PicoVRWindowsSDK_Unity

开发说明文档

版本：v_2.0.2

北京小鸟看看科技有限公司

目录

1 引言.....5
1.1 编写目的.....5
1.2 背景.....5
2 支持设备.....5
2.1 虚拟现实头盔.....5
3 开发环境.....6
3.1 推荐配置.....6
3.1.1 Unity 版本兼容.....6
3.2 PicoHome 行业版安装.....7
4 SDK Unity 包结构.....7
4.1 PicoVRSDK 目录.....7
4.2 PicoVRSDK.....8
4.2.1 Prefabs.....8
4.2.2 Resources.....8
4.2.3 Scripts.....8
4.3 PicoPaymentSDK_PC.....9
4.3.1 从开发者平台获取 AppKey、AppID 和 AppSecret 的方法.....9
4.3.2 支付 API 接口以及使用说明.....15
4.3.3 开发者平台游戏内配置使用说明.....16
4.3.4 内购 API 接口以及使用说明.....18
5 PicoVRSDK 开发过程指导.....20
5.1 PicoVRSDK 开发指导.....20
5.1.1 创建工程.....20
5.1.2 导入开发包.....21
5.1.3 替换相机.....22
5.1.4 设置场景.....23
5.1.5 FPS 使用.....24
5.1.6 模拟运行.....25

5.1.7 模拟操作.....26
5.1.8 编辑器调试.....26
5.1.9 开发基于 Pico Tracking Kit 的游戏.....27
5.1.10 PC 平台应用程序发布.....28
6 PicoVRSDK 功能接口说明.....29
6.1 PicoVRSDK 预置件.....29
6.2 PicoVRSDK 脚本.....30
6.2.1 PicoVRConfigProfile 脚本.....31
6.2.2 PicoVRBaseDevice 脚本.....31
6.2.3 PicoVRUnityDevice 脚本.....31
6.2.4 PicoVRWinPCDevice 脚本.....31
6.2.5 PVRPluginEvent 脚本.....31
6.2.6 SightInputModule 脚本.....31
6.2.7 PicoVREye 脚本.....32
6.2.8 PicoVREyeManager 脚本.....32
6.2.9 PicoVRHeadTrack 脚本.....33
6.2.10 PicoVRHandTrack 脚本.....34
6.2.11 PicoVRPose 脚本.....34
6.2.12 PicoVRManager 脚本.....35
6.3 Unity API.....35
6.3.1 初始化接口.....35
6.3.2 重置头部追踪姿态接口.....36
6.3.3 过程控制暂停接口.....36
6.3.4 Rendertexture 配置接口.....36
6.3.5 获取 Rendertexture 尺寸接口.....36
6.3.6 更新姿态旋转与位置状态接口.....37
6.3.7 获取头戴距离传感器状态接口.....37
6.3.8 获取手柄姿态接口.....38
6.3.9 获取手柄位置接口.....38
6.3.10 获取手柄状态接口.....38
6.3.11 触发手柄震动接口.....39
6.3.12 获取摄像头连接状态接口.....40
6.4 Windows API 说明.....40
6.4.1 SDK 初始化.....40
6.4.2 SDK 初始化注销.....41
6.4.3 获取头戴姿态数据.....41
6.4.4 获取头戴位置信息.....41
6.4.5 获取头戴分辨率信息.....42
6.4.6 获取 Render Texture 大小.....42

6.4.7 获取头戴连接状态.....43
6.4.8 头部追踪姿态重置.....43
6.4.9 获取头戴距离传感器状态接口.....44
6.4.10 获取手柄姿态数据.....44
6.4.11 获取手柄位置信息.....45
6.4.12 获取手柄状态信息.....45
6.4.13 触发手柄震动接口.....46
6.4.14 获取摄像头连接状态.....47
6.4.15 游戏支付验证.....48
6.4.16 游戏道具内购.....48
7 FAQ.....49

1 引言

1.1 编写目的

撰写“PicoVRWindowsSDK_Unity”开发说明文档是为了帮助开发者能够快速地开发基于 PicoSDK 的应用。

本文档主要面向对象为：技术人员、开发者。

1.2 背景

PicoVRWindowsSDK_Unity 是北京小鸟看看科技有限公司为了配合广大的 Unity 3D 软件开发者开发，适用于 Pico Neo 虚拟现实头盔而推出的 SDK（Software Development Kit）软件开发工具包。开发包主要支持头部跟踪、畸变校正、双目视差等功能，为开发者提供便利支持。本说明文档中的 SDK 通过 unity package 包的格式发布，接入 SDK 前，需要配置好 Unity 开发环境，再导入开发包，在 Assets\PicoVRSDK\Scenes 文件夹下我们提供了 demo 场景供开发者参考。

2 支持设备

2.1 虚拟现实头盔

厂商	产品
小鸟看看	Pico Neo DK Pico Neo DKS

3 开发环境

3.1 推荐配置

软件名称	版本信息
操作系统	Windows 7 64 位 Windows 8 64 位 Windows 10 64 位
内存	8G 及以上
显卡	NVIDIA GeForce GTX 970 及以上； AMD Radeon R9 290 及以上
显卡驱动	NVIDIA 显卡：GeForce Game Ready Driver 361.91 及以上； AMD 显卡：AMD Radeon Software Crimson Edition 16.3.2 及以上 （推荐更新到 NVIDIA 和 AMD 官网最新显卡驱动）
Unity3D	5.2 及以上

3.1.1 Unity 版本兼容

PicoVRWindowsSDK_Unity 可以适配 Unity5.2 及以上版本

3.2 PicoHome 行业版安装

从官网 http://www.picovr.com/picohome.html 下载 PicoHome 行业版。运行 PicoHome 行业版安装包，按照提示一步步安装。安装完成后需要重启系统。PicoHome 启动界面如下。

图 3.1 PicoHome 主界面

4 SDK Unity 包结构

4.1 PicoVRSDK 目录

成功导入 PicoVRSDK.unitypackage 后可以在工程中看到如下图所示的层级目录

- “PicoPaymentSDK_PC”目录下提供 Windows 开发支付相关 API
- “PicoVRSDK”目录下提供开发 VR 应用的 SDK

- “Plugins”目录下存放程序需要运行的.dll 文件

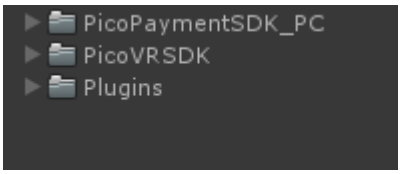


图 4.1 SDK 目录结构

4.2 PicoVRSDK

4.2.1 Prefabs

- PicoVR.prefab：三维立体显示预置件

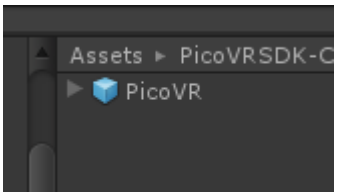


图 4.2 PicoVR.prefab

4.2.2 Resources

- 编辑器模式使用的 shader 文件

图 4.3 shader 文件

4.2.3 Scripts

- SDK 核心脚本文件

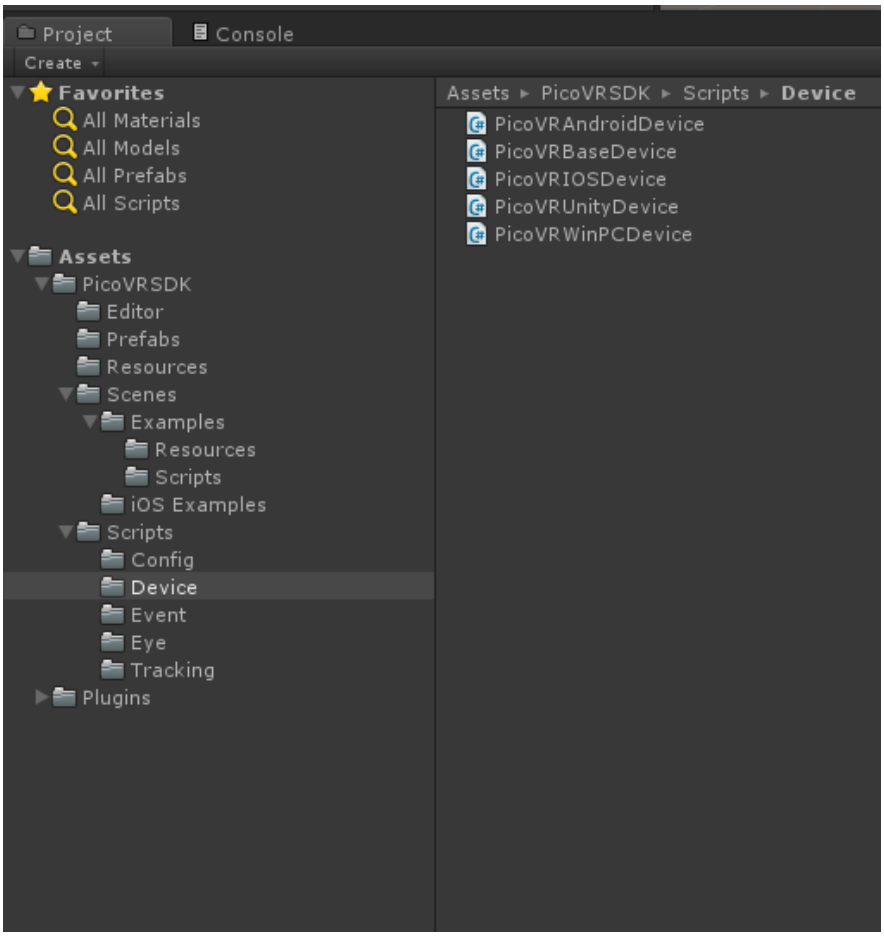


图 4.4 SDK 核心脚本文件

4.3 PicoPaymentSDK_PC

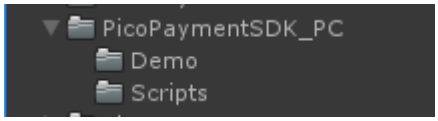


图 4.5 Pico 支付 SDK

- Demo 以及 Scripts 提供支付使用的 demo

4.3.1 从开发者平台获取 AppKey、AppID 和 AppSecret 的方法

目前开发者平台支持 PicoStore、PicoHome、PicoVR 的应用发布及管理功能。开发者在接入支付 SDK 时，需要在开发者平台创建应用并获取相应字符串。申请流程如下：

1.登录开发者平台并注册 Pico 会员（http://dev.picovr.com/）

在申请开发者前，需要先成为 Pico 会员。

2.申请成为开发者

开发者分为个人开发者和企业开发者，请根据实际情况进行申请。审核提交后，我们会在 3 个工作日内进行反馈，请及时查看开发者平台状态。

3.查看商户 ID

申请成为开发者后，点击右上角昵称可以查看到开发者 ID，即商户 ID。

图 4.6 商户 ID

4.创建应用

开发者可以从管理中心进入到创建应用阶段。通过创建应用可以获取到 AppKey、AppID 和 AppSecret 等字符串。

图 4.7 创建应用

点击创建应用后，可以选择要发布的平台。

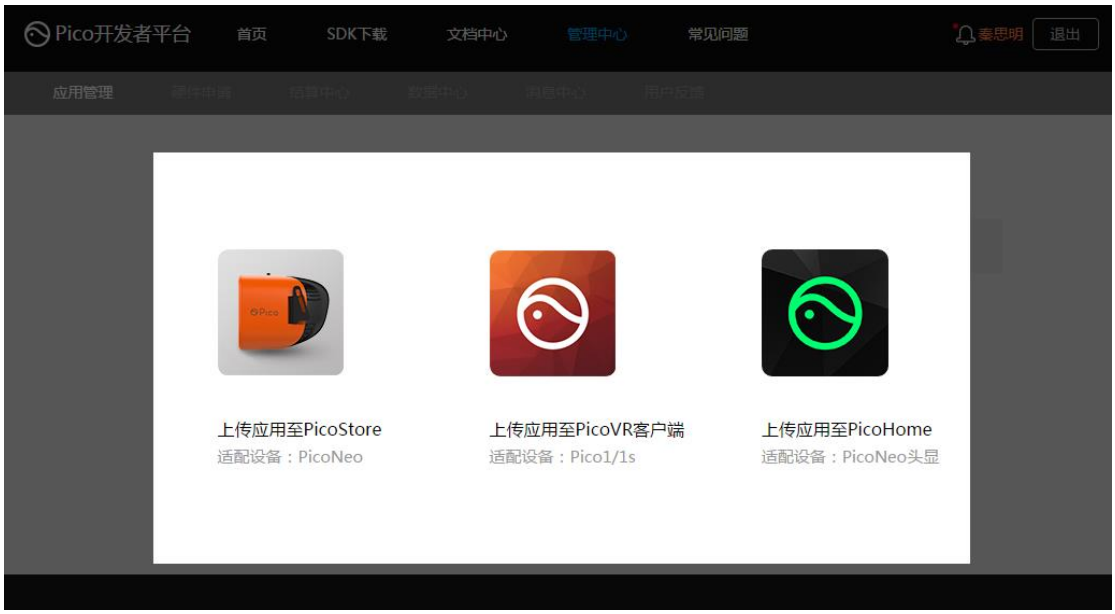


图 4.8 选择发布平台

进入相应平台后即可完善应用的相关信息（具体规则见开发者前端页面）。请重点注意上图标红位置，请谨慎填写应用类型，一经填写是无法进行修改的！游戏类应用如果存在道具内付的情况，我们要求开发者必须采用开发者后台增加商品码的方式进行统一管理。

图 4.9 完善信息

成功创建应用后，开发者平台会对其分配字符串。

图 4.10 分配字符串

上传 PicoHome 游戏时需要注意文件格式，且需要在游戏安装包内进行简单的配置才可以最终生效。资源大小由平台自动算出，版本名称需要由开发者自行填写。

图 4.11 上传应用

5.PicoHome 平台 Pico 游戏配置文件说明

第一步：为了保证游戏在 PicoHome 中顺利运行，需要在游戏包可执行文件（.exe 文件）同级别目录下制作一个 manifest.ini 文件。

（注意：.exe 为启动游戏的主程序文件）

第二步：manifest.ini 文件格式如下：

exe=”游戏可执行文件名称”

如下图示例。

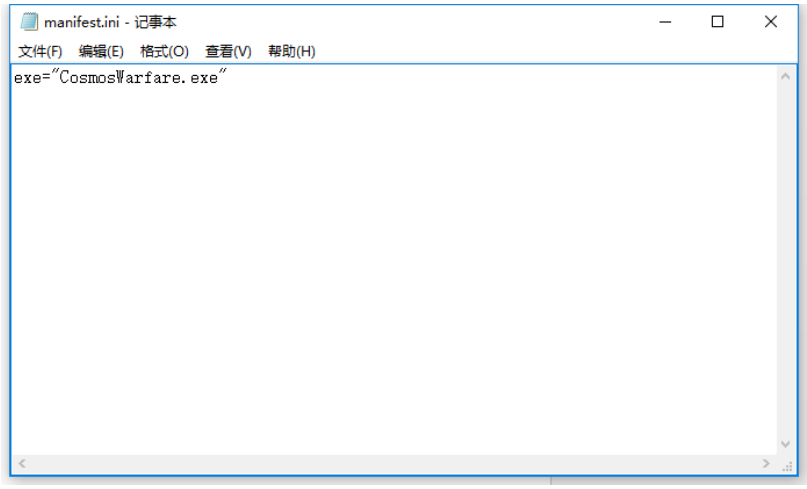


图 4.12 manifest 格式示例

最终文件包格式如下图示例：

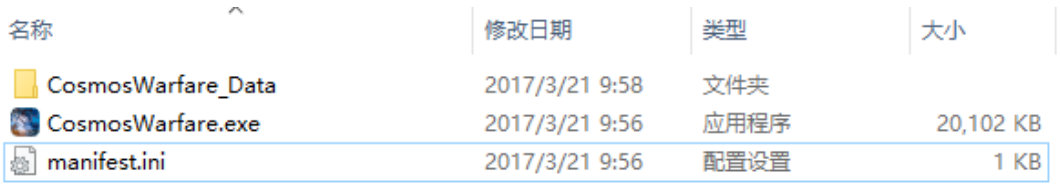


图 4.13 最终文件包格式

第三步：在 PicoHome 平台上上线的 pico 游戏，在后台部署时需要将游戏文件打包成 .zip 文件。（注意：直接将所有游戏文件打包，而不是将存放游戏的文件夹打包）。如下图示例：

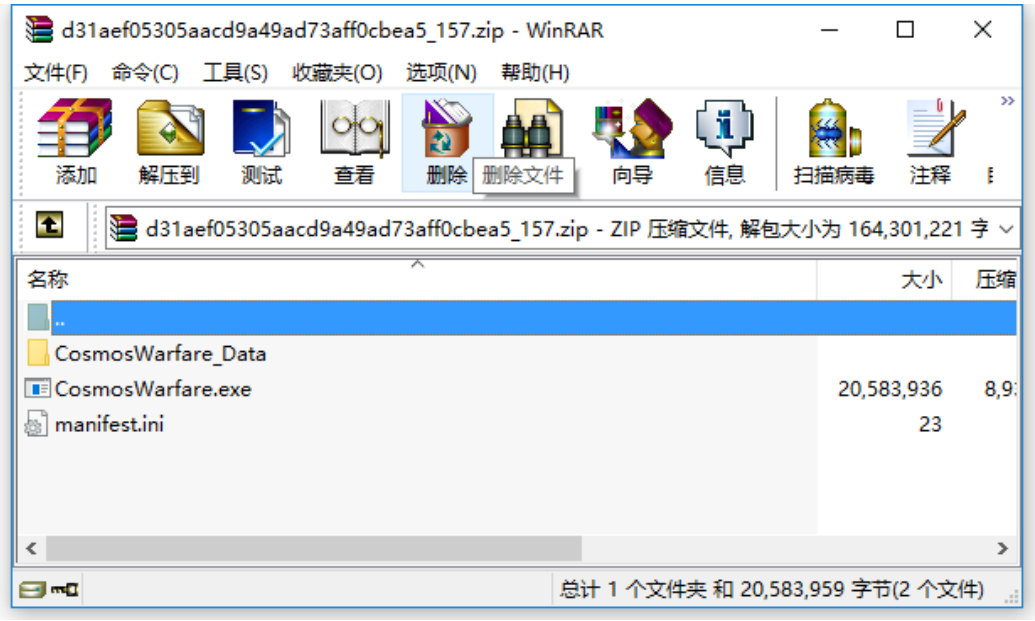


图 4.14 正确示例

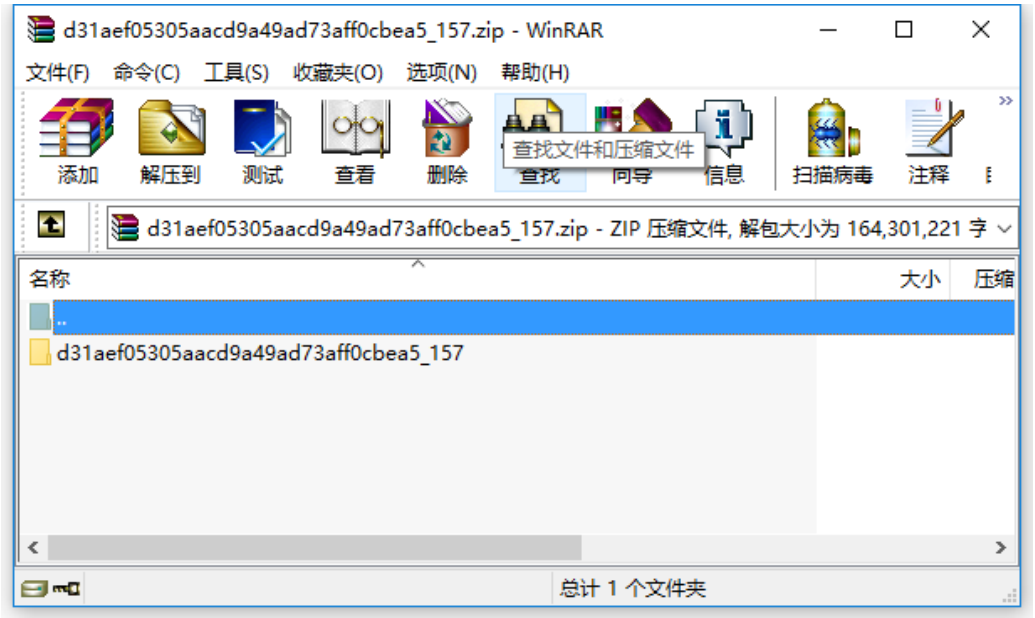


图 4.15 错误示例

4.3.2 支付 API 接口以及使用说明

- 导入 PicoPayment.dll，调用 PicoVerify 接口

```
private int PicoVerify(string gameId)
{
    return PVR_Verify(gameId);
```

```
}
[DllImport("PicoPayment")]
private static extern int
PVR_Verify([MarshalAs(UnmanagedType.LPStr)]string  b_gameId);
```

- API 说明
  - int PicoVerify(string gameId)
  - 函数功能：调用支付验证接口
  - 输入参数：gameId，游戏 ID，即开发者在 Pico 开发者平台上注册申请的 AppID
  - 输出参数：无
  - 返回值：

    0：验证通过！

    1：未登录 PicoHome！

    2：未购买该游戏！

    3：无法启动 PicoHome！

    其他：调用失败！

4.3.3 开发者平台游戏内配置使用说明

配置前提：游戏内支付配置仅针对“游戏类型”的应用生效！应用类型的选择在创建应用时勾选，一经选择无法修改。

创建游戏后会引导开发者进行游戏内支付配置。建议在上传应用前先进行游戏配置，减少审核过程中出现漏查漏测的现象。

图 4.16 游戏内支付配置

游戏内配置采用商品码配置的方式进行，开发者需要对游戏内支付的道具做出相应配置才可生效。

图 4.17 游戏内支付配置

商品码的规则定义为‘首位为字母，仅允许输入字母及数字，不超过 20 位字符’。不同道具间的商品码不能重复。

道具类型分为可消耗道具和不可消耗道具。可消耗道具为可重复购买的商品，如金币、血瓶等；不可消耗道具为一次性购买产品，如武器、解锁关卡。

4.3.4 内购 API 接口以及使用说明

- 导入 PicoPayment.dll，调用 PVR_Purchase 接口

```
public int PicoPurchase(string[] list )
{
   return PVR_Purchase(list[0], list[1], list[2], list[3], list[4], list[5],
    list[6]);
 }
[DllImport("PicoPayment")]
private static extern int
PVR_Purchase([MarshalAs(UnmanagedType.LPStr)]string mch_id,
[MarshalAs(UnmanagedType.LPStr)]string app_id ,
```

[MarshalAs(UnmanagedType.LPStr)]string app_key,

[MarshalAs(UnmanagedType.LPStr)]string app_secret,

[MarshalAs(UnmanagedType.LPStr)]string total_fee,

[MarshalAs(UnmanagedType.LPStr)]string pay_code,

[MarshalAs(UnmanagedType.LPStr)]string out_trade_no);

- API 说明
  - int PVR_Purchase(const char* mch_id, const char* app_id, const char* app_key, const char* app_secret, const char* total_fee, const char* pay_code, const char* out_trade_no)
  - 函数功能：调用游戏内购接口
  - 输入参数：mch_id 开发者 id

    app_id 开发者在 Pico 开发者平台上注册申请的 app_id

    app_key 开发者在 Pico 开发者平台上注册申请的 app_key

    app_secret 开发者在 Pico 开发者平台上注册申请的 app_secret

    total_fee 要购买的道具金额

    pay_code 商品代码

    out_trade_no 商户内部订单号，32 个字符以内，可包含字母，不同订单不能相同
  - 输出参数：无
  - 返回值：

    0：成功

    1：密码错误

    2：登录失败

3：余额不足

4：创建订单失败

5：购买失败

5 PicoVRSDK 开发过程指导

5.1 PicoVRSDK 开发指导

5.1.1 创建工程

- 点击 File 菜单

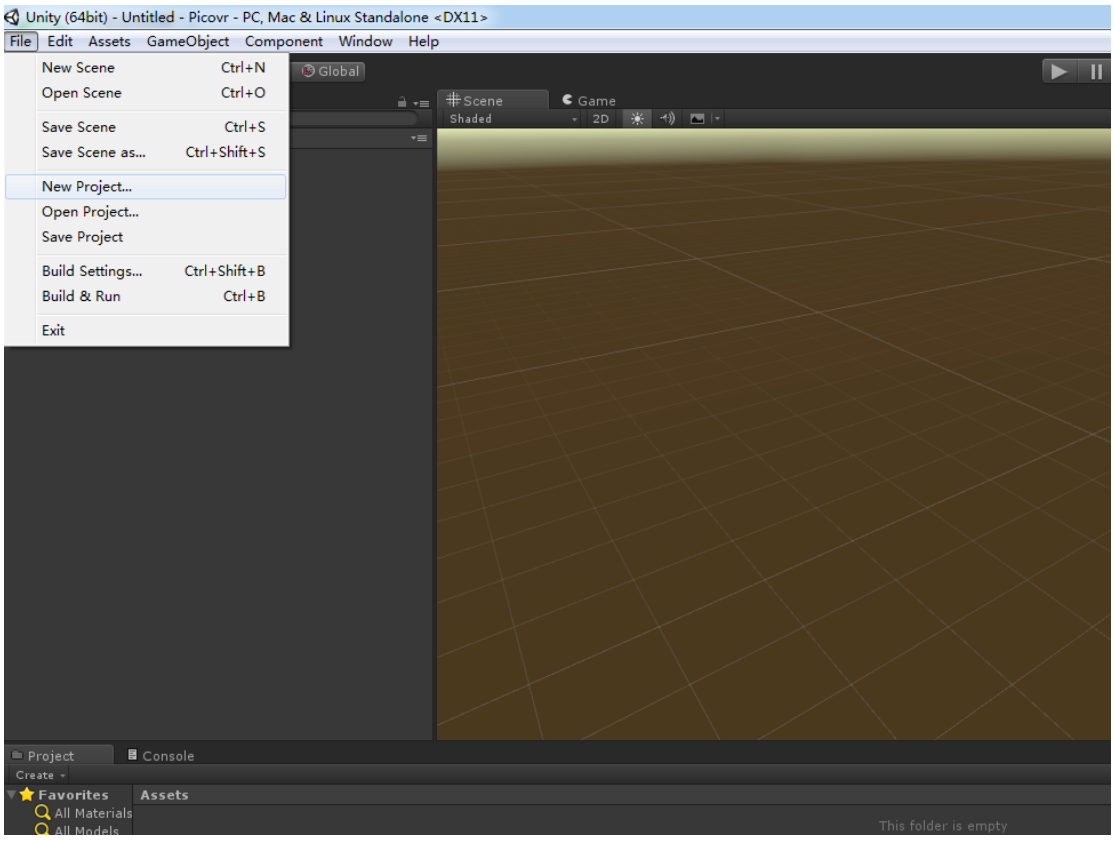


图 5.1 File 菜单

- 选择 New Project

- 按照 Unity 的引导新建一个工程

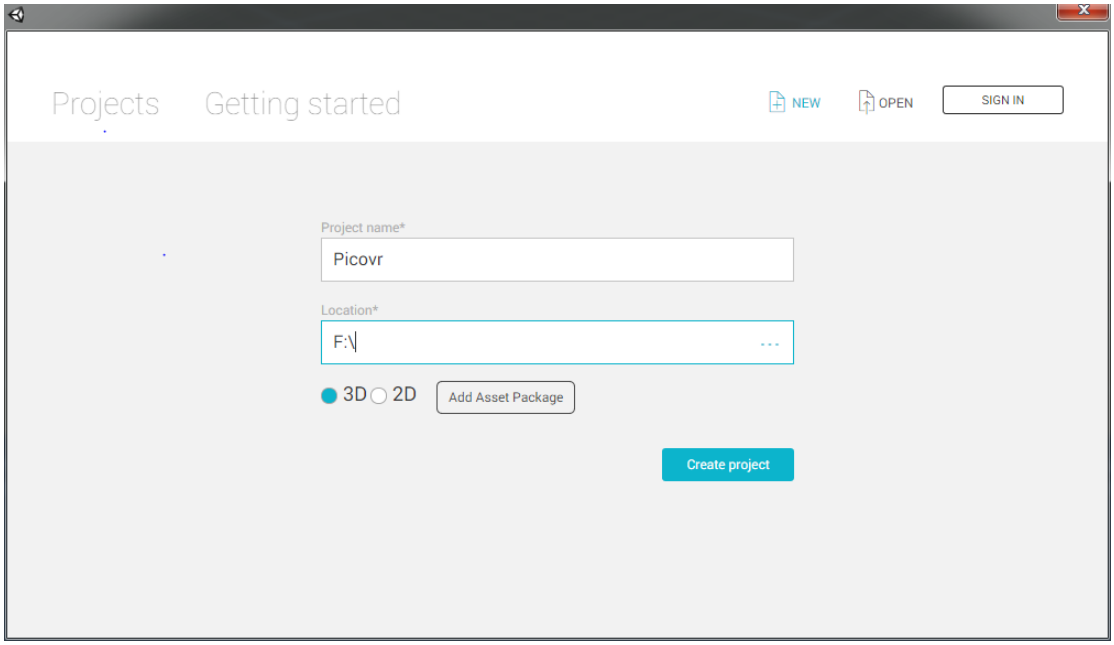


图 5.2 创建 unity 工程

5.1.2 导入开发包

- 点击 Assets 菜单
- 选择 Import Package 菜单的 Custom Package

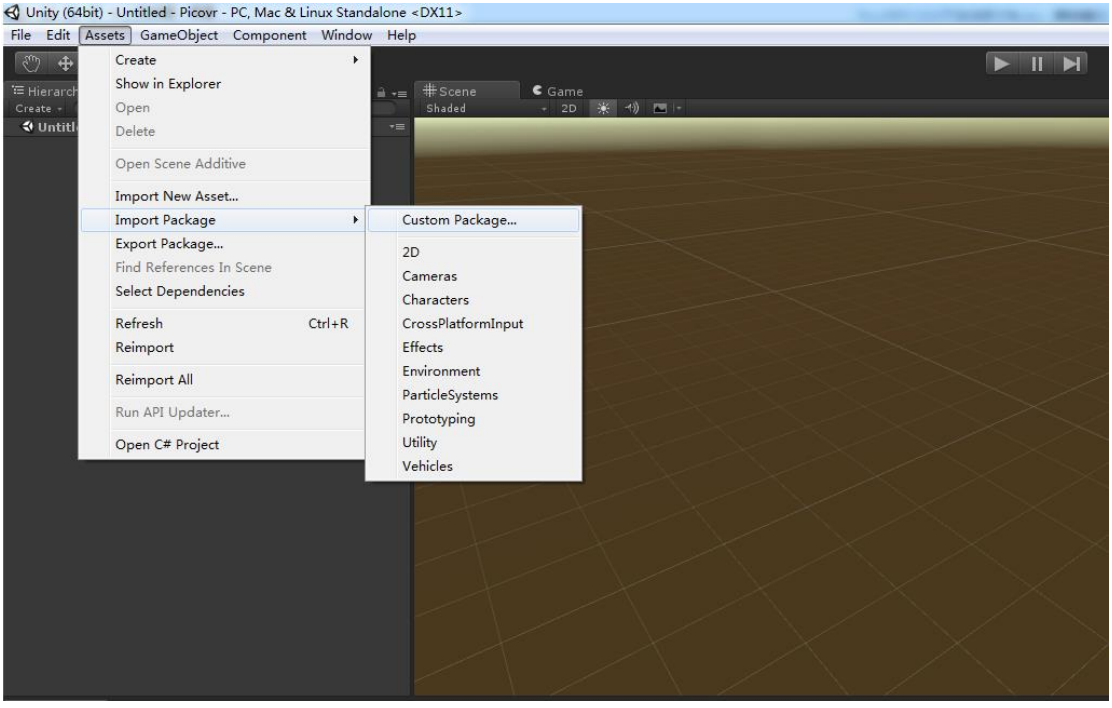


图 5.3 Import Package

- 浏览到 PicoVRSDK.unitypackage
- 点击 Import，导入所有内容

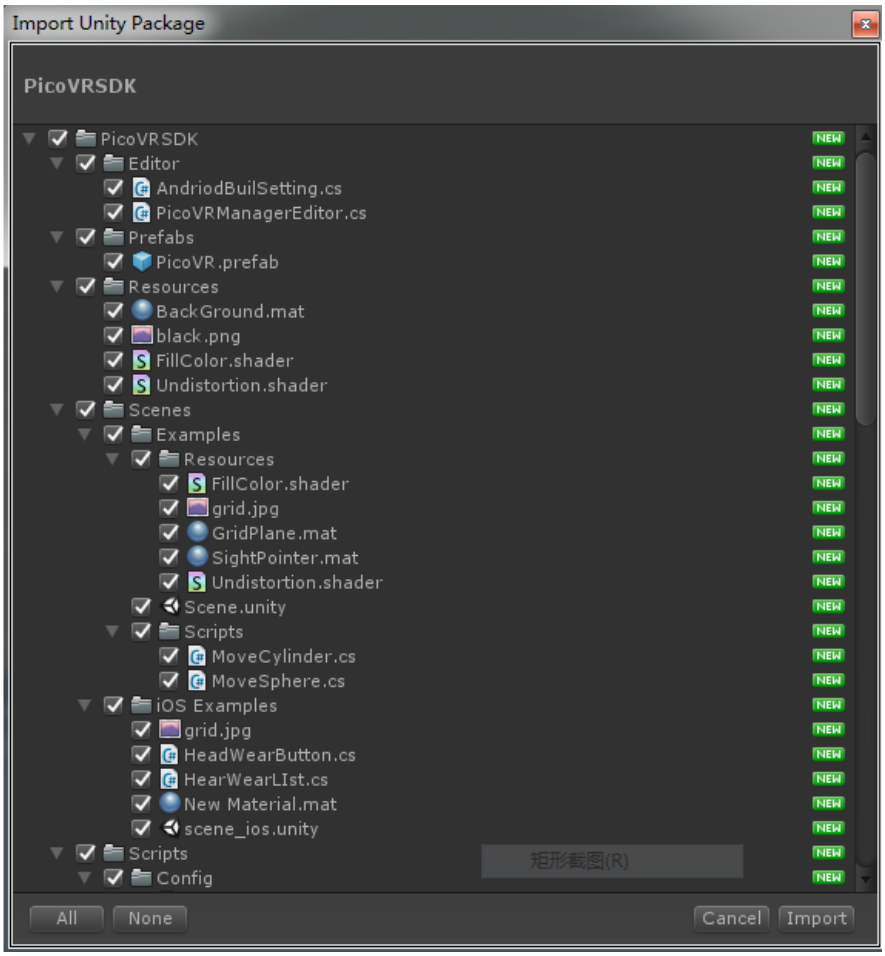


图 5.4 导入 Package

5.1.3 替换相机

- 删除掉场景中自带的 MainCamera
- 找到 Project 视图中 Prefabs 文件夹

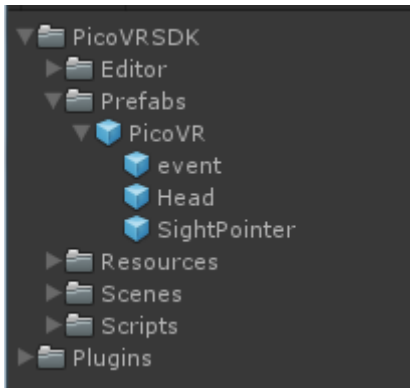


图 5.5 Prefabs 文件夹

- 将 PicoVR.prefab 拖动至场景
- 设置 position 为“0,0,0”（这里为测试使用，开发者可以把该预置件拖动至想要的场景位置）

5.1.4 设置场景

- 创建 cube1，设置其 position 信息如下

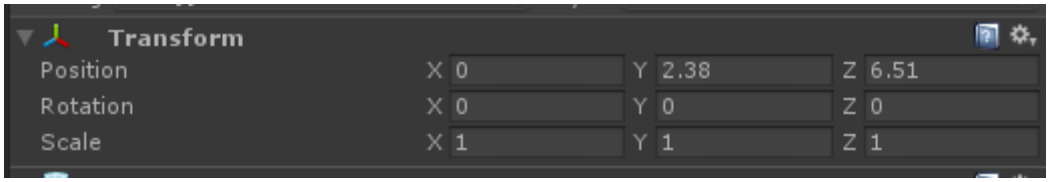


图 5.6 cube1 属性

- 创建 cube2，设置其 position 信息如下

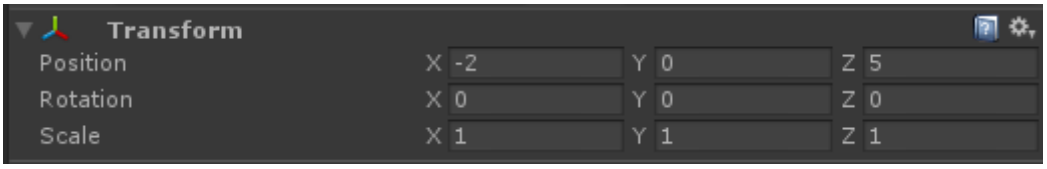


图 5.7 cube2 属性

- 创建 cube3，设置其 position 信息如下

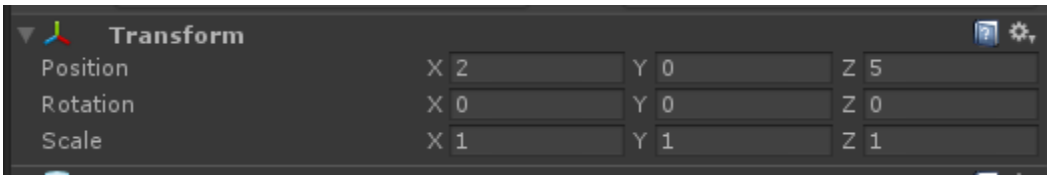


图 5.8 cube3 属性

- 配置完成后场景中的层级如图所示：

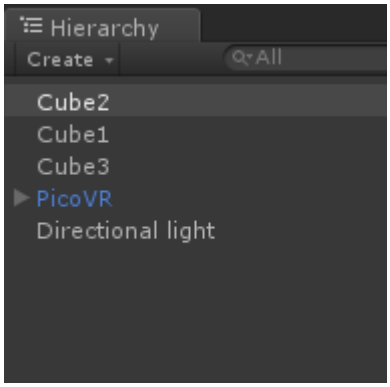


图 5.9 Scene 场景层级

5.1.5 FPS 使用

- FPS 显示仅作为开发测试使用，FPS 计算说明参见 FPS_S.cs 代码注释。
- FPS 使用方法：点击勾选下图 Show FPS。

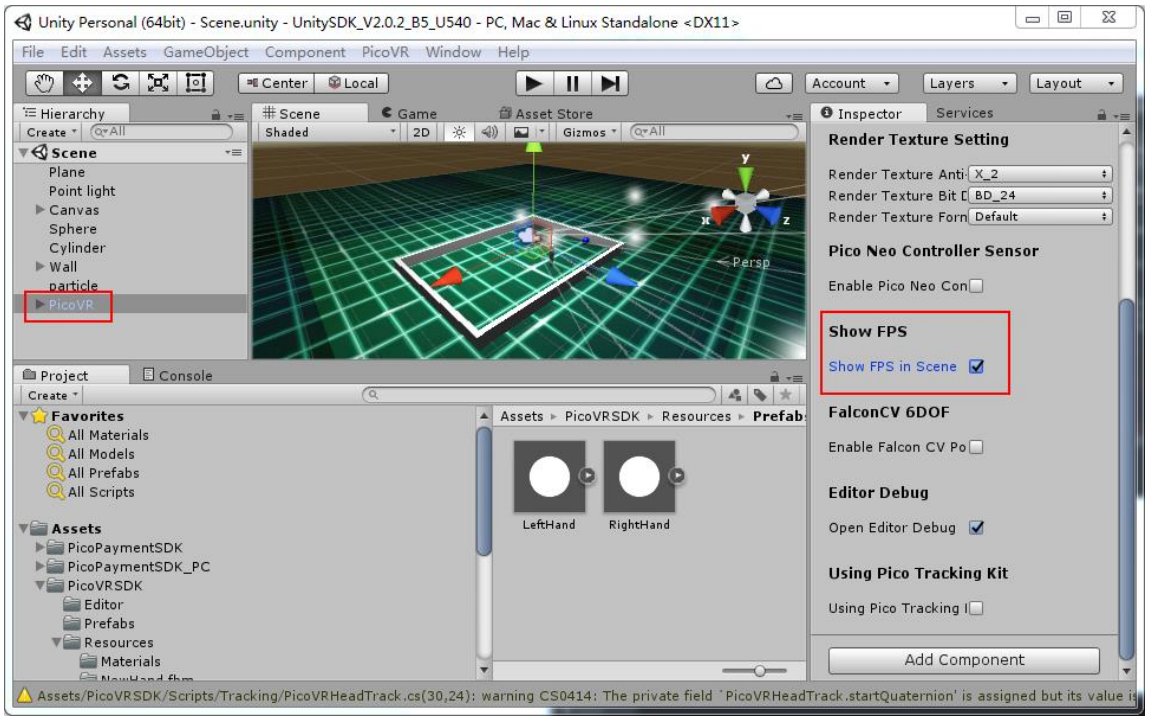


图 5.10 显示 FPS

5.1.6 模拟运行

- 点击编辑器运行，结果如下

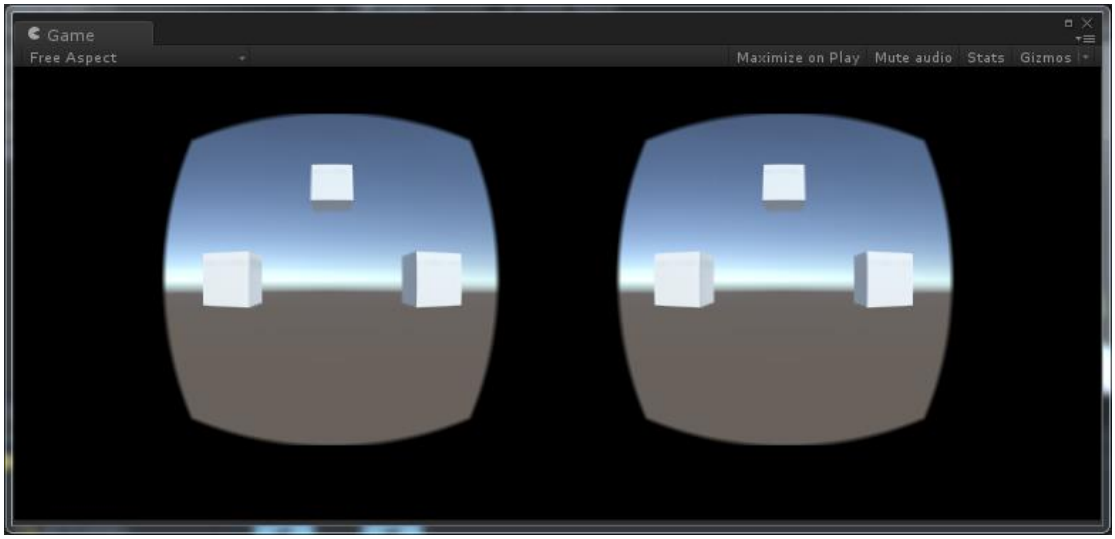


图 5.11 运行示例图

5.1.7 模拟操作

- 按住 Alt+鼠标左键，移动鼠标，画面跟着上下左右转动

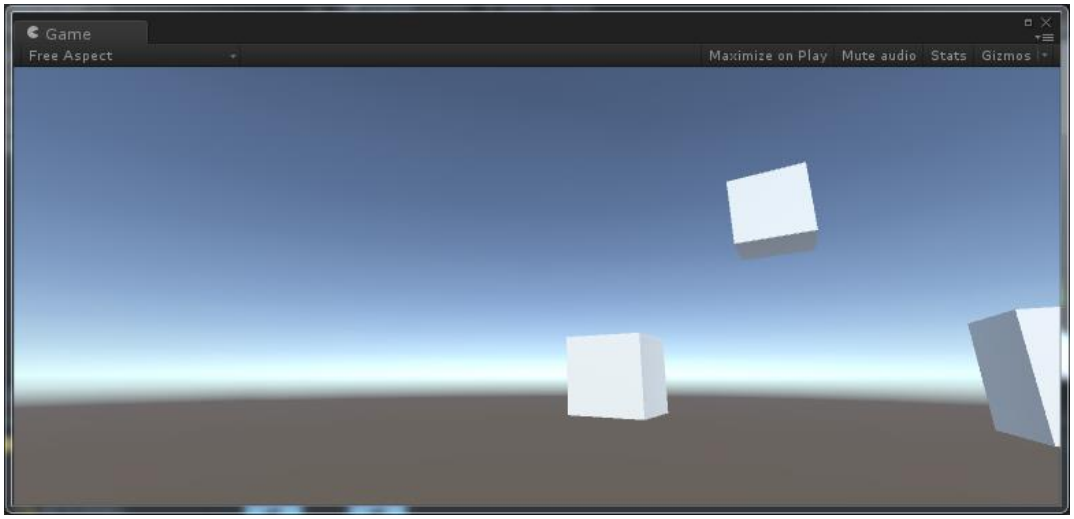


图 5.12 模拟操作示意图

5.1.8 编辑器调试

- 选中 PicoVR prefab ,在 Inspector 窗口中的 Editor Debug 选项中 ,选中 Open Editor Debug。然后点运行按钮即可看到头戴中的 Demo 画面，并且场景随头戴的晃动而改变。

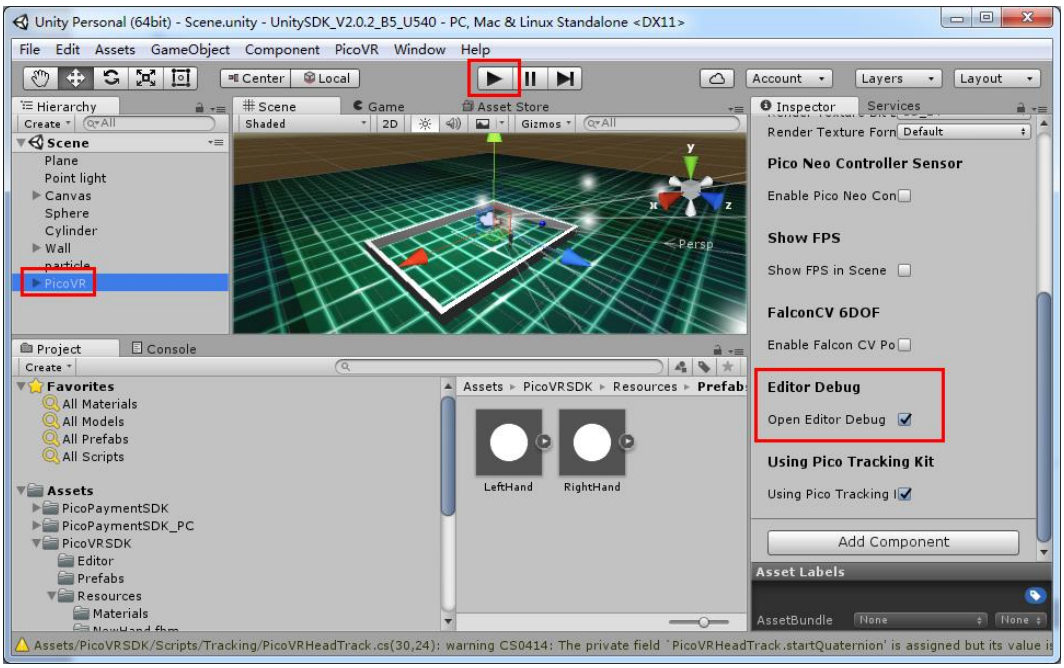


图 5.13 编辑器调试示意图

5.1.9 开发基于 Pico Tracking Kit 的游戏

- 如果要开发使用 Pico Tracking Kit 的游戏，则需要选中 Using Pico Tracking Kit。否则，如果开发基于单头盔的游戏，则不要选中该项。

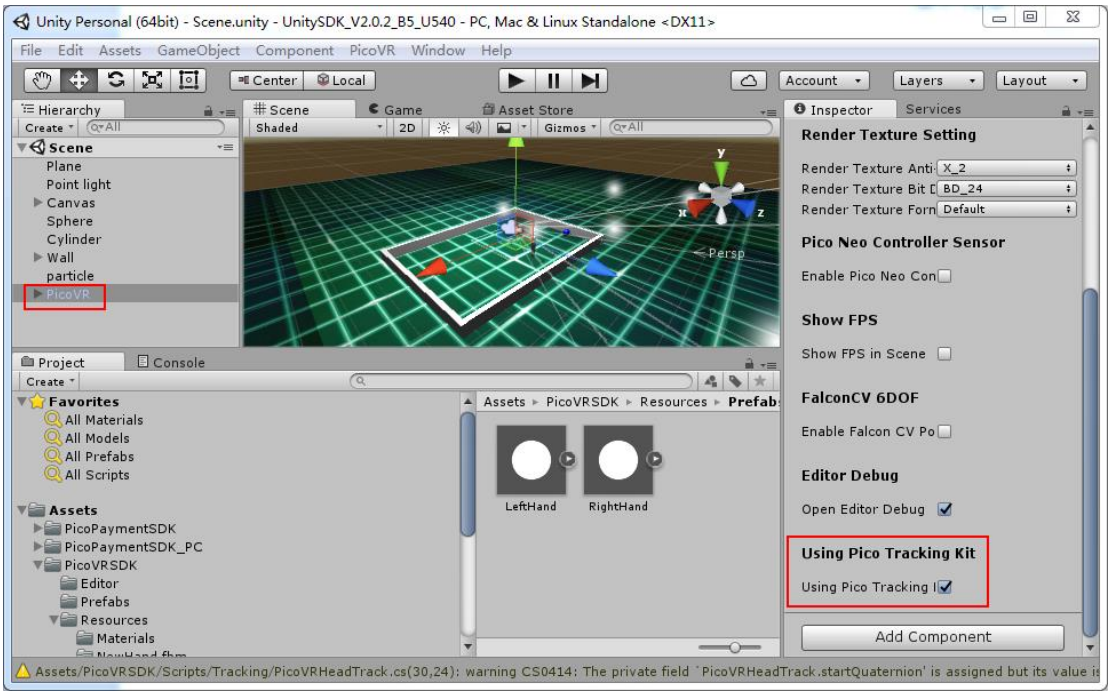


图 5.14 启用 PicoTrackingKit 示意图

5.1.10 PC 平台应用程序发布

- 在 Build settings 中首先添加场景

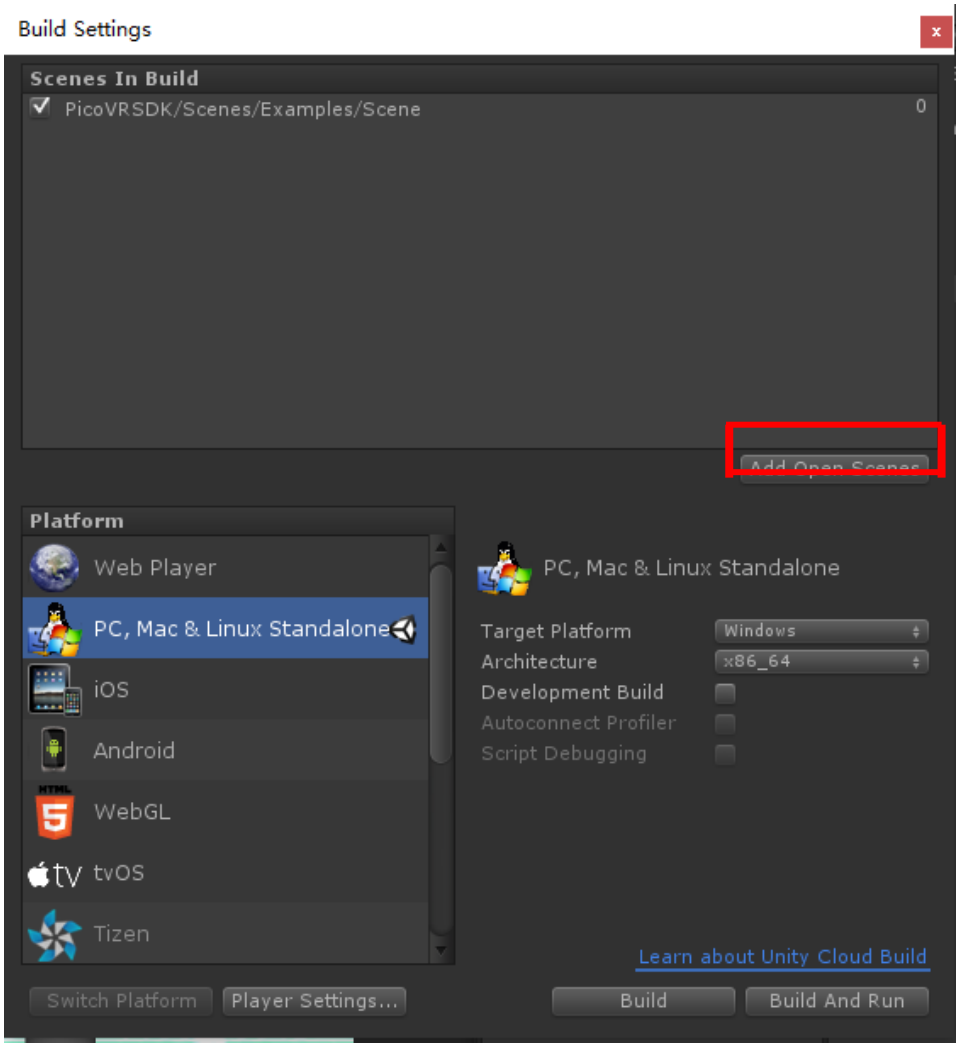


图 5.15 添加场景

- 在 Platform 选项中选择 PC、Mac&Linux Standalone，点击 Switch Platform。

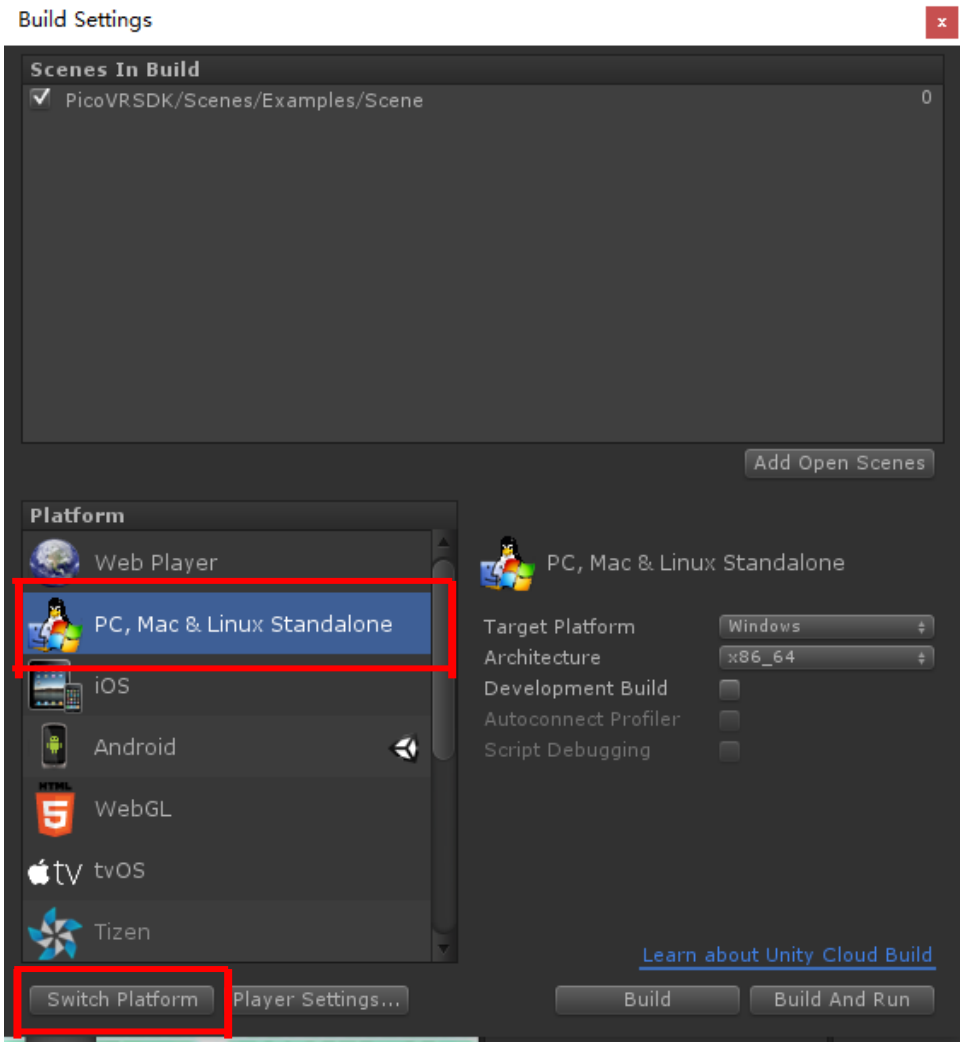


图 5.16 Switch Platform

- 点击 Build，PC 应用打包成功

6 PicoVRSDK 功能接口说明

6.1 PicoVRSDK 预置件

- 控制场景内相机的位置，PicoVR.prefab
- 该预置件组织整理了 Head 的相关层级及功能

图 6.1 PicoVR.prefab 层级

- 在 Unity 编辑中建议使用 PicoVR 预置件。使用方法如下：
  - 新建场景
  - 删除场景中的 Main Camera
  - 使用\Assets\PicoVRSDK\Prefabs\下 PicoVR 预置件

6.2 PicoVRSDK 脚本

- 脚本目录结构如下：

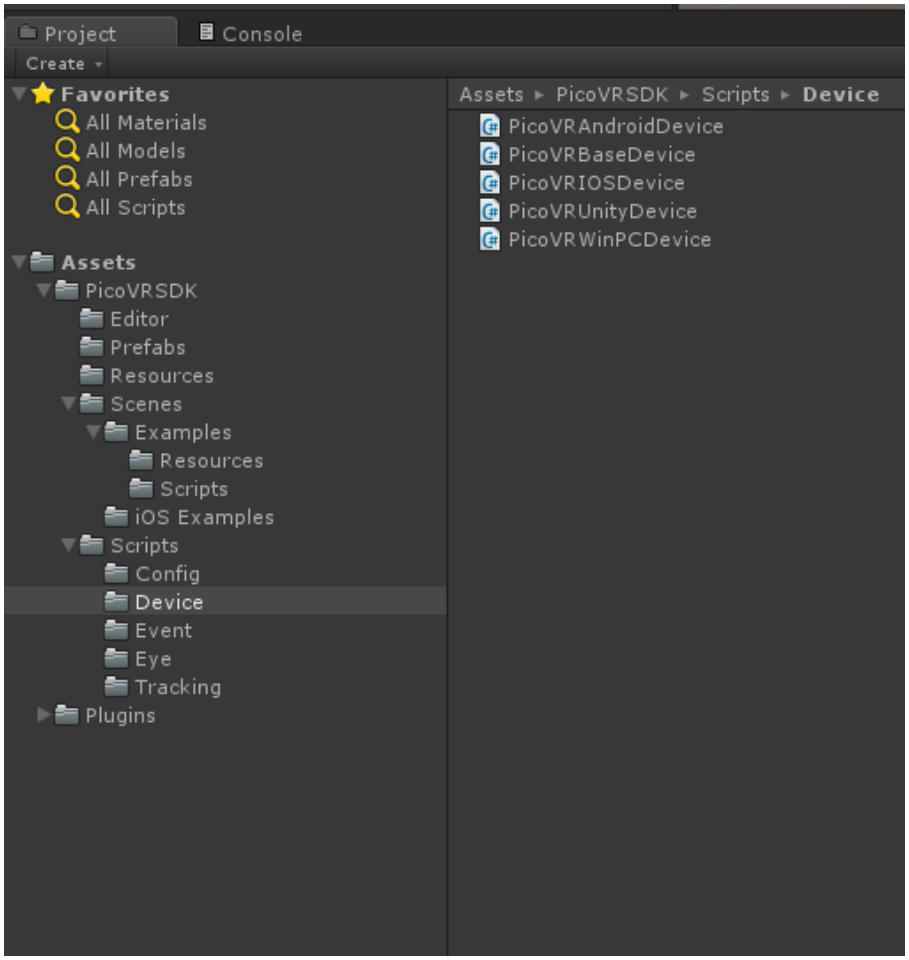


图 6.2 脚本目录结构

6.2.1 PicoVRConfigProfile 脚本

- 位置\Assets\PicoVRSDK\Scripts\Config\
- 主要功能：作为参数配置的文件，普通类

6.2.2 PicoVRBaseDevice 脚本

- 位置\Assets\ PicoVRSDK\Scripts\Device\
- 主要功能：依据平台使用不同脚本，普通类

6.2.3 PicoVRUnityDevice 脚本

- 位置\Assets\PicoVRSDK\Scripts\Device\
- 主要功能：依据平台使用不同脚本，普通类，继承 PicoVRBaseDevice

6.2.4 PicoVRWinPCDevice 脚本

- 位置\Assets\PicoVRSDK\Scripts\Device\
- 主要功能：依据平台使用不同脚本，普通类，继承 PicoVRBaseDevice

6.2.5 PVRPluginEvent 脚本

- 位置\Assets\PicoVRSDK\Scripts\Event\
- 主要功能：底层事件交互包括渲染，过程控制等

6.2.6 SightInputModule 脚本

- 位置\Assets\PicoVRSDK\Scripts\Event\

- 主要功能：光标拾取
- 编辑器可修改参数：

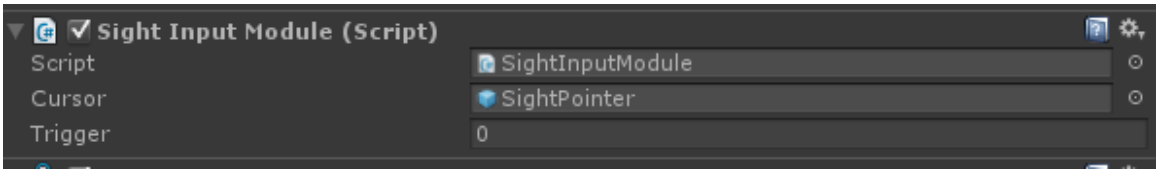


图 6.3 SightInputModule 可编辑属性

- Cursor： 光标 object
- Trigger：控制呼出键盘， 1 表示可以呼出， 0 表示不能呼出

6.2.7 PicoVREye 脚本

- 位置\Assets\PicoVRSDK\Scripts\Eye\
- 主要功能：左右相机管理控制脚本
- 编辑器可修改参数：

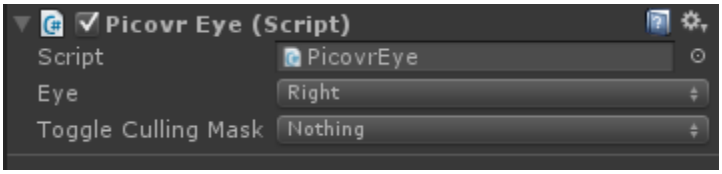


图 6.4 Picovr Eye 可编辑属性

- Eye： 左右眼标志位
- Toggle Culling Mask： 相机剔除 mask

6.2.8 PicoVREyeManager 脚本

- 位置\Assets\PicoVRSDK\Scripts\Eye\
- 主要功能：控制左右眼的场景绘制
- 编辑器可修改参数：

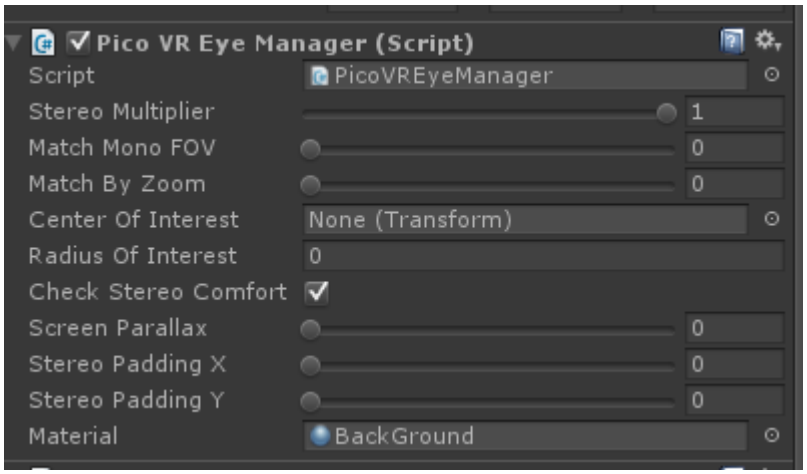


图 6.5 Picovr eye controller 可编辑属性

- Direct Render： 直接输出或者是通过缓冲区输出
- Stereo Multiplier： 设置相机 level
- Match Mono Fov： 立体视角 FOV 调整幅度设置与 Match By Zoom 配合使用
- Match By Zoom： 当立体视角 FOV 调整幅度为 0 时，zoom 不起作用，当立体视角 FOV 调整幅度大于 0，立体视角可放大缩小

6.2.9 PicoVRHeadTrack 脚本

- 位置位置\Assets\PicoVRSDK\Scripts\Tracking\
- 主要功能：模拟跟踪头部旋转和位置
- 编辑器可修改参数：

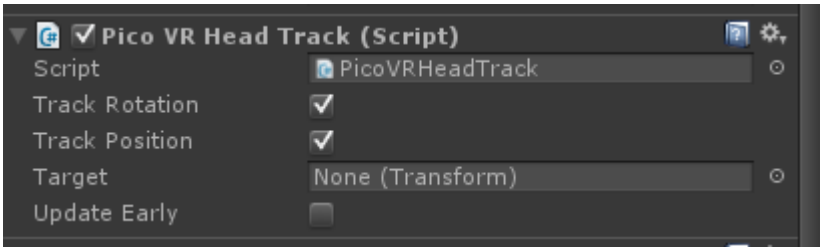


图 6.6 Picovr head 可编辑属性

- Track Rotation： 是否启用头部旋转跟踪

- Track Position： 是否启用头部的位置跟踪
- Target： 三维坐标信息
- Update Early： 每帧跟踪时机

6.2.10 PicoVRHandTrack 脚本

- 位置位置\Assets\PicoVRSDK\Scripts\Tracking\
- 主要功能：模拟跟踪手部旋转和位置
- 编辑器可修改参数

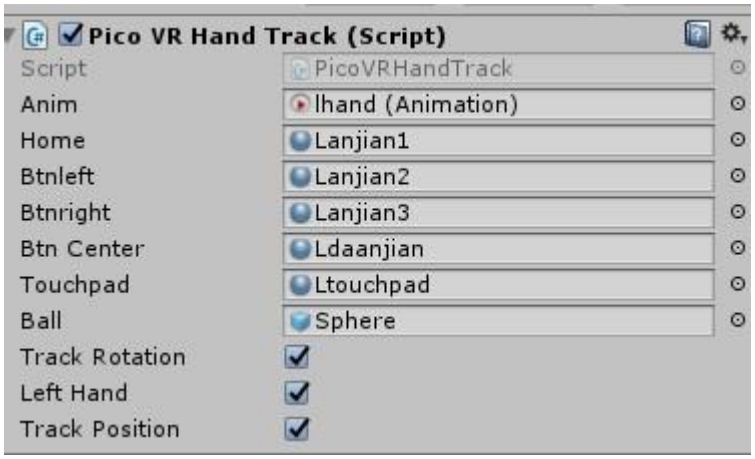


图 6.7 Picovr hand 可编辑属性

- Track Rotation： 是否启用头部旋转跟踪
- Left Hand：是否是左手柄
- Track Position： 是否启用头部的位置跟踪

6.2.11 PicoVRPose 脚本

- 位置位置\Assets\PicoVRSDK\Scripts\Tracking\
- 主要功能：定义与设置头部位置与旋转，普通类

6.2.12 PicoVRManager 脚本

- 位置\Assets\PicoVRSDK\Scripts\
- 主要功能：SDK camera 总控制
- 编辑器可修改参数：

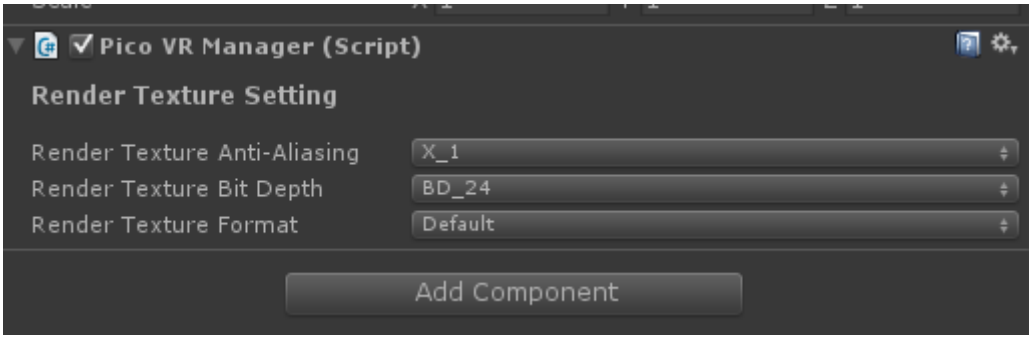


图 6.8 Rendertexture 设置

- Render Texture Anti-Aliasing： Render Texture 抗锯齿选项
- Render Texture Bit Depth： Render Texture 位深选项
- Render Texture Format： Render Texture 格式选项

6.3 Unity API

6.3.1 初始化接口

- private void InitDevice()
- 函数功能：初始化功能
- 输入参数：无
- 返回值：无
- 接口调用：在 PicoVRManager 类中调用

6.3.2 重置头部追踪姿态接口

- public void ResetHeadTrack()
- 函数功能：重置头部追踪姿态
- 输入参数：无
- 返回值：无
- 接口调用:在 PicoVRManager 类中直接用

6.3.3 过程控制暂停接口

- private void OnApplicationPause(bool pause)
- 函数功能：过程控制暂停接口
- 输入参数：bool pause
- 返回值：无

6.3.4 Rendertexture 配置接口

- private void ConfigureEyeTexture(int eyeTextureIndex)
- 函数功能：左右眼 Rendertexture 配置
- 输入参数：Rendertexture id
- 返回值：无

6.3.5 获取 Rendertexture 尺寸接口

- public override Vector2 GetStereoScreenSize()
- 函数功能：获取 Rendertexture 尺寸

- 输入参数：无
- 返回值：Rendertexture 的长宽数据
- 返回值定义： Vector2 类型 Rendertexture 的长宽数据

6.3.6 更新姿态旋转与位置状态接口

- public override void UpdateState()
- 函数功能：更新姿态旋转与位置，获取最新的姿态信息用于画面渲染
- 输入参数：无
- 返回值：无

6.3.7 获取头戴距离传感器状态接口

- public int GetPsensorState()
- 函数功能：获得头戴距离传感器的状态
- 输入参数：无
- 返回值：

-1: 头戴未连接

0: 带上头戴返回的状态值

1: 远离时返回的状态值

2: 获取状态出错

3: 距离传感器被禁用，此为头戴固件版本太低导致，需要更新头戴固件版本到最新版本

6.3.8 获取手柄姿态接口

- pvrQuaternion GetHandTrackedPose(int index)
- 函数功能：获取手柄姿态
- 输入参数：index：手柄序号：0 为左手柄，1 为右手柄
- 返回值：返回手柄姿态四元数数据（w，x，y，z）
- 备注：该接口在 PicoVRWinPCDevice.cs 脚本中实现。

6.3.9 获取手柄位置接口

- void GetControllerPosition(int index, ref pvrVector3 pos, ref int uPosEdgeOut)
- 函数功能：获取手柄位置
- 输入参数：index：手柄序号：0 为左手柄，1 为右手柄
- 输出参数 :pvrVector3_t 类型的位置信息 pos（x, y, z）;int 类型的 uPosEdgeOut 为出界信息： 0 表示定位正确，1 表示遮挡或其他原因导致定位目标消失，2 表示目标出界，3 表示目标过近出界，4 表示目标过远出界。
- 返回值：无
- 备注：该接口在 PicoVRWinPCDevice.cs 脚本中实现。

6.3.10 获取手柄状态接口

- bool PVR_GetControllerState(int unWhichDevice, ref ControllerState state)
- 函数功能：获取手柄的的状态信息
- 输入参数：int unWhichDevice，0 为左手柄，1 为右手柄

- 输出参数：ControllerState 类型的状态信息

```
struct ControllerState

{
public int btnPressed;        //Bit 0:带按键 Touch, 1: Step Trigger,
                                  2:Home, 3:Right, 4:Left, 5:X, 6:Y, 7:reserve
public int touchPadX;          //0-255
public int touchPadY;          //0-255
public int triggerAnalog;    //0-255
public int stateOfCharge;     //15,50,75,90,other num is disable
public int isConnect;          //0:DisConnect 1:Connect
};
```

- 返回值：

  true：获取手柄状态信息成功

  false：获取手柄状态信息失败

- 备注：该接口在 PicoVRWinPCDevice.cs 脚本中实现。

6.3.11 触发手柄震动接口

- void ControllerShake(int device, int microSec)
- 函数功能：触发手柄震动
- 输入参数：

  device：代表触发哪一只手柄，0：左手柄，1：右手柄

microSec：手柄震动强度，范围为 0-255，总共 256 个震动强度等级

- 输出参数：无
- 返回值：

  true：调用成功

  false：调用失败

- 备注：该接口在 PicoVRHandTrack.cs 脚本中实现。

6.3.12 获取摄像头连接状态接口

- bool GetCameraConnectState()
- 函数功能：获取摄像头连接状态
- 输入参数：无
- 输出参数：无
- 返回值：摄像头连接状态：true 为已连接，false 为未连接。
- 备注：该接口在 PicoVRWinPCDevice.cs 脚本中实现。

6.4 Windows API 说明

6.4.1 SDK 初始化

- bool PVR_Init()
- 函数功能：初始化 SDK，创建相应资源。
- 输入参数：无
- 返回值：

  true: 初始化成功

false：初始化失败

6.4.2 SDK 初始化注销

- bool PVR_Shutdown()
- 函数功能：注销 SDK，释放相应资源
- 输入参数：无
- 返回值：

  true：注销成功

  false：注销失败

6.4.3 获取头戴姿态数据

- bool PVR_GetTrackedPose(pvrQuaternion_t& pose);
- 函数功能：获取头戴的姿态数据
- 输入参数：无
- 输出参数：pvrQuaternion_t 类型四元数信息（w, x, y, z）
- 返回值：

  true：获取头戴姿态数据成功

  false：获取头戴姿态数据失败

- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.4 获取头戴位置信息

- bool PVR_GetTrackedPosition(pvrVector3_t& position)
- 函数功能：获取头戴的位置信息

- 输入参数：无
- 输出参数：pvrVector3_t 类型的位置信息（x, y, z）
- 返回值：

  true：获取头戴位置信息成功

  false：获取头戴位置信息失败
- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.5 获取头戴分辨率信息

- bool PVR_GetScreenResolutionSize (int &hResolution, int &vResolution)
- 函数功能：获取头戴屏幕的分辨率
- 输入参数：无
- 输出参数：屏幕水平分辨率 hResolution，屏幕垂直分辨率 vResolution
- 返回值：

  true：获取头戴屏幕分辨率成功

  false：获取头戴屏幕分辨率失败
- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.6 获取 Render Texture 大小

- bool PVR_GetRenderTargetSize(int &width, int &height)
- 函数功能：获取 Render Texture 大小
- 输入参数：无
- 输出参数：Render Texture 宽度大小 width，Render Texture 高度大小 height

- 返回值：

  true：获取 Render Texture 大小成功

  false：获取 Render Texture 大小失败

- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.7 获取头戴连接状态

- bool PVR_IsHmdConnected()
- 函数功能：获取头戴连接状态
- 输入参数：无
- 输出参数：无
- 返回值：

  true：头戴已连接

  false：头戴未连接

- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.8 头部追踪姿态重置

- bool PVR_ResetSensor()
- 函数功能：重置头部追踪姿态
- 输入参数：无
- 输出参数：无
- 返回值：

  true：重置成功

false：重置失败

- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.9 获取头戴距离传感器状态接口

- bool PVR_GetPsensorStatus(int &status)
- 函数功能：获得头戴距离传感器的状态
- 输入参数：无
- 输出参数：status：

-1: 头戴未连接

0: 带上头戴返回的状态值

1: 远离时返回的状态值

2: 获取状态出错

3: 距离传感器被禁用，此为头戴固件版本太低导致，需要更新头戴固件版本到最新版本

- 返回值：

true：调用成功

false：调用失败

- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.10 获取手柄姿态数据

- bool PVR_GetControllerPose(int unWhichDevice, pvrQuaternion &q);
- 函数功能：获取手柄的的姿态数据

- 输入参数：int unWhichDevice，0 为左手柄，1 为右手柄
- 输出参数：pvrQuaternion_t 类型四元数信息（w, x, y, z）
- 返回值：

  true：获取手柄姿态数据成功

  false：获取手柄姿态数据失败
- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.11 获取手柄位置信息

- bool PVR_GetControllerPosition(int unWhichDevice, pvrVector3 &v, int &uPosEdgeOut);
- 函数功能：获取手柄的的位置信息
- 输入参数：int unWhichDevice，0 为左手柄，1 为右手柄。
- 输出参数：pvrVector3_t 类型的位置信息 v（x, y, z）；int 类型的 uPosEdgeOut 为出界信息： 0 表示定位正确，1 表示遮挡或其他原因导致定位目标消失，2 表示目标出界，3 表示目标过近出界，4 表示目标过远出界。
- 返回值：

  true：获取手柄位置信息成功

  false：获取手柄位置信息失败
- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.12 获取手柄状态信息

- bool PVR_GetControllerState(int unWhichDevice, ControllerState &state);

- 函数功能：获取手柄的的状态信息
- 输入参数：int unWhichDevice，0 为左手柄，1 为右手柄
- 输出参数：ControllerState 类型的状态信息

```
struct ControllerState

{
public int btnPressed;        //Bit 0:带按键 Touch, 1: Step Trigger,
                                  2:Home, 3:Right, 4:Left, 5:X, 6:Y, 7:reserve
public int touchPadX;          //0-255
public int touchPadY;          //0-255
public int triggerAnalog;    //0-255
public int stateOfCharge;     //15,50,75,90,other num is disable
public int isConnect;          //0:DisConnect 1:Connect
};
```

- 返回值：

  true：获取手柄状态信息成功

  false：获取手柄状态信息失败

- 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.13 触发手柄震动接口

- bool PVR_TriggerControllerShake(int unWhichDevice, int usDurationMicroSec)

- ✧ 函数功能：触发手柄震动
- ✧ 输入参数：

  unWhichDevice：代表触发哪一只手柄，0：左手柄，1：右手柄

  usDurationMicroSec：手柄震动强度，范围为 0-255，总共 256 个震动强度等级
- ✧ 输出参数：无
- ✧ 返回值：

  true：调用成功

  false：调用失败
- ✧ 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.14 获取摄像头连接状态

- ✧ bool PVR_GetCameraConnectStatus(bool &isConnected)
- ✧ 函数功能：获取摄像头连接状态
- ✧ 输入参数：无
- ✧ 输出参数：摄像头连接状态：true：摄像头已连接，false：摄像头未连接
- ✧ 返回值：

  true：调用成功

  false：调用失败
- ✧ 备注： 在调用该接口之前需要确保 PVR_Init()已经调用过并返回 true。

6.4.15 游戏支付验证

- int PVR_Verify(const char * b_gameId)
- 函数功能：游戏支付验证
- 输入参数 :b_gameId ,游戏 ID ,即开发者在 Pico 开发者平台上注册申请的 AppID
- 输出参数：无
- 返回值：

  0：验证通过

  1：未登录 PicoHome

  2：未购买游戏

  3：无法启动 PicoHome

  其他：调用失败

6.4.16 游戏道具内购

- int PVR_Purchase(const char* mch_id, const char* app_id, const char* app_key, const char* app_secret, const char* total_fee, const char* pay_code, const char* out_trade_no)
- 函数功能：调用游戏内购接口
- 输入参数：mch_id 开发者在 Pico 开发者平台上注册申请的商户号

  app_id 开发者在 Pico 开发者平台上注册申请的 app_id

  app_key 开发者在 Pico 开发者平台上注册申请的 app_key

  app_key 开发者在 Pico 开发者平台上注册申请的 app_secret

  total_fee 要购买的道具金额

pay_code 商品代码

out_trade_no 商户内部订单号，32 个字符以内，可包含字母，不同订单不能相同

- ✧ 输出参数：无
- ✧ 返回值：

0：成功

1：密码错误

2：登录失败

3：余额不足

4：创建订单失败

5：购买失败

7 FAQ

问: 启动游戏或 Demo 后，连接头戴，头戴中不显示游戏内容

答: 目前还不支持未连接头戴就启动游戏或者 Demo 的情况，也就是说，在启动游戏之前必须保证头戴已经连接成功。

问: Windows7 系统，用 Unity5.4.1, 5.4.4 或者 5.5.1 开发 PC 游戏，在 Editor 模式下运行会导致 Unity 退出，或者打包后运行游戏，游戏会崩溃，而用其他版本的 Unity 没有此问题。

答：该问题与操作系统缺少相应的更新有关，目前只发现在 Win7 操作系统下。安装链接中的更新即可解决：

https://www.microsoft.com/en-us/download/details.aspx?id=36805

问: Camera Culling Mask 如何设置

答: PicoVR 中 LeftEye 以及 RightEye 的 Camera 中可设置属性 Culling Mask:

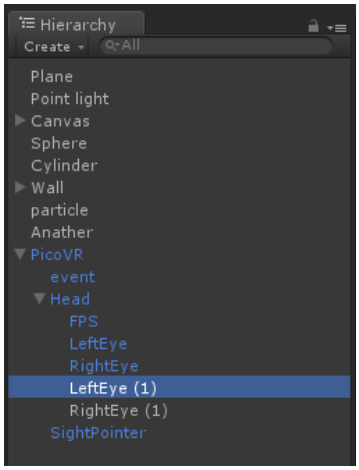
- 如果需要代码修改 toggle culling mask,可参见普通相机修改方式
- 对于左右眼需要看不同物体的时候, 通过 PicoVR 左右眼相机 Culling Mask 来控制显示 Layer, 左眼相机 Culling Mask 不勾选右眼物体的 Layer,右眼相机 Culling Mask 不勾选左眼物体的 Layer.

问: 同一场景是否支持多 Camera

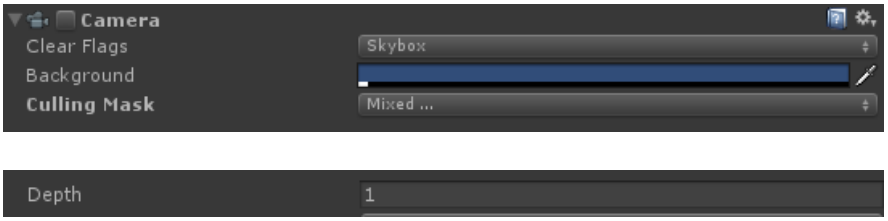
答: 支持，有两种方法。

方法 1：

- 展开 PicoVR，分别右键点击 LeftEye 和 RightEye，执行 Duplicate 命令，复制出另外两个副 Eye，如下图所示：

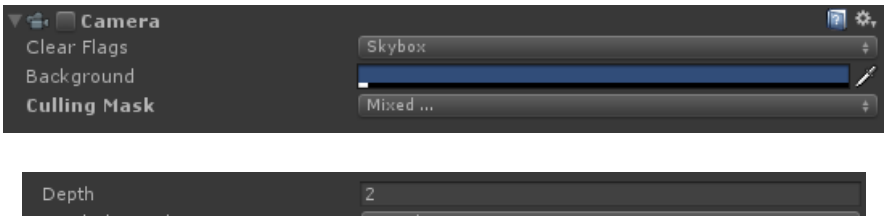


- 分别修改 LeftEye 和 RightEye 中的 Culling Mask，使其看不到特定的层，如下图所示：



- 分别修改 LeftEye (1) 和 RightEye (1) 中的 Culling Mask，使其只能看到特定的层，

并修改其 Depth，使其高于对应的主 Eye 的 Depth，如下图所示



方法 2：

- 分别在 LeftEye 和 RightEye 下创建空的 Gameobject，命名为 CameraGroup。把所有相机拖到其中一个 Eye 的 CameraGroup 下，然后复制一份到另一个 Eye 的 CameraGroup 下，如下图所示：

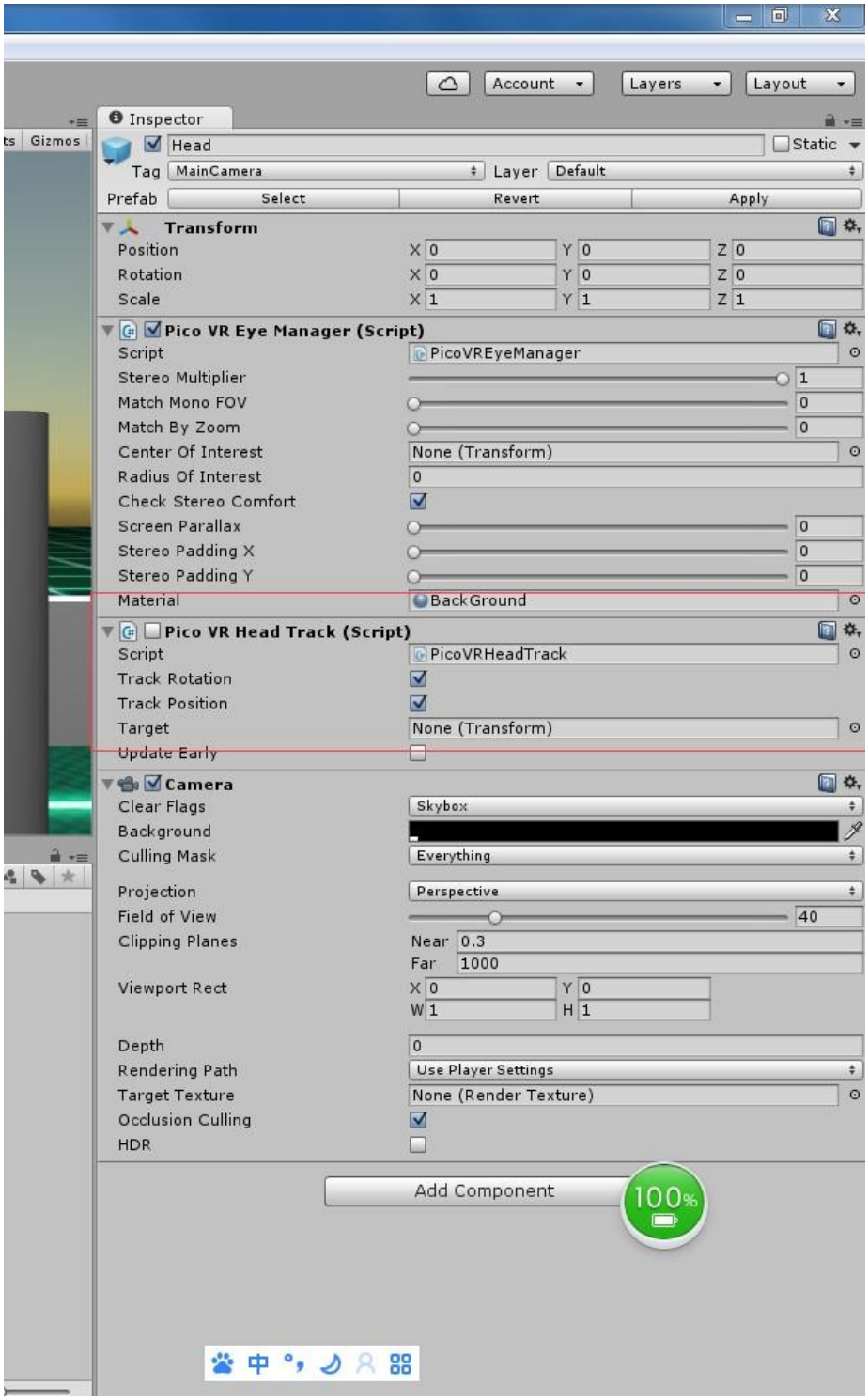
- 在这些 Camera 里，targetTexture 为空的 Camera 会绘制到所在 Eye 的 RenderTexture 上，targetTexture 不为空的 Camera 不作处理。
- Reset 每个 Camera 的 Transform，通过控制 Eye 的移动旋转来控制 Camera
- 调整各个 Camera 的 Depth ,最好保证 LeftEye 和 RightEye 的各个 Camera 的 Depth 都不一样。
- 把 CameraGroupPost.cs 分别拖到 LeftEye 和 RightEye 的相机里 Depth 最大的那个 Camera 上，左右 Eye 各一个。

  注意：这种方法需要引入单独的扩展包，以上两种，支持多 Camera 对渲染帧率会有一定的影响，请审慎使用。

问: 如何将 sensor 作用于场景其他物体

答: 按以下步骤进行设置

- 将场景中 PicoVR 中的 head 上 PicoVRHeadTrack 脚本取消勾选。



- 选中想要作用的物体，添加 PicoVRHeadTrack 脚本。