尝试通过拖动鼠标来进行自定义墙体的绘制

MeshComponent

UMeshComponent继承自UPrimitiveComponent ，这也就和之前渲染代理部分联系上了

计算组件的边界。默认行为是一个大小为零的边界框/球体

virtual FBoxSphereBounds CalcBounds(const FTransform& LocalToWorld) const;

要重写自己的组件应该就要重新定义这个吧

Void SendRenderDynamicData\_Concurrent() : UActorComponent

调用此函数将此组件的动态数据发送到呈现线程

Void [CreateRenderState\_Concurrent()](https://docs.unrealengine.com/en-US/API/Runtime/Engine/Components/UActorComponent/CreateRenderState_Concurrent/index.html)

用于为该组件创建任何呈现线程信息

/\*\* 获取应用于网格的材料的场景相关信息。仅在游戏线程中有效。 \*/

FMaterialRelevance GetMaterialRelevance(ERHIFeatureLevel::Type InFeatureLevel) const;

/\*\* Main tick function for the Component 在ActorComponent中 \*/

UPROPERTY(EditDefaultsOnly, Category="ComponentTick")

struct FActorComponentTickFunction PrimaryComponentTick;

/\*\* 指示此组件的动态数据需要在框架的末尾发送。 ActorComponent中 \*/

void MarkRenderDynamicDataDirty();

/\*\*

\* Used to create any rendering thread information for this component

\* @warning This is called concurrently on multiple threads (but never the same component concurrently)

在ActorComponent中，可以为该组件创建一个类似于，渲染线程调用的方法的函数 并发

\*/

virtual void CreateRenderState\_Concurrent();

FVertexBuffer 、 IndexBuffer 、 LocalVertexFactory

BUF\_Static 缓冲区将被写入一次。

BUF\_Dynamic 缓冲偶尔会被写入，GPU只读，CPU只读。数据生存期将持续到下一次更新，否则缓冲区将被销毁

FDynamicMeshVertex 用于动态网格的顶点类型

PositionComponent等，不同的分别对应FDynamicMeshVertex中的Position啊TextureCoordinate等

类只存储位，初始化为0，并有一个|=操作符

//提供一组复合材料(组件/网格)

FMaterialRelevance

渲染代理

/\*\* Defines the domain of a material. \*/

UENUM()

enum EMaterialDomain

{

/\*\* 材料的属性描述了一个3d表面。 \*/

MD\_Surface UMETA(DisplayName = "Surface"),

# 控制相关

控制镜头方式：

Apawn取身上的Controller，

APlayerController\* const PC = CastChecked<APlayerController>(Controller);

之后计算摄像机偏移等，进行镜头移动处理，例如

void APlayerController::UpdateRotation( float DeltaTime )

{

// Calculate Delta to be applied on ViewRotation

FRotator DeltaRot(RotationInput);

FRotator ViewRotation = GetControlRotation();

if (PlayerCameraManager)

{

PlayerCameraManager->ProcessViewRotation(DeltaTime, ViewRotation, DeltaRot);

}

SHADER、材质相关

材质参数集

反射，获取类等相关

// Special canonical package for FindObject, ParseObject.

//用于FindObject, ParseObject的特殊规范包。

#define ANY\_PACKAGE ((UPackage\*)-1)

FString UFunctionNode::GetEnumValString(FFunctionType EnumToGet)

{

//UEnum\* EnumPtr = Cast<UEnum>(EnumToGet);

//FindObject 创建一个该枚举类型的对象，如果找到该枚举类型的对象，则不创建

UEnum\* EnumPtr = FindObject<UEnum>(ANY\_PACKAGE, TEXT("FFunctionType"), true);

FString CurShooterDataStr(EnumPtr->GetNameByValue((int)EnumToGet).ToString());//转化相应枚举字符串

FString ClassName;

CurShooterDataStr.Split("::",nullptr, &ClassName);

return ClassName;

}

使用该方法可以获取Emun的值得STRING

核心函数，获取并且执行对应的Static函数

FString FunctionName = GetEnumValString(M\_FunctionType);

UClass\* uclass = LibraryObj.Get();

UFunction\* func = GetFunctionInLibrary(\*FunctionName);//LibraryObj.Get()->FindFunctionByName(\*FunctionName);

if (func)

{

struct MyClass\_Func\_Parms //定义一个结构用来包装参数和返回值，就像在gen.cpp里那样

{

//float param1;

UObject\* param1;

UObject\* ReturnValue;

};

MyClass\_Func\_Parms params;

//params.param1 = 1;

params.param1 = this;

uclass->ProcessEvent(func, &params);

SaveData = params.ReturnValue;

}

LibraryObj是一个TSubClassOf 也可以直接用某个Obj 这个是调用静态方法

UFunction\* UFunctionNode::GetFunctionInLibrary(FName FuncitonName)

{

UClass\* uclass = LibraryObj.Get();

UFunction\* func = LibraryObj.Get()->FindFunctionByName(FuncitonName);

return func;

}