文档旨在帮助用户使用 IVR 接口开发能够在一体机上运行并显示的应用程序

IVR 接口使用说明

IDEALSEE 2016.11.1

目录

1. 文件清单

ivr.h	包含了需要用到的各种数据类型与函数接口声明
libivr.so	通过此 so 自动链接系统库，调用实际需要的函数

2. IVR 接口集成方法

在代码中通过 include ivr.h 来调用接口函数，用时在 app 中集成 libivr.so 即可。具体调用流程可参考后面的“渲染时序”

3. 接口中定义的数据类型

类型	说明
IVRQuatf	四元向量，其分量为 x, y, z, w
IVRVector3f	三元向量，其分量为 x, y, z,
IVRInstance	表示 VR 相关上下文的句柄
IVRHmdType	表示设备类型
IVREyeType	IVREye_Left = 0, IVREye_Right = 1
IVRHmdInfo	float eyeTextureFov[2]：　左右眼的 FOV
	float ipd　以米为单位
	IVRHmdType type：　设备类型
IVRStatus	函数的返回类型，IVRStatus_OK 为成功，其它类型表示有异常
	IVR_STATUS_OK, //return if success
	IVR_STATUS_ERROR,
	IVR_STATUS_INIT_FAIL,
	IVR_STATUS_UNKOWN

4. 接口说明

1) 初始化函数

IVRInstance ivrInit(jobject activity , JavaVM * VRJavaVM)

返回实例对象句柄，后续的函数调用都要使用这个句柄，其包含了线程中与 VR 相关的上下文，同时也初始化了 VR 中需要使用到的一些全局变量。返回 NULL 表示初始化失败。

在调用 IVRINIT 之前请确保 EGL 上下文已经初始化完毕。在初始化 EGL 之前调用此函数会导致错误退出

IVRStatus ivrDeInit(IVRInstance instance)

用于销毁句柄，若没有调用，则会引起内存泄漏

2) RESUME/PAUSE 函数

IVRStatus ivrResume(IVRInstance instance)

IVRStatus ivrPause(IVRInstance instance)

这两个函数需要在 activity 的 onResume/onPause 中调用，用于控制 VR 的进入与退出。

参数 **instance** 从 **IVR_Init** 的返回值得到

请确保调用 IVRRESUME/ IVRPAUSE 的线程与调用初始化函数 IVRINIT 的是同一个线程。

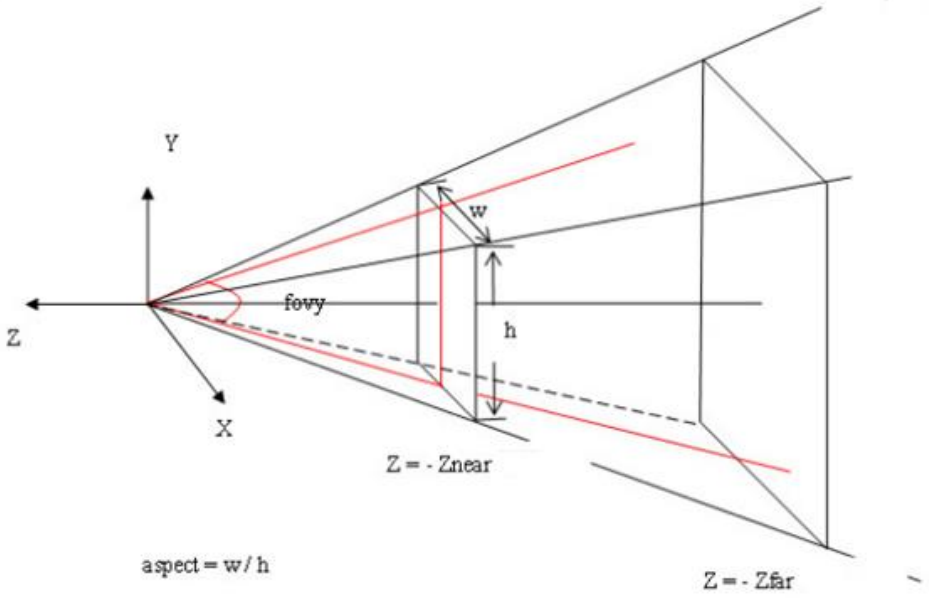
3) 获取设备信息函数

IVRStatus ivrGetDeviceInfo(IVRInstance instance , IVRHmdInfo& info)

获取 VR 设备的信息，提供包括投影矩阵等信息，信息以 **IVRHmdInfo** 结构表示。

简单介绍一下其中两个参数 eyeTextureFov 和 ipd 的使用方法。

使用参数 EYETEXTUREFOV 创建投影矩阵



eyeTextureFov 即为图中的 fovy，通过此参数可以创建投影矩阵。在此给出一个参考公式，请根据实际情况使用合适的投影矩阵公式：

$$(x' \quad y' \quad z' \quad w') = (x \quad y \quad z \quad 1)\begin{pmatrix} \frac{f}{a} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{znear + zfar}{znear - zfar} & -1 \\ 0 & 0 & \frac{2 \times znear \times zfar}{znear - zfar} & 0 \end{pmatrix}$$

其中，$f = 1/tg(fovy/2\deg), a = aspect$

使用参数 IPD 调整 MODEL_VEIW 矩阵

参数 ipd 即为瞳距。在一体机上运行的程序每一帧需要对左右眼各绘制一幅图像因此其立体渲染的典型步骤为：

1. 创建一个主相机：应用程序控制唯一一个主相机，所有的逻辑动画等等都只针对主相机，这样，我们在处理相机与其他物体交互的时候就变得简单统一，此外需要注意主相机不参与真正的渲染。主相机还有一个好处，就是可以在双立体视图和传统单一视图之间切换。
2. 用两个相机渲染：除了主相机，VR 程序需要外加两个相机，用来真正处理渲染。这两个相机需要与主相机的位置和朝向保持一致，只是有一点不同，就是它们分别要向左向右偏移一点点，用来模拟我们的瞳距。
3. 渲染至两个视口：VR 应用分别创建的左右眼的渲染相机，它们的视口宽度都是屏幕宽度的一半，高度都为屏幕高度。

可以看到在第二步中就需要使用到瞳距参数 ipd，参考使用方法为：

```
Const float eyeOffset = ( eye ? -0.5f : 0.5f ) * ipd ;

Translate_matrix = {    1.0f , 0.0f , 0.0f , eyeOffset ,

                        0.0f , 1.0f , 0.0f , 0.0f ,

                        0.0f , 0.0f , 1.0f , 0.0f ,

                        0.0f , 0.0f , 0.0f , 1.0f};

Multiply(Translate_matrix , current_View_Matrix);
```

4) 头部姿势函数

IVRStatus ivrGetHMDPose(IVRInstance instance , IVRQuatf &orientation , IVRVector3f &position , float &fov)

通过此 API 可以从 sensor 里得到当前头部的 Rotation、Position 以及 FOV 信息。

参数 **orientation** 为四元向量，存储 Rotation 信息

参数 **position** 为三元向量，存储位移信息，在没有实现手势跟踪的情况下暂时无用

通过 **orientation** 可以算出 view 矩阵

参数 **fov** 则可以用来计算投影矩阵

IVRStatus ivrGetHMDPoseByEye(IVRInstance instance , IVREyeType eye , IVRQuatf &orientation , float &fov)

根据左右眼得到矫正后的的视图矩阵，建议在计算视图矩阵时使用此函数分别得到左右眼的 **orientation** 以及 **fov** 并以此计算视图矩阵与投影矩阵。

*只需要调用**ivrGetHMDPose** 和**ivrGetHMDPoseByEye** 其中一个函数获取头部姿态信息即可。*

IVRStatus ivrResetPose(IVRInstance instance)

用于重置头部姿势，主要是为了配合一体机的重置功能，通常情况下可以不使用

5) 纹理设置函数

IVRStatus ivrSetTexture(IVRInstance instance , IVREyeType eye, int textureId)

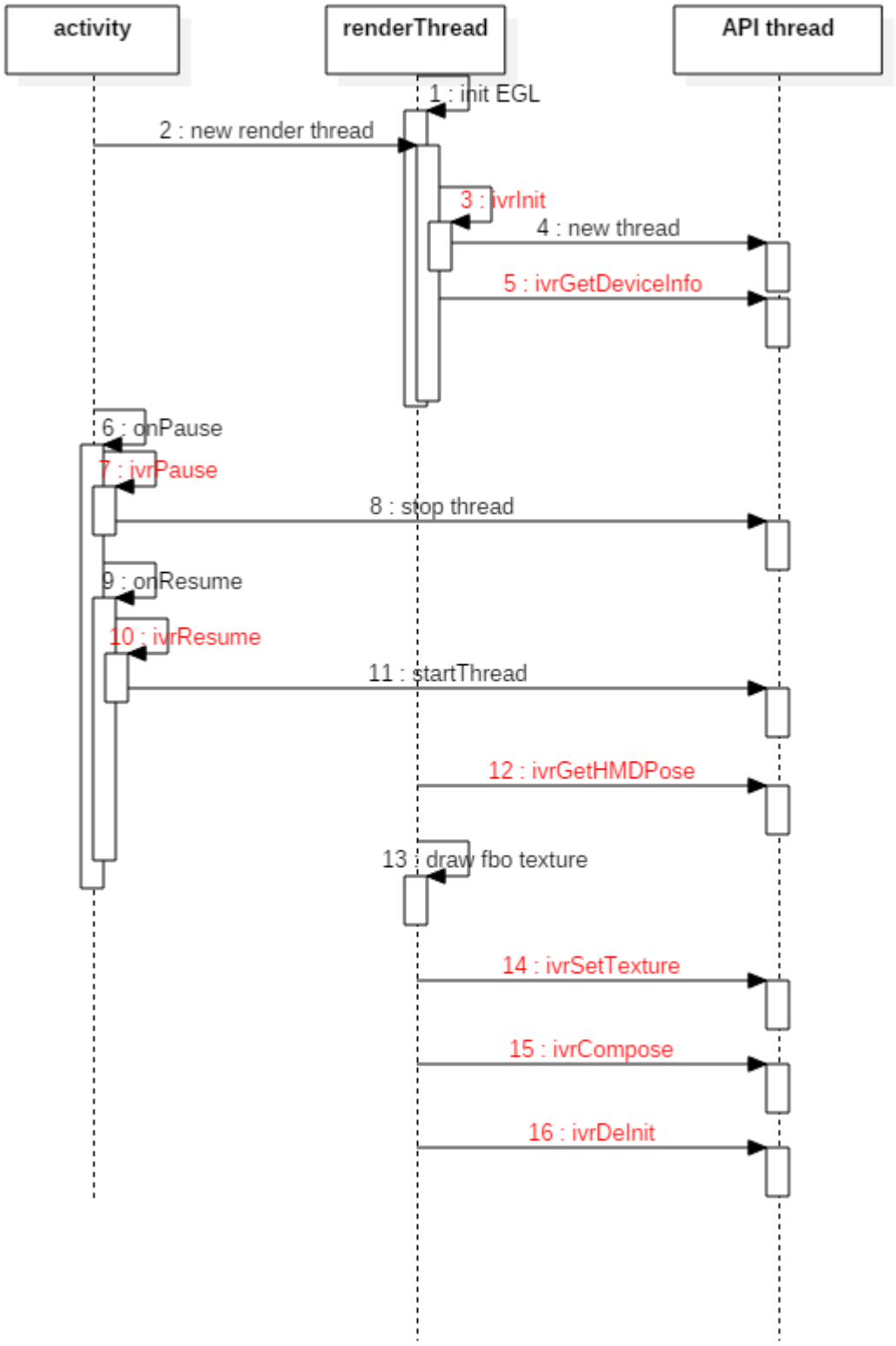
用于设置左右眼纹理的函数，参数 **eye** 为 IVREyeType 中的其中一个。

6) 渲染函数

IVRStatus ivrSetTexture(IVRInstance instance ,IVREyeType eye, int textureId)

调用设备渲染接口将传入的纹理绘制到屏幕。渲染之前应当确保调用过 **ivrinit, ivrSetTexture，ivrGetHMDPose / ivrGetHMDPoseByEye**

5. 渲染时序


activity
renderThread
API thread
1 : init EGL
2 : new render thread
3 : ivrInit
4 : new thread
5 : ivrGetDeviceInfo
6 : onPause
7 : ivrPause
8 : stop thread
9 : onResume
10 : ivrResume
11 : startThread
12 : ivrGetHMDPose
13 : draw fbo texture
14 : ivrSetTexture
15 : ivrCompose
16 : ivrDeInit