Prioritization

Prioritize objects for replication with Iris.

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

PREREQUISITE TOPICS

In order to understand and use the content on this page, make sure you are familiar with the following topics:

- Introduction to Iris

The **Iris Prioritization System** prioritizes actors for replication. Not every replicated object or property is updated each frame due to limited bandwidth. To determine what objects are most important to replicate each frame, the prioritization system ranks objects and actors by assigning them a floating point *replication priority*. The higher the priority relative to other objects, the more likely it is for an object to be replicated. Different priority ranges have different impacts on whether an object is considered for replication:

- 0.0 up to, but not including 1.0: Object is not considered for replication this frame.

- 1.0 and above: Object is considered for replication this frame if it has updated, replicated properties.

Replication priority accumulates across network ticks until an object is replicated and its priority is reset. This means that even low priority objects eventually reach a high enough priority to be considered for replication. Prioritization is accomplished with batches of objects to encourage minimal code and data cache misses.

Iris provides two types of prioritization:

- Static
- Dynamic

Static Prioritizers

Static prioritizers assign a constant priority number to an object or actor. To assign a static priority to an object, follow these steps:

1. Include the necessary Iris files in order to access required Iris functionality.

```
#if UE_WITH_IRIS
#include "Net/Iris/ReplicationSystem/ReplicationSystemUtil.h"
#include "Iris/ReplicationSystem/ReplicationSystem.h"
#include "Net/Iris/ReplicationSystem/ActorReplicationBridge.h"
#endif UE_WITH_IRIS
```

Copy full snippet

2. In your gameplay code, retrieve the replication system and replication bridge for your replicated object.

```
// The actor for which you want to control filtering
AActor* RepActorPtr;

UReplicationSystem* ReplicationSystem =
UE::Net::FReplicationSystemUtil::GetReplicationSystem(RepActorPtr);
UActorReplicationBridge* ReplicationBridge =
UE::Net::FReplicationSystemUtil::GetActorReplicationBridge(RepActorPtr);
```

Copy full snippet

3. Retrieve the `FNetRefHandle` for your replicated object. Iris uses this identifier to locate your object within the replication system.

```
	UE::Net::FNetRefHandle RepActorNetRefHandle = ReplicationBridge->Ge
```

Copy full snippet

4. Set your object to use a static priority.

```
	ReplicationSystem->SetStaticPriority(RepActorNetRefHandle, 1.0f);
```

Copy full snippet

Dynamic Prioritizers

Dynamic prioritizers assign a variable priority number to an object or actor depending on the logic defined by the prioritizer you choose. Unreal Engine (UE) provides some dynamic prioritizers to help you:

- `SphereNetObjectPrioritizer`
- `SphereWithOwnerBoostNetObjectPrioritizer`
- `NetObjectCountLimiter`

You can either choose a built-in dynamic prioritizer or create your own. For more information on how to create your own dynamic prioritizer, see Create a Custom Dynamic Prioritizer.

Specify a Dynamic Prioritizer

To assign a dynamic prioritizer for a replicated object, follow these steps:

1. Include the necessary Iris files and retrieve references to the replication system and replication bridge.

```
#if UE_WITH_IRIS
#include "Net/Iris/ReplicationSystem/ReplicationSystemUtil.h"
#include "Iris/ReplicationSystem/ReplicationSystem.h"
#include "Net/Iris/ReplicationSystem/ActorReplicationBridge.h"
```

```
#endif UE_WITH_IRIS

// The actor for which you want to control filtering
AActor* RepActorPtr;

UReplicationSystem* ReplicationSystem =
UE::Net::FReplicationSystemUtil::GetReplicationSystem(RepActorPtr);
UActorReplicationBridge* ReplicationBridge =
UE::Net::FReplicationSystemUtil::GetActorReplicationBridge(RepActorPtr);
```

Copy full snippet

2. Retrieve the `FNetRefHandle` for your replicated object. Iris uses this identifier to locate your object within the replication system.

```
    UE::Net::FNetRefHandle RepActorNetRefHandle = ReplicationBridge->Ge
```

Copy full snippet

3. Retrieve the handle for the dynamic prioritizer you want to use.

```
// To use a custom prioritizer, replace the SphereNetObjectPrioritizer
with the name of your custom prioritizer
FNetObjectPrioritizerHandle PrioritizerHandle = ReplicationSystem-
>GetPrioritizerHandle(FName("SphereNetObjectPrioritizer"));
```

Copy full snippet

If you intend to use a custom dynamic prioritizer, ensure that your prioritizer is configured in your `DefaultEngine.ini` file. For more information, see Configure a Custom Dynamic Prioritizer section.

4. Assign your object to use the dynamic prioritizer.

```
if (!ReplicationSystem->SetPrioritizer(RepActorNetRefHandle,
PrioritizerHandle))
{
UE_LOG(LogTemp, Warning, TEXT("Failed to assign dynamic prioritizer."));

// Dynamic prioritizer assignment failed, assign a fall-back static priority
ReplicationSystem->SetStaticPriority(ObjectHandle, 1.0f);
}
```

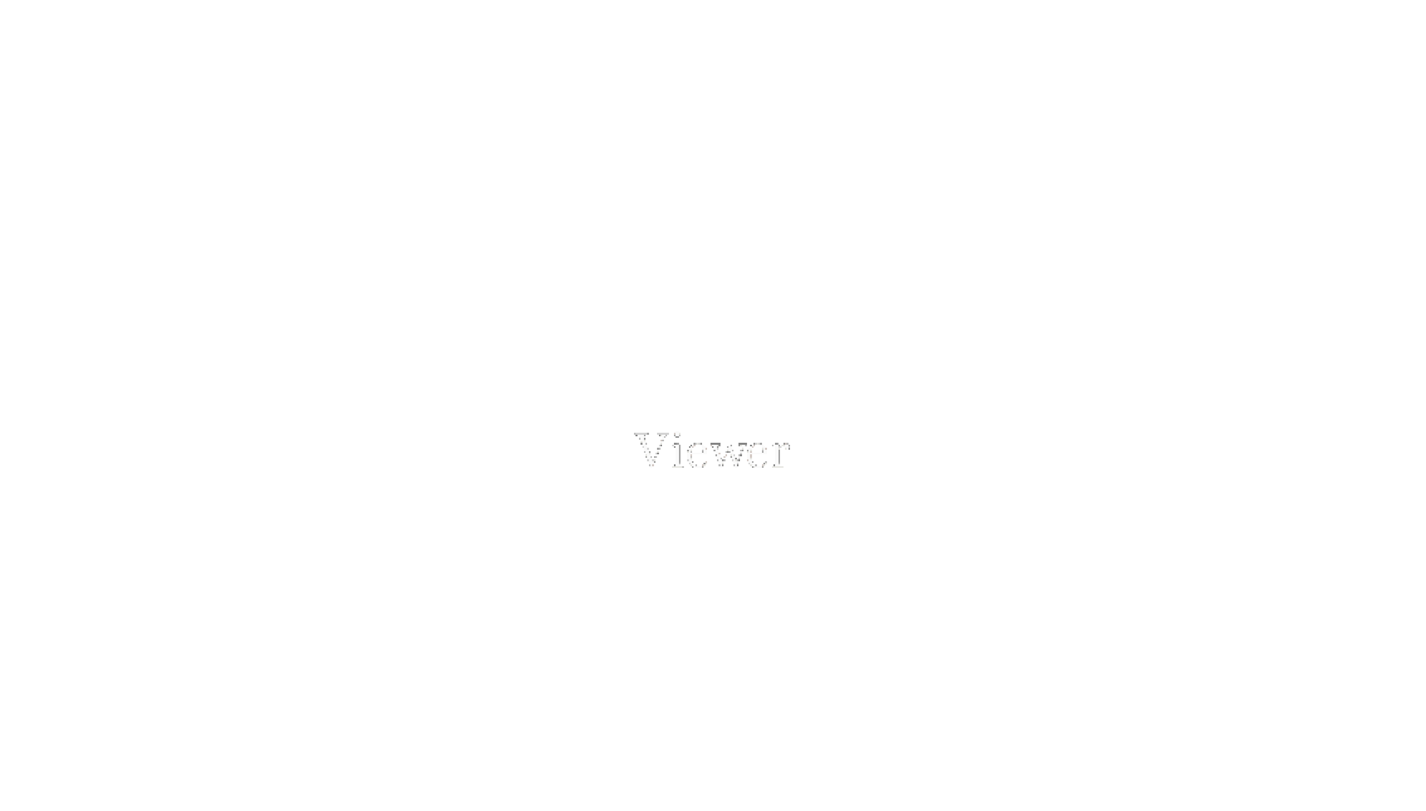
Copy full snippet

Provided Dynamic Prioritizers Reference

Sphere Prioritizer

This dynamic prioritizer is used by default for all actors with a replicated world location. You can visualize this prioritizer as two concentric spheres with finite radius.

- Objects inside the inner sphere have highest priority.
- Objects outside the inner sphere but inside the outer sphere have a dynamic priority based on where they are located.
- Objects outside the outer sphere have the lowest priority.



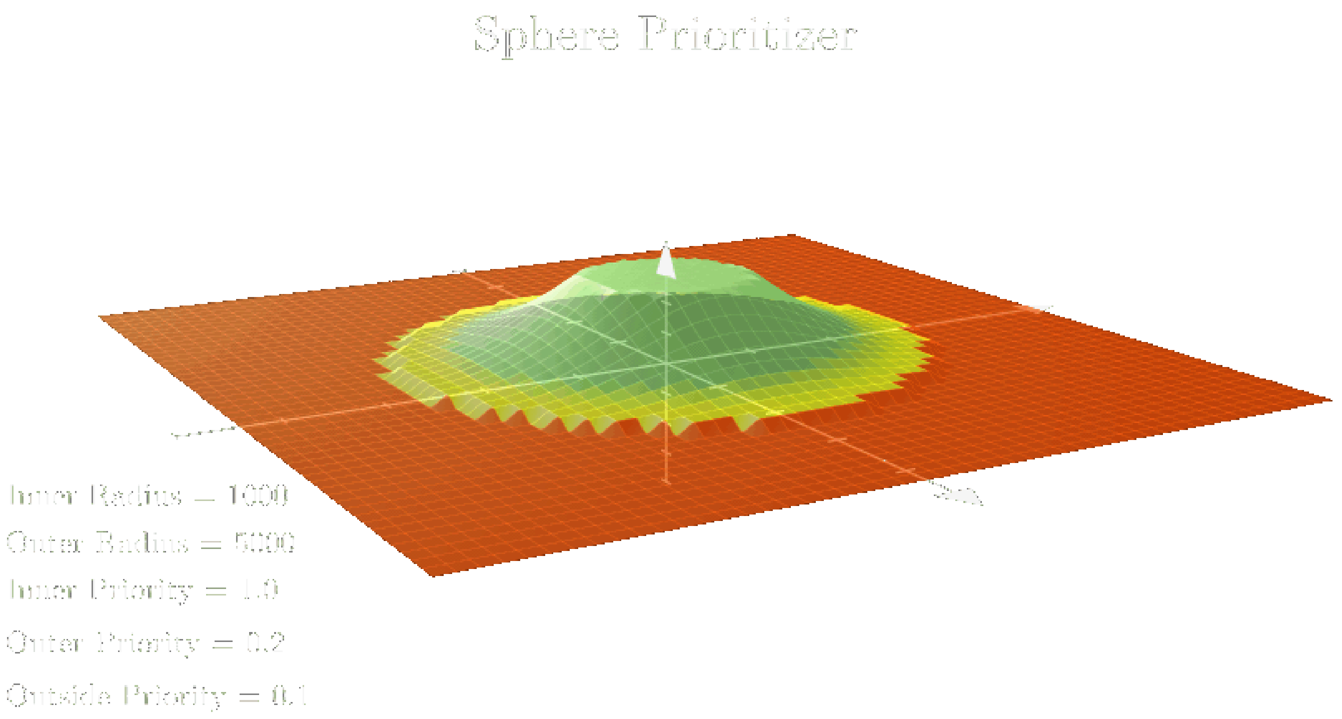
Visualization of the sphere prioritizer for objects in space relative to the viewer.

The following provides more detail into how the sphere prioritizer works. The viewer (player) is represented at the center. Objects are prioritized depending on where they are located in relation to the viewer. The priority of an object located in the inner sphere is the inner priority. The priority of an object located outside the outer sphere has the outside priority. Objects located between the inner sphere and the outer sphere have a variable priority that falls off with the squared distance between the object and the viewer.

The colors used in these visualizations represent the following:

- Green: Inner Priority
- Yellow: Outer Priority
- Red: Outside Priority

The following gradient surface plot shows the priority of an object as a function of its location in relation to the viewer, located at the origin:



Priority of object as a function of its location relative to the viewer in the same XY-plane. Red corresponds to the outside priority, yellow corresponds to outer priority, and green corresponds to inner priority.

Configure Sphere Prioritizer

You can configure the `SphereNetObjectPrioritizer` in an engine configuration file such as `DefaultEngine.ini`:

```
[Script/Iris.SphereNetObjectPrioritizerConfig]
InnerRadius=<INNER_RADIUS>
OuterRadius=<OUTER_RADIUS>
InnerPriority=<INNER_PRIORITY>
OuterPriority=<OUTER_PRIORITY>
OutsidePriority=<OUTSIDE_PRIORITY>
```

Copy full snippet

For example, the default configuration settings are:

```
[Script/Iris.SphereNetObjectPrioritizerConfig]
InnerRadius=1000.0
OuterRadius=5000.0
InnerPriority=1.0
OuterPriority=0.2
OutsidePriority=0.1
```

Copy full snippet

Sphere With Owner Boost Prioritizer

The `SphereWithOwnerBoostNetObjectPrioritizer` is the same as the Sphere Prioritizer, but provides a priority boost for the owning connection. In these cases, the prioritizer adds the value of `OwnerPriorityBoost` to the priority that is normally calculated by the sphere prioritizer.

Configure Sphere With Owner Boost Prioritizer

You can configure the `SphereWithOwnerBoostNetObjectPrioritizer` in an engine configuration file such as `DefaultEngine.ini`:

```
[Script/Iris.SphereWithOwnerBoostNetObjectPrioritizerConfig]
InnerRadius=<INNER_RADIUS>
OuterRadius=<OUTER_RADIUS>
OwnerPriorityBoost=<OWNER_BOOST_PRIORITY>
InnerPriority=<INNER_PRIORITY>
```

```
OuterPriority=<OUTER_PRIORITY>
OutsidePriority=<OUTSIDE_PRIORITY>
```

Copy full snippet

Object Count Limiter

The `NetObjectCountLimiter` limits the number of objects considered for replication each frame.

Configure Object Count Limiter

You can configure the `NetObjectCountLimiter` in an engine configuration file such as `DefaultEngine.ini`:

```
[Script/Iris.NetObjectCountLimiterConfig]
MaxObjectCount=<MAX_COUNT>
Priority=<PRIORITY>
OwningConnectionPriority=<OWNER_PRIORITY>
bEnableOwnedObjectsFastLane=[true | false]
```

Copy full snippet

For example, the default configuration settings are:

```
[Script/Iris.NetObjectCountLimiterConfig]
MaxObjectCount=2
Priority=1.0
OwningConnectionPriority=1.0
bEnableOwnedObjectsFastLane=true
```

Copy full snippet

Create a Custom Dynamic Prioritizer

You can create a custom dynamic prioritizer by implementing the `UNetObjectPrioritizer` interface located in:

- `..\Engine\Source\Runtime\Experimental\Iris\Core\Public\Iris\ReplicationSystem\Prioritization\NetObjectPrioritizer.h`

Configure a Custom Dynamic Prioritizer

To register your custom prioritizer with the replication system, you must configure your prioritizer in the Engine configuration file hierarchy, preferably in your project's `DefaultEngine.ini` file:

```
[/Script/IrisCore.NetObjectPrioritizerDefinitions]
+NetObjectPrioritizerDefinitions=(PrioritizerName=<PRIORITIZER_NAME>,
ClassName=/Script/<MODULE_NAME>.<PRIORITIZER_NAME>,
ConfigClassName=/Script/<CONFIG_MODULE_NAME>.<PRIORITIZER_CONFIG_NAME>)
```

Copy full snippet

where:

- `PRIORITIZER_NAME` is the name of the custom prioritizer.
- `MODULE_NAME` is the name of the module where your prioritizer is defined.
- `CONFIG_MODULE_NAME` is the name of the module where your prioritizer's associated configuration is defined.
- `PRIORITZER_CONFIG_NAME` is the name of the custom prioritizer's configuration class.

There is a special `DefaultPrioritizer` that, if set, is automatically used for all objects with a valid `WorldLocation`.

Use a Custom Dynamic Prioritizer

To use your custom dynamic prioritizer for a replicated object, follow the steps in Specify a Dynamic Prioritizer and use the name of your prioritizer in the call to `GetPrioritizerHandle` in Step 3.