，选中 Camera Offset 下的 Right Teleportation Ray 对象，在 Inspector 面板中找到 XR Controller（Action-based）组件下 Input 下的 Select Action 属性配置，当前 Select Action 对应的 Reference 是 XRI RightHand Interaction/Select，也就是右手控制器的 Grip 按键，将这个 Action 的 Reference 映射成别的按键，就可以改变触发传送的按键。单击 Select Action 下 Reference 属性右侧的圆点，在弹出的对话框中搜索 activate，选中 XRI RightHand Interaction/Activate 选项，此时右手控制器触发传送的按键就换成了 Trigger 按键。Right Teleportation Ray 对象此时的相关属性设置如图 5-15 所示。

对 LeftTeleportationRay 对象也做类似设置。

在 Hierarchy 中展开 XR Origin→Camera Offset，选中 LeftTeleportationRay 对象，在 Inspector 面板中 XR Controller（Action-based）组件下 Input 下的 Select Action 属性配置，当前此 Action 对应的 Reference 是 XRI LeftHand Interaction/Select，也就是左手控制器的

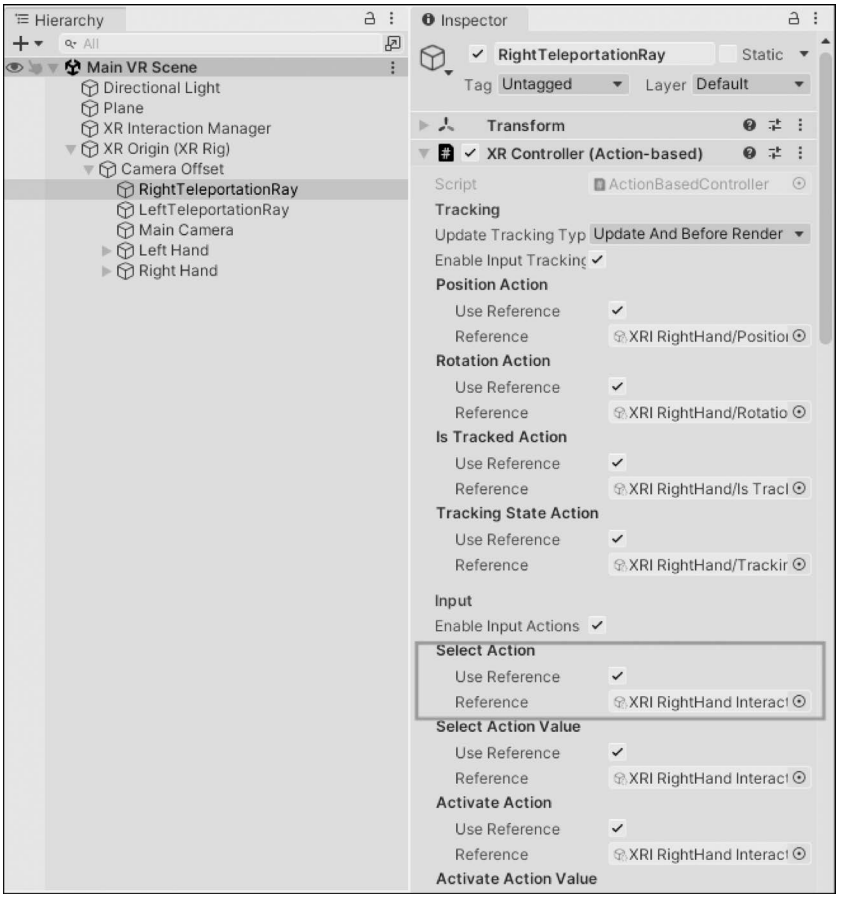

图 5-15　将右手触发传送的按键从 Grip 键变更为 Trigger 键

Grip 按键，将这个 Action 的 Reference 映射成别的按键，就可以改变触发传送的按键。单击 Select Action 下 Reference 属性右侧的圆点，在弹出的对话框中搜索 activate，选中 XRI LeftHand Interaction/Activate 选项，此时左手控制器触发传送的按键就换成了 Trigger 按键。LeftTeleportationRay 对象此时的相关属性设置如图 5-16 所示。

5.5.5　测试和总结

单击 Unity 编辑器的 Play 按钮运行测试，戴上 VR 头盔观察游戏场景，按下左右手任一控制器的 Trigger 按键，在指定传送区域尝试传送移动，观察到虽然线性存在一定扭曲，但是不妨碍松开 Trigger 按键后成功传送到指定地点，如图 5-17 所示。

本节的主要操作步骤如下：

在 Hierarchy 中选中 XR Origin，在 Inspector 面板追加组件 Teleportation Provider，将 Inspector 面板上方的 Locomotion System 组件拖曳到 System 属性框中。

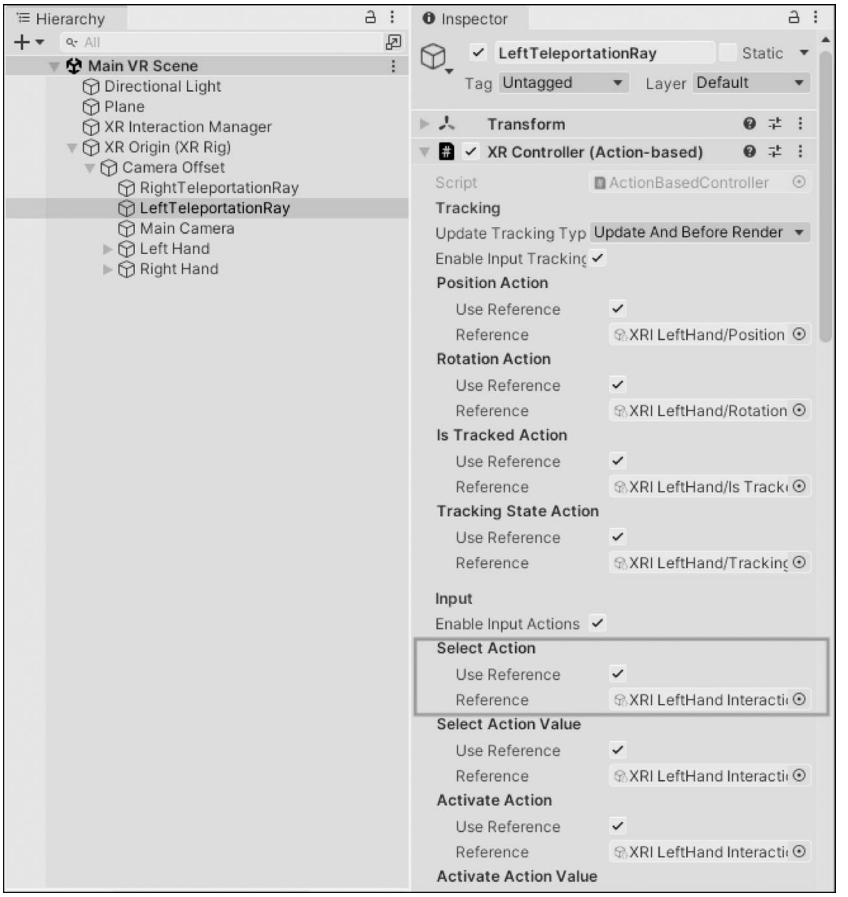

图 5-16　将左手触发传送的按键从 Grip 键变更为 Trigger 键

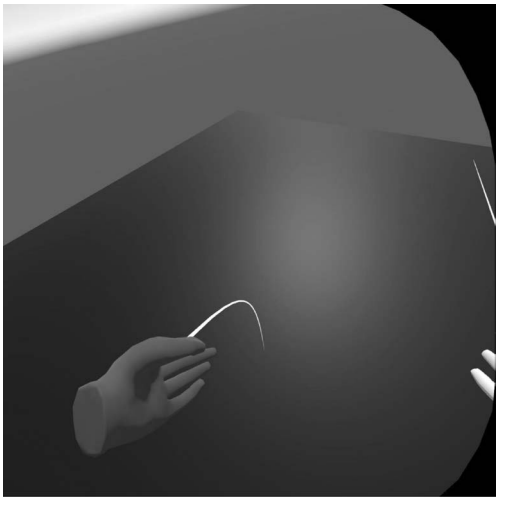

图 5-17　按下 Trigger 键触发传送移动

在 Hierarchy 中选中 Plane 对象，在 Inspector 面板中追加 TeleportationArea 组件，将 Inspector 面板中的 Mesh Collider 拖曳到 TeleportationArea 组件的 Colliders 属性列表中。

在 Hierarchy 中展开 XR Origin→Camera Offset，选中 Right Teleportation Ray 对象，找到 XR Controller(Action-based)组件下的 SelectAction 项下的 Reference 属性框，将关联按键重选为 XRI RightHand Interaction/Activate。

同样在 Hierarchy 的 XR Origin→Camera Offset 下选中 LeftTeleportationRay 对象，找到 XR Controller(Action-based)组件下的 SelectAction 项下的 Reference 属性框，将关联按键重选为 XRI LeftHand Interaction/Activate。

5.6　实现指定目标锚点的传送移动

除了可以在指定传送区域内的传送移动外，另一种常见的传送移动模式是面向指定锚点位置的传送移动。

5.6.1　创建锚点

为了创建指定目标锚点的传送移动，在 Hierarchy 中选中根目录 Main VR Scene，右击后在弹出的快捷菜单中选择 GameObject→3D Object→Plane，新建一个 Plane 对象，默认名称为 Plane(1)。

在 Scene 面板中选中新建的 Plane(1)对象，按下键盘上的 W 键，出现平移坐标轴，拖曳红色的 x 轴，往箭头方向平移 Plane(1)对象到现有 Plane 的左边，如图 5-18 所示。

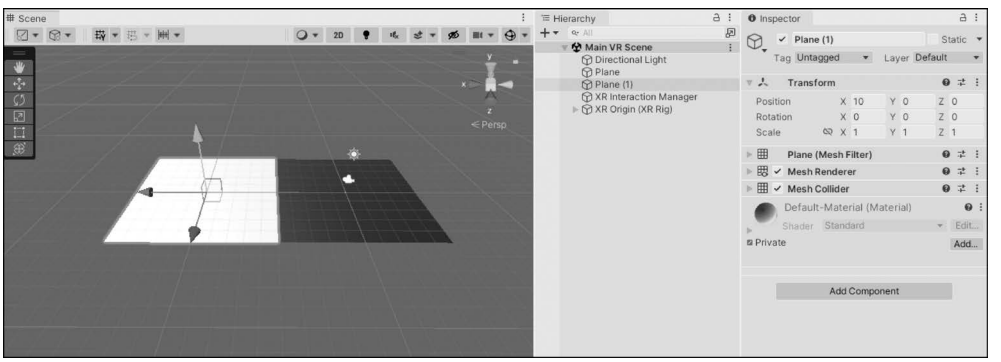

图 5-18　新建 Plane 对象

在 Hierarchy 中，首先右击根目录 Main VR Scene，在弹出的快捷菜单中选择 GameObject → Create Empty，创建一个空对象，然后将这个空对象重命名为 TeleportationAnchor。

继续在新建的 TeleportationAnchor 上右击，在弹出的快捷菜单中选择 TeleportationAnchor→Cylinder，新建一个圆柱体作为子对象，如图 5-19 所示。

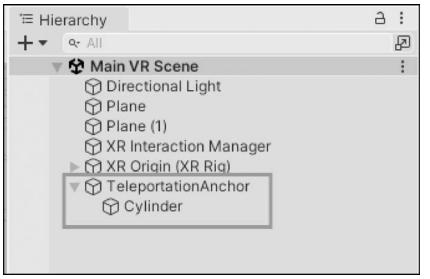

图 5-19　新建 TeleportationAnchor 对象和 Cylinder 对象

5.6.2　创建透明材质

为新建的圆柱体 Cylinder 创建 material。在 Project 面板中展开 Assets→Materials，打开 Materials 文件夹。在空白处右击，从快捷菜单中选择 Create→Material，将这个新建的 Material 重命名为 Teleportation Anchor Material，如图 5-20 所示。

图 5-20　新建 Teleportation Anchor Material 材质资源

进一步设置新建的 Teleportation Anchor Material，保持 Project 面板中选中 Teleportation Anchor Material 材质资源文件，在 Inspector 面板中，将 Shader 属性重选为 Legacy Shader→Transparent→Diffuse，打开 Main Color 属性并修改 RGB 属性，设定一个自己喜欢的颜色，例如蓝色。再调整 A 属性获得一定的透明度，如图 5-21 所示。

把设置完成的 Teleportation Anchor Material 对象从 Project 面板拖曳到 Scene 面板中的圆柱体上。应用了 Teleportation Anchor Material 材质的圆柱体立即具备了相应的透明度，如图 5-22 所示。

5.6.3　完善锚点组件配置

在 Hierarchy 中继续保持选中圆柱体对象 Cylinder，在 Inspector 面板中将 Transform 组件下的 Position 属性设置为 X = 0，Y = 1，Z = 0。该设置的效果是当选中 TeleportationAnchor 对象时，锚点(Pivot)位于圆柱的底部，效果如图 5-23 所示。

在 Hierarchy 中选中 TeleportationAnchor 对象，在 Inspector 面板中单击 Add Component 按钮，在弹出的对话框中搜索 TeleportationAnchor 组件，单击该条目追加一个 TeleportationAnchor 组件。在 TeleportationAnchor 组件下可以看到 Colliders 属性列表，从 Hierarchy 中将 Cylinder 拖曳到 TeleportationAnchor 组件下的 Colliders 属性列表中，再从 Hierarchy 中将 TeleportationAnchor 对象拖曳到 TeleportationAnchor 组件下 TeleportationAnchorTransform 属性框中，这样一个传送锚点的设置就完成了，如图 5-24 所示。

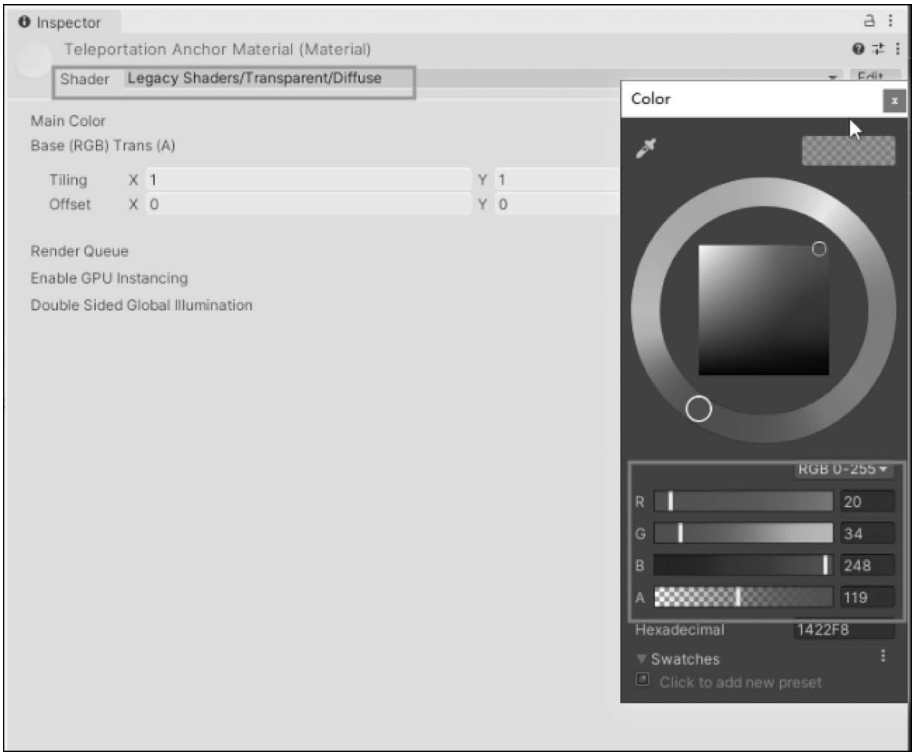

图 5-21　Teleportation Anchor Material 的设置

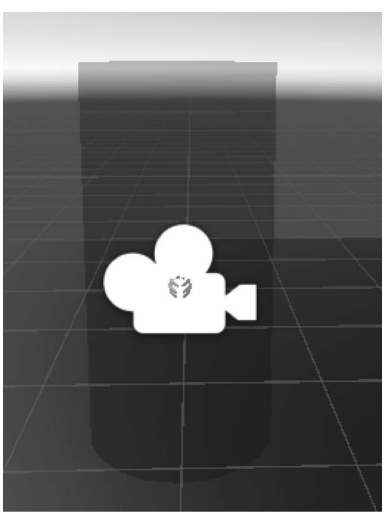

图 5-22　圆柱体立即具备透明度

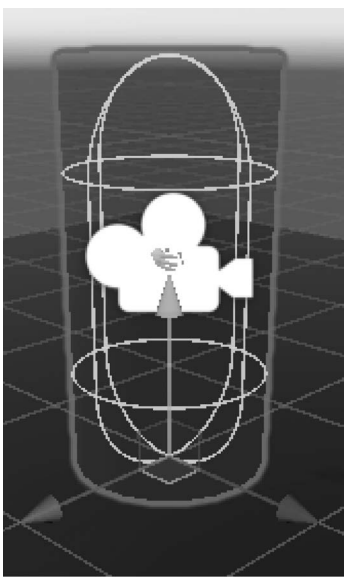

图 5-23　设置锚点位置

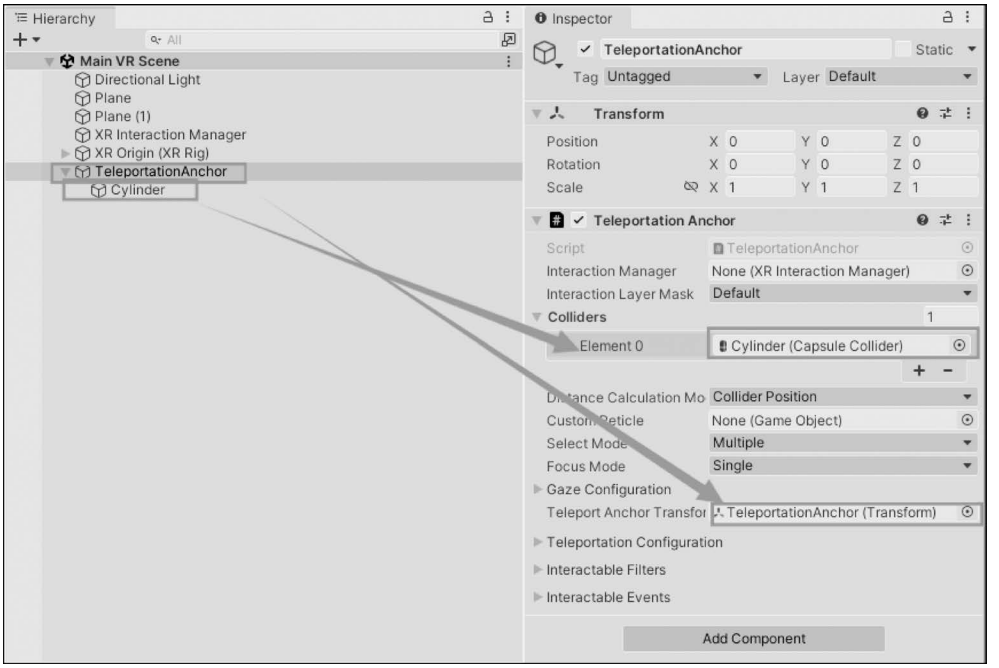

图 5-24　传送锚点的组件设置

作为传送移动目标的第 1 个圆柱体锚点创建完成，将这个对象复制后可以重复使用。在 Hierarchy 中选中 TeleportationAnchor，按快捷键 Ctrl＋D 复制 3 次，将复制后对象沿着 x 或 z 轴方向移动到 Plane(1)范围内的不同地点，如图 5-25 所示。

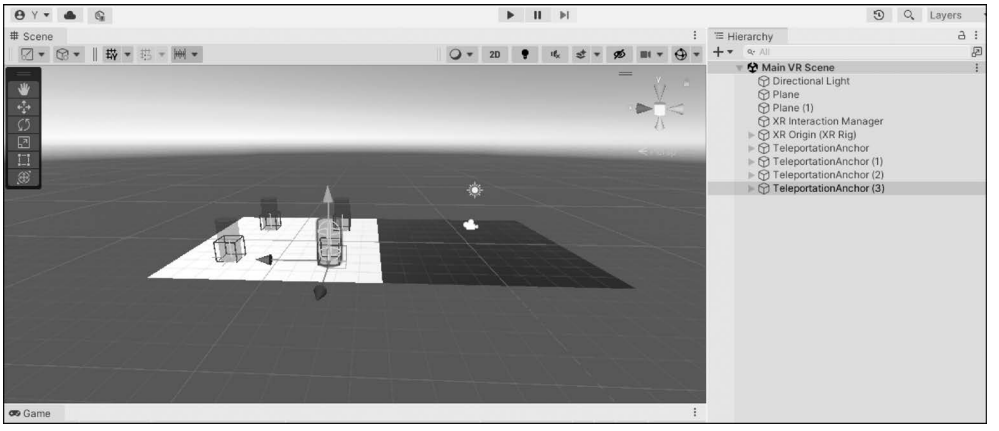

图 5-25　复制 4 个 TeleportationAnchor 对象

在 Unity 编辑器单击 Play 按钮运行测试，戴上 VR 头盔，尝试用左手或右手控制器的 Trigger 按键触发传送移动。观察到可以通过 Trigger 按键在各个锚点间实现传送移动。同时注意到在 Plane(1)范围内无法传送到锚点以外的地面位置。这是因为 Plane(1)自身

未被设置为 Teleportation Area。

5.6.4　指定每个锚点的主体移动后朝向

测试过程中同时发现传送移动后玩家面对的方向是随机的。

如果想要指定传送移动到锚点后的身体朝向，则可以选中该锚点所在圆柱体对象进行设置。在 Hierarchy 中选中 Teleportation Anchor，在 Inspector 面板中找到 Teleportation Anchor 组件下 Teleportation Configuration 部分的 Match Orientation 属性，将这个属性改选为 Target Up And Forward。这个选项的作用是规定玩家传送到当前锚点后 Forward 方向对齐 Teleportation Anchor 对象 Transform 坐标轴的 z 轴正方向(蓝色箭头)，Upward 方向对齐 Teleportation Anchor 对象 Transform 坐标轴的 y 轴正方向(绿色箭头)。

TeleportationAnchor 组件的完整设置如图 5-26 所示。

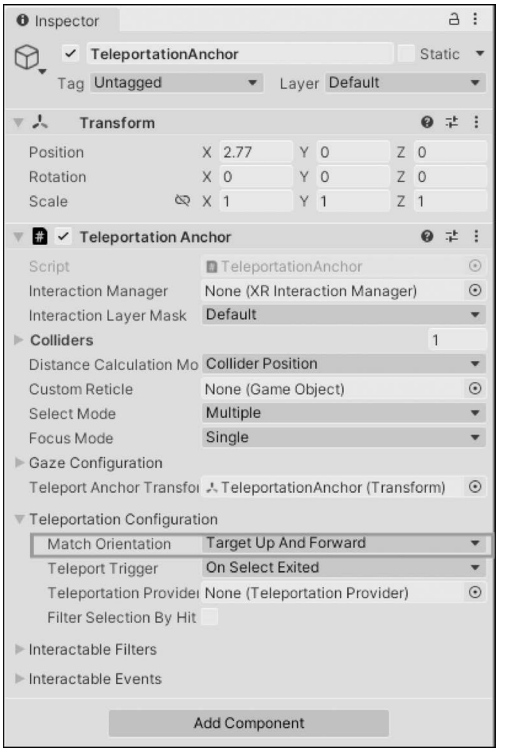

图 5-26　TeleportationAnchor 组件指定传送后面对方向

5.6.5　测试和总结

再次运行测试，传送到各个锚点位置后，观察到玩家朝向与锚点的 Forward 方向一致。

本节的主要操作步骤如下：

在 Hierarchy 中新建 Plane 对象，默认名称为 Plane(1)。在 Scene 面板中将 Plane(1)的

位置移动到现有 Plane 对象的左边。

在 Hierarchy 下新建一个空对象，命名为 Teleportation Anchor。在 Teleportation Anchor 下新建 Cylinder 子对象。

在 Project 面板中展开 Assets→Materials，新建一个 Material 资源文件，命名为 Teleportation Anchor Material。将 Shader 属性修改为 Legacy Shaders/Transparent/Diffuse。修改 Main Color 属性，调用调色板后调整到自己偏好的 RBG 和透明度 alpha 值。

在 Hierarchy 中展开 Teleportation Anchor，选中 Teleportation Anchor 对象下的圆柱体 Cylinder。将新建的材质资源文件 Teleportation Anchor Material 拖曳到圆柱体上。

在 Inspector 面板中将 Transform 组件下的 Position 属性设置为 X＝0，Y＝1，Z＝0。追加 Teleportation Anchor 组件，从 Hierarchy 中将圆柱体 Cylinder 对象拖曳到 Colliders 属性列表中，再将 Inspector 上方的 Teleportation Anchor 组件拖曳到 Teleportation Anchor Transform 属性框中。

复制 3 次锚点对象 Teleportation Anchor，将复制体放置在场景中 Plane(1)上的不同位置。

在 Hierarchy 中选中 Teleportation Anchor，在 Inspector 面板中找到 Teleportation Anchor 组件的 Teleportation Configuration 属性设置部分，将 Match Orientation 属性设置为 Target Up And Forward。

5.7 自定义 Ray 的外观

目前实现了区域和锚点两种传送移动方法，这一节继续介绍如何自定义和美化指示射线 Ray 的外观。

5.7.1 设置射线的线性

首先自定义 Ray 的线性。

在 hierarchy 中展开 XR Origin→Camera Offset，选中 Right Teleportation Ray，在 Inspector 面板中找到 XR Ray Interactor 组件下的 Raycast Configuration 部分，将代表线型的 LineType 属性重选为 Bezier Curve(贝塞尔曲线)。

贝塞尔曲线(Bezier Curve)是一种用于计算机图形学和相关领域的曲线定义方法。它由法国数学家 Pierre Bézier 在 20 世纪 60 年代提出，被广泛地应用于曲线建模、字体设计、动画路径、CAD/CAM 系统等。贝塞尔曲线通过控制点来定义，曲线的形状由这些控制点的位置决定。在这里我们只需知道将线性选为 Bezier Curve 后指示射线将从单调的直线变为优美的曲线，如图 5-27 所示。

在 Unity 编辑器中单击 Play 按钮运行测试，戴上 VR 头盔后用控制器的 Trigger 按键触发传送移动，可以发现，用于指示传送移动目标地点的 Ray 从原本的直线线型变为一条优美的曲线线型。这条曲线就叫贝塞尔曲线，可以通过 Inspector 面板中 LineType 下的属

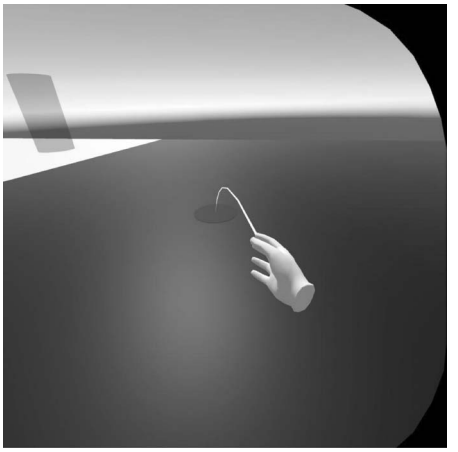

图 5-27　传送指示射线线性变为更自然的贝塞尔曲线

性来控制这条曲线的呈现。

在 Hierarchy 中保持选中 Right Teleportation Ray，在 Inspector 面板中将 XR Ray Interactor 组件下 Raycast Configuration→Line Type 下的 End Point Distance 属性设置为 5，将 End Point Height 设置为－7，将 Control Point Distance 设置为 3，将 Control Point Height 设置为－0.3。这样就完成了这条贝塞尔曲线的设定，如图 5-28 所示。

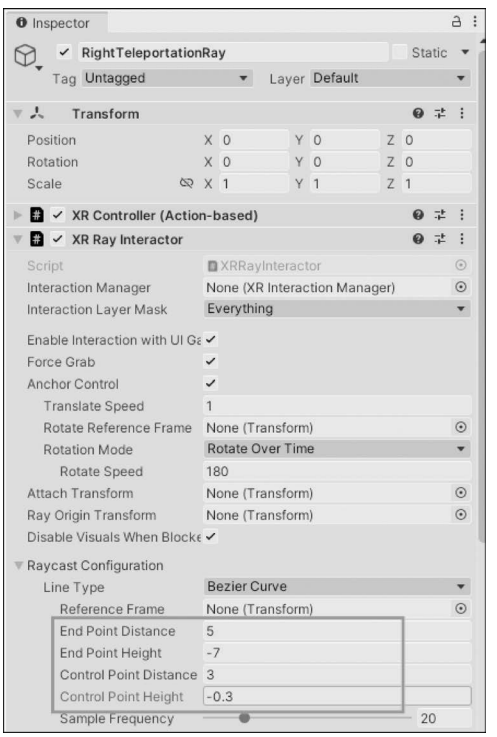

图 5-28　Right Teleportation Ray 的设置

对 Left Teleportation Ray 也做类似设置。

在 Hierarchy 中展开 XR Origin→Camera Offset，选中 Left Teleportation Ray，在 Inspector 面板中找到 XR Ray Interactor 组件下的 Raycast Configuration 部分，将代表线型的 LineType 属性重选为 Bezier Curve。将 Raycast Configuration→Line Type 下的 End Point Distance 属性设置为 5，将 End Point Height 设置为－7，将 Control Point Distance 设置为 3，将 Control Point Height 设置为－0.3。这样就完成了左手贝塞尔曲线的设定。Left Teleportation Ray 的 Raycast Configuration 设定此时如图 5-29 所示。

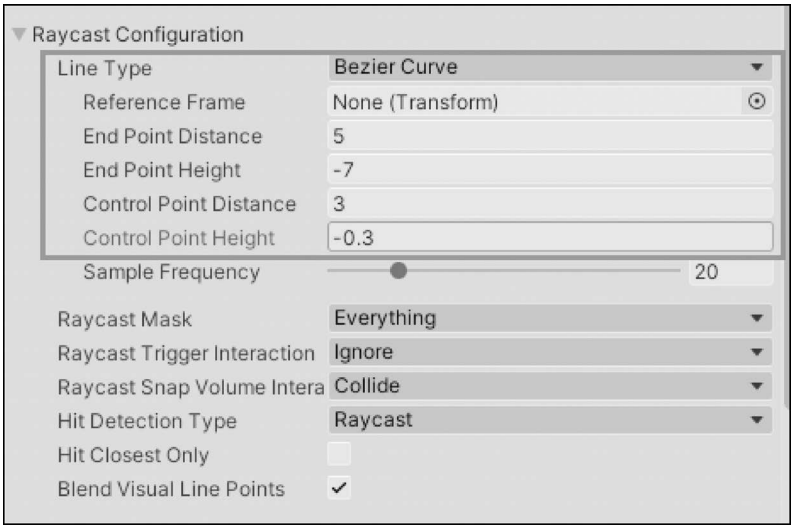

图 5-29　XR Ray Interactor 组件下 Raycast Configuration 部分的设置

5.7.2　自定义射线落点的样式

接下来继续美化 Ray 落点的呈现。在 Hierarchy 中右击 Right Teleportation Ray 对象，在快捷菜单中选择 3D Object→Cylinder，新建一个圆柱体子对象。将这个圆柱体对象 Cylinder 重命名为 Reticle。

在 Hierarchy 中选中 Reticle，在 Inspector 面板中找到 Transform 部分，将 Scale 属性设置为 X＝0.5，Y＝0.01，Z＝0.5，圆柱体 Reticle 呈现饼状。在 Inspector 面板中将 Reticle 的 Capsule Collider 组件移除，最终 Hierarchy 结构和 Inspector 设置如图 5-30 所示。

在 Projects 面板中展开 Assets→Materials 文件夹，将 Teleportation Anchor Material 拖曳到 Hierarchy 或 Scene 的 Reticle 对象上，使 Reticle 应用 Teleportation Anchor Material 材质，如图 5-31 所示。

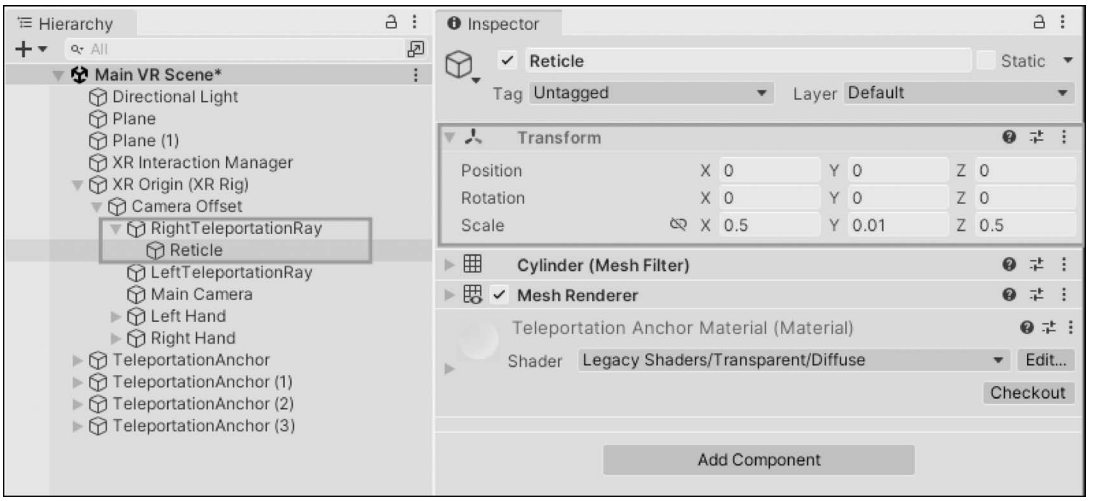

图 5-30　创建并设置 Reticle 对象后的 Hierarchy 结构与 Inspector 设置

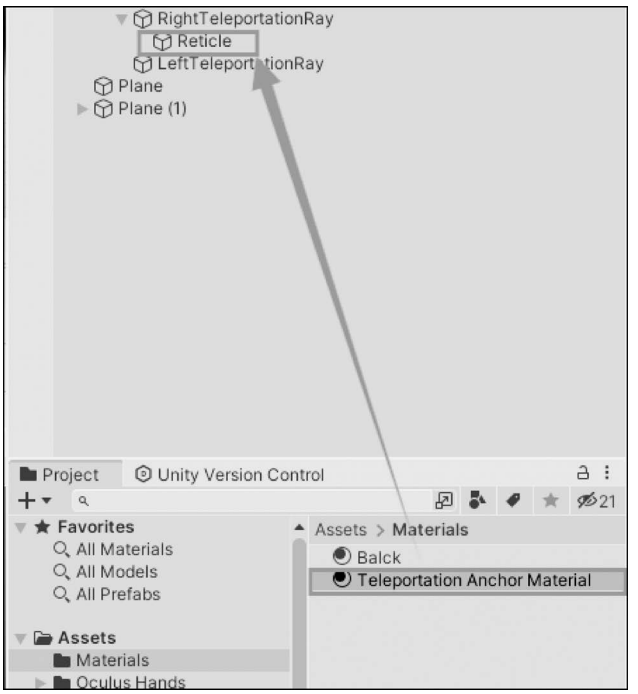

图 5-31　对 Reticle 应用 Teleportation Anchor Material 材质

在 Hierarchy 中展开 XR Origin→Camera Offset，选中 Right Teleportation Ray，从 Project 面板中将自定义落点 Reticle 对象拖曳到 Inspector 面板中组件 XR Interactor Line Visual 下的 Reticle 属性框中，如图 5-32 所示。

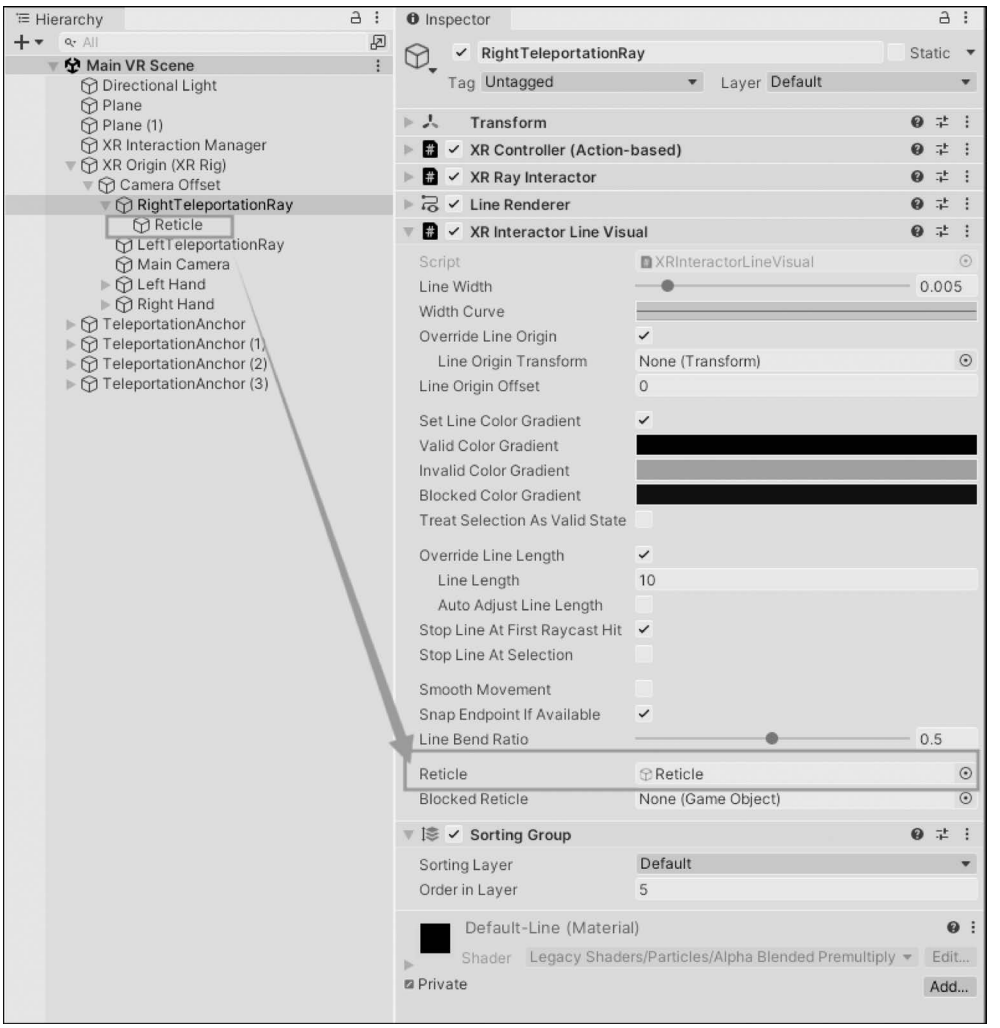

图 5-32　Reticle 对象关联 XR Interactor Line Visual 组件下的 Reticle 属性

在 Hierarchy 中选中 Reticle 对象，按快捷键 Ctrl＋D 复制为新对象 Reticle(1)。将 Reticle(1)拖曳到 Left Teleportation Ray 对象下，并重命名为 Reticle。

在 Hierarchy 中展开 XR Origin→Camera Offset，选中 Left Teleportation Ray 对象，从 Project 面板中将自定义落点 Reticle 对象拖曳到 Inspector 面板中 XR Interactor Line Visual 组件下的 Reticle 属性框中，如图 5-33 所示。

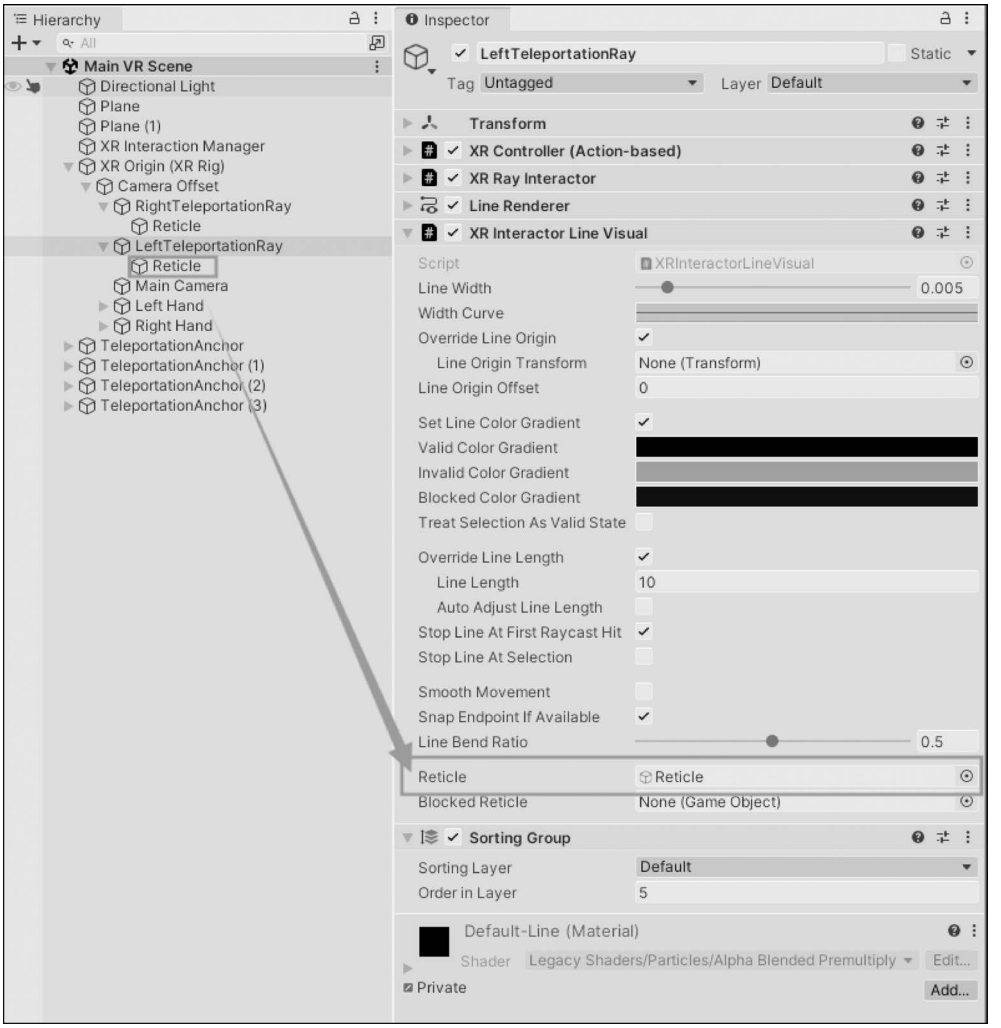

图 5-33　Reticle 对象关联 XR Interactor Line Visual 组件下的 Reticle 属性

5.7.3　测试和总结

单击 Unity 编辑器的 Play 按钮运行测试，用左右手控制器的 Trigger 按键触发传送移动，发现射线与地面碰撞的落点以自定义的形式显现，落点显得更为美观了。

本节的主要步骤如下：

在 Hierarchy 中选中 Right Teleportation Ray 对象，在 Inspector 面板中找到 XR Ray Interactor 组件下的 RayCast Configuration 部分，设置 LineType = Bezier Curve，EndPointDistance = 5，EndPointHeight = −7，ControlPointDistance = 3，ControlPointHeight = −0.3。

在 Hierarchy 中选中 Left Teleportation Ray 对象，在 Inspector 面板中找到 XR Ray Interactor 组件下的 RayCast Configuration 部分，设置 LineType = Bezier Curve，EndPointDistance=5，EndPointHeight=−7，ControlPointDistance=3，ControlPointHeight=−0.3。

在 Right Teleportation Ray 下新建圆柱子对象，命名为 Reticle。

选中 Reticle 对象后在 Inspector 面板中将 Transform 组件下的 Scale 属性设置为 X=0.5，Y=0.01，Z=0.5。移除 Collider 组件。对 Reticle 应用 Teleportation Anchor Material 材质。

选中 Right Teleportation Ray 对象，找到 XR Interactor Line Visual 组件，从 Hierarchy 将刚才新建的 Reticle 对象拖曳到 Inspector 的 Reticle 属性框中。

将 Reticle 对象复制为子对象并拖曳到 Left Teleportation Ray 对象下，将 Left Teleportation Ray 对象下的复制 Reticle 对象重命名，拖曳到 Inspector 面板的 XR Interactor Line Visual 组件部分下的 Reticle 属性框中。

5.8 使传送指示射线只在传送时出现

目前即使玩家并未试图使用控制器触发传送移动，用于指示传送移动目标点的指示射线 Ray 也总是呈现的，这显然是不合理的。

5.8.1 控制传送指示射线的显隐

为了解决这个问题，需要在 XR Origin 中引入一个新的自定义脚本组件来控制传送指示射线 Ray 只在需要的场合出现。

这个脚本的内容非常简单，基本实现逻辑是先检查左右手两个控制器的 Trigger 按键阈值是否超过 0.1，如果超过这个阈值，则说明玩家按下 Trigger 按键较深，可以确定玩家意图触发传送移动，此时呈现 Left Teleportation Ray 或 Right Teleportation Ray 对象。如果左右手两个控制器的 Trigger 按键阈值低于 0.1，则说明玩家未碰触或只是轻微碰触到 Trigger 按键，没有明确触发传送移动的意图，此时隐藏 Left Teleportation Ray 和 Right Teleportation Ray 对象。

在 Hierarchy 中选中 XR Origin。在 Inspector 面板中单击 Add Component 按钮，在弹出的对话框中单击 New Script，将此自定义脚本组件命名为 ActivateTeleportationRay，如图 5-34 所示。最后单击 Create and Add 按钮，完成自定义脚本的追加。

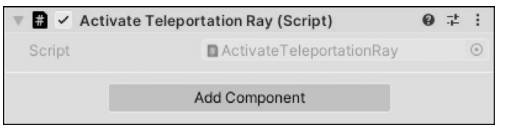

图 5-34 创建自定义脚本组件 ActivateTeleportationRay

双击 Script 属性框，打开 VS 脚本编辑器，开始编写 ActivateTeleportationRay 脚本的具体内容。

第一部分用于引入需要的包，包括 XR. Interaction. Toolkit 包和 InputSystem 包，代码如下：

```
using UnityEngine.XR.Interaction.Toolkit;
using UnityEngine.InputSystem;
```

第二部分用于声明变量。声明两个 GameObject 类型的公共变量 leftTeleportation 和 rightTeleportation，这两个公共变量在 Unity 编辑器的 Inspector 面板中用于关联 Hierarchy 下的 leftTeleportationRay 和 rightTeleportationRay 对象，代码如下：

```
public GameObject leftTeleportation;
public GameObject rightTeleportation;
```

继续声明两个 InputActionProperty 类型的公共变量 leftActiviate 和 rightActiviate，用于获取左右手控制器的 Trigger 按键的数值，代码如下：

```
public InputActionProperty leftActiviate;
public InputActionProperty rightActiviate;
```

第三部分用于编写 Update()函数。在 Update 逻辑中用 leftActivate 和 rightActivate 的数值是否大于 0.1 作为显示或是隐藏 leftTeleportation 和 rightTeleportation 的判断条件。

ActivateTeleportationRay 脚本的完整代码内容如下：

```
//第 5 章 - 只在意图传送时呈现传送指示射线 - 脚本 ActivateTeleportationRay
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.Interaction.Toolkit;
using UnityEngine.InputSystem;

public class ActivateTeleportationRay : MonoBehaviour
{
    //左手的传送指示射线(游戏对象)
    public GameObject leftTeleportation;

    //右手的传送指示射线(游戏对象)
    public GameObject rightTeleportation;

    //左手传送激活动作属性(用于检测输入)
    public InputActionProperty leftActiviate;

    //右手传送激活动作属性(用于检测输入)
    public InputActionProperty rightActiviate;

    //Start 是 Unity 的生命周期方法，在游戏开始时调用一次
```

```
    void Start()
    {
        //此处暂未实现任何初始化逻辑
    }

    //Update 是 Unity 的生命周期方法,每帧调用一次
    void Update()
    {
        //检测右手的传送激活输入值,如果值大于 0.1,则激活右手传送指示射线
        rightTeleportation.SetActive(rightActiviate.action.ReadValue<float>() > 0.1f);

        //检测左手的传送激活输入值,如果值大于 0.1,则激活左手传送指示射线
        leftTeleportation.SetActive(leftActiviate.action.ReadValue<float>() > 0.1f);
    }
}
```

5.8.2 配置自定义脚本组件 ActivateTeleportationRay

回到 Unity 编辑器界面,在 Hierarchy 中保持选中 XR Origin,在 Inspector 面板中找到自定义脚本组件 ActivateTeleportationRay 的部分,开始对 ActivateTeleportationRay 组件下的公共变量属性进行关联设置。

从 Hierarchy 中将 Right Teleportation Ray 拖曳到 Inspector 面板中 ActivateTeleportationRay 组件下的 Right Teleportation 属性。将 Left Teleportation Ray 拖曳到 Inspector 面板中 ActivateTeleportationRay 组件下的 Left Teleportation 属性。

继续在 Inspector 面板中勾选 ActivateTeleportationRay 组件下的 Right Activate 部分下的 Use Reference 选框,Action 选取 XRI RightHand Interaction/Activate Value。继续勾选 ActivateTeleportationRay 组件下的 Left Activate 下的 Use Reference 选框,Action 选取 XRI LeftHand Interaction/Activate Value。

ActivateTeleportationRay 组件的完整设置如图 5-35 所示。

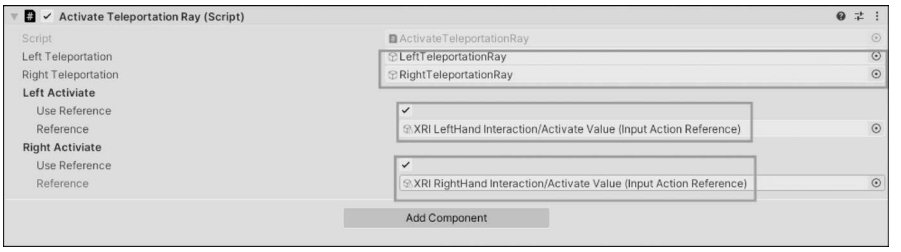

图 5-35 ActivateTeleportationRay 组件的完整设置

5.8.3 测试和总结

在 Unity 编辑器中单击 Play 按钮运行测试,戴上 VR 头盔后尝试用左手或右手控制器

按下 Trigger 按键触发传送移动，观察到只有按下 Trigger 按键到一定深度时，才会出现传送移动的指示射线。

接下来介绍 Update()和 FixedUpdate()函数的异同。

在 Unity 脚本中，Update()和 FixedUpdate()是两个常用的方法，它们用于在每帧更新游戏对象的状态和行为。虽然它们的作用类似，但在调用时机和用途上有一些区别。

Update()函数是在每帧中被调用的，它用于更新游戏对象的状态、位置、旋转等信息。一般用于处理与游戏对象位置、输入、动画等相关的逻辑。

Update()函数的调用频率是不固定的，取决于游戏的帧率。在每帧中，Update()函数都会被调用一次，但如果游戏的帧率发生变化，则 Update()函数的调用次数也会发生相应变化。

FixedUpdate()函数是在固定的时间间隔内被调用的，它用于处理与物理相关的逻辑，例如刚体运动、碰撞检测等。

FixedUpdate()函数的调用频率是固定的，不受游戏帧率的影响。在默认情况下，它每秒调用一次，并且在物理系统更新之后执行，确保物理计算的稳定性和准确性。

由于 Update()函数的调用频率不固定，而 FixedUpdate()函数的调用频率是固定的，因此 Update()函数适用于处理与游戏对象位置、输入、动画等相关的逻辑，而 FixedUpdate()函数适用于处理与物理相关的逻辑，例如刚体运动、碰撞检测等。

本节的主要操作步骤如下：

在 Hierarchy 中选中 XR Origin，在 Inspector 面板中新增自定义脚本组件 Activate Teleportation Ray。

打开脚本编写逻辑，先引入需要的命名空间，代码如下：

```
using UnityEngine.XR.Interaction.Toolkit;
using UnityEngine.InputSystem;
```

声明逻辑中需要使用的变量，代码如下：

```
public GameObject leftTeleportation;
public GameObject rightTeleportation;
public InputActionProperty leftActiviate;
public InputActionProperty rightActiviate;
```

编写 Update()函数逻辑，检查左右两个控制器的按键阈值是否超过 0.1，如果超过，则说明玩家按下按钮意图移动，激活 LeftTeleportation 或 RightTeleportation 对象，代码如下：

```
rightTeleportation.SetActive(rightActiviate.action.ReadValue<float>() > 0.1f);
leftTeleportation.SetActive(leftActiviate.action.ReadValue<float>() > 0.1f);
```