项目1

基于 Unity 和 GSXR 搭建 XR 应用框架

任务 1.1 GSXR 概 述

1. 概述

GSXR（General Standard for XR）标准通过提供统一的应用开发标准和设备对接规范，帮助开发者实现快速分发和多平台覆盖，更好地解决硬件终端平台耦合、标准差异化严重等问题，增强软件适配性。GSXR 互通标准具体包含互通规范、开发套件、测评系统三部分，其中互通规范涵盖应用层及设备层两大层面的接口定义，可以支持 3DOF 及 6DOF 设备，对于一体机及分体机均适用。

2. GSXR 互通接口规范

GSXR 互通接口规范提供了 XR 标准应用程序接口（Application Programming Interface，API）定义与功能描述，包括虚拟现实（Virtual Reality，VR）、增强现实（Augmented Reality，AR）和混合现实（Mixed Reality，MR）相关应用的标准接口。XR 应用开发者可以调用格式一致的 XR 功能函数开发 XR 应用，而无须再为不同 Runtime 接口进行适配，只需专注于 XR 内容开发，且开发出的 GSXR 应用可运行于各个支持 GSXR 标准的 XR 设备上。目前，此规范只适用于 Android 系统。

3. GSXR 接口适用范围

1）XR 设备插件

XR 设备厂家应根据设备所支持的功能，提供对应的 XR 函数实现，例如上报头显或手柄设备状态、设备跟踪及输入数据等。设备厂家必须依据自身设备的硬件、固件，以及算法设计实现 GSXR 设备接口，以提供 Runtime 需求的设备数据。

2）XR Runtime

XR Runtime 必须根据产品设计，正确完成 XR 设备插件的初始化，并依据 XR 应用的功能需求，在适当的时候，调用对应的 GSXR 设备接口获取设备数据。同时配合 XR Runtime 本身的架构及功能设计，提供 XR 应用程序即时的设备交互数据，确保 XR 应用

在 XR 设备上正确运行。

4. 术语及缩略语

GSXR 接口规范适用的缩略语及全称如表 1-1 所示。

表 1-1　缩略语及全程

缩略语	全　　称
GSXR	General Standard for XR
规范	GSXR 互通接口规范
XR 应用	GSXR 应用程序
XR Runtime	GSXR Runtime
XR 设备	GSXR 设备
VR	Virtual Reality，虚拟现实
AR	Augmented Reality，增强现实
MR	Mixed Reality，混合现实
API（接口）	Application Programming Interface，应用程序接口
HMD（头显）	Head-Mounted Display，头戴式显示器
IPD	Inter Pupillary Distance，双眼瞳距
DoF	Degree of Freedom，空间自由度
V-Sync	Vertical Synchronization，垂直同步
CPU	Central Processing Unit，中央处理单元
GPU	Graphics Processing Unit，图形处理单元

5. GSXR 主流程

GSXR 应用程序对于互通接口的操作，从创建 Runtime 实例开始（该实例是 Runtime 的生成基础），所有 Runtime 之后的操作都将依据该实例进行。然后，Runtime 进行 XR 设备连线的接入，赋予设备正确的状态及组态，确认跟踪、输入 / 输出、图形及显示系统正常工作。接着创建渲染器实例及纹理队列并开始渲染循环。在渲染循环中，XR 应用程序等待 Runtime 返回正确的 V-Sync 信号，从而开始获取 Runtime 的事件、位姿（实时位置与姿态）、输入数据，并获取纹理队列中的图形缓冲进行视图渲染，提交渲染帧，Runtime 进行后续 XR 的图形算法处理并输出显示屏。GSXR 主流程详细示意图如图 1-1 所示。

6. GSXR 基本架构

如果把一个 XR 应用的软件实现简单分成三层，那么 GSXR 属于中间层部分，向上为应用开发者提供统一的 XR 应用开发接口，向下为平台厂商提供 GSXR 各功能的实现规范标准，如图 1-2 所示。

GSXR 提供给上层开发者完整的 GSXR 生命周期接口控制函数，包括 GSXR 的初始化和销毁，每帧追踪位姿的更新、渲染以及纹理的提交等接口。GSXR 还提供了专门针对当下比较流行的游戏引擎 Unity 3D/Unreal Engine 的插件封装，操作简单方便，同时提供更开放的原生接口，适用于使用其他引擎的开发者。

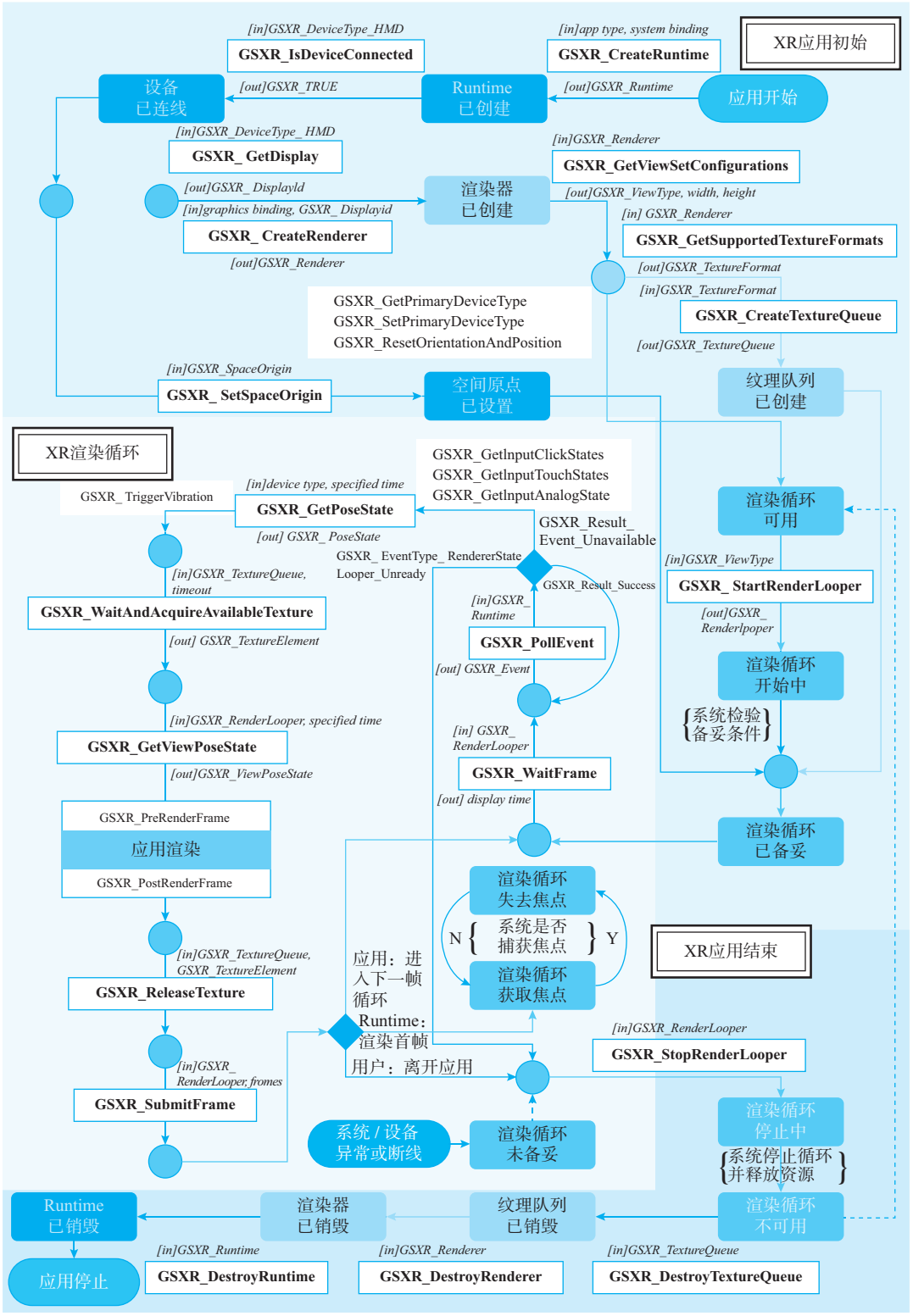

图 1-1　GSXR 主流程详细示意图

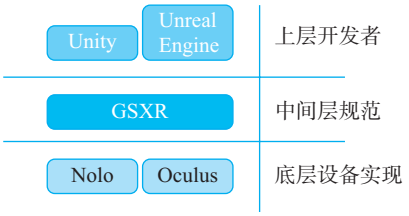

图 1-2　GSXR 基本架构示意图

7. GSXR 主要组成部分

GSXR 规范的主要内容是服务于人的视觉和触觉两大感知系统，其中，视觉主要体现在三维立体图像通过电子屏幕的显示画面进入人眼，与大脑建立视觉联系；而触觉主要依靠控制手柄在虚拟三维空间的位姿变化以及振动马达反馈到人手臂的触觉系统，与大脑建立联系。通过人的这两大感知系统，让大脑沉浸在虚拟环境中。

1）GSXR Runtime 模块

该模块作为 GSXR 的上下文管理模块，包含了各种完成 XR 任务所需的信息，包括获取屏幕设备信息、图形渲染能力信息、追踪能力信息、输入设备信息、特性功能信息等，以及包含 GSXR 任务的启动和结束等内容。以追踪能力信息为例，开发者通过 GSXR 获取设备信息的 API 函数接口，获取设备是否支持 6 个自由度的追踪能力（或者仅支持 3 个自由度的追踪能力），是否支持手柄控制器，以及手柄控制器是否具备 6 个自由度或者 3 个自由度的追踪能力。Runtime 模块还包含 GSXR 中其他模块生命周期的管控，例如渲染模块的创建、初始化、销毁，以及特性模块的创建、初始化及销毁。GSXR Runtime 类示意图如图 1-3 所示。

图 1-3　GSXR Runtime 类示意图

2）GSXR 渲染模块内容

（1）图形渲染 API。图形渲染 API 是用来访问计算机 GPU 硬件的软件接口，目前主流的图形渲染 API 包括 OpenGL、Direct3D、Meta 以及 Vulkan。在移动平台上普遍使用 OpenGL-ES 图形渲染 API。无论是 GSXR 应用程序本身的内容渲染，还是 GSXR Runtime 对于应用提交的内容帧所进行的 Timewarp 渲染管线后处理渲染，都应该使用操作系统所

支持的图形渲染 API 进行渲染。故应用程序在 XR 渲染器创建之初，必须将应用程序使用的图形渲染 API 特化信息提供给 Runtime，XR 渲染器才能正确地进行后续的 XR 后处理渲染。软件开发访问 GPU 硬件的过程示意图如图 1-4 所示。

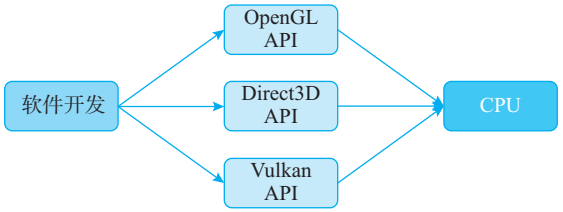

图 1-4　软件开发访问 GPU 硬件的过程示意图

（2）GSXR 视图集配置。一个标准的 GSXR 显示设备，既可能是穿戴于头上的头显，也可能是握持于手上的手持式设备，不同性质的显示设备所需求的渲染视图也可能不同。通常头显需要渲染左右两眼的双视图，而手持式设备则只需要渲染单视图。同样地，显示设备的规格差异将产生不同的视图配置，也将影响 GSXR Runtime 渲染器在建构渲染循环时的帧缓冲操作。视图集是视图的集合体，其中可能包含一个或多个视图，同一视图集中的各视图配置全部相同。一个视图集通常对应一组显示输出（可能为实体显示设备，如 VR 头显；也可能为虚拟显示设备，如无线投屏）。GSXR Runtime 经由视图集配置描述显示设备的组态及其对应的纹理结构，GSXR 应用程序则需要根据视图集配置，指定其图形渲染 API 相关参数进行渲染，才可以渲染出合适的图像并输出到对应此视图集配置的显示设备。显示设备有主屏、副屏之分，一般 GSXR 应用程序只以主屏为渲染对象（且多数情况也只存在主屏）。主屏对应创建渲染器时，输入的由显示设备获取的 DisplayId，而后用于渲染主屏视图的视图姿态就从这个显示设备类型的设备姿态转换而来。副屏通常为主屏的延伸屏幕，使用与主屏相同的设备姿态并根据副屏视图配置按照需求进行副屏渲染。在某些特定的应用情境下，同一渲染器可能同时存在主屏与副屏，GSXR 应用程序可根据应用本身的设计决定是否同时渲染主副屏，但至少会渲染主屏。头显左右眼双视图如图 1-5 所示。

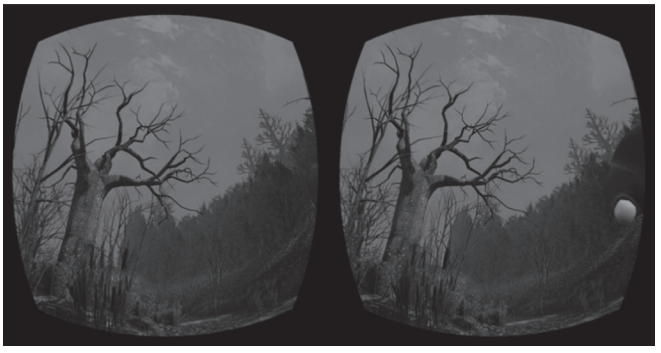

图 1-5　头显左右眼双视图

（3）GSXR 纹理队列。GSXR 纹理队列也被称作纹理交换链（Swap Chain），是一组纹理缓冲器（Buffer）。GSXR 应用程序按照 GSXR Runtime 支持的纹理格式及参数，根据

自身的渲染需求创建纹理队列，并于渲染循环中循序从队列中获取可用纹理进行内容渲染。纹理队列如图 1-6 所示。

图 1-6　纹理队列示意图

（4）GSXR 渲染循环。GSXR 应用程序在准备好（也称备妥）所有渲染要素后，便可开始进行渲染循环。一个标准的渲染循环需先等待 Runtime 返回 V-Sync 信号，在轮询事件队列、获取输入及位姿数据后，便可从纹理队列中获取纹理进行内容渲染，然后将该纹理图像提交给 Runtime 进行后续的 XR 后处理渲染，如此反复循环便为渲染循环。GSXR 渲染循环如图 1-7 所示。

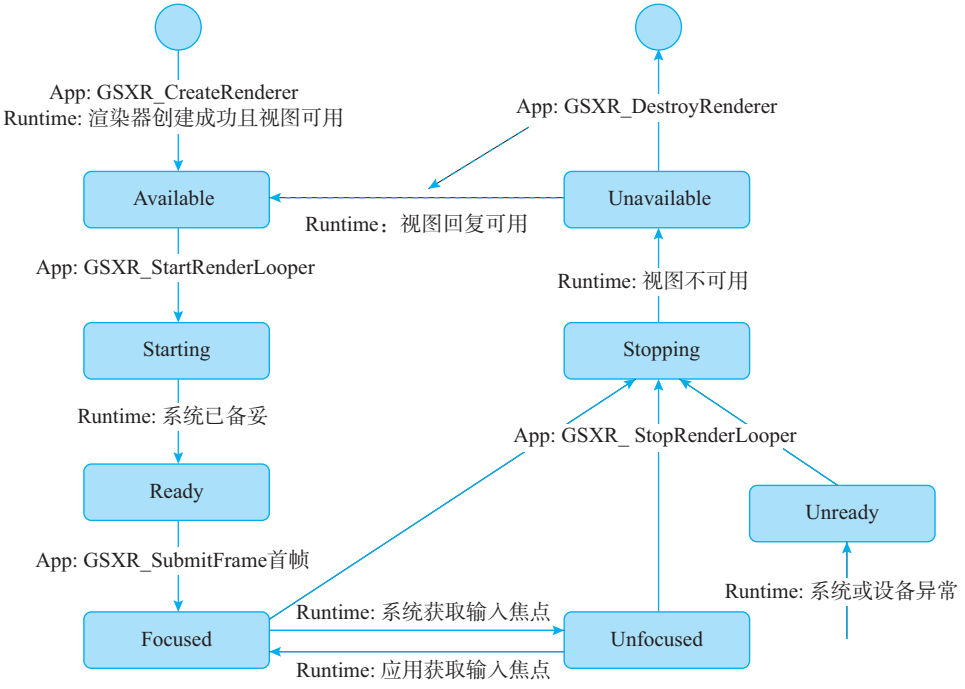

图 1-7　GSXR 渲染循环示意图

（5）GSXR 渲染器的生命周期。GSXR 应用程序必须先完成 GSXR 渲染器的实例创建及初始化后，才可以提供 GSXR 应用程序后续渲染循环所需要的功能。在 GSXR 应用程序不再使用此渲染器时，应用程序必须停止 GSXR 渲染器并销毁实例，将资源返回给系统，此流程就是 GSXR 渲染器的生命周期。

（6）GSXR 渲染循环状态。渲染循环状态各自代表着不同的运行情况，各状态也有各自的形成条件，GSXR Runtime 必须根据当前状态发送状态事件给应用程序，应用程序必须根据上报的状态事件进行适当的处置，才可确保渲染循环正确运行。本任务介绍 8 种渲染循环状态的定义、GSXR Runtime 及应用程序在不同状态所应采取的处理动作。

① Available 状态。当 XR 应用程序成功创建渲染器，且系统及显示设备所对应的视

图集状态正常，渲染循环即进入 Available 状态，代表此视图及其渲染循环已经处于可用状态。此时 GSXR Runtime 需发送 GSXR_EventType_RendererState_Looper_Available 事件给应用程序，当应用程序获取此事件时即可调用 GSXR_StartRenderLooper 开始渲染循环。

② Starting 状态。当 GSXR 应用程序成功调用 GSXR_StartRenderLooper 开始渲染循环，渲染循环即进入 Starting 状态，代表此渲染循环处于起始状态。此时 GSXR Runtime 需发送 GSXR_EventType_RendererState_Looper_Starting 事件给应用程序，并根据 GSXR Runtime 自身的设计检视进入下一 Ready 状态的必要条件，如 TextureQueue 是否已创建、空间原点是否已设置等。

③ Ready 状态。当 XR Runtime 备妥渲染循环的必要资源，也确认 XR 应用程序完成所有必要动作，渲染循环即进入 Ready 状态，代表此渲染循环处于已备妥状态。此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Ready 事件给应用程序，应用程序便可开始进行应用的帧循环渲染。

④ Focused 状态。当 GSXR 应用程序完成第一次帧渲染，并将渲染帧提交给 GSXR Runtime 之后，渲染循环即进入 Focused 状态，代表应用内容已经可见且当前输入焦点属于应用程序（在此状态，应用程序可以接收设备输入数据进行内容交互），此时 Runtime 需发送 GSXR_EventType_RendererState_Looper_Focused 事件给应用程序。

⑤ Unfocused 状态。当 GSXR Runtime 将系统渲染层（如系统选单）覆盖于应用程序合成层之上，并从应用程序中获取输入焦点后，渲染循环即进入 Unfocused 状态，代表应用内容虽可见但当前输入焦点不属于应用程序（在此状态下，应用程序无法接收设备输入数据进行内容交互），此时 GSXR Runtime 需发送 GSXR_EventType_RendererState_Looper_Unfocused 事件给应用程序。

⑥ Unready 状态。当 GSXR Runtime 检测到系统或设备发生异常或断线时，渲染循环即进入 Unready 状态，代表渲染循环已无法正常工作，处于未备妥状态，此时 GSXR Runtime 需发送 GSXR_EventType_RendererState_Looper_Unready 事件给应用程序，当应用程序获取此事件时必须调用 GSXR_StopRenderLooper 停止渲染循环。

⑦ Stopping 状态。当 GSXR 应用程序成功调用 GSXR_StopRenderLooper 结束渲染循环，渲染循环即进入 Stopping 状态，代表此渲染循环处于结束状态。此时 GSXR Runtime 需发送 GSXR_EventType_RendererState_Looper_Stopping 事件给应用程序，并释放此渲染循环的系统资源。

⑧ Unavailable 状态。当 GSXR Runtime 释放完渲染循环的必要资源，渲染循环即进入 Unavailable 状态，代表此渲染循环已经处于不可用状态，此时 GSXR Runtime 需发送 GSXR_EventType_RendererState_Looper_Unavailable 事件给应用程序。待 GSXR Runtime 确认系统及设备状态正常后再重新进入 Available 状态。

（7）GSXR 帧同步与视图姿态。为了确保 GSXR 应用程序的渲染帧与显示同步，GSXR 提供了 GSXR_WaitFrame 函数限制 GSXR 应用渲染循环，XR Runtime 根据实际显示设备的刷新节奏调节 GSXR 应用程序调用 GSXR_WaitFrame 后的返回时机，并在返回函数时输出下一渲染帧的预测显示时间。XR 应用程序必须以此预测时间调用 GSXR_GetPoseState 获取设备姿态，以及调用 GSXR_GetViewPoseState 获取视图姿态，以预测该帧显示时的姿态进行内容渲染，补偿渲染与显示之间的时间差所造成的延迟。GSXR 应

用程序要正确渲染出适用 XR 显示设备的视图，必须先获取视图姿态以及投影参数。在开始渲染循环时所输入的视图形态（GSXR_ViewType）所对应的视图集配置（GSXR_ViewSetConfiguration），其成员视图数目（viewCount）便是 GSXR 应用程序调用 GSXR_GetViewPoseState 函数获取视图姿态的个数，GSXR 应用程序需根据各个视图的视图姿态及投影参数进行内容渲染，再将渲染帧提交给 GSXR Runtime。

（8）GSXR 帧提交与合成层。GSXR 应用程序在等待 GSXR_WaitFrame 返回并完成内容渲染后，必须调用渲染帧提交函数 GSXR_SubmitFrame，将渲染帧提交给 GSXR Runtime 渲染管线进行 GSXR 渲染后处理。GSXR 应用程序提交的渲染帧信息，可包含一个或多个合成层，再由 GSXR Runtime 进行合成，若输入的 GSXR_FrameSubmitInfo 的 layerCount 为 0，则 GSXR Runtime 必须清除当前的显示内容。GSXR 应用程序必须根据合成层的类型以及内容需求，选用相应的合成层结构体，备妥该合成层结构体所需的功能选项及参数，并指定此合成层所对应的渲染纹理子图像及视图姿态，将完整的合成层信息正确记录在 submitInfo 中，方可调用 GSXR_SubmitFrame 函数，提交渲染帧。合成层结构体将 GSXR_CompositionLayerHeader 结构作为结构标头，指定其合成层类型及合成层功能选项，合成层结构标头后的其余成员则根据该合成层类型分别制定，GSXR 应用程序必须输入正确的合成层信息，GSXR Runtime 方可正确输出最终的合成结果。

（9）预设置与后设置渲染。GSXR 提供延伸的渲染功能函数，供 GSXR 应用程序在内容渲染时启用 GSXR Runtime 支持的延伸渲染功能。渲染功能类型定义在 GSXR_RenderFunctionType 中。GSXR 应用程序使用的延伸渲染功能信息，需在 GSXR_FrameRenderInfo 中指定使用功能的所需参数，并在 GSXR_WaitFrame 之后及应用程序开始渲染帧内容之前，调用 GSXR_PreRenderFrame 预设置此内容帧所要使用的延伸渲染功能，下一帧循环也会持续沿用此设定，直至应用程序在 GSXR_PostRenderFrame 后设置停用此功能。GSXR_PreRenderFrame 及 GSXR_PostRenderFrame 并不是 GSXR 渲染循环的必须函数，GSXR 应用程序可根据自身内容的需要选择使用与否，使用时 GSXR_FrameRenderInfo 的 functionCount 不可为 0，且指定使用的延伸渲染功能必须是 GSXR Runtime 所支持的功能。

（10）注视点渲染。GPU 及其图形渲染 API 若支持注视点渲染功能，则 XR 应用程序便可使用此功能针对非注视点区域进行较低解析度的内容渲染，以节省资源及时间。注视点参数信息通过 GSXR_FoveatedParameters 加以描述，其（focalX,focalY）坐标以眼球中心为平面原点，并将两侧区间归一化成 –1~1 的数值，注视点（focalX,focalY）坐标便落于此区间中，再加上此注视点视场角的角度信息 foveaFov，以及非注视点区域的视觉品质 peripheralQuality，即形成完整的注视点参数信息。

任务 1.2 GSXR 插件下载及环境配置

GSXR 插件下载及环境配置. mp4

在正式开发 XR 应用前，开发者需要配置相关的开发环境，根据规范选择对应的插件工具，从而为后续 XR 应用功能开发打下基础。本任务主要介绍 XR Plugin Management 和 GSXR UnityXR Plugin 的下载、安装和配置。

1. 安装 XR Plugin Management

【步骤 1】在 Unity 菜单栏中，选择 Edit→Project Settings，如图 1-8 所示。

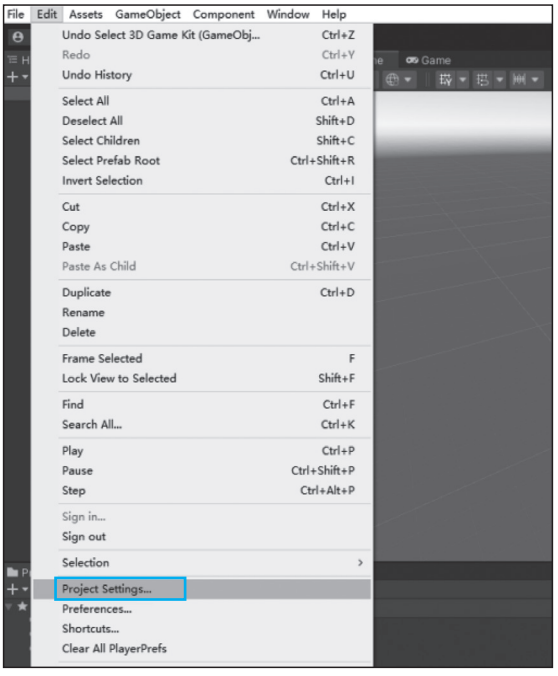

图 1-8　选择 Project Settings 选项

【步骤 2】在 XR Plugin Management 中单击 Install XR Plugin Management 进行安装，如图 1-9 所示。

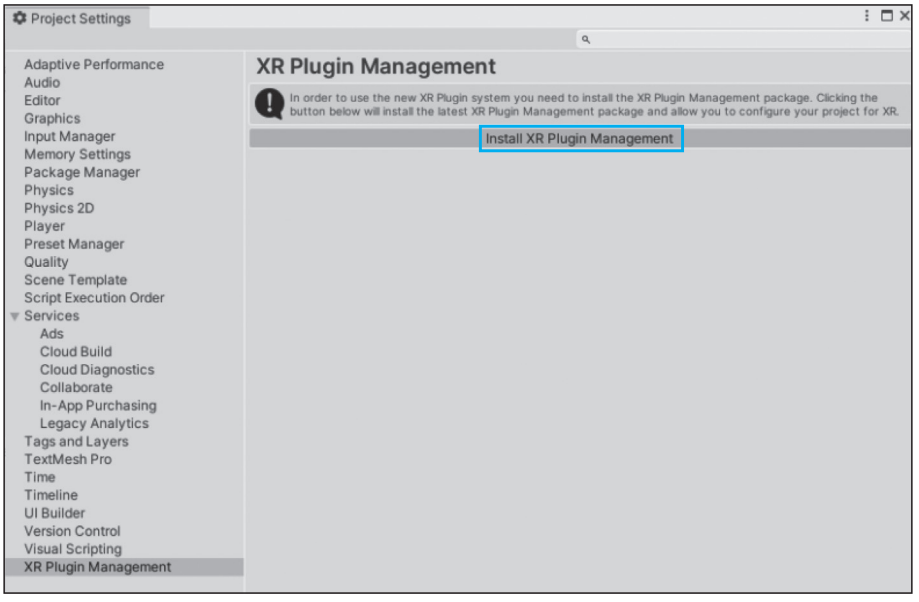

图 1-9　安装 XR Plugin Management

【步骤 3】安装成功，如图 1-10 所示。

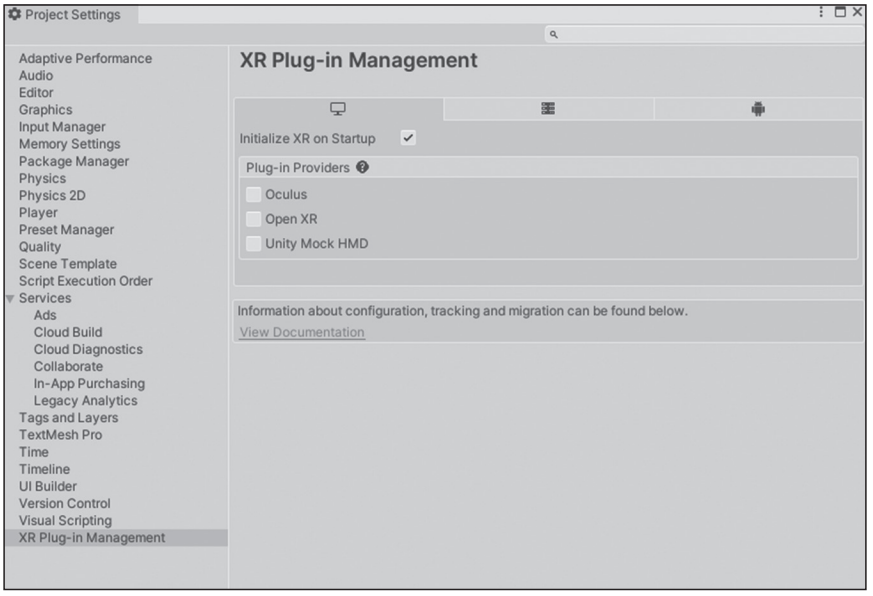

图 1-10　XR Plugin Management 安装成功

2. 在 Unity 中导入 GSXR UnityXR Plugin

【步骤 1】登录 GSXR 官网下载 GSXR UnityXR Plugin。

【步骤 2】单击 GSXR Unity SDK 下的“SDK 下载”，如图 1-11 所示。

图 1-11　GSXR UnityXR Plugin 下载界面

【步骤 3】在 Unity 菜单栏中，选择 Window→Package Manager，如图 1-12 所示。

【步骤 4】从磁盘中添加 GSXR UnityXR Plugin。单击“+”按钮打开下拉菜单，选择 Add package from disk，如图 1-13 所示。