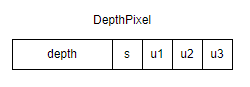
# 获取深度缓存

## 深度像素格式



DepthPixel

## Way1：直接使用ENQUEUE\_RENDER\_COMMAND命令获取(效率较低)

全屏可以解决像素不对应的问题。

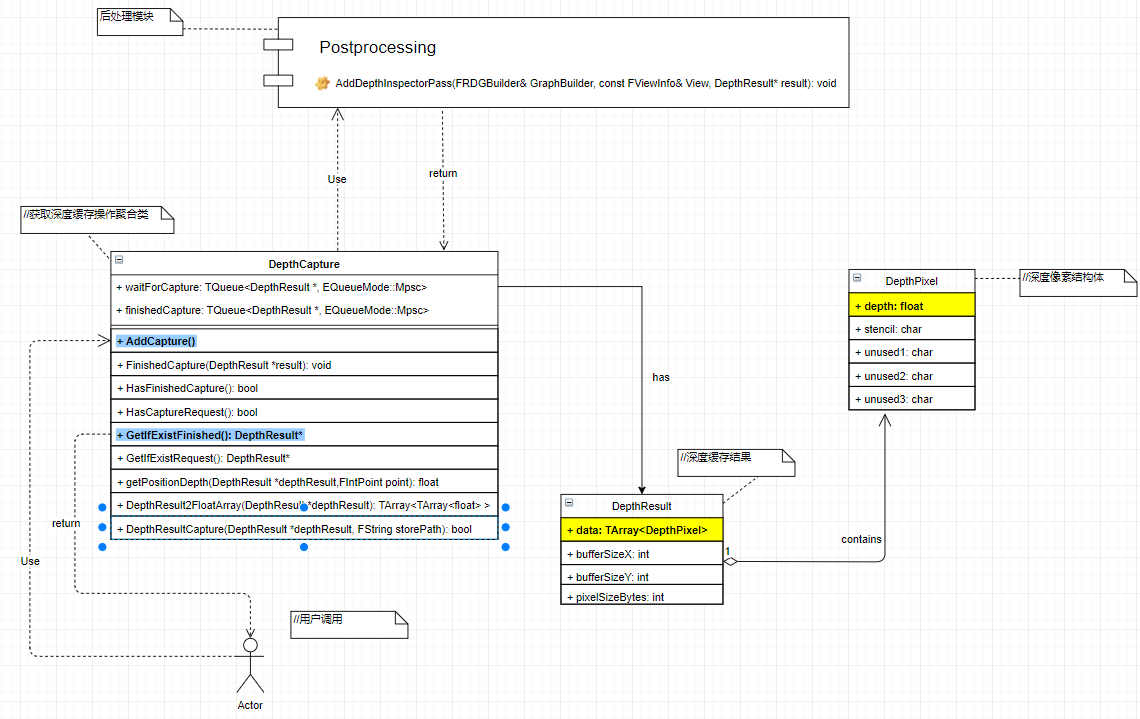
在任意tick函数或者其他函数添加以下的命令：

struct DepthPixel //定义深度像素结构体  
 {  
 float depth;  
 char stencil;  
 char unused1;  
 char unused2;  
 char unused3;  
 };  
 float\* cpuDataPtr; // Texture深度值数组首地址  
 TArray<DepthPixel> mydata; //最终获取色深度值数据  
 FIntPoint buffsize; //深度长宽大小X和Y  
  
 ENQUEUE\_RENDER\_COMMAND(ReadSurfaceFloatCommand)(//将读取深度数据的命令推给渲染线程进行执行  
 [&cpuDataPtr, &mydata, &buffsize](FRHICommandListImmediate& RHICmdList) //&cpuDataPtr, &mydata, &buffsize为传入的外部参数  
 {  
 FSceneRenderTargets::Get(RHICmdList).AdjustGBufferRefCount(RHICmdList, 1);  
 FTexture2DRHIRef uTex2DRes = FSceneRenderTargets::Get(RHICmdList).GetSceneDepthSurface();   
 buffsize = uTex2DRes->GetSizeXY();  
 uint32 sx = buffsize.X;  
 uint32 sy = buffsize.Y;  
 mydata.AddUninitialized(sx \* sy);  
 uint32 Lolstrid = 0;  
 cpuDataPtr = (float\*)RHILockTexture2D(uTex2DRes,0,RLM\_ReadOnly,Lolstrid,true); // 加锁 获取可读depth Texture深度值数组首地址  
 memcpy(mydata.GetData(), cpuDataPtr, sx \* sy \* sizeof(DepthPixel)); //复制深度数据  
 RHIUnlockTexture2D(uTex2DRes, 0, true); //解锁  
 FSceneRenderTargets::Get(RHICmdList).AdjustGBufferRefCount(RHICmdList, -1);   
 });  
 FlushRenderingCommands(); //等待渲染线程执行  
 mydata; //最终获取深度数据

最终返回的mydata数据就是最终的深度值数组，其中每个深度值的结构是DepthPixel，其中一个成员为depth，另外四个不不使用。其中使用上面的几个命令需要添加“RHI.h”和"RHIResources.h"头文件。

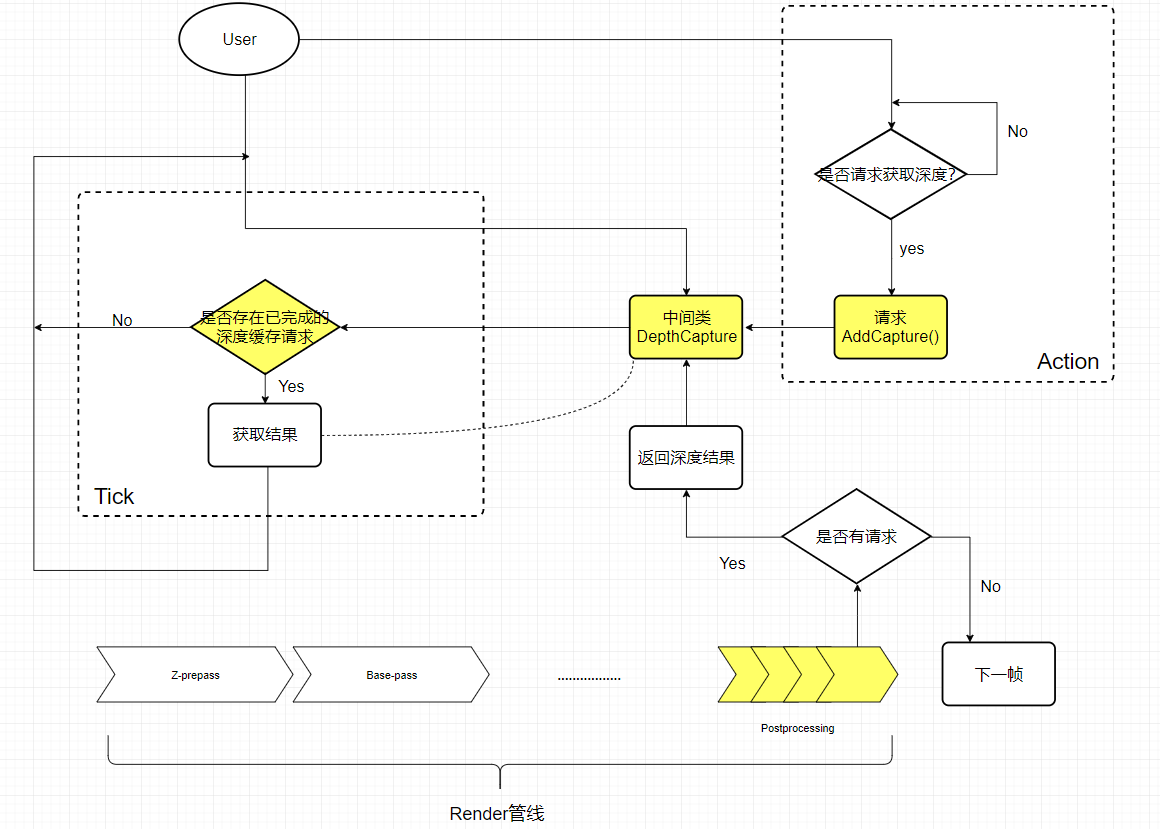
## Way2：写个请求类读取(集成)

UML图：



depth\_UML

流程图：



流程图

### 1. 首先在项目的build.cs文件添加：

添加引擎源码地址

// 添加引擎源码地址  
 string EnginePath = "C:/Program Files (x86)/UE4+VS2017/UnrealEngine/";  
 PrivateIncludePaths.AddRange(  
 new string[] {  
 EnginePath + "Source/Runtime/Renderer/Private",  
 EnginePath + "Source/Runtime/Renderer/Private/CompositionLighting",  
 EnginePath + "Source/Runtime/Renderer/Private/PostProcess"  
 }  
 );

添加引依赖项



依赖项

### 2. 类实现

将下面类代码复制到PostProcessing.h文件任意位置：

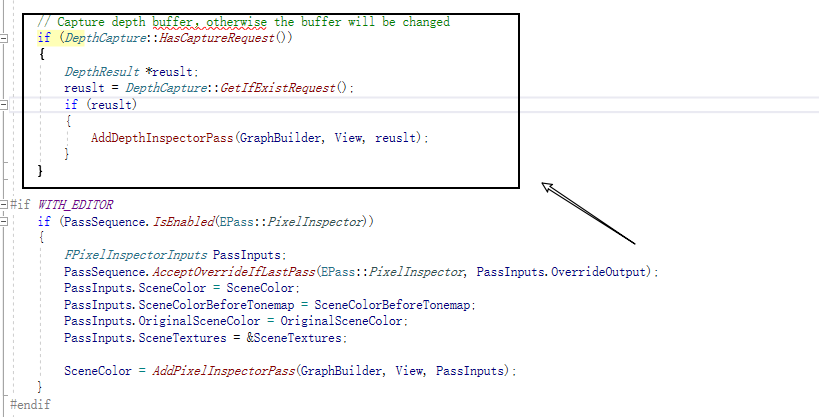
*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*Get Depth Class\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  
  
*/\* 存储一个像素的缓存*  
 *depth 深度缓存*  
 *stencil （抠图缓存）\*/*  
*struct DepthPixel*  
*{*  
 *float depth;*  
 *char stencil;*  
 *char unused1;*  
 *char unused2;*  
 *char unused3;*  
*};*  
  
*/\* 存储整个视窗的缓存*  
 *data 像素缓存数组*  
 *bufferSizeX 缓存大小X*  
 *bufferSizeY 缓存大小Y*  
 *pixelSizeBytes 像素缓存字节数\*/*  
**struct** DepthResult  
{  
 TArray<DepthPixel> data;  
 int bufferSizeX;  
 int bufferSizeY;  
 int pixelSizeBytes;  
};  
  
*/\* 获取深度缓存的类 \*/*  
**class** RENDERER\_API DepthCapture  
{  
**public**:  
 */\* 静态成员，当用户发出一个获取深度缓存的请求后，waitForCapture长度加1，新增DepthResult内容为空*  
 *当系统完成一个深度缓存的请求后，waitForCapture长度减一 \*/*  
 static TQueue<DepthResult \*, EQueueMode::Mpsc> waitForCapture;  
 */\* 静态成员，当系统完成一个深度缓存的请求后，finishedCapture长度加1，*  
 *新增DepthResult含有深度缓存信息 \*/*  
 static TQueue<DepthResult \*, EQueueMode::Mpsc> finishedCapture;  
  
**public**:  
 */\*用户发出一个获取深度缓存的请求时调用\*/*  
 static void AddCapture()  
 {  
 waitForCapture.Enqueue(**new** DepthResult());  
 }  
 */\*系统完成一个深度缓存请求后调用\*/*  
 static void FinishedCapture(DepthResult \*result)  
 {  
 finishedCapture.Enqueue(result);  
 }  
 */\*返回是否存在已经完成的请求\*/*  
 static bool HasFinishedCapture()  
 {  
 **return** !finishedCapture.IsEmpty();  
 }  
 */\*如果存在已完成的请求，返回一个深度结果\*/*  
 static DepthResult\* GetIfExistFinished()  
 {  
 DepthResult\* result = NULL;  
 **if** (!finishedCapture.IsEmpty())  
 {  
 finishedCapture.Dequeue(result);  
 }  
 **return** result;  
 }  
 */\*返回是否存在等待系统执行的请求\*/*  
 static bool HasCaptureRequest()  
 {  
 **return** !waitForCapture.IsEmpty();  
 }  
 */\*如果存在待完成的请求，返回一个深度结果（为空）\*/*  
 *static DepthResult\* GetIfExistRequest()*  
 *{*  
 *DepthResult\* result = NULL;*  
 *if (!waitForCapture.IsEmpty())*  
 *{*  
 *waitForCapture.Dequeue(result);*  
 *}*  
 *return result;*  
 *}*

/////////////////////辅助方法////////////////////////  
   
 /\*\*\*\*\*\*\*\*\*\* 获取某一点的深度数据，返回float  
 depthResult 深度数据结果  
 point 屏幕某个点的位置\*/  
 static float getPositionDepth(DepthResult \*depthResult,FIntPoint point) {  
 return depthResult->data[point.Y \* depthResult->bufferSizeX + point.X].depth;  
 }  
  
  
 /\*\*\*\*\*\*\*\*\*\* 深度数据转化为二维浮点数组，返回float的二维数组  
 depthResult 深度数据结果\*/  
 static TArray<TArray<float> > DepthResult2FloatArray(DepthResult \*depthResult) {  
 TArray<TArray<float> > result;  
 for (size\_t i = 0; i < depthResult->bufferSizeY; ++i)  
 {  
 TArray<float> line;  
 for (size\_t j = 0; j < depthResult->bufferSizeX; ++j)  
 {  
 line.Add(depthResult->data[i \* depthResult->bufferSizeX + j].depth);  
 }  
 result.Add(line);  
 }  
 return result;  
 }  
   
 /\*\*\*\*\*\*\*\*\*\* 将深度结果转化为bmp灰度图，返回是否存储成功的bool值  
 depthResult 深度数据结果  
 storePath 存储路径，名字后缀必须为.bmg\*/  
 static bool DepthResultCapture(DepthResult \*depthResult, FString storePath) {  
 bool imageSavedOk = false;  
 TArray<FColor> RawPixels;  
 RawPixels.AddUninitialized(depthResult->bufferSizeX \* depthResult->bufferSizeY);  
  
 for (uint64 i = 0; i < depthResult->bufferSizeX \* depthResult->bufferSizeY; ++i)  
 {  
 // Switch Blue changes.  
 uint8 v = depthResult->data[i].depth \* 10000000;  
 const uint8 PR = v > 255 ? 255 : v;  
 RawPixels[i].B = PR;  
 }  
 // 保存为灰度图  
 imageSavedOk = FFileHelper::CreateBitmap(\*storePath, depthResult->bufferSizeX, depthResult->bufferSizeY, RawPixels.GetData());  
  
 return imageSavedOk;  
 }  
};  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

将下面类中静态成员初始化和添加执行获取代码代码复制到PostProcessing.cpp文件任意位置：

/\*类静态成员的定义\*/  
TQueue<DepthResult \*, EQueueMode::Mpsc> DepthCapture::waitForCapture;  
TQueue< DepthResult \*, EQueueMode::Mpsc> DepthCapture::finishedCapture;  
  
/\*获取深度缓存\*/  
void AddDepthInspectorPass(FRDGBuilder& GraphBuilder, const FViewInfo& View, DepthResult\* result)  
{  
  
 RDG\_EVENT\_SCOPE(GraphBuilder, "DepthInspector");  
 {  
 // 获取渲染对象  
 FSceneRenderTargets& renderTargets = FSceneRenderTargets::Get(GRHICommandList.GetImmediateCommandList());  
  
 // 定义拷贝参数  
 uint32 striped = 0;  
 FIntPoint size = renderTargets.GetBufferSizeXY();  
 result->bufferSizeX = size.X;  
 result->bufferSizeY = size.Y;  
 result->data.AddUninitialized(size.X \* size.Y);  
 // 获取视窗某一帧的深度缓存对象  
 FRHITexture2D\* depthTexture = (FRHITexture2D \*)renderTargets.SceneDepthZ->GetRenderTargetItem().TargetableTexture.GetReference();  
  
 // 执行拷贝深度缓存操作，将GPU显存中的缓存信息拷贝到CPU内存中，返回指向这块CPU内存的首地址  
 void\* buffer = RHILockTexture2D(depthTexture, 0, EResourceLockMode::RLM\_ReadOnly, striped, true);  
 // 将缓存结果拷贝到result，用于输出  
 memcpy(result->data.GetData(), buffer, size.X \* size.Y \* 8);  
  
 // 必须执行解锁语句，否则被锁住的GPU缓存信息将不能释放  
 RHIUnlockTexture2D(depthTexture, 0, true);  
  
 // 拷贝结果入队  
 DepthCapture::FinishedCapture(result);  
 }  
}  
////////////////////////////////////////

PostProcessing.cpp中该位置添加以下代码：



添加代码

代码如下：

// Capture depth buffer，otherwise the buffer will be changed  
 if (DepthCapture::HasCaptureRequest())  
 {  
 DepthResult \*reuslt;  
 reuslt = DepthCapture::GetIfExistRequest();  
 if (reuslt)  
 {  
 AddDepthInspectorPass(GraphBuilder, View, reuslt);  
 }  
 }

### 3. 调用

绑定一个事件(Action)发出获取深度缓存数据的请求，事件的函数如下，发出请求：

void captureDepth() {  
 DepthCapture::AddCapture(); // 发出获取深度缓存的请求

}

在Tick函数中检查是否完成深度缓存的获取，完成之后可以获取；使用以下的代码可以获取深度值，获取的结果为result：

void Tick(float DeltaTime)  
{  
 // 如果存在已完成的深度缓存请求  
 if (DepthCapture::HasFinishedCapture())  
 {  
 DepthResult \*result;  
 // 获取已完成的深度缓存结果  
 result = DepthCapture::GetIfExistFinished();  
 if (result)  
 {  
 int n = result->data.Num();  
 //this is test  
 GEngine->AddOnScreenDebugMessage(-1, -1, FColor::Blue, FString::Printf(TEXT("Get Depth Size: %d "), n));  
 /\* 相关辅助函数  
 TArray<TArray<float> > floatResult = DepthCapture::DepthResult2FloatArray(result); // 深度结果转化为二维浮点数组  
 bool saveImageOk = DepthCapture::DepthResultCapture(result, "D://depth.bmp"); // 保存深度灰度图  
 float positionDepth = DepthCapture::GetPositionDepth(result, FIntPoint(500,500)); // 获取某一点的深度值  
 \*/  
 }  
 }  
}