OpenPoseForUnrealPlugin制作与应用过程

安装前置条件

版本要求

OpenPose: at least v1.5.0

Unreal: at least v4.26.2

cuda: v10.0

cudnn: v7.5.0

只要能将OpenPose安装并成功运行，理论上即可正确使用OpenPoseForUnreal插件

安装帖子很多，我参照了这个[Openpose1.5.0+VS2017+CUDA10+cuDNN7.5+WIN10安装部署教程（C++和Python API）_小EZ的博客-CSDN博客]

OpenPose插件

Openpose DLL导出与调用原理

OpenPose源码在使用cmake构建时有选项`BUILD_UNITY_SUPPORT`，勾选之后使用cmake generate之后得到visual studio文件。使用VS打开并编译之后即可得到带有所需的`openpose.dll`

`BUILD_UNITY_SUPPORT`对应的代码中有两个关键的回调函数：

```
// Output callback register in Unity
typedef void(__stdcall * OutputCallback) (void * ptrs, int ptrSize, int * sizes,
int sizeSize, uchar outputType); // unityBinding.cpp
typedef void(__stdcall * DebugCallback) (const char* const str, int
type);//errorAndLog.cpp
```

相对应的，在unreal插件中，也需要按照相同的方式定义这两个回调函数：

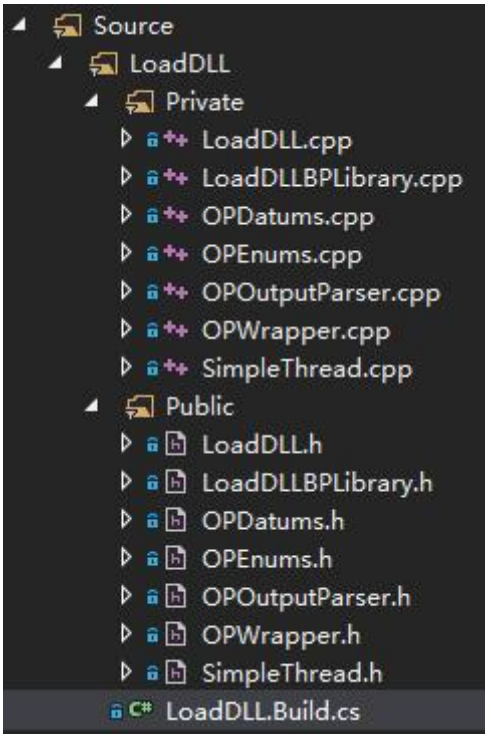
```
// Define callback function
typedef void(__stdcall * OutputCallback) (void * ptrs, int ptrSize, int * sizes,
int sizeSize, uint8 outputType);
typedef void(__stdcall * DebugCallback) (const char* const str, int type);
```

这样就可以在unreal程序运行的同时开启openpose程序的运行，并从openpose程序那得到当前摄像头内容识别之后得到的骨架信息，并传递到unreal程序当中。

Unreal插件结构

OpenPoseForUnreal插件在VS工程文件当中的名称如下：`LoadDLL`。磁盘中位置为工程文件根目录下`Plugins-LoadDLL`当中。

`Plugins\LoadDLL\Source\LoadDLL`中包含着关键代码。

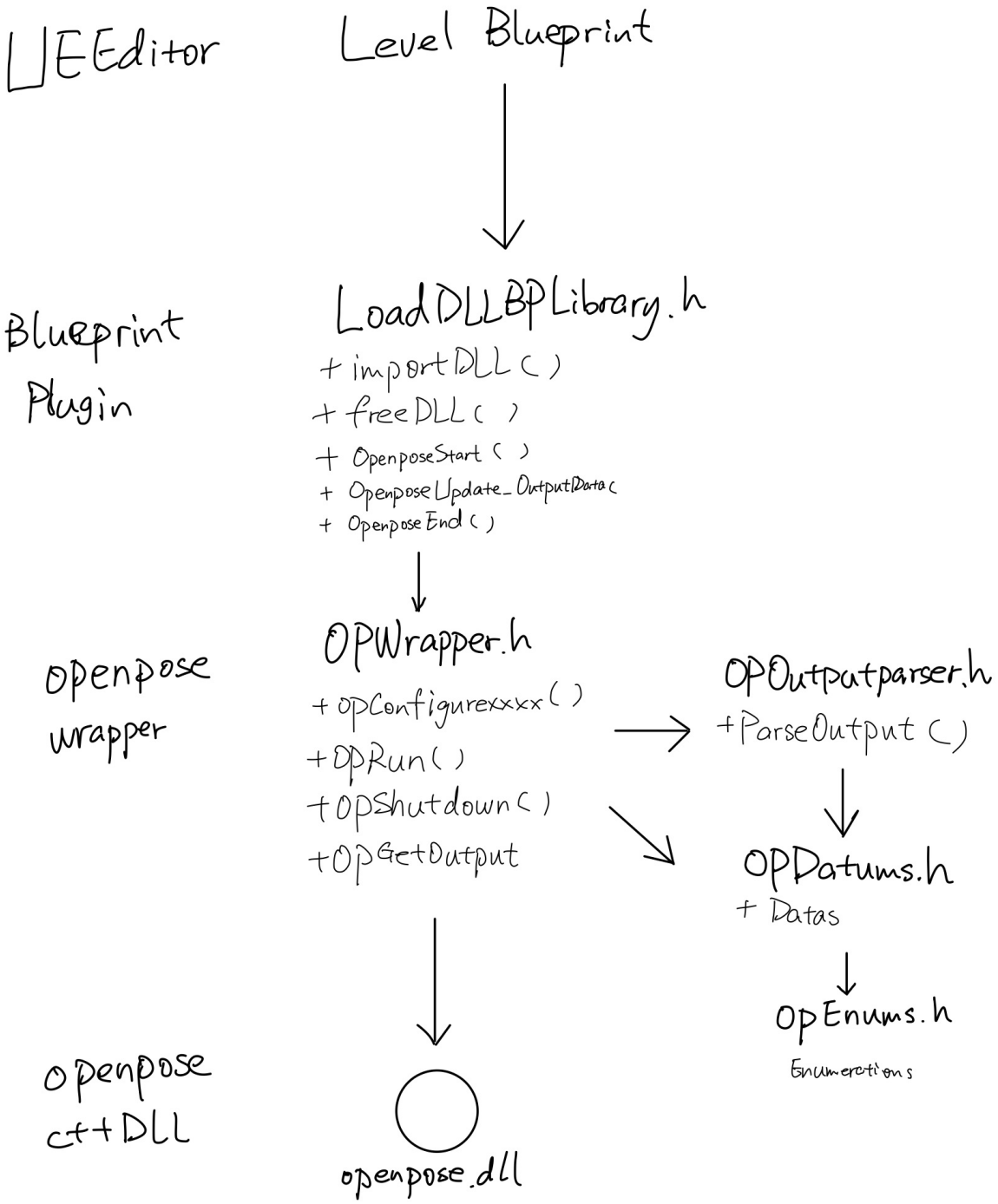


`LoadDLL.Build.cs`中包含了插件需要的第三方库与各项依赖，这保证了插件的正常运行。

`LoadDLL.h LoadDLL.cpp`描述了插件在作为一个模块启动时(对应函数`StartupModule`)或是结束时(对应函数`ShutdownModule`)需要执行的步骤。此处函数`StartupModule`执行的是一些openpose运行所需的第三方库的加载，若是加载不成功，则会在输出日志中报错：`"%s not loaded"`

`LoadDLLBPLibrary.cpp`则有两部分功能：加载dll中的函数并封装；向编辑器提供可以在蓝图中调用的函数。其中存在类型为`OPDatum`的静态成员`m_Datum`，从`OPWrapper.h`数据缓冲中`dataBuffer`提取骨架数据，经过处理后向unreal程序输出。

具体代码组织结构如下所示：



`OPWrapper.h` 调用 `LoadDLLBPLibrary.h` 中封装的openpose函数，并提供函数供 `LoadDLLBPLibrary.h` 中提供给外界的蓝图函数调用。`OPWrapper.h` 中定义了唯一的 `currentData`（类型为 `OPDatums`）与识别到的openpose数据缓冲 `dataBuffer`（类型为 `queue<OPDatum>`）。`OPWrapper` 类中的函数与参数均为static，便于 `LoadDLLBPLibrary.h` 直接调用。

`OPOutputparser.h` 用来解析openpose程序向unreal程序发送的信息，处理为各个关键点的数据之后更新 `currentData`，使得在被压入数据缓冲 `dataBuffer` 时的 `currentData` 已经经过了数据的解析

`OPDatums.h` 中描述了存储结构 `OPDatum` 以及一些程序中需要使用到的自定义数据结构 `MultiArray` 与 `Pair` 等。

`OPEnums.h` 中描述了需要使用到的枚举类型

添加Log的方式

添加通用Log类型

在需要log的地方如下编写即可：

```
UE_LOG(LogTemp, Error, TEXT("Trying to shutdown, while OpenPose is not
running"));
UE_LOG(LogTemp, Warning, TEXT("Trying to shutdown, while OpenPose is not
running"));
```

添加单独Log类型

参考`OPOutputParser.h`与`OPOutputParser.cpp`中的方式。

在`OPOutputParser.h`中做出如下声明：

```
//General Log
DECLARE_LOG_CATEGORY_EXTERN(OPOutputParserLOG, Log, All);
```

在`OPOutputParser.cpp`中做出如下声明：

```
//General Log
DEFINE_LOG_CATEGORY(OPOutputParserLOG);
```

之后在需要进行Log的时候，带上`OPOutputParserLOG`即可：

```
UE_LOG(OPOutputParserLOG, Error, TEXT("Function ParseImage Image size length
invalid: %d"), sizeArray.Num());
```

关于插件中DLL存放位置

`LoadDLL.Build.cs`中描述了插件中第三方DLL的保存的相对路径

LoadDLL.cpp 中则描述了实际插件在加载过程需要导入的dll的相对路径，具体代码如下所示:

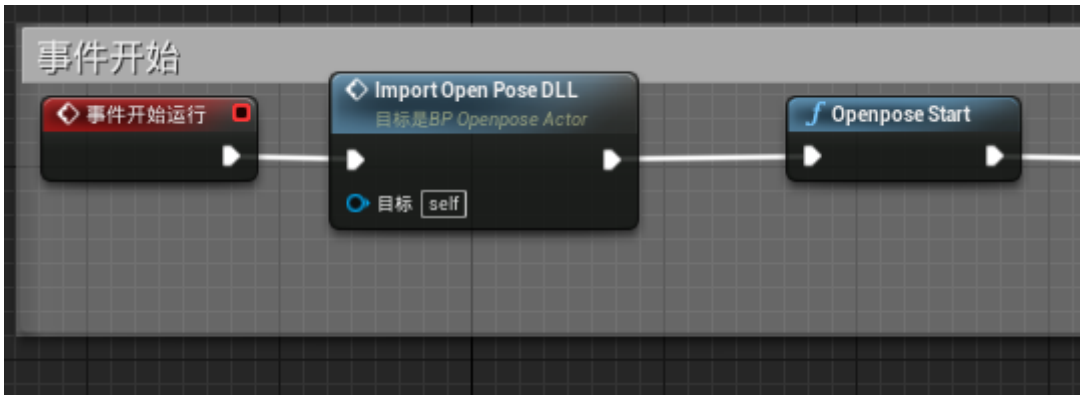
```
#if PLATFORM_WINDOWS
#if WITH_EDITOR
    RootOpenCVPath = FPaths::ProjectPluginsDir() /
FString::Printf(TEXT("LoadDll/ThirdParty/OpenPose/lib/"));
#else
    RootOpenCVPath = FPaths::ProjectDir() /
FString::Printf(TEXT("Binaries/Win64/ThirdParty/"));
    RootOpenCVPath = FPaths::ConvertRelativePathToFull(RootOpenCVPath);
    //RootOpenCVPath = FString::Printf(TEXT("./ThirdParty/lib/"));
#endif

#endif
```

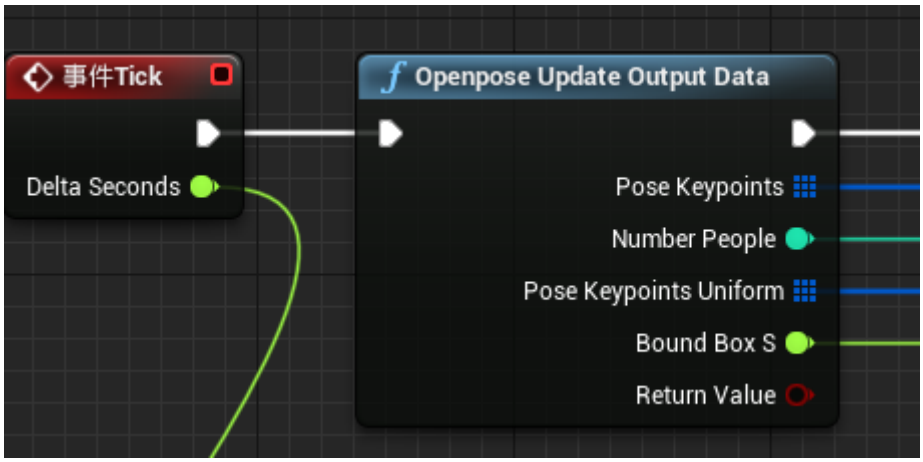
WITH_EDITOR 的引入是为了区分打包过程与非打包过程，此处虽然告知了打包之后去哪里寻找对应的dll，但是还是无法work，目前的解决方案为将所需dll拷贝到可执行文件目录下。

插件使用方式

新建蓝图类OpenPoseActor，设置为Actor类使得能够将其拖入场景中。

在OpenPoseActor的事件图表中添加如下图所示的节点





节点`ImportOpenPoseDLL`将`openpose.dll`导入，节点`OpenposeStart`使用默认参数启动Openpose程序【注意，参数目前是在VS当中写死的，因此若需要更改参数则需要更改VS工程文件中的代码并重新编译】

事件Tick每一帧调用一次，节点`OpenposeUpdateOutputData`在收到Openpose骨架数据信息之后输出对应的数据。

`PoseKeypoints`：归一化处理之后得到的骨架关键点数据，应用于之后的动作识别

`NumberPeople`：识别到的人体数据

`PoseKeypointsUniform`：归一化之前的骨架关键点数据。与摄像头分辨率有关，例如摄像头分辨率为1280x720时，骨架关键点数据就在【0~1280， 0~720】的范围中。