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;

}

# 家具文件结构等说明记录

#### **格子地面是GroundObj**

其中的

UPROPERTY()

class UGroundGridMgrComponent\* GridMgr;

是记录格子数据相关的组件，里面包含了

//格子的二维数组NestArray

UPROPERTY()

TArray< FNestedArray > GridDataListNested;

//用来单独存储被占用的格子，遍历时能减少大量计算

UPROPERTY()

TArray<FNestedArray> BlockGridDataList;

/\* 当前地面上的家具列表,正在操作中的家具不会计入列表中，但是会参与阻挡计算 \*/

UPROPERTY()

TArray<class ADecorationBase\*> DecorationList;

#### 地面家具等的基类是ADecorationBase

里面有 /\* 格子Mgr \*/

UPROPERTY()

class UDecorationGridMgrComponent\* GridMgr;

是家具的格子管理控件，包含了

//格子宽高

UPROPERTY()

FVector2D GridRowColumn;

//格子List

UPROPERTY()

TArray< FNestedArray > GridDataList;

protected:

/\* 自身管理的家具Actor \*/

UPROPERTY()

class ADecorationBase\* DecorationActor;

/\* 放入其中的地面的总GridMgr \*/

UPROPERTY()

class UGroundGridMgrComponent\* GroundGridMgr;

数据，GroundGridMgr对应的就是当前格子地面的地面格子管理器。

# 序列化保存记录

所有的家具数据存储在地面上，所以只要把地面序列化保存起来，就可以了。

<https://blog.csdn.net/mohuak/article/details/83027211>

每个uasset对应一个包，UPackage

#### UE4对象类类型引用和类默认对象（Class Default Object,简称CDO）

UCLASS宏为UObject提供了一个描述其基于虚幻的类型的UCLASS的引用。每个UCLASS都维护一个名为“类默认对象”的对象，简称CDO。 CDO本质上是一个默认的“模板”对象，由类构造函数生成，之后未修改。可以为给定的Object实例检索UCLASS和CDO，尽管它们通常应该被认为是只读的。可以使用GetClass()函数随时访问Object实例的UCLASS。

原文链接：https://blog.csdn.net/luomogenhaoqi/article/details/88994343

一个外国友人胸弟的序列化代码等

<https://forums.unrealengine.com/development-discussion/c-gameplay-programming/88477-spawning-actors-from-serialized-data>

FArchive有个字段为

/\*\* Whether this archive is for loading data. \*/

uint8 ArIsLoading : 1;

/\*\* Whether this archive is for saving data. \*/

uint8 ArIsSaving : 1;

在Actor::Serialize中有对其进行判断对loading和Saving进行不同的处理。

断点跟踪拖动一个Actor放入Level，会调用ActorFactory的CreateActor，进而走到UWorld::SpawnActor()

修改Detail上的属性时，UObject会调用Modify，其中会调用UObjectGlobals的函数

bool SaveToTransactionBuffer(UObject\* Object, bool bMarkDirty)

该函数会将该object通过

// Save a copy of the object to the transactor

GUndo->SaveObject( Object );

保存下来，之后会通过SaveObject调用FObjectRecord中的SerializeObject将该Object重新序列化。

经简单OBJ测试，序列化是一个比较可行的方案！但是对于每一个自定义的UObject、Actor都要设置operator<<

#### 20.3.13 新序列化文章测试：

<https://blog.csdn.net/mohuak/article/details/83027211>

<https://wiki.unrealengine.com/Save_System,_Read_%26_Write_Any_Data_to_Compressed_Binary_Files>

<http://runedegroot.com/saving-and-loading-actor-data-in-unreal-engine-4/#comment-93>

主要是参照第三个

首先定义一个接口，用来进行基础的ActorSaveDataLoaded以及ActorSaveDataSaved();

创建一个Struct用来存储Actor的基本序列化数据

再创建一个整个游戏的用来存储序列化数据的数据结构。

最后，最重要的一步，创建一个继承自FObjectAndNameAsStringProxyArchive的struct用来保存Actor的数据！只有继承自该结构的Ar才可以正确进行Actor的序列化！

UGameplayStatics::GetAllActorsWithInterface(GetWorld(), USaveableActorInterface::StaticClass(), Actors);

可以获取到所有继承了某个接口的Actor