Incremental Garbage Collection

Improved garbage collection system for UObjects.

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

Unreal Engine (UE) uses a mark-and-sweep garbage collector to manage UObject memory. For soft-real-time applications, garbage collectors have historically had one major drawback: the potential to generate gameplay hitches while the garbage collector determines which objects' memory can be reclaimed. In UE, this process is called *reachability analysis*. UE has always relied on this phase of garbage collection to complete within a single frame, which temporarily stops all UObject processing, in particular, gameplay. The more objects reachability analysis has to scan, the longer the pause, and it usually doesn't take much to introduce visible gameplay hitches as a result. There are several ways programmers can circumvent this, such as:

- keeping tight UObject budgets
- using UObject pools
- disabling garbage collection during normal gameplay

However, these workarounds tend to increase code complexity and overall project costs.

Incremental Reachability Analysis

UE improves upon this using *incremental reachability analysis*. Users now have the ability to split the garbage collector's reachability analysis phase across multiple frames with a configurable per-frame, soft time limit. The engine tracks UObject references between reachability iterations through `TObjectPtr` properties. That is, any assignment to a `TObjectPtr`-exposed `UPROPERTY` immediately marks the object as reachable while garbage collection is in progress. This is also known as a garbage collector *write barrier*.

The engine has already been converted to use `TObjectPtr` instead of raw C++ pointers in any place that exposes UObjects to the garbage collector, including any UObject or `FGCObject` `AddReferencedObjects` functions. To use incremental reachability analysis in projects built with Unreal Engine, it's critical to convert all `UPROPERTY` instances to use `TObjectPtr` instead of raw C++ otherwise garbage collection might reclaim some of UObjects' memory too early. We are initially releasing this feature as experimental, as it's still possible that the reachability analysis phase can exceed the specified time limit.

Enable Incremental Reachability Analysis

Incremental reachability analysis can be enabled with the following console variables when added to your project's `DefaultEngine.ini`:

```
[ConsoleVariables]
gc.AllowIncrementalReachability=1 ; enables Incremental Reachability Analysis
gc.AllowIncrementalGather=1 ; enables Incremental Gather Unreachable Objects
gc.IncrementalReachabilityTimeLimit=0.002 ; sets the soft time limit to 2ms
```

Copy full snippet

Additional Console Variables

We also provide an additional set of console variables for stress-testing and debugging purposes:

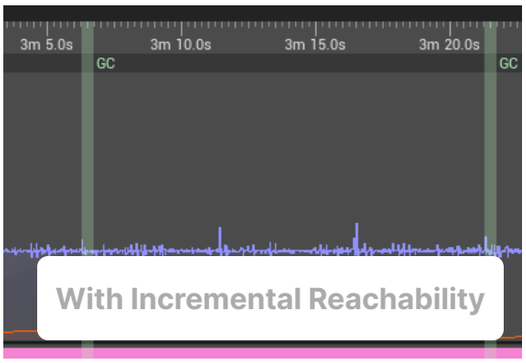
Console Variable	Description	Type
`gc.DelayReachabilityIteration`	Delay reachability analysis by the specified number of frames. Used for stress-testing the GC barrier.	`<INTEGER>`: Number of Frames (default: 10)
`gc.VerifyNoUnreachableObjects`	Run a test after reachability analysis is complete to make sure no reachable (valid) object is referencing an unreachable (soon to be destroyed) object.	0: disabled, 1: enabled
`gc.ContinuousIncrementalGC`	Keep restarting incremental garbage collection after the previous one has completed.	0: disabled, 1: enabled

Performance Comparison

Below is an Unreal Insights visualization of a sample project's performance graph with incremental reachability turned off. The blue line represents the total frame time and the

orange line shows the time taken up by reachability analysis. Unreal Insights plots a continuous graph line between single events separated by multiple frames; even though it may seem like garbage collection is running throughout the entire timeline, it only actually runs for a single frame at a time.

In the first image of the following graph comparison, we can clearly see a spike each time garbage collection runs (denoted by the "GC" labels at the top of the timeline).



In the second image, with incremental reachability turned on, we can see that the GC lag spikes are gone, and that incremental reachability is now split across multiple frames (represented by the now wider light green bars).