Large World Coordinates

An Overview to Large World Coordinates and how it is used in Unreal Engine 5.

Learn to use this **Beta** feature, but use caution when shipping with it.

Large World Coordinates (**LWC**) introduces support for double-precision data variant types in **Unreal Engine 5**(**UE5**) where extensive changes are being implemented across all engine systems to improve floating-point precision. These systems include **Architectural Visualization**, **Simulation**, **Rendering** (**Niagara** and **HLSL** code), and projects with massive world scale.

In **Unreal Engine 4**(**UE4**), 32-bit float precision types would restrict the size of the world. LWC vastly improves the size of your projects by providing a 64-bit double to your core data types. These new changes will enable you to build massive worlds and greatly improve Actor placement accuracy and orientation precision. Large World Coordinates are ready for you to use when you begin a new project in UE5.

Upgrading your Project to Unreal Engine 5

When upgrading your UE4 project to an UE5, We generally recommend you refer to the Migration Guide, as you may experience a few edge cases where your code base experiences precision loss. This is not as much of a cause of concern for projects that are not using Large

World Coordinates. However, for projects that plan to use the double type to increase the scale of their worlds, we recommend you use the Large World Coordinates Conversion Guidelines.

Experimenting with Large Worlds

The default WORLD_MAX size is 88 million kilometers and the world bounds check remains enabled.

If you want to use the UE4 WORLD_MAX size of 21 kilometers, you can set the global value `UE_USE_UE4_WORLD_MAX` located in the `EngineDefines.h`, then recompile the engine.

This value may change before future releases of Unreal Engine and may exhibit stability issues that will continually be optimized throughout the development of Unreal Engine 5.

Blueprints

In Blueprints, floats now display with the appropriate single or double precision subtype. This new type supports feature parity with both float and double. All existing Blueprints and Blueprint types (**UMG**, **Control Rig**, **Animation Blueprints**) have been implicitly converted to use either precision type without any need for you to update your previous work.

Source Code Interface

Source code may now expose both float and double types. The **Unreal Header Tool**(**UHT**) will interpret any Blueprint-accessible floating-point type in code as a Blueprint Float with the appropriate single (C++ float) or double (C++ double) precision subtype, enabling automatic conversion of Float values of either precision supplied by any Blueprint node.

Exposing UFUNCTION Property Specifiers Expecting Float Values

Any method marked with a UFUNCTION Property Specifier that contains a float data value risks introducing imprecision, because the Blueprint Float value is cast to the lower precision float. It is important to audit any existing UFUNCTION property. This helps to determine if switching the parameters or return values to double may be necessary to avoid precision issues in the future. Switching between float and double types is safe to do at any time, and in either direction.

This applies to any K2 nodes that you may have constructed or exposed in code.

Rendering

In Unreal Engine you have various coordinate spaces and transformations that describe how an object can exist in the world. **World space**(coordinates of your level / world) and **Local space**(relative to a specific object), for additional information refer to the Coordinate Space Terminology documentation.

Each object that can be placed in your project's world has a three coordinate axis, an orientation, and an origin. Refer to Transforming Actors for additional information.

Shaders

New HLSL types have been introduced along with Large World Coordinates and can be found in the `LargeWorldCoordinates.ush` file. Refer to the LWC Rendering Doc for additional information on how to convert your shader code to UE5.

Niagara

In Niagara, the implementation is different from the main engine. To keep particle effects performant, the data is stored as a set of floats instead of using doubles. A sufficiently large world will be divided into grid cells. The first float then defines the Niagara system in that grid cell, and the second represents which grid cell the system is in.

To accommodate this additional information, there is a new data type called Position for storing the vectors referenced by Particles.Position. Refer to the Large World Coordinates in Niagara document for additional information.

Chaos

Chaos Destruction Physics continues to function unabridged using the double type. The engine implicitly casts in both directions, for example:

```
FVector3f -> Chaos::FVec3!

However, the only explicit cast is from:

Chaos::FVec3 -> FVector3f
//This would catch precision loss from downgrading to a float

```

Copy full snippet

With further release versions of Unreal Engine 5, the FVector casts will continue to implicitly upgrade your floats to double. If you want to cast your double to a float, then you will need to explicitly perform a conversion. Refer to the Large World Coordinates Conversion Guidelines for additional information.

Large World Coordinates Project Conversion Guidelines

This document serves as a guide to convert your UE4 Projects to UE5 with minimal precision loss.

Large World Coordinates Rendering Overview.

An overview of Rendering Large World Coordinates.