// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "Async/AsyncWork.h"

#include "CoreMinimal.h"

#include "UObject/ObjectMacros.h"

#include "UObject/UObjectGlobals.h"

#include "UObject/Object.h"

#include "UObject/StrongObjectPtr.h"

#include "Misc/Guid.h"

#include "Templates/SubclassOf.h"

#include "Templates/UnrealTemplate.h"

#include "Engine/EngineTypes.h"

#include "UObject/ScriptMacros.h"

#include "Interfaces/Interface\_AssetUserData.h"

#include "RenderCommandFence.h"

#include "Interfaces/Interface\_CollisionDataProvider.h"

#include "Interfaces/Interface\_AsyncCompilation.h"

#include "MeshUVChannelInfo.h"

#include "Engine/StreamableRenderAsset.h"

#include "Templates/UniquePtr.h"

#include "StaticMeshSourceData.h"

#include "PerPlatformProperties.h"

#include "MeshTypes.h"

#include "PerQualityLevelProperties.h"

#if UE\_ENABLE\_INCLUDE\_ORDER\_DEPRECATED\_IN\_5\_2

#include "Components.h"

#include "StaticMeshResources.h"

#include "RenderAssetUpdate.h"

#endif

#include "StaticMesh.generated.h"

class FSpeedTreeWind;

class FStaticMeshLODGroup;

class UAssetUserData;

class UMaterialInterface;

class UNavCollisionBase;

class UStaticMeshComponent;

class UStaticMeshDescription;

class FStaticMeshUpdate;

class UPackage;

struct FMeshDescription;

struct FStaticMeshLODResources;

/\*-----------------------------------------------------------------------------

Async Static Mesh Compilation

-----------------------------------------------------------------------------\*/

enum class EStaticMeshAsyncProperties : uint32

{

None                    = 0,

RenderData              = 1 << 0,

//OccluderData            = 1 << 1,

SourceModels            = 1 << 2,

SectionInfoMap          = 1 << 3,

OriginalSectionInfoMap  = 1 << 4,

NavCollision            = 1 << 5,

LightmapUVVersion       = 1 << 6,

BodySetup               = 1 << 7,

LightingGuid            = 1 << 8,

ExtendedBounds          = 1 << 9,

NegativeBoundsExtension = 1 << 10,

PositiveBoundsExtension = 1 << 11,

StaticMaterials         = 1 << 12,

LightmapUVDensity       = 1 << 13,

IsBuiltAtRuntime        = 1 << 14,

MinLOD                  = 1 << 15,

LightMapCoordinateIndex = 1 << 16,

LightMapResolution      = 1 << 17,

HiResSourceModel = 1 << 18,

All                     = MAX\_uint32

};

inline const TCHAR\* ToString(EStaticMeshAsyncProperties Value)

{

switch (Value)

{

case EStaticMeshAsyncProperties::None:

return TEXT("None");

case EStaticMeshAsyncProperties::RenderData:

return TEXT("RenderData");

case EStaticMeshAsyncProperties::SourceModels:

return TEXT("SourceModels");

case EStaticMeshAsyncProperties::SectionInfoMap:

return TEXT("SectionInfoMap");

case EStaticMeshAsyncProperties::OriginalSectionInfoMap:

return TEXT("OriginalSectionInfoMap");

case EStaticMeshAsyncProperties::NavCollision:

return TEXT("NavCollision");

case EStaticMeshAsyncProperties::LightmapUVVersion:

return TEXT("LightmapUVVersion");

case EStaticMeshAsyncProperties::BodySetup:

return TEXT("BodySetup");

case EStaticMeshAsyncProperties::LightingGuid:

return TEXT("LightingGuid");

case EStaticMeshAsyncProperties::ExtendedBounds:

return TEXT("ExtendedBounds");

case EStaticMeshAsyncProperties::NegativeBoundsExtension:

return TEXT("NegativeBoundsExtension");

case EStaticMeshAsyncProperties::PositiveBoundsExtension:

return TEXT("PositiveBoundsExtension");

case EStaticMeshAsyncProperties::StaticMaterials:

return TEXT("StaticMaterials");

case EStaticMeshAsyncProperties::LightmapUVDensity:

return TEXT("LightmapUVDensity");

case EStaticMeshAsyncProperties::IsBuiltAtRuntime:

return TEXT("IsBuiltAtRuntime");

case EStaticMeshAsyncProperties::MinLOD:

return TEXT("MinLOD");

case EStaticMeshAsyncProperties::LightMapCoordinateIndex:

return TEXT("LightMapCoordinateIndex");

case EStaticMeshAsyncProperties::LightMapResolution:

return TEXT("LightMapResolution");

case EStaticMeshAsyncProperties::HiResSourceModel:

return TEXT("HiResSourceModel");

default:

check(false);

return TEXT("Unknown");

}

}

ENUM\_CLASS\_FLAGS(EStaticMeshAsyncProperties);

class FStaticMeshPostLoadContext;

class FStaticMeshBuildContext;

#if WITH\_EDITOR

// Any thread implicated in the static mesh build must have a valid scope to be granted access to protected properties without causing any stalls.

class FStaticMeshAsyncBuildScope

{

public:

FStaticMeshAsyncBuildScope(const UStaticMesh\* StaticMesh)

{

PreviousScope = StaticMeshBeingAsyncCompiled;

StaticMeshBeingAsyncCompiled = StaticMesh;

}

~FStaticMeshAsyncBuildScope()

{

check(StaticMeshBeingAsyncCompiled);

StaticMeshBeingAsyncCompiled = PreviousScope;

}

static bool ShouldWaitOnLockedProperties(const UStaticMesh\* StaticMesh)

{

return StaticMeshBeingAsyncCompiled != StaticMesh;

}

private:

const UStaticMesh\* PreviousScope = nullptr;

// Only the thread(s) compiling this static mesh will have full access to protected properties without causing any stalls.

static thread\_local const UStaticMesh\* StaticMeshBeingAsyncCompiled;

};

/\*\*

 \* Worker used to perform async static mesh compilation.

 \*/

class FStaticMeshAsyncBuildWorker : public FNonAbandonableTask

{

public:

UStaticMesh\* StaticMesh;

TUniquePtr<FStaticMeshPostLoadContext> PostLoadContext;

TUniquePtr<FStaticMeshBuildContext> BuildContext;

/\*\* Initialization constructor. \*/

FStaticMeshAsyncBuildWorker(

UStaticMesh\* InStaticMesh,

TUniquePtr<FStaticMeshBuildContext>&& InBuildContext)

: StaticMesh(InStaticMesh)

, PostLoadContext(nullptr)

, BuildContext(MoveTemp(InBuildContext))

{

}

/\*\* Initialization constructor. \*/

FStaticMeshAsyncBuildWorker(

UStaticMesh\* InStaticMesh,

TUniquePtr<FStaticMeshPostLoadContext>&& InPostLoadContext)

: StaticMesh(InStaticMesh)

, PostLoadContext(MoveTemp(InPostLoadContext))

, BuildContext(nullptr)

{

}

FORCEINLINE TStatId GetStatId() const

{

RETURN\_QUICK\_DECLARE\_CYCLE\_STAT(FStaticMeshAsyncBuildWorker, STATGROUP\_ThreadPoolAsyncTasks);

}

void DoWork();

};

struct FStaticMeshAsyncBuildTask : public FAsyncTask<FStaticMeshAsyncBuildWorker>

{

FStaticMeshAsyncBuildTask(

UStaticMesh\* InStaticMesh,

TUniquePtr<FStaticMeshPostLoadContext>&& InPostLoadContext)

: FAsyncTask<FStaticMeshAsyncBuildWorker>(InStaticMesh, MoveTemp(InPostLoadContext))

, StaticMesh(InStaticMesh)

{

}

FStaticMeshAsyncBuildTask(

UStaticMesh\* InStaticMesh,

TUniquePtr<FStaticMeshBuildContext>&& InBuildContext)

: FAsyncTask<FStaticMeshAsyncBuildWorker>(InStaticMesh, MoveTemp(InBuildContext))

, StaticMesh(InStaticMesh)

{

}

const UStaticMesh\* StaticMesh;

};

#endif // #if WITH\_EDITOR

/\*-----------------------------------------------------------------------------

Legacy mesh optimization settings.

-----------------------------------------------------------------------------\*/

/\*\* Optimization settings used to simplify mesh LODs. \*/

UENUM()

enum ENormalMode : int

{

NM\_PreserveSmoothingGroups,

NM\_RecalculateNormals,

NM\_RecalculateNormalsSmooth,

NM\_RecalculateNormalsHard,

TEMP\_BROKEN,

ENormalMode\_MAX,

};

UENUM()

enum EImportanceLevel : int

{

IL\_Off,

IL\_Lowest,

IL\_Low,

IL\_Normal,

IL\_High,

IL\_Highest,

TEMP\_BROKEN2,

EImportanceLevel\_MAX,

};

/\*\* Enum specifying the reduction type to use when simplifying static meshes. \*/

UENUM()

enum EOptimizationType : int

{

OT\_NumOfTriangles,

OT\_MaxDeviation,

OT\_MAX,

};

/\*\* Old optimization settings. \*/

USTRUCT()

struct FStaticMeshOptimizationSettings

{

GENERATED\_USTRUCT\_BODY()

/\*\* The method to use when optimizing the skeletal mesh LOD \*/

UPROPERTY()

TEnumAsByte<enum EOptimizationType> ReductionMethod;

/\*\* If ReductionMethod equals SMOT\_NumOfTriangles this value is the ratio of triangles [0-1] to remove from the mesh \*/

UPROPERTY()

float NumOfTrianglesPercentage;

/\*\*If ReductionMethod equals SMOT\_MaxDeviation this value is the maximum deviation from the base mesh as a percentage of the bounding sphere. \*/

UPROPERTY()

float MaxDeviationPercentage;

/\*\* The welding threshold distance. Vertices under this distance will be welded. \*/

UPROPERTY()

float WeldingThreshold;

/\*\* Whether Normal smoothing groups should be preserved. If false then NormalsThreshold is used \*\*/

UPROPERTY()

bool bRecalcNormals;

/\*\* If the angle between two triangles are above this value, the normals will not be

smooth over the edge between those two triangles. Set in degrees. This is only used when PreserveNormals is set to false\*/

UPROPERTY()

float NormalsThreshold;

/\*\* How important the shape of the geometry is (EImportanceLevel). \*/

UPROPERTY()

uint8 SilhouetteImportance;

/\*\* How important texture density is (EImportanceLevel). \*/

UPROPERTY()

uint8 TextureImportance;

/\*\* How important shading quality is. \*/

UPROPERTY()

uint8 ShadingImportance;

FStaticMeshOptimizationSettings()

: ReductionMethod( OT\_MaxDeviation )

, NumOfTrianglesPercentage( 1.0f )

, MaxDeviationPercentage( 0.0f )

, WeldingThreshold( 0.1f )

, bRecalcNormals( true )

, NormalsThreshold( 60.0f )

, SilhouetteImportance( IL\_Normal )

, TextureImportance( IL\_Normal )

, ShadingImportance( IL\_Normal )

{

}

/\*\* Serialization for FStaticMeshOptimizationSettings. \*/

inline friend FArchive& operator<<( FArchive& Ar, FStaticMeshOptimizationSettings& Settings )

{

Ar << Settings.ReductionMethod;

Ar << Settings.MaxDeviationPercentage;

Ar << Settings.NumOfTrianglesPercentage;

Ar << Settings.SilhouetteImportance;

Ar << Settings.TextureImportance;

Ar << Settings.ShadingImportance;

Ar << Settings.bRecalcNormals;

Ar << Settings.NormalsThreshold;

Ar << Settings.WeldingThreshold;

return Ar;

}

};

/\*-----------------------------------------------------------------------------

UStaticMesh

-----------------------------------------------------------------------------\*/

/\*\*

 \* Per-section settings.

 \*/

USTRUCT()

struct FMeshSectionInfo

{

GENERATED\_USTRUCT\_BODY()

/\*\* Index in to the Materials array on UStaticMesh. \*/

UPROPERTY()

int32 MaterialIndex;

/\*\* If true, collision is enabled for this section. \*/

UPROPERTY()

bool bEnableCollision;

/\*\* If true, this section will cast shadows. \*/

UPROPERTY()

bool bCastShadow;

/\*\* If true, this section will be visible in ray tracing Geometry. \*/

UPROPERTY()

bool bVisibleInRayTracing;

/\*\* If true, this section will affect lighting methods that use Distance Fields. \*/

UPROPERTY()

bool bAffectDistanceFieldLighting;

/\*\* If true, this section will always considered opaque in ray tracing Geometry. \*/

UPROPERTY()

bool bForceOpaque;

/\*\* Default values. \*/

FMeshSectionInfo()

: MaterialIndex(0)

, bEnableCollision(true)

, bCastShadow(true)

, bVisibleInRayTracing(true)

, bAffectDistanceFieldLighting(true)

, bForceOpaque(false)

{

}

/\*\* Default values with an explicit material index. \*/

explicit FMeshSectionInfo(int32 InMaterialIndex)

: MaterialIndex(InMaterialIndex)

, bEnableCollision(true)

, bCastShadow(true)

, bVisibleInRayTracing(true)

, bAffectDistanceFieldLighting(true)

, bForceOpaque(false)

{

}

/\*\* Comparison for mesh section info. \*/

friend bool operator==(const FMeshSectionInfo& A, const FMeshSectionInfo& B);

friend bool operator!=(const FMeshSectionInfo& A, const FMeshSectionInfo& B);

};

/\*\*

 \* Map containing per-section settings for each section of each LOD.

 \*/

USTRUCT()

struct FMeshSectionInfoMap

{

GENERATED\_USTRUCT\_BODY()

/\*\* Maps an LOD+Section to the material it should render with. \*/

UPROPERTY()

TMap<uint32,FMeshSectionInfo> Map;

/\*\* Serialize. \*/

void Serialize(FArchive& Ar);

/\*\* Clears all entries in the map resetting everything to default. \*/

ENGINE\_API void Clear();

/\*\* Get the number of section for a LOD. \*/

ENGINE\_API int32 GetSectionNumber(int32 LODIndex) const;

/\*\* Return true if the section exist, false otherwise. \*/

ENGINE\_API bool IsValidSection(int32 LODIndex, int32 SectionIndex) const;

/\*\* Gets per-section settings for the specified LOD + section. \*/

ENGINE\_API FMeshSectionInfo Get(int32 LODIndex, int32 SectionIndex) const;

/\*\* Sets per-section settings for the specified LOD + section. \*/

ENGINE\_API void Set(int32 LODIndex, int32 SectionIndex, FMeshSectionInfo Info);

/\*\* Resets per-section settings for the specified LOD + section to defaults. \*/

ENGINE\_API void Remove(int32 LODIndex, int32 SectionIndex);

/\*\* Copies per-section settings from the specified section info map. \*/

ENGINE\_API void CopyFrom(const FMeshSectionInfoMap& Other);

/\*\* Returns true if any section of the specified LOD has collision enabled. \*/

bool AnySectionHasCollision(int32 LodIndex) const;

};

USTRUCT()

struct FAssetEditorOrbitCameraPosition

{

GENERATED\_USTRUCT\_BODY()

FAssetEditorOrbitCameraPosition()

: bIsSet(false)

, CamOrbitPoint(ForceInitToZero)

, CamOrbitZoom(ForceInitToZero)

, CamOrbitRotation(ForceInitToZero)

{

}

FAssetEditorOrbitCameraPosition(const FVector& InCamOrbitPoint, const FVector& InCamOrbitZoom, const FRotator& InCamOrbitRotation)

: bIsSet(true)

, CamOrbitPoint(InCamOrbitPoint)

, CamOrbitZoom(InCamOrbitZoom)

, CamOrbitRotation(InCamOrbitRotation)

{

}

/\*\* Whether or not this has been set to a valid value \*/

UPROPERTY()

bool bIsSet;

/\*\* The position to orbit the camera around \*/

UPROPERTY()

FVector CamOrbitPoint;

/\*\* The distance of the camera from the orbit point \*/

UPROPERTY()

FVector CamOrbitZoom;

/\*\* The rotation to apply around the orbit point \*/

UPROPERTY()

FRotator CamOrbitRotation;

};

#if WITH\_EDITOR

/\*\* delegate type for pre mesh build events \*/

DECLARE\_MULTICAST\_DELEGATE\_OneParam(FOnPreMeshBuild, class UStaticMesh\*);

/\*\* delegate type for post mesh build events \*/

DECLARE\_MULTICAST\_DELEGATE\_OneParam(FOnPostMeshBuild, class UStaticMesh\*);

#endif

//~ Begin Material Interface for UStaticMesh - contains a material and other stuff

USTRUCT(BlueprintType)

struct FStaticMaterial

{

GENERATED\_USTRUCT\_BODY()

ENGINE\_API FStaticMaterial();

ENGINE\_API FStaticMaterial(class UMaterialInterface\* InMaterialInterface

, FName InMaterialSlotName = NAME\_None

#if WITH\_EDITORONLY\_DATA

, FName InImportedMaterialSlotName = NAME\_None

#endif

);

friend FArchive& operator<<(FArchive& Ar, FStaticMaterial& Elem);

ENGINE\_API friend bool operator==(const FStaticMaterial& LHS, const FStaticMaterial& RHS);

ENGINE\_API friend bool operator==(const FStaticMaterial& LHS, const UMaterialInterface& RHS);

ENGINE\_API friend bool operator==(const UMaterialInterface& LHS, const FStaticMaterial& RHS);

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = StaticMesh)

TObjectPtr<class UMaterialInterface> MaterialInterface;

/\*This name should be use by the gameplay to avoid error if the skeletal mesh Materials array topology change\*/

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = StaticMesh)

FName MaterialSlotName;

/\*This name should be use when we re-import a skeletal mesh so we can order the Materials array like it should be\*/

UPROPERTY(VisibleAnywhere, Category = StaticMesh)

FName ImportedMaterialSlotName;

/\*\* Data used for texture streaming relative to each UV channels. \*/

UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = StaticMesh)

FMeshUVChannelInfo UVChannelData;

};

enum EImportStaticMeshVersion

{

// Before any version changes were made

BeforeImportStaticMeshVersionWasAdded,

// Remove the material re-order workflow

RemoveStaticMeshSkinxxWorkflow,

StaticMeshVersionPlusOne,

LastVersion = StaticMeshVersionPlusOne - 1

};

USTRUCT()

struct FMaterialRemapIndex

{

GENERATED\_USTRUCT\_BODY()

FMaterialRemapIndex()

{

ImportVersionKey = 0;

}

FMaterialRemapIndex(uint32 VersionKey, TArray<int32> RemapArray)

: ImportVersionKey(VersionKey)

, MaterialRemap(RemapArray)

{

}

UPROPERTY()

uint32 ImportVersionKey;

UPROPERTY()

TArray<int32> MaterialRemap;

};

/\*\*

 \* A StaticMesh is a piece of geometry that consists of a static set of polygons.

 \* Static Meshes can be translated, rotated, and scaled, but they cannot have their vertices animated in any way. As such, they are more efficient

 \* to render than other types of geometry such as USkeletalMesh, and they are often the basic building block of levels created in the engine.

 \*

 \* @see https://docs.unrealengine.com/latest/INT/Engine/Content/Types/StaticMeshes/

 \* @see AStaticMeshActor, UStaticMeshComponent

 \*/

UCLASS(hidecategories=Object, customconstructor, MinimalAPI, BlueprintType, config=Engine)

class UStaticMesh : public UStreamableRenderAsset, public IInterface\_CollisionDataProvider, public IInterface\_AssetUserData, public IInterface\_AsyncCompilation

{

GENERATED\_UCLASS\_BODY()

#if WITH\_EDITOR

/\*\* Notification when bounds changed \*/

DECLARE\_MULTICAST\_DELEGATE\_OneParam(FOnExtendedBoundsChanged, const FBoxSphereBounds&);

/\*\* Notification when anything changed \*/

DECLARE\_MULTICAST\_DELEGATE(FOnMeshChanged);

#endif

public:

ENGINE\_API ~UStaticMesh();

private:

#if WITH\_EDITOR

/\*\* Used as a bit-field indicating which properties are currently accessed/modified by async compilation. \*/

*std*::*atomic*<uint32> LockedProperties;

void AcquireAsyncProperty(EStaticMeshAsyncProperties AsyncProperties = EStaticMeshAsyncProperties::All);

void ReleaseAsyncProperty(EStaticMeshAsyncProperties AsyncProperties = EStaticMeshAsyncProperties::All);

ENGINE\_API void WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties AsyncProperties) const;

#else

FORCEINLINE void AcquireAsyncProperty(EStaticMeshAsyncProperties AsyncProperties = EStaticMeshAsyncProperties::All) {};

FORCEINLINE void ReleaseAsyncProperty(EStaticMeshAsyncProperties AsyncProperties = EStaticMeshAsyncProperties::All) {};

FORCEINLINE void WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties AsyncProperties) const {}

#endif

/\*\* Pointer to the data used to render this static mesh. \*/

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

TUniquePtr<class FStaticMeshRenderData> RenderData;

public:

#if WITH\_EDITOR

bool IsCompiling() const override { return AsyncTask != nullptr || LockedProperties.load(*std*::*memory\_order\_relaxed*) != 0; }

#else

FORCEINLINE bool IsCompiling() const { return false; }

#endif

ENGINE\_API FStaticMeshRenderData\* GetRenderData();

ENGINE\_API const FStaticMeshRenderData\* GetRenderData() const;

ENGINE\_API void SetRenderData(TUniquePtr<class FStaticMeshRenderData>&& InRenderData);

void RequestUpdateCachedRenderState() const;

#if WITH\_EDITORONLY\_DATA

static const float MinimumAutoLODPixelError;

private:

/\*\* Imported raw mesh bulk data. \*/

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY(Setter = None, Getter = None)

TArray<FStaticMeshSourceModel> SourceModels;

/\*\* Optional hi-res source data \*/

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY(Setter = None, Getter = None)

FStaticMeshSourceModel HiResSourceModel;

void SetLightmapUVVersion(int32 InLightmapUVVersion)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightmapUVVersion);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

LightmapUVVersion = InLightmapUVVersion;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

/\*\* Map of LOD+Section index to per-section info. \*/

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY()

FMeshSectionInfoMap SectionInfoMap;

/\*\*

 \* We need the OriginalSectionInfoMap to be able to build mesh in a non destructive way. Reduce has to play with SectionInfoMap in case some sections disappear.

 \* This member will be update in the following situation

 \* 1. After a static mesh import/reimport

 \* 2. Postload, if the OriginalSectionInfoMap is empty, we will fill it with the current SectionInfoMap

 \*

 \* We do not update it when the user shuffle section in the staticmesh editor because the OriginalSectionInfoMap must always be in sync with the saved rawMesh bulk data.

 \*/

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY()

FMeshSectionInfoMap OriginalSectionInfoMap;

public:

static FName GetSectionInfoMapName()

{

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return GET\_MEMBER\_NAME\_CHECKED(UStaticMesh, SectionInfoMap);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

/\*\* The LOD group to which this mesh belongs. \*/

UPROPERTY(EditAnywhere, AssetRegistrySearchable, Category=LodSettings)

FName LODGroup;

/\*\*

 \* If non-negative, specify the maximum number of streamed LODs. Only has effect if

 \* mesh LOD streaming is enabled for the target platform.

 \*/

UPROPERTY()

FPerPlatformInt NumStreamedLODs;

/\* The last import version \*/

UPROPERTY()

int32 ImportVersion;

UPROPERTY()

TArray<FMaterialRemapIndex> MaterialRemapIndexPerImportVersion;

private:

/\* The lightmap UV generation version used during the last derived data build \*/

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY()

int32 LightmapUVVersion;

public:

int32 GetLightmapUVVersion() const

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightmapUVVersion);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return LightmapUVVersion;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

/\*\* If true, the screen sizes at which LODs swap are computed automatically. \*/

UPROPERTY()

uint8 bAutoComputeLODScreenSize : 1;

/\*\*

\* If true on post load we need to calculate Display Factors from the

\* loaded LOD distances.

\*/

uint8 bRequiresLODDistanceConversion : 1;

/\*\*

 \* If true on post load we need to calculate resolution independent Display Factors from the

 \* loaded LOD screen sizes.

 \*/

uint8 bRequiresLODScreenSizeConversion : 1;

/\*\* Materials used by this static mesh. Individual sections index in to this array. \*/

UPROPERTY()

TArray<TObjectPtr<UMaterialInterface>> Materials\_DEPRECATED;

/\*\* Settings related to building Nanite data. \*/

UPROPERTY(EditAnywhere, Category=NaniteSettings)

FMeshNaniteSettings NaniteSettings;

#endif // #if WITH\_EDITORONLY\_DATA

#if WITH\_EDITOR

UFUNCTION(BlueprintCallable, Category = StaticMesh)

bool IsLODScreenSizeAutoComputed() const

{

return bAutoComputeLODScreenSize;

}

#endif

/\*\* Check the QualitLevel property is enabled for MinLod. \*/

bool IsMinLodQualityLevelEnable() const;

UPROPERTY()

/\*PerQuality override. Note: Enable PerQuality override in the Project Settings/ General Settings/ UseStaticMeshMinLODPerQualityLevels\*/

/\* Allow more flexibility to set various values driven by the Scalability or Device Profile.\*/

FPerQualityLevelInt MinQualityLevelLOD;

static FName GetQualityLevelMinLODMemberName()

{

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return GET\_MEMBER\_NAME\_CHECKED(UStaticMesh, MinQualityLevelLOD);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

const FPerQualityLevelInt& GetQualityLevelMinLOD() const

{

return MinQualityLevelLOD;

}

void SetQualityLevelMinLOD(FPerQualityLevelInt InMinLOD)

{

MinQualityLevelLOD = MoveTemp(InMinLOD);

}

UFUNCTION(BlueprintPure, Category = StaticMesh)

void GetMinimumLODForQualityLevels(TMap<FName, int32>& QualityLevelMinimumLODs) const

{

#if WITH\_EDITORONLY\_DATA

for (const TPair<int32, int32>& Pair : GetQualityLevelMinLOD().PerQuality)

{

QualityLevelMinimumLODs.Add(QualityLevelProperty::QualityLevelToFName(Pair.Key), Pair.Value);

}

#endif

}

UFUNCTION(BlueprintPure, Category = StaticMesh)

int32 GetMinimumLODForQualityLevel(const FName& QualityLevel) const

{

#if WITH\_EDITORONLY\_DATA

int32 QualityLevelKey = QualityLevelProperty::FNameToQualityLevel(QualityLevel);

if (const int32\* Result = GetQualityLevelMinLOD().PerQuality.Find(QualityLevelKey))

{

return \*Result;

}

#endif

return INDEX\_NONE;

}

UFUNCTION(BlueprintCallable, Category = StaticMesh, Meta = (ToolTip = "Allow to override min lod quality levels on a staticMesh and it Default value (-1 value for Default dont override its value)."))

void SetMinLODForQualityLevels(const TMap<EPerQualityLevels, int32>& QualityLevelMinimumLODs, int32 Default = -1)

{

#if WITH\_EDITORONLY\_DATA

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::MinLOD);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

MinQualityLevelLOD.PerQuality = QualityLevelProperty::ConvertQualtiyLevelData(QualityLevelMinimumLODs);

MinQualityLevelLOD.Default = Default >= 0 ? Default : MinQualityLevelLOD.Default;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

#endif

}

UFUNCTION(BlueprintPure, Category = StaticMesh)

void GetMinLODForQualityLevels(TMap<EPerQualityLevels, int32>& QualityLevelMinimumLODs, int32& Default) const

{

#if WITH\_EDITORONLY\_DATA

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::MinLOD);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

QualityLevelMinimumLODs = QualityLevelProperty::ConvertQualtiyLevelData(MinQualityLevelLOD.PerQuality);

Default = MinQualityLevelLOD.Default;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

#endif

}

/\*Choose either PerPlatform or PerQuality override. Note: Enable PerQuality override in the Project Settings/ General Settings/ UseStaticMeshMinLODPerQualityLevels\*/

ENGINE\_API int32 GetMinLODIdx(bool bForceLowestLODIdx = false) const;

ENGINE\_API int32 GetDefaultMinLOD() const;

ENGINE\_API void SetMinLODIdx(int32 InMinLOD);

ENGINE\_API static void OnLodStrippingQualityLevelChanged(IConsoleVariable\* Variable);

/\*\* Minimum LOD to use for rendering.  This is the default setting for the mesh and can be overridden by component settings. \*/

UE\_DEPRECATED(4.27, "Please do not access this member directly; use UStaticMesh::GetMinLOD() or UStaticMesh::SetMinLOD().")

UPROPERTY()

FPerPlatformInt MinLOD;

const FPerPlatformInt& GetMinLOD() const

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::MinLOD);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return MinLOD;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

void SetMinLOD(FPerPlatformInt InMinLOD)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::MinLOD);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

MinLOD = MoveTemp(InMinLOD);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

UFUNCTION(BlueprintPure, Category=StaticMesh)

void GetMinimumLODForPlatforms(TMap<FName, int32>& PlatformMinimumLODs) const

{

#if WITH\_EDITORONLY\_DATA

PlatformMinimumLODs = GetMinLOD().PerPlatform;

#endif

}

UFUNCTION(BlueprintPure, Category=StaticMesh)

int32 GetMinimumLODForPlatform(const FName& PlatformName) const

{

#if WITH\_EDITORONLY\_DATA

if (const int32\* Result = GetMinLOD().PerPlatform.Find(PlatformName))

{

return \*Result;

}

#endif

return INDEX\_NONE;

}

UFUNCTION(BlueprintCallable, Category=StaticMesh)

void SetMinimumLODForPlatforms(const TMap<FName, int32>& PlatformMinimumLODs)

{

#if WITH\_EDITORONLY\_DATA

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::MinLOD);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

MinLOD.PerPlatform = PlatformMinimumLODs;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

#endif

}

UFUNCTION(BlueprintCallable, Category=StaticMesh)

void SetMinimumLODForPlatform(const FName& PlatformName, int32 InMinLOD)

{

#if WITH\_EDITORONLY\_DATA

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::MinLOD);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

MinLOD.PerPlatform.Add(PlatformName, InMinLOD);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

#endif

}

#if WITH\_EDITORONLY\_DATA

/\*\*

 \* Returns true if this SM should have Nanite built for it.

 \* This also includes the result of IsNaniteForceEnabled().

 \*/

ENGINE\_API bool IsNaniteEnabled() const;

/\*\*

 \* Returns true if this SM should always have Nanite data built.

 \* This forces the SM to be Nanite even if the flag in the editor is set to false.

 \*/

ENGINE\_API bool IsNaniteForceEnabled() const;

#endif

// TODO: Temp/deprecated hack - Do not call

inline bool IsNaniteLandscape() const

{

return GetName().StartsWith(TEXT("LandscapeNaniteMesh"));

}

private:

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY(BlueprintGetter = GetStaticMaterials, BlueprintSetter = SetStaticMaterials, Category = StaticMesh)

TArray<FStaticMaterial> StaticMaterials;

public:

static FName GetStaticMaterialsName()

{

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return GET\_MEMBER\_NAME\_CHECKED(UStaticMesh, StaticMaterials);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

TArray<FStaticMaterial>& GetStaticMaterials()

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::StaticMaterials);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return StaticMaterials;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

UFUNCTION(BlueprintGetter)

const TArray<FStaticMaterial>& GetStaticMaterials() const

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::StaticMaterials);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return StaticMaterials;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

UFUNCTION(BlueprintSetter)

void SetStaticMaterials(const TArray<FStaticMaterial>& InStaticMaterials)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::StaticMaterials);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

StaticMaterials = InStaticMaterials;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

private:

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY()

float LightmapUVDensity;

public:

void SetLightmapUVDensity(float InLightmapUVDensity)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightmapUVDensity);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

LightmapUVDensity = InLightmapUVDensity;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

float GetLightmapUVDensity() const

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightmapUVDensity);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return LightmapUVDensity;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

UE\_DEPRECATED(4.27, "Please do not access this member directly; use UStaticMesh::GetLightMapResolution() or UStaticMesh::SetLightMapResolution().")

UPROPERTY(EditAnywhere, Category=StaticMesh, *meta*=(ClampMax = 4096, ToolTip="The light map resolution", FixedIncrement="4.0"))

int32 LightMapResolution;

int32 GetLightMapResolution() const

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightMapResolution);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return LightMapResolution;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

void SetLightMapResolution(int32 InLightMapResolution)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightMapResolution);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

LightMapResolution = InLightMapResolution;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

static FName GetLightMapResolutionName()

{

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return GET\_MEMBER\_NAME\_CHECKED(UStaticMesh, LightMapResolution);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

/\*\* The light map coordinate index \*/

UE\_DEPRECATED(4.27, "Please do not access this member directly; use UStaticMesh::GetLightMapCoordinateIndex() or UStaticMesh::SetLightMapCoordinateIndex().")

UPROPERTY(EditAnywhere, AdvancedDisplay, Category=StaticMesh, *meta*=(ToolTip="The light map coordinate index", UIMin = "0", UIMax = "3"))

int32 LightMapCoordinateIndex;

int32 GetLightMapCoordinateIndex() const

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightMapCoordinateIndex);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return LightMapCoordinateIndex;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

void SetLightMapCoordinateIndex(int32 InLightMapCoordinateIndex)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightMapCoordinateIndex);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

LightMapCoordinateIndex = InLightMapCoordinateIndex;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

static FName GetLightMapCoordinateIndexName()

{

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return GET\_MEMBER\_NAME\_CHECKED(UStaticMesh, LightMapCoordinateIndex);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

/\*\* Useful for reducing self shadowing from distance field methods when using world position offset to animate the mesh's vertices. \*/

UPROPERTY(EditAnywhere, AdvancedDisplay, Category = StaticMesh)

float DistanceFieldSelfShadowBias;

private:

// Physics data.

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY(EditAnywhere, transient, duplicatetransient, Instanced, Category = StaticMesh)

TObjectPtr<class UBodySetup> BodySetup;

public:

UBodySetup\* GetBodySetup() const

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::BodySetup);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return BodySetup;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

void SetBodySetup(UBodySetup\* InBodySetup)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::BodySetup);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

BodySetup = InBodySetup;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

static FName GetBodySetupName()

{

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return GET\_MEMBER\_NAME\_CHECKED(UStaticMesh, BodySetup);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

/\*\*

 \* Specifies which mesh LOD to use for complex (per-poly) collision.

 \* Sometimes it can be desirable to use a lower poly representation for collision to reduce memory usage, improve performance and behaviour.

 \* Collision representation does not change based on distance to camera.

 \*/

UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = StaticMesh, *meta*=(DisplayName="LOD For Collision"))

int32 LODForCollision;

/\*\*

 \* Whether to generate a distance field for this mesh, which can be used by DistanceField Indirect Shadows.

 \* This is ignored if the project's 'Generate Mesh Distance Fields' setting is enabled.

 \*/

UPROPERTY(EditAnywhere, Category=StaticMesh)

uint8 bGenerateMeshDistanceField : 1;

/\*\* If true, strips unwanted complex collision data aka kDOP tree when cooking for consoles.

On the Playstation 3 data of this mesh will be stored in video memory. \*/

UPROPERTY()

uint8 bStripComplexCollisionForConsole\_DEPRECATED:1;

/\*\* If true, mesh will have NavCollision property with additional data for navmesh generation and usage.

    Set to false for distant meshes (always outside navigation bounds) to save memory on collision data. \*/

UPROPERTY(EditAnywhere, Category=Navigation)

uint8 bHasNavigationData:1;

/\*\*

Mesh supports uniformly distributed sampling in constant time.

Memory cost is 8 bytes per triangle.

Example usage is uniform spawning of particles.

\*/

UPROPERTY(EditAnywhere, AdvancedDisplay, Category = StaticMesh)

uint8 bSupportUniformlyDistributedSampling : 1;

/\*\*

If true, complex collision data will store UVs and face remap table for use when performing

    PhysicalMaterialMask lookups in cooked builds. Note the increased memory cost for this

functionality.

\*/

UPROPERTY(EditAnywhere, AdvancedDisplay, Category = StaticMesh)

uint8 bSupportPhysicalMaterialMasks : 1;

/\*\*

 \* If true, a ray tracing acceleration structure will be built for this mesh and it may be used in ray tracing effects

 \*/

UPROPERTY(EditAnywhere, Category = RayTracing)

uint8 bSupportRayTracing : 1;

UPROPERTY()

uint8 bDoFastBuild : 1;

private:

UPROPERTY()

uint8 bIsBuiltAtRuntime\_DEPRECATED : 1;

public:

UE\_DEPRECATED(5.0, "IsBuiltAtRuntime() is no longer used.")

bool IsBuiltAtRuntime() const

{

return false;

}

UE\_DEPRECATED(5.0, "SetIsBuiltAtRuntime() is no longer used.")

void SetIsBuiltAtRuntime(bool InIsBuiltAtRuntime)

{

}

protected:

/\*\* Tracks whether InitResources has been called, and rendering resources are initialized. \*/

uint8 bRenderingResourcesInitialized:1;

public:

/\*\*

 \* If true, will keep geometry data CPU-accessible in cooked builds, rather than uploading to GPU memory and releasing it from CPU memory.

 \* This is required if you wish to access StaticMesh geometry data on the CPU at runtime in cooked builds (e.g. to convert StaticMesh to ProceduralMeshComponent)

 \*/

UPROPERTY(EditAnywhere, AdvancedDisplay, Category = StaticMesh)

uint8 bAllowCPUAccess:1;

/\*\*

 \* If true, a GPU buffer containing required data for uniform mesh surface sampling will be created at load time.

 \* It is created from the cpu data so bSupportUniformlyDistributedSampling is also required to be true.

 \*/

UPROPERTY(EditAnywhere, AdvancedDisplay, Category = StaticMesh)

uint8 bSupportGpuUniformlyDistributedSampling : 1;

/\*\* A fence which is used to keep track of the rendering thread releasing the static mesh resources. \*/

FRenderCommandFence ReleaseResourcesFence;

#if WITH\_EDITORONLY\_DATA

/\*\* Importing data and options used for this mesh \*/

UPROPERTY(EditAnywhere, Instanced, Category=ImportSettings)

TObjectPtr<class UAssetImportData> AssetImportData;

/\*\* Path to the resource used to construct this static mesh \*/

UPROPERTY()

FString SourceFilePath\_DEPRECATED;

/\*\* Date/Time-stamp of the file from the last import \*/

UPROPERTY()

FString SourceFileTimestamp\_DEPRECATED;

/\*\* Information for thumbnail rendering \*/

UPROPERTY(VisibleAnywhere, Instanced, AdvancedDisplay, Category=StaticMesh)

TObjectPtr<class UThumbnailInfo> ThumbnailInfo;

/\*\* The stored camera position to use as a default for the static mesh editor \*/

UPROPERTY()

FAssetEditorOrbitCameraPosition EditorCameraPosition;

/\*\* If the user has modified collision in any way or has custom collision imported. Used for determining if to auto generate collision on import \*/

UPROPERTY(EditAnywhere, Category = Collision)

bool bCustomizedCollision;

#endif // WITH\_EDITORONLY\_DATA

private:

/\*\* Unique ID for tracking/caching this mesh during distributed lighting \*/

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

FGuid LightingGuid;

public:

const FGuid& GetLightingGuid() const

{

#if WITH\_EDITORONLY\_DATA

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightingGuid);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return LightingGuid;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

#else

static const FGuid NullGuid( 0, 0, 0, 0 );

return NullGuid;

#endif // WITH\_EDITORONLY\_DATA

}

void SetLightingGuid(const FGuid& InLightingGuid = FGuid::NewGuid())

{

#if WITH\_EDITORONLY\_DATA

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::LightingGuid);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

LightingGuid = InLightingGuid;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

#endif // WITH\_EDITORONLY\_DATA

}

/\*\*

 \* Array of named socket locations, set up in editor and used as a shortcut instead of specifying

 \* everything explicitly to AttachComponent in the StaticMeshComponent.

 \*/

UPROPERTY()

TArray<TObjectPtr<class UStaticMeshSocket>> Sockets;

/\*\* Data that is only available if this static mesh is an imported SpeedTree \*/

TSharedPtr<class FSpeedTreeWind> SpeedTreeWind;

/\*\* Bound extension values in the positive direction of XYZ, positive value increases bound size \*/

UE\_DEPRECATED(4.27, "Please do not access this member directly; use UStaticMesh::GetPositiveBoundsExtension() or UStaticMesh::SetPositiveBoundsExtension.")

UPROPERTY(EditDefaultsOnly, AdvancedDisplay, Category = StaticMesh)

FVector PositiveBoundsExtension;

const FVector& GetPositiveBoundsExtension() const

{

// No need for WaitUntilAsyncPropertyReleased here as this is only read during async Build/Postload

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return PositiveBoundsExtension;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

void SetPositiveBoundsExtension(FVector InPositiveBoundsExtension)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::PositiveBoundsExtension);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

PositiveBoundsExtension = InPositiveBoundsExtension;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

static FName GetPositiveBoundsExtensionName()

{

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return GET\_MEMBER\_NAME\_CHECKED(UStaticMesh, PositiveBoundsExtension);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

/\*\* Bound extension values in the negative direction of XYZ, positive value increases bound size \*/

UE\_DEPRECATED(4.27, "Please do not access this member directly; use UStaticMesh::GetNegativeBoundsExtension() or UStaticMesh::SetNegativeBoundsExtension.")

UPROPERTY(EditDefaultsOnly, AdvancedDisplay, Category = StaticMesh)

FVector NegativeBoundsExtension;

const FVector& GetNegativeBoundsExtension() const

{

// No need for WaitUntilAsyncPropertyReleased here as this is not modified during async Build/Postload

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return NegativeBoundsExtension;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

void SetNegativeBoundsExtension(FVector InNegativeBoundsExtension)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::NegativeBoundsExtension);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

NegativeBoundsExtension = InNegativeBoundsExtension;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

static FName GetNegativeBoundsExtensionName()

{

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return GET\_MEMBER\_NAME\_CHECKED(UStaticMesh, NegativeBoundsExtension);

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

/\*\* Original mesh bounds extended with Positive/NegativeBoundsExtension \*/

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY()

FBoxSphereBounds ExtendedBounds;

public:

const FBoxSphereBounds& GetExtendedBounds() const

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::ExtendedBounds);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

return ExtendedBounds;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

}

void SetExtendedBounds(const FBoxSphereBounds& InExtendedBounds)

{

WaitUntilAsyncPropertyReleased(EStaticMeshAsyncProperties::ExtendedBounds);

PRAGMA\_DISABLE\_DEPRECATION\_WARNINGS

ExtendedBounds = InExtendedBounds;

PRAGMA\_ENABLE\_DEPRECATION\_WARNINGS

#if WITH\_EDITOR

OnExtendedBoundsChanged.Broadcast(InExtendedBounds);

#endif

}

#if WITH\_EDITOR

FOnExtendedBoundsChanged OnExtendedBoundsChanged;

FOnMeshChanged OnMeshChanged;

/\*\* This transient guid is use by the automation framework to modify the DDC key to force a build. \*/

FGuid BuildCacheAutomationTestGuid;

#endif

protected:

/\*\*

 \* Index of an element to ignore while gathering streaming texture factors.

 \* This is useful to disregard automatically generated vertex data which breaks texture factor heuristics.

 \*/

UPROPERTY()

int32 ElementToIgnoreForTexFactor;

/\*\* Array of user data stored with the asset \*/

UPROPERTY(EditAnywhere, AdvancedDisplay, Instanced, Category = StaticMesh)

TArray<TObjectPtr<UAssetUserData>> AssetUserData;

friend class FStaticMeshCompilingManager;

friend class FStaticMeshAsyncBuildWorker;

friend struct FStaticMeshUpdateContext;

friend class FStaticMeshUpdate;

public:

#if WITH\_EDITORONLY\_DATA

UPROPERTY(Instanced)

TObjectPtr<class UObject> EditableMesh\_DEPRECATED;

UPROPERTY(EditAnywhere, Category = Collision)

TObjectPtr<class UStaticMesh> ComplexCollisionMesh;

#endif

/\*\*

 \* Registers the mesh attributes required by the mesh description for a static mesh.

 \*/

UE\_DEPRECATED(4.24, "Please use FStaticMeshAttributes::Register to do this.")

ENGINE\_API static void RegisterMeshAttributes( FMeshDescription& MeshDescription );

#if WITH\_EDITORONLY\_DATA

/\*

 \* Return the MeshDescription associate to the LODIndex. The mesh description can be created on the fly if it was null

 \* and there is a FRawMesh data for this LODIndex.

 \*/

ENGINE\_API FMeshDescription\* GetMeshDescription(int32 LodIndex) const;

/\*

 \* Clone the MeshDescription associated to the LODIndex.

 \*

 \* This will make a copy of any pending mesh description that hasn't been committed or will deserialize

 \* from the bulkdata or rawmesh directly if no current working copy exists.

 \*/

ENGINE\_API bool CloneMeshDescription(int32 LodIndex, FMeshDescription& OutMeshDescription) const;

ENGINE\_API bool IsMeshDescriptionValid(int32 LodIndex) const;

ENGINE\_API FMeshDescription\* CreateMeshDescription(int32 LodIndex);

ENGINE\_API FMeshDescription\* CreateMeshDescription(int32 LodIndex, FMeshDescription MeshDescription);

/\*\* Structure that defines parameters passed into the commit mesh description function \*/

struct FCommitMeshDescriptionParams

{

FCommitMeshDescriptionParams()

: bMarkPackageDirty(true)

, bUseHashAsGuid(false)

{}

/\*\*

\* If set to false, the caller can be from any thread but will have the

\* responsability to call MarkPackageDirty() from the main thread.

\*/

bool bMarkPackageDirty;

/\*\*

\* Uses a hash as the GUID, useful to prevent recomputing content already in cache.

\*/

bool bUseHashAsGuid;

};

/\*\*

 \* Serialize the mesh description into its more optimized form.

 \*

 \* @param LodIndex Index of the StaticMesh LOD.

 \* @param Params Different options to use when committing mesh description

 \*/

ENGINE\_API void CommitMeshDescription(int32 LodIndex, const FCommitMeshDescriptionParams& Params = FCommitMeshDescriptionParams());

/\*\*

 \* Clears the cached mesh description for the given LOD.

 \* Note that this does not empty the bulk data.

 \*/

ENGINE\_API void ClearMeshDescription(int32 LodIndex);

/\*\*

 \* Clears cached mesh descriptions for all LODs.

 \*/

ENGINE\_API void ClearMeshDescriptions();

ENGINE\_API bool LoadHiResMeshDescription(FMeshDescription& OutMeshDescription) const;

ENGINE\_API bool CloneHiResMeshDescription(FMeshDescription& OutMeshDescription) const;

ENGINE\_API FMeshDescription\* CreateHiResMeshDescription();

ENGINE\_API FMeshDescription\* CreateHiResMeshDescription(FMeshDescription MeshDescription);

ENGINE\_API FMeshDescription\* GetHiResMeshDescription() const;

ENGINE\_API bool IsHiResMeshDescriptionValid() const;

ENGINE\_API void CommitHiResMeshDescription(const FCommitMeshDescriptionParams& Params = FCommitMeshDescriptionParams());

ENGINE\_API void ClearHiResMeshDescription();

/\*\*

 \* Performs a Modify on the StaticMeshDescription object pertaining to the given LODIndex

 \*/

ENGINE\_API bool ModifyMeshDescription(int32 LodIndex, bool bAlwaysMarkDirty = true);

/\*\*

 \* Performs a Modify on StaticMeshDescription objects for all LODs

 \*/

ENGINE\_API bool ModifyAllMeshDescriptions(bool bAlwaysMarkDirty = true);

/\*\*

 \* Performs a Modify on the hi-res StaticMeshDescription

 \*/

ENGINE\_API bool ModifyHiResMeshDescription(bool bAlwaysMarkDirty = true);

/\*\*

 \* Get AssetImportData for the static mesh

 \*/

class UAssetImportData\* GetAssetImportData() const

{

return AssetImportData;

}

/\*\*

 \* Set AssetImportData for the static mesh

 \*/

void SetAssetImportData(class UAssetImportData\* InAssetImportData)

{

AssetImportData = InAssetImportData;

}

/\*\*

 \* Adds an empty UV channel at the end of the existing channels on the given LOD of a StaticMesh.

 \* @param LODIndex Index of the StaticMesh LOD.

 \* @return true if a UV channel was added.

 \*/

ENGINE\_API bool AddUVChannel(int32 LODIndex);

/\*\*

 \* Inserts an empty UV channel at the specified channel index on the given LOD of a StaticMesh.

 \* @param LODIndex Index of the StaticMesh LOD.

 \* @param UVChannelIndex Index where to insert the UV channel.

 \* @return true if a UV channel was added.

 \*/

ENGINE\_API bool InsertUVChannel(int32 LODIndex, int32 UVChannelIndex);

/\*\*

 \* Removes the UV channel at the specified channel index on the given LOD of a StaticMesh.

 \* @param LODIndex Index of the StaticMesh LOD.

 \* @param UVChannelIndex Index where to remove the UV channel.

 \* @return true if the UV channel was removed.

 \*/

ENGINE\_API bool RemoveUVChannel(int32 LODIndex, int32 UVChannelIndex);

/\*\*

 \* Sets the texture coordinates at the specified UV channel index on the given LOD of a StaticMesh.

 \* @param LODIndex Index of the StaticMesh LOD.

 \* @param UVChannelIndex Index where to remove the UV channel.

 \* @param TexCoords The texture coordinates to set on the UV channel.

 \* @return true if the UV channel could be set.

 \*/

ENGINE\_API bool SetUVChannel(int32 LODIndex, int32 UVChannelIndex, const TMap<FVertexInstanceID, FVector2D>& TexCoords);

#endif

/\*\* Create an empty StaticMeshDescription object, to describe a static mesh at runtime \*/

UFUNCTION(BlueprintCallable, Category="StaticMesh")

static ENGINE\_API UStaticMeshDescription\* CreateStaticMeshDescription(UObject\* Outer = nullptr);

/\*\* Builds static mesh LODs from the array of StaticMeshDescriptions passed in \*/

UFUNCTION(BlueprintCallable, Category="StaticMesh")

ENGINE\_API void BuildFromStaticMeshDescriptions(const TArray<UStaticMeshDescription\*>& StaticMeshDescriptions, bool bBuildSimpleCollision = false, bool bFastBuild = true);

/\*\* Return a new StaticMeshDescription referencing the MeshDescription of the given LOD \*/

UFUNCTION(BlueprintCallable, Category="StaticMesh")

ENGINE\_API UStaticMeshDescription\* GetStaticMeshDescription(int32 LODIndex);

struct FBuildMeshDescriptionsLODParams

{

/\*\*

 \* If true, Tangents will be stored at 16 bit vs 8 bit precision.

 \*/

bool bUseHighPrecisionTangentBasis = false;

/\*\*

 \* If true, UVs will be stored at full floating point precision.

 \*/

bool bUseFullPrecisionUVs = false;

};

 /\*\* Structure that defines parameters passed into the build mesh description function \*/

struct FBuildMeshDescriptionsParams

{

FBuildMeshDescriptionsParams()

: bMarkPackageDirty(true)

, bUseHashAsGuid(false)

, bBuildSimpleCollision(false)

, bCommitMeshDescription(true)

, bFastBuild(false)

, bAllowCpuAccess(false)

{}

/\*\*

 \* If set to false, the caller can be from any thread but will have the

 \* responsibility to call MarkPackageDirty() from the main thread.

 \*/

bool bMarkPackageDirty;

/\*\*

 \* Uses a hash as the GUID, useful to prevent recomputing content already in cache.

 \* Set to false by default.

 \*/

bool bUseHashAsGuid;

/\*\*

 \* Builds simple collision as part of the building process. Set to false by default.

 \*/

bool bBuildSimpleCollision;

/\*\*

 \* Commits the MeshDescription as part of the building process. Set to true by default.

 \*/

bool bCommitMeshDescription;

/\*\*

 \* Specifies that the mesh will be built by the fast path (mandatory in non-editor builds).

 \* Set to false by default.

 \*/

bool bFastBuild;

/\*\*

 \* Ored with the value of bAllowCpuAccess on the static mesh. Set to false by default.

 \*/

bool bAllowCpuAccess;

/\*\*

 \* Extra optional LOD params. Overrides any previous settings from source model build settings.

 \*/

TArray<FBuildMeshDescriptionsLODParams> PerLODOverrides;

};

/\*\*

 \* Builds static mesh render buffers from a list of MeshDescriptions, one per LOD.

 \*/

ENGINE\_API bool BuildFromMeshDescriptions(const TArray<const FMeshDescription\*>& MeshDescriptions, const FBuildMeshDescriptionsParams& Params = FBuildMeshDescriptionsParams());

/\*\* Builds a LOD resource from a MeshDescription \*/

ENGINE\_API void BuildFromMeshDescription(const FMeshDescription& MeshDescription, FStaticMeshLODResources& LODResources);

/\*\*

 \* Returns the number of UV channels for the given LOD of a StaticMesh.

 \* @param LODIndex Index of the StaticMesh LOD.

 \* @return the number of UV channels.

 \*/

ENGINE\_API int32 GetNumUVChannels(int32 LODIndex);

/\*\* Pre-build navigation collision \*/

private:

UE\_DEPRECATED(5.0, "This must be protected for async build, always use the accessors even internally.")

UPROPERTY(VisibleAnywhere, transient, duplicatetransient, Instanced, Category = Navigation, *meta* = (EditCondition = "bHasNavigationData"))

TObjectPtr<UNavCollisionBase> NavCollision;

public:

ENGINE\_API void SetNavCollision(UNavCollisionBase\*);

ENGINE\_API UNavCollisionBase\* GetNavCollision() const;

ENGINE\_API bool IsNavigationRelevant() const;

/\*\*

 \* Default constructor

 \*/

ENGINE\_API UStaticMesh(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get());

//~ Begin UObject Interface.

#if WITH\_EDITOR

ENGINE\_API virtual void PreEditChange(FProperty\* PropertyAboutToChange) override;

ENGINE\_API virtual void PostEditChangeProperty(FPropertyChangedEvent& PropertyChangedEvent) override;

ENGINE\_API virtual void PostEditUndo() override;

ENGINE\_API virtual EDataValidationResult IsDataValid(class FDataValidationContext& Context) const override;

ENGINE\_API virtual void GetAssetRegistryTagMetadata(TMap<FName, FAssetRegistryTagMetadata>& OutMetadata) const override;

ENGINE\_API virtual void WillNeverCacheCookedPlatformDataAgain() override;

ENGINE\_API virtual void ClearCachedCookedPlatformData(const ITargetPlatform\* TargetPlatform) override;

ENGINE\_API virtual void ClearAllCachedCookedPlatformData() override;

ENGINE\_API virtual void BeginCacheForCookedPlatformData(const ITargetPlatform\* TargetPlatform) override;

ENGINE\_API virtual bool IsCachedCookedPlatformDataLoaded(const ITargetPlatform\* TargetPlatform) override;

ENGINE\_API void SetLODGroup(FName NewGroup, bool bRebuildImmediately = true, bool bAllowModify = true);

ENGINE\_API void BroadcastNavCollisionChange();

FOnExtendedBoundsChanged& GetOnExtendedBoundsChanged() { return OnExtendedBoundsChanged; }

FOnMeshChanged& GetOnMeshChanged() { return OnMeshChanged; }

/\*

 \* Add or change the LOD data specified by LodIndex with the content of the sourceStaticMesh.

 \*

 \* @Param SourceStaticMesh - The data we want to use to add or modify the specified lod

 \* @Param LodIndex - The lod index we want to add or modify.

 \* @Param SourceDataFilename - The source filename thta need to set into the LOD to allow re-import.

 \*/

ENGINE\_API bool SetCustomLOD(const UStaticMesh\* SourceStaticMesh, int32 LodIndex, const FString& SourceDataFilename);

//SourceModels API

ENGINE\_API FStaticMeshSourceModel& AddSourceModel();

UFUNCTION(BlueprintCallable, Category="StaticMesh")

ENGINE\_API void SetNumSourceModels(int32 Num);

ENGINE\_API void RemoveSourceModel(int32 Index);

ENGINE\_API const TArray<FStaticMeshSourceModel>& GetSourceModels() const;

ENGINE\_API FStaticMeshSourceModel& GetSourceModel(int32 Index);

ENGINE\_API const FStaticMeshSourceModel& GetSourceModel(int32 Index) const;

ENGINE\_API int32 GetNumSourceModels() const;

ENGINE\_API bool IsSourceModelValid(int32 Index) const;

ENGINE\_API TArray<FStaticMeshSourceModel>&& MoveSourceModels();

ENGINE\_API void SetSourceModels(TArray<FStaticMeshSourceModel>&& SourceModels);

ENGINE\_API FStaticMeshSourceModel& GetHiResSourceModel();

ENGINE\_API const FStaticMeshSourceModel& GetHiResSourceModel() const;

ENGINE\_API FStaticMeshSourceModel&& MoveHiResSourceModel();

ENGINE\_API void SetHiResSourceModel(FStaticMeshSourceModel&& SourceModel);

ENGINE\_API FMeshSectionInfoMap& GetSectionInfoMap();

ENGINE\_API const FMeshSectionInfoMap& GetSectionInfoMap() const;

ENGINE\_API FMeshSectionInfoMap& GetOriginalSectionInfoMap();

ENGINE\_API const FMeshSectionInfoMap& GetOriginalSectionInfoMap() const;

ENGINE\_API bool IsAsyncTaskComplete() const;

/\*\* Try to cancel any pending async tasks.

 \*  Returns true if there is no more async tasks pending, false otherwise.

 \*/

ENGINE\_API bool TryCancelAsyncTasks();

TUniquePtr<FStaticMeshAsyncBuildTask> AsyncTask;

#endif // WITH\_EDITOR

ENGINE\_API virtual void Serialize(FArchive& Ar) override;

ENGINE\_API virtual void PostInitProperties() override;

ENGINE\_API virtual void PostLoad() override;

#if WITH\_EDITORONLY\_DATA

ENGINE\_API static void DeclareConstructClasses(TArray<FTopLevelAssetPath>& OutConstructClasses, const UClass\* SpecificSubclass);

#endif

ENGINE\_API virtual bool IsPostLoadThreadSafe() const override;

ENGINE\_API virtual void BeginDestroy() override;

ENGINE\_API virtual bool IsReadyForFinishDestroy() override;

ENGINE\_API virtual void GetAssetRegistryTags(TArray<FAssetRegistryTag>& OutTags) const override;

ENGINE\_API virtual FString GetDesc() override;

ENGINE\_API virtual void GetResourceSizeEx(FResourceSizeEx& CumulativeResourceSize) override;

ENGINE\_API virtual bool CanBeClusterRoot() const override;

//~ End UObject Interface.

//~ Begin UStreamableRenderAsset Interface

ENGINE\_API virtual int32 CalcCumulativeLODSize(int32 NumLODs) const final override;

ENGINE\_API virtual FIoFilenameHash GetMipIoFilenameHash(const int32 MipIndex) const final override;

ENGINE\_API virtual bool DoesMipDataExist(const int32 MipIndex) const final override;

ENGINE\_API virtual bool HasPendingRenderResourceInitialization() const final override;

ENGINE\_API virtual bool StreamOut(int32 NewMipCount) final override;

ENGINE\_API virtual bool StreamIn(int32 NewMipCount, bool bHighPrio) final override;

ENGINE\_API virtual EStreamableRenderAssetType GetRenderAssetType() const final override;

//~ End UStreamableRenderAsset Interface

/\*\*

\* Cancels any pending static mesh streaming actions if possible.

\* Returns when no more async loading requests are in flight.

\*/

ENGINE\_API static void CancelAllPendingStreamingActions();

/\*\*

 \* Contains all the parameters required to build the mesh.

 \*/

struct FBuildParameters

{

// Required to work around a Clang bug

#if PLATFORM\_COMPILER\_CLANG

FBuildParameters()

: bInSilent(false)

, OutErrors(nullptr)

, bInRebuildUVChannelData(false)

, bInEnforceLightmapRestrictions(false)

{

}

#endif

// If true will not popup a progress dialog.

bool bInSilent = false;

// If provided, will contain the errors that occurred during this process.This will prevent async static mesh compilation because OutErrors could get out of scope.

TArray<FText>\* OutErrors = nullptr;

// Whether to completely rebuild the UV Channel data after the render data has been computed.

bool bInRebuildUVChannelData = false;

// Whether to call EnforceLightmapRestrictions as part of the build process.

bool bInEnforceLightmapRestrictions = false;

};

/\*\*

 \* Rebuilds renderable data for this static mesh, automatically made async if enabled.

 \* @param bInSilent If true will not popup a progress dialog.

 \* @param [out] OutErrors If provided, will contain the errors that occurred during this process. This will prevent async static mesh compilation because OutErrors could get out of scope.

 \*/

ENGINE\_API void Build(bool bInSilent, TArray<FText>\* OutErrors = nullptr);

/\*\*

 \* Rebuilds renderable data for a batch of static meshes.

 \* @param InStaticMeshes The list of all static meshes to build.

 \* @param bInSilent If true will not popup a progress dialog.

 \* @param InProgressCallback If provided, will be used to abort task and report progress to higher level functions (should return true to continue, false to abort).

 \* @param [out] OutErrors If provided, will contain the errors that occurred during this process. This will prevent async static mesh compilation because OutErrors could get out of scope.

 \*/

ENGINE\_API static void BatchBuild(const TArray<UStaticMesh\*>& InStaticMeshes, bool bInSilent, TFunction<bool(UStaticMesh\*)> InProgressCallback = nullptr, TArray<FText>\* OutErrors = nullptr);

/\*\*

 \* Rebuilds renderable data for this static mesh, automatically made async if enabled.

 \* @param BuildParameters   Contains all the information required to build the mesh.

 \*/

ENGINE\_API void Build(const FBuildParameters& BuildParameters = FBuildParameters());

/\*\*

 \* Rebuilds renderable data for a batch of static meshes.

 \* @param InStaticMeshes The list of all static meshes to build.

 \* @param BuildParameters     Contains all the parameters required to build the mesh.

 \* @param InProgressCallback If provided, will be used to abort task and report progress to higher level functions (should return true to continue, false to abort).

 \*/

ENGINE\_API static void BatchBuild(const TArray<UStaticMesh\*>& InStaticMeshes, const FBuildParameters& BuildParameters = FBuildParameters(), TFunction<bool(UStaticMesh\*)> InProgressCallback = nullptr);

/\*\*

 \* Initialize the static mesh's render resources.

 \*/

ENGINE\_API virtual void InitResources();

/\*\*

 \* Releases the static mesh's render resources.

 \*/

ENGINE\_API virtual void ReleaseResources();

/\*\*

 \* Update missing material UV channel data used for texture streaming.

 \*

 \* @param bRebuildAll If true, rebuild everything and not only missing data.

 \*/

ENGINE\_API void UpdateUVChannelData(bool bRebuildAll);

/\*\*

 \* Returns the material bounding box. Computed from all lod-section using the material index.

 \*

 \* @param MaterialIndex Material Index to look at

 \* @param TransformMatrix Matrix to be applied to the position before computing the bounds

 \*

 \* @return false if some parameters are invalid

 \*/

ENGINE\_API FBox GetMaterialBox(int32 MaterialIndex, const FTransform& Transform) const;

/\*\*

 \* Returns the UV channel data for a given material index. Used by the texture streamer.

 \* This data applies to all lod-section using the same material.

 \*

 \* @param MaterialIndex the material index for which to get the data for.

 \* @return the data, or null if none exists.

 \*/

ENGINE\_API const FMeshUVChannelInfo\* GetUVChannelData(int32 MaterialIndex) const;

/\*\*

 \* Returns the number of vertices for the specified LOD.

 \*/

ENGINE\_API int32 GetNumVertices(int32 LODIndex) const;

/\*\*

 \* Returns the number of triangles in the render data for the specified LOD.

 \*/

UFUNCTION(BlueprintPure, Category = StaticMesh)

ENGINE\_API int32 GetNumTriangles(int32 LODIndex) const;

/\*\*

 \* Returns the number of tex coords for the specified LOD.

 \*/

ENGINE\_API int32 GetNumTexCoords(int32 LODIndex) const;

/\*\*

 \* Returns the number of vertices of the Nanite representation of this mesh.

 \*/

ENGINE\_API int32 GetNumNaniteVertices() const;

/\*\*

 \* Returns the number of triangles of the Nanite representation of this mesh.

 \*/

ENGINE\_API int32 GetNumNaniteTriangles() const;

/\*\*

 \* Returns the number of LODs used by the mesh.

 \*/

UFUNCTION(BlueprintPure, Category = "StaticMesh", *meta*=(ScriptName="GetNumLods"))

ENGINE\_API int32 GetNumLODs() const;

/\*\*

 \* Returns true if the mesh has data that can be rendered.

 \*/

ENGINE\_API bool HasValidRenderData(bool bCheckLODForVerts = true, int32 LODIndex = INDEX\_NONE) const;

/\*\*

 \* Returns true if the mesh has valid Nanite render data.

 \*/

ENGINE\_API bool HasValidNaniteData() const;

/\*\*

 \* Returns the number of bounds of the mesh.

 \*

 \* @return The bounding box represented as box origin with extents and also a sphere that encapsulates that box

 \*/

UFUNCTION( BlueprintPure, Category="StaticMesh" )

ENGINE\_API FBoxSphereBounds GetBounds() const;

/\*\* Returns the bounding box, in local space including bounds extension(s), of the StaticMesh asset \*/

UFUNCTION(BlueprintPure, Category="StaticMesh")

ENGINE\_API FBox GetBoundingBox() const;

/\*\* Returns number of Sections that this StaticMesh has, in the supplied LOD (LOD 0 is the highest) \*/

UFUNCTION(BlueprintPure, Category = "StaticMesh")

ENGINE\_API int32 GetNumSections(int32 InLOD) const;

/\*\*

 \* Gets a Material given a Material Index and an LOD number

 \*

 \* @return Requested material

 \*/

UFUNCTION(BlueprintPure, Category = "StaticMesh")

ENGINE\_API UMaterialInterface\* GetMaterial(int32 MaterialIndex) const;

/\*\*

 \* Adds a new material and return its slot name

 \*/

UFUNCTION(BlueprintCallable, Category = "StaticMesh")

ENGINE\_API FName AddMaterial(UMaterialInterface\* Material);

/\*\*

 \* Gets a Material index given a slot name

 \*

 \* @return Requested material

 \*/

UFUNCTION(BlueprintPure, Category = "StaticMesh")

ENGINE\_API int32 GetMaterialIndex(FName MaterialSlotName) const;

ENGINE\_API int32 GetMaterialIndexFromImportedMaterialSlotName(FName ImportedMaterialSlotName) const;

/\*\*

 \* Returns the render data to use for exporting the specified LOD. This method should always

 \* be called when exporting a static mesh.

 \*/

ENGINE\_API const FStaticMeshLODResources& GetLODForExport(int32 LODIndex) const;

/\*\*

 \* Static: Processes the specified static mesh for light map UV problems

 \*

 \* @param InStaticMesh Static mesh to process

 \* @param InOutAssetsWithMissingUVSets Array of assets that we found with missing UV sets

 \* @param InOutAssetsWithBadUVSets Array of assets that we found with bad UV sets

 \* @param InOutAssetsWithValidUVSets Array of assets that we found with valid UV sets

 \* @param bInVerbose If true, log the items as they are found

 \*/

ENGINE\_API static void CheckLightMapUVs( UStaticMesh\* InStaticMesh, TArray< FString >& InOutAssetsWithMissingUVSets, TArray< FString >& InOutAssetsWithBadUVSets, TArray< FString >& InOutAssetsWithValidUVSets, bool bInVerbose = true );

//~ Begin Interface\_CollisionDataProvider Interface

ENGINE\_API virtual bool GetPhysicsTriMeshData(struct FTriMeshCollisionData\* CollisionData, bool InUseAllTriData) override;

ENGINE\_API virtual bool ContainsPhysicsTriMeshData(bool InUseAllTriData) const override;

ENGINE\_API virtual bool PollAsyncPhysicsTriMeshData(bool InUseAllTriData) const override;

ENGINE\_API virtual bool GetTriMeshSizeEstimates(struct FTriMeshCollisionDataEstimates& OutTriMeshEstimates, bool bInUseAllTriData) const override;

private:

bool GetPhysicsTriMeshDataCheckComplex(struct FTriMeshCollisionData\* CollisionData, bool bInUseAllTriData, bool bInCheckComplexCollisionMesh);

bool ContainsPhysicsTriMeshDataCheckComplex(bool InUseAllTriData, bool bInCheckComplexCollisionMesh) const;

public:

virtual bool WantsNegXTriMesh() override

{

return true;

}

ENGINE\_API virtual void GetMeshId(FString& OutMeshId) override;

//~ End Interface\_CollisionDataProvider Interface

/\*\* Return the number of sections of the StaticMesh with collision enabled \*/

int32 GetNumSectionsWithCollision() const;

//~ Begin IInterface\_AssetUserData Interface

ENGINE\_API virtual void AddAssetUserData(UAssetUserData\* InUserData) override;

ENGINE\_API virtual void RemoveUserDataOfClass(TSubclassOf<UAssetUserData> InUserDataClass) override;

ENGINE\_API virtual UAssetUserData\* GetAssetUserDataOfClass(TSubclassOf<UAssetUserData> InUserDataClass) override;

ENGINE\_API virtual const TArray<UAssetUserData\*>\* GetAssetUserDataArray() const override;

//~ End IInterface\_AssetUserData Interface

/\*\*

 \* Create BodySetup for this staticmesh if it doesn't have one

 \*/

ENGINE\_API void CreateBodySetup();

/\*\*

 \* Calculates navigation collision for caching

 \*/

ENGINE\_API void CreateNavCollision(const bool bIsUpdate = false);

/\*\*

 \* Delete current NavCollision and create a new one if needed

 \*/

ENGINE\_API void RecreateNavCollision();

/\*\* Configures this SM as bHasNavigationData = false and clears stored NavCollision \*/

ENGINE\_API void MarkAsNotHavingNavigationData();

/\*\*

 \* Add a socket object in this StaticMesh.

 \*/

UFUNCTION(BlueprintCallable, Category = "StaticMesh")

ENGINE\_API void AddSocket(UStaticMeshSocket\* Socket);

/\*\*

 \* Find a socket object in this StaticMesh by name.

 \* Entering NAME\_None will return NULL. If there are multiple sockets with the same name, will return the first one.

 \*/

UFUNCTION(BlueprintPure, Category = "StaticMesh")

ENGINE\_API class UStaticMeshSocket\* FindSocket(FName InSocketName) const;

/\*\*

 \* Remove a socket object in this StaticMesh by providing it's pointer. Use FindSocket() if needed.

 \*/

UFUNCTION(BlueprintCallable, Category = "StaticMesh")

ENGINE\_API void RemoveSocket(UStaticMeshSocket\* Socket);

/\*\*

 \* Returns a list of sockets with the provided tag.

 \*/

UFUNCTION(BlueprintPure, Category = "StaticMesh")

ENGINE\_API TArray<UStaticMeshSocket\*> GetSocketsByTag(const FString& InSocketTag) const;

/\*\*

 \* Returns vertex color data by position.

 \* For matching to reimported meshes that may have changed or copying vertex paint data from mesh to mesh.

 \*

 \* @param VertexColorData (out)A map of vertex position data and its color. The method fills this map.

 \*/

ENGINE\_API void GetVertexColorData(TMap<FVector3f, FColor>& VertexColorData);

/\*\*

 \* Sets vertex color data by position.

 \* Map of vertex color data by position is matched to the vertex position in the mesh

 \* and nearest matching vertex color is used.

 \*

 \* @param VertexColorData A map of vertex position data and color.

 \*/

ENGINE\_API void SetVertexColorData(const TMap<FVector3f, FColor>& VertexColorData);

/\*\* Removes all vertex colors from this mesh and rebuilds it (Editor only \*/

ENGINE\_API void RemoveVertexColors();

/\*\* Make sure the Lightmap UV point on a valid UVChannel \*/

ENGINE\_API void EnforceLightmapRestrictions(bool bUseRenderData = true);

/\*\* Calculates the extended bounds \*/

ENGINE\_API void CalculateExtendedBounds();

inline bool AreRenderingResourcesInitialized() const { return bRenderingResourcesInitialized; }

/\*\* Helper function for resource tracking, construct a name using the mesh's path name and LOD index . \*/

static FName GetLODPathName(const UStaticMesh\* Mesh, int32 LODIndex);

#if WITH\_EDITOR

/\*\*

 \* Sets a Material given a Material Index

 \*/

UFUNCTION(BlueprintCallable, Category = "StaticMesh")

ENGINE\_API void SetMaterial(int32 MaterialIndex, UMaterialInterface\* NewMaterial);

/\*\*

 \* Returns true if LODs of this static mesh may share texture lightmaps.

 \*/

ENGINE\_API bool CanLODsShareStaticLighting() const;

/\*\*

 \* Retrieves the names of all LOD groups.

 \*/

ENGINE\_API static void GetLODGroups(TArray<FName>& OutLODGroups);

/\*\*

 \* Retrieves the localized display names of all LOD groups.

 \*/

ENGINE\_API static void GetLODGroupsDisplayNames(TArray<FText>& OutLODGroupsDisplayNames);

ENGINE\_API void GenerateLodsInPackage();

ENGINE\_API virtual void PostDuplicate(bool bDuplicateForPIE) override;

/\*\* Get multicast delegate broadcast prior to mesh building \*/

FOnPreMeshBuild& OnPreMeshBuild() { return PreMeshBuild; }

/\*\* Get multicast delegate broadcast after mesh building \*/

FOnPostMeshBuild& OnPostMeshBuild() { return PostMeshBuild; }

/\* Return true if the reduction settings are setup to reduce a LOD\*/

ENGINE\_API bool IsReductionActive(int32 LODIndex) const;

/\* Get a copy of the reduction settings for a specified LOD index. \*/

ENGINE\_API struct FMeshReductionSettings GetReductionSettings(int32 LODIndex) const;

/\*\* Get whether this mesh should use LOD streaming for the given platform. \*/

bool GetEnableLODStreaming(const class ITargetPlatform\* TargetPlatform) const;

/\* Get a static mesh render data for requested platform. \*/

static FStaticMeshRenderData& GetPlatformStaticMeshRenderData(UStaticMesh\* Mesh, const ITargetPlatform\* Platform);

private:

/\*\*

 \* Converts legacy LODDistance in the source models to Display Factor

 \*/

void ConvertLegacyLODDistance();

/\*\*

 \* Converts legacy LOD screen area in the source models to resolution-independent screen size

 \*/

void ConvertLegacyLODScreenArea();

/\*\*

 \* Fixes up static meshes that were imported with sections that had zero triangles.

 \*/

void FixupZeroTriangleSections();

/\*\*

\* Converts legacy RawMesh to MeshDescription.

\*/

void ConvertLegacySourceData();

/\*\*

 \* Verify if the static mesh can be built.

 \*/

bool CanBuild() const;

/\*\*

 \* Initial step for the static mesh building process - Can't be done in parallel.

 \*/

void BeginBuildInternal(FStaticMeshBuildContext\* Context = nullptr);

/\*\*

 \* Build the static mesh

 \*/

bool ExecuteBuildInternal(const FBuildParameters& BuildParameters);

/\*\*

 \* Complete the static mesh building process - Can't be done in parallel.

 \*/

void FinishBuildInternal(const TArray<UStaticMeshComponent\*>& InAffectedComponents, bool bHasRenderDataChanged, bool bShouldComputeExtendedBounds = true);

/\*\*

 \* Get an estimate of the peak amount of memory required to build this mesh.

 \*/

int64 GetBuildRequiredMemory() const;

#if WITH\_EDITORONLY\_DATA

/\*\*

 \* Deserialize MeshDescription for the specified LodIndex from BulkData, DDC or RawMesh.

 \*/

bool LoadMeshDescription(int32 LodIndex, FMeshDescription& OutMeshDescription) const;

#endif

public:

/\*\*

 \* Caches derived renderable data.

 \*/

ENGINE\_API void CacheDerivedData();

/\*\*

 \* Caches derived renderable for cooked platforms currently active.

 \*/

ENGINE\_API void PrepareDerivedDataForActiveTargetPlatforms();

private:

// Filled at CommitDescription time and reused during build

TOptional<FBoxSphereBounds> CachedMeshDescriptionBounds;

// Notification about missing Nanite required shader models.

TWeakPtr<class SNotificationItem> ShaderModelNotificationPtr;

void CheckForMissingShaderModels();

FOnPreMeshBuild PreMeshBuild;

FOnPostMeshBuild PostMeshBuild;

/\*\*

 \* Fixes up the material when it was converted to the new staticmesh build process

 \*/

bool bCleanUpRedundantMaterialPostLoad;

/\*\*

 \* Guard to ignore re-entrant PostEditChange calls.

 \*/

bool bIsInPostEditChange = false;

#endif // #if WITH\_EDITOR

/\*\*

 \* Initial step for the Post Load process - Can't be done in parallel.

 \*/

void BeginPostLoadInternal(FStaticMeshPostLoadContext& Context);

/\*\*

 \* Thread-safe part of the Post Load

 \*/

void ExecutePostLoadInternal(FStaticMeshPostLoadContext& Context);

/\*\*

 \* Complete the static mesh postload process - Can't be done in parallel.

 \*/

void FinishPostLoadInternal(FStaticMeshPostLoadContext& Context);

};

class FStaticMeshCompilationContext

{

public:

FStaticMeshCompilationContext();

// Non-copyable

FStaticMeshCompilationContext(const FStaticMeshCompilationContext&) = delete;

FStaticMeshCompilationContext& operator=(const FStaticMeshCompilationContext&) = delete;

// Movable

FStaticMeshCompilationContext(FStaticMeshCompilationContext&&) = default;

FStaticMeshCompilationContext& operator=(FStaticMeshCompilationContext&&) = default;

bool bShouldComputeExtendedBounds = false;

bool bIsEditorLoadingPackage = false;

};

class FStaticMeshPostLoadContext : public FStaticMeshCompilationContext

{

public:

bool bNeedsMeshUVDensityFix = false;

bool bNeedsMaterialFixup = false;

bool bIsCookedForEditor = false;

};

class FStaticMeshBuildContext : public FStaticMeshCompilationContext

{

public:

FStaticMeshBuildContext(const UStaticMesh::FBuildParameters& InBuildParameters)

: BuildParameters(InBuildParameters)

{

}

UStaticMesh::FBuildParameters BuildParameters;

bool bHasRenderDataChanged = false;

};

namespace UE::Private::StaticMesh

{

#if WITH\_EDITOR

ENGINE\_API FString BuildStaticMeshDerivedDataKey(const ITargetPlatform\* TargetPlatform, UStaticMesh\* Mesh, const FStaticMeshLODGroup& LODGroup);

#endif

}