MERCUNA

Mercuna 3D Navigation
Unreal Engine User Guide

v2.7

MERCUNA

Contents

Installing

The Mercuna middleware is integrated into Unreal Engine as a standard plugin compatible with **Unreal Engine 5.0.1 or later**.

- **Binary evaluation** - The binary evaluation version of Mercuna must be installed as an Engine plugin - unzip the archive and copy the Mercuna directory to the **Plugins/Marketplace** directory (you may need to create the Marketplace subdirectory) within your Unreal Engine 5 install, e.g. *UnrealEngineDir*/Engine/Plugins/Marketplace/Mercuna.

- **Source version** - For source versions of Mercuna, it can be used either as a Game plugin or, if you are building the engine from source, as an Engine plugin. Simply copy the Mercuna directory into the **Plugins** directory in your Game/Engine folder.

If you have also licensed Mercuna Ground Navigation, then the plugin will provide both ground and 3D navigation capability. See the Mercuna Ground Navigation User Guide for documentation of the Ground Navigation capabilities.

Once installed, the Mercuna components, actors and menu will automatically be available when you next start the editor.

Configuring the Navigation Octree

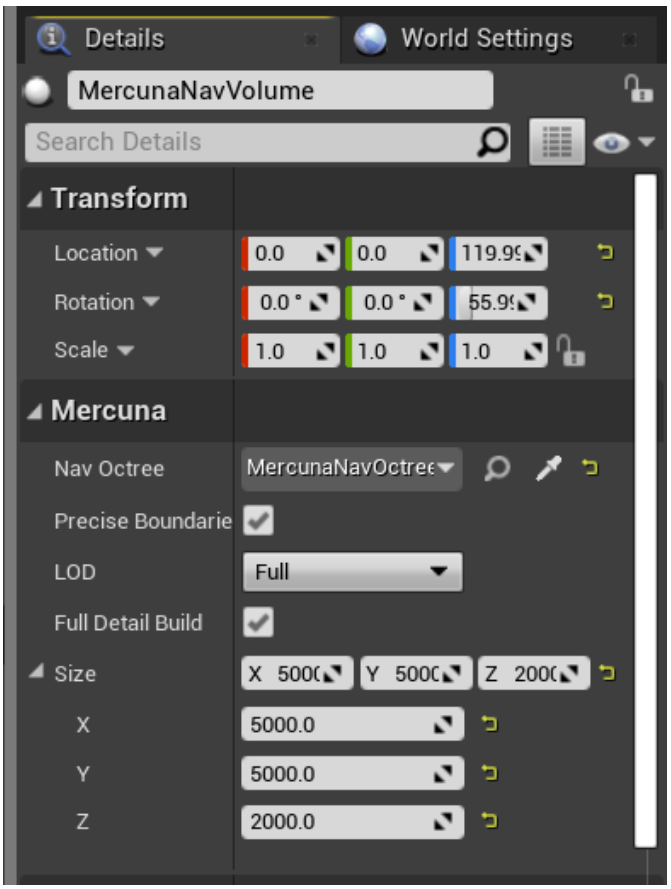
A **Mercuna Nav Octree** is used to find paths for pawns through 3D space, much like Unreal's 2D Nav Mesh is used to navigate pawns over terrain.

Nav Octree is generated in the level everywhere that is within a **Mercuna Nav Octree Volume**. A Nav Octree can simultaneously support multiple actor sizes (within limits), so in most cases a single octree should be sufficient. However, more complex setups with multiple Nav Octrees in a single level are possible if support for very different actor sizes is required.

Multiple Nav Octree Volumes can be present in one level, and overlapping Nav Octree Volumes that are linked to the same Nav Octree allow pawns to navigate seamlessly between them. All space outside of the Nav Octree Volumes is treated as unnavigable.

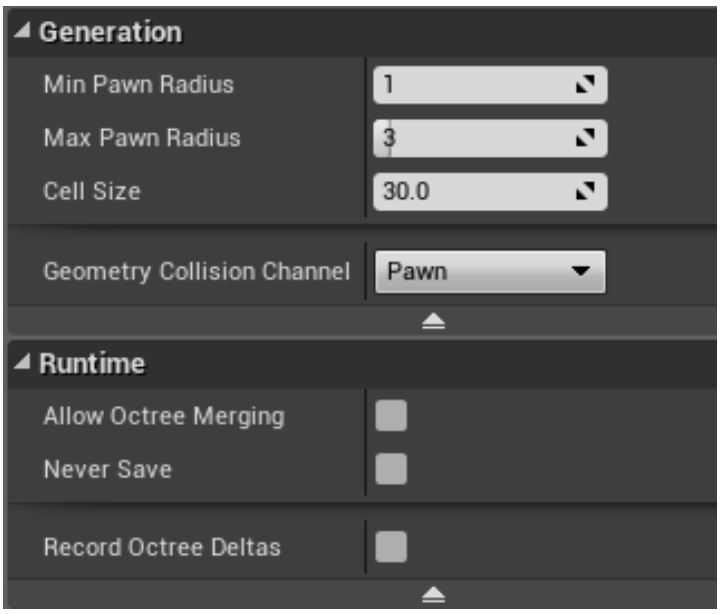
Setting up the navigable space

To set up a navigable volume, first add a Mercuna Nav Octree Volume actor into the level, and size as required. Only boxes are supported, so to describe more complex boundaries of the navigable space, multiple Nav Octree Volumes can be configured.



Example settings for Mercuna Nav Octree Volume

The octree generation parameters to be used in this level are configured on the Mercuna Nav Octree actor. Upon creation, the parameters are set to default values. These defaults can be modified in the Mercuna project settings.



Example settings for Mercuna Nav Octree

The parameters determine how detailed the representation of the navigable space is in the octree, and the sizes of pawns that can accurately navigate through it.

The **cell size** determines the side length of the cubes that make up the lowest level of the octree. Cells are considered unnavigable if there is any level geometry within them, so the larger the cell size the greater the error margin in the representation of the geometry.

The **minimum** and **maximum pawn radius** determine what navigation data is stored in the octree. Paths will never go closer to geometry than the minimum pawn radius, and the octree stores data to allow paths to be found with up to the maximum radius clearance from geometry.

The radius is expressed as multiples of cell size, so for a cell size of 30, a minimum radius of 2 and a maximum radius of 5, entities of radius between 60 and 150 can be accurately navigated through the level. Entities that are smaller than the minimum radius will navigate successfully, but might not take paths through small gaps they could fit through. Entities that are larger than the maximum radius aren't supported, as their paths might make them collide with geometry.

The **Restrict to Surface** option limits the octree to be generated only within the configured max pawn radius of surfaces. This means that agents can't move through open spaces but instead are only able to navigate close to geometry.

The **Never Save** property indicates whether the navigation data will be built from procedurally generated data at runtime, so any navigation data generated in the editor should not be saved (see Runtime Octree Building below).

Finally, the **Geometry Collision Channel** specifies the collision channel used to query whether geometry should be considered blocking for navigation. For a collider to be considered by navigation it must have "Can Ever Affect Navigation" set to true, and give a Block response to this collision channel.

If **Precise Boundaries** is ticked on the Nav Octree Volume, the boundaries of the navigable volume are considered hard edges, beyond which your pawns can't move. However, if the Precise Boundaries option is turned off then navigable space will extend up to a high power of 2 boundary in the octree. This allows a significant memory saving when there is open space around the navigable volume and you don't need to precisely specify the edge of the volume your agents will move within.

Level of Detail

It is possible to specify that part of the navigation octree is built at a different level of detail by setting the **LOD** on a Nav Octree Volume. This would normally be used when you want high precision navigation in particular parts of the level without the expense of using a high resolution octree across the entire level.

The normal set up is to create a large nav volume with ½ or ¼ level of detail, and then specify smaller overlapping nav volumes that generate particular areas at full detail. It is also possible to set the large volume to full detail and add smaller volumes at lower detail.

When generating the octree, Mercuna assumes that smaller nav volumes override the level of detail of overlapping larger volumes.

The **Full Detail Build** option on the Nav Octree Volume specifies whether the volume is built at the reduced level of detail or at full detail. If it is built at full detail then the geometry is voxelised as normal and then the LOD is applied when the octree is stored. If full detail build is switched off then the octree is voxelized as if the voxel size in that volume has been increased, so for example a ½ LOD area in an octree with a voxel size of 30 will have an effective voxel size of 60 for the purposes of building the octree in this volume. Note that the settings on the octree itself are not affected, the cell size and min and max radius there are always in terms of full level of detail cells.

Full Detail Build should only be switched off if you want to reduce the CPU cost of runtime octree builds for areas set to a lower LOD, normally it is best to have it switched on in order to increase the accuracy of the generated octree.

In low level of detail areas, agents won't be able to navigate as close to walls or through as small gaps as they would if full LOD was used. However, the generation time and memory usage for low LOD volumes is substantially less than for full LOD volumes.

Multiple Octrees

Mercuna supports multiple Nav Octrees within a single level, which is normally used to allow navigation for agents of very different sizes. In this case, you must select which Nav Octree each Nav Octree Volume is associated with - each Nav Octree Volume can only be linked to a single Nav Octree.

By default, Mercuna is configured to automatically link Nav Octree Volumes to Nav Octrees, which means that no manual configuration is required when there is only a single octree in the level. If no Octree is present in a level, Mercuna will automatically create a Nav Octree actor when the first Nav Octree Volume is added to the level, and link the Nav Octree Volume to it.

If automatic linking is disabled (by unticking "Auto Link Nav Volumes with Graphs" in Project Settings -> Plugins -> Mercuna) then you must manually create Mercuna Nav Octrees and link the Nav Octree Volumes to them. This can help avoid errors when using multiple octrees, as the Octree must be set explicitly, rather than Nav Volumes potentially being automatically linked to a different Octree to the one that was intended.

Octree Transform

The Mercuna Nav Octree is usually positioned at the world origin and aligned with the world's axis. Its transform should not be directly modified. Nav Octree Volumes can be moved and rotated. However, the Octree and all of its other Nav Octree Volumes must then be updated to have the same orientation.

This is enforced in the Editor. Changing the rotation of a Nav Octree Volume in the Editor will automatically change the rotation of its linked Octree (along with all of the Octree's other associated Nav and Modifier Volumes). After any transform change the Octree will need to be completely rebuilt. Nav Octree Volumes that are linked to a new or different Nav Octree will automatically adopt the orientation of that Octree .

During gameplay, levels that are streamed in containing prebuilt Octrees can be transformed (including rotated) via overall level transforms. However, individual Mercuna actors, including both the Nav Octree Volumes and Nav Octrees should not be moved or rotated at runtime.

Nav Seeds

In order to identify which regions should be considered navigable, Mercuna requires you to place **Mercuna Nav Seed** actors into levels. This allows uninteresting regions, such as the small isolated areas inside hollow geometry or large areas outside of the level boundaries, to be excluded and avoids pawn positions getting clamped to the wrong side of polygons.

A Mercuna Nav Seed needs to be placed in the main part of the level where pawns will move. This seed is used during construction of the octree to find all connected reachable cells, by flood filling the region starting at the nav seed. If you have multiple disconnected areas in your level where you expect pawns to move, a seed must be placed in each separate area.

Note that if you also have Mercuna Ground Navigation, the nav seed will also seed any nav octree.

Exclusion Volumes

A particular box volume can be completely excluded from navigable space by adding a **Mercuna Nav Exclusion Volume** actor to the level. By default the exclusion volume will be applied to all nav octrees. If you have multiple octrees in a level, and only want the exclusion volume to apply to one of them, then you can explicitly specify which octree the exclusion volume is linked to via the volume's properties.

The exclusion volume may be rotated and scaled.

Building the Octree

Unless runtime generation is being used, the Octree must be built after it is first configured and after any change to the level geometry. This can be done by selecting Build Octree from the Mercuna menu (accessed by clicking the Mercuna button in the Toolbar). An on screen notification displays the progress of the octree construction.

When multiple Octrees are present in the level, the menu changes to show Build All Octrees and Build Selected Octree. This allows you to build all the octrees at once, or select a specific Octree and just build that one.

If full Mercuna logging is enabled (see Logging section below), then you will see the generation progress for each navigation volume in the output log, and the total memory consumption of the octree is reported.

Commandlet

If you wish to automate the building of the Octrees, for example as part of a job run regularly each night, they can be built using the MercunaNavGraphBuilderCommandlet. This can be run from the command line using:

```
UnrealEditor.exe ProjectName.uproject MapName
-run=MercunaNavGraphBuilderCommandlet [-AlwaysBuild]
```

By default only octrees that have been saved as dirty will be rebuilt and resaved, however you can force all octrees to be built using the AlwaysBuild switch.

Memory usage and Performance

The main influences on memory usage and performance are:

- **Cell size**: Smaller cell size octrees use significantly more memory and take longer to generate.
- **Density of geometry**: Large open navigable volumes are stored efficiently, and are quick to pathfind through, volumes with dense geometry and narrow corridors use more memory and take longer to find paths through.
- **Maximum pawn radius**: A larger maximum pawn radius will take longer to generate and slightly increase memory usage.

Single Threaded Mode

When Unreal is running single threaded, Mercuna will only schedule jobs from the module Tick function. The budget per tick, specified in seconds, is set in the Mercuna section of Project Settings:

When in single threaded mode, nav graph queries maybe time sliced over multiple frames. While rebuilding the nav octree works in single threaded mode, it is not recommended as you can easily schedule more work than it is possible to perform on a single thread.

Multiple levels

Mercuna supports both level streaming and world composition by saving the octree that is relevant to each streamed or composed level (sub-level) within that level.

When using level streaming or world composition, Mercuna Nav Octree Volumes and Mercuna Nav Seeds should be placed in the sub-level so that they are loaded and unloaded at the correct times.

When editing the Persistent level you will see one Mercuna Nav Octree in each loaded sub-level, this octree is automatically associated with the Nav Octree Volumes and Nav Seeds that are in that sub-level.

In order to allow seamless navigation between octrees loaded from different sub-levels, Mercuna merges together octrees loaded from sublevels at runtime. To enable this feature, the octrees in the sublevels must have exactly the same octree settings, have the same orientation, and have the **Allow Nav Graph Merging** option set.

If **Allow Nav Graph Merging** is false, or if the octree settings don't match then octrees won't be merged and multiple octrees will exist at runtime. In this case you can manually switch pawns between octrees using the **Set Nav Octree** function on the Mercuna 3D Navigation Component.

Generating Octrees in Multiple Levels

The best way to generate the sub-level octrees is from the Persistent level. Load all your sub-levels using the levels window and then select Build All Octrees from the Mercuna menu. This ensures that geometry that overlaps between levels is correctly represented in each level's octree. Once generation is complete, save all the sub-levels.

When placing Nav Octree Volumes in sub-levels for octrees that will merge together on sub-level load, you should have at least one voxel size worth of overlap between the nav volumes.

If it is not possible to load all levels due to memory constraints, go through each sub-level containing a Mercuna Nav Octree in turn. Load a sub-level and all levels that contain geometry overlapping Nav Octree Volumes within the sub-level, generate the octrees in that sub-level by selecting them and using the Build Selected Octree option from the Mercuna menu. Then save the sub-level and go on to the next.

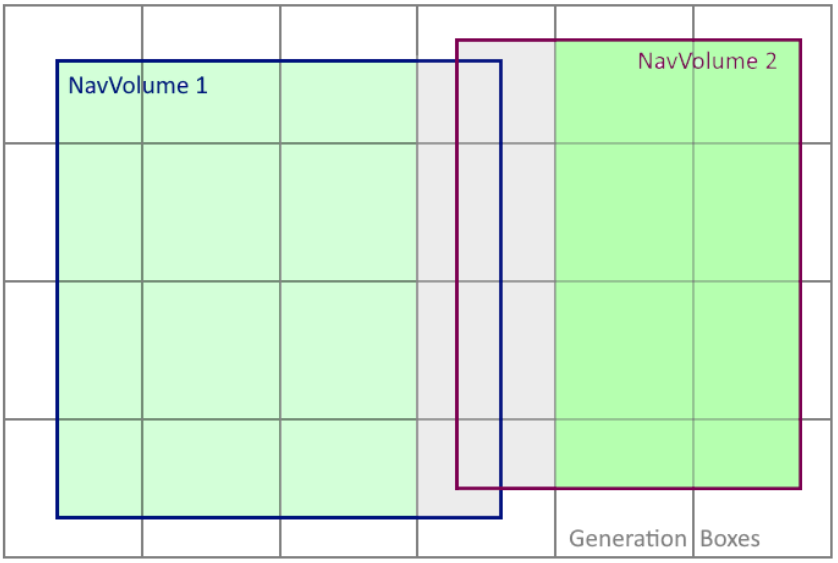
Aligning Nav Octree Volumes for Optimum Merge Performance

Agents will be able to seamlessly navigate a merged octree providing the Nav Octree Volumes in the sub-levels are overlapping or touching.

The process of merging the Nav Octree can be quite CPU intensive, however if the Nav Octree Volumes in sub-levels are aligned correctly then the amount of work Mercuna has to do when merging can be reduced.

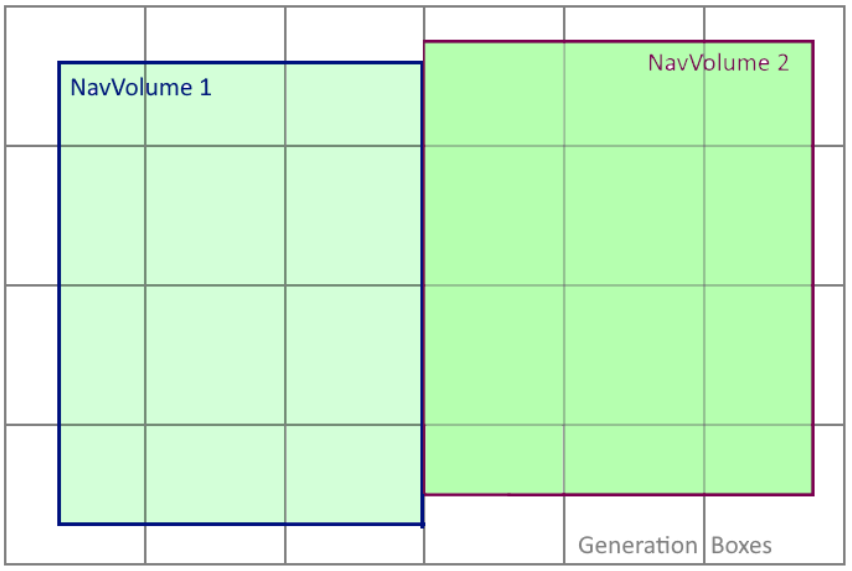
Internally, the Nav Octree is broken up into boxes that are 64 octree cells on a side. The connectivity within these boxes is baked into the octree data, and the connectivity between the boxes is stored in a separate connectivity graph to allow hierarchical pathfinding.

If volumes in the sub-levels aren't aligned then the connectivity within boxes where the octree volumes overlap needs to be recomputed when the sub-levels are streamed in - this is shown by the grey regions in the diagram below.



Connectivity information all along the merged edge needs to be rebuilt

The CPU cost can be minimised by aligning the boundaries of touching Nav Octree Volumes on a multiple of 64 times the Cell Size configured on the Nav Octree:

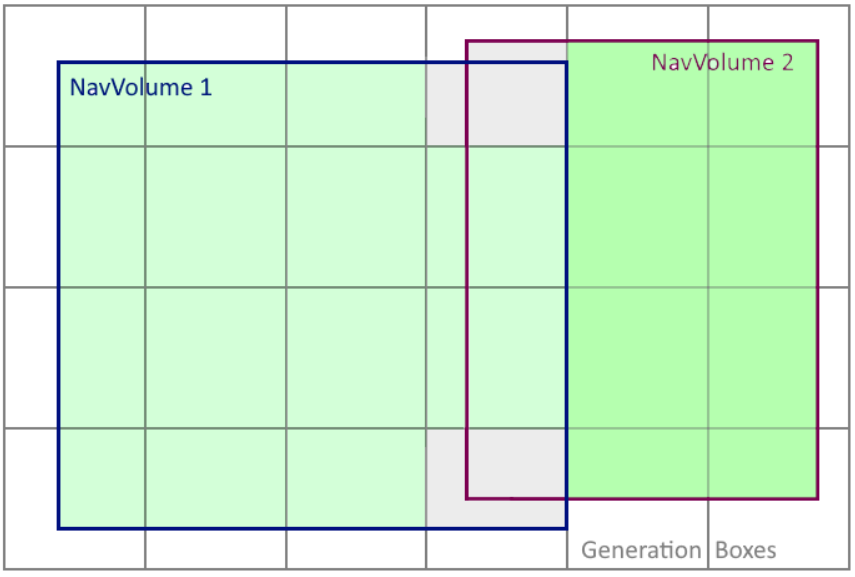


No connectivity information need to be rebuilt

In this case only the connectivity between the boxes needs to be computed, which is relatively cheap.

The alignment is with respect to the world origin, appropriately rotated if the octree has a non-zero rotation specified.

If it is not possible to set the Nav Octree Volumes up this way due to how the Octree aligns with gameplay regions, then it is best to increase the amount of overlap so that individual 64x64x64 boxes can be read in their entirety from one of the sub-levels. This can significantly reduce the amount of connectivity rebuilding that is required:



Only connectivity at corners needs to be rebuilt

World Partition (UE5 only) - experimental

Mercuna supports UE5's new World Partition system in a way that is transparent to the user and is designed to be simple to use.

In contrast to Level Streaming or World Composition maps, where a Nav Octree per sublevel is required, in a World Partition map you only need a single Nav Octree and then can freely place Nav Octree Volumes that can span large regions.

Internally Mercuna automatically splits the octree data into chunks and saves each part within an invisible grid of MerNavDataChunk actors. As these chunk actors are loaded and unloaded by the World Partition system, both in the editor and at runtime, the corresponding part of the octree will be merged in on load, or purged from memory on unload.

The whole map can therefore be covered by a single Nav Octree. Nav Octrees are stored as non-spatially loaded actors and will always be present, whereas the Nav Volumes/Modifier Volumes/Exclusion Volumes are stored as spatial actors that are streamed in and out by the World Partition system as required.

Building

When the Nav Octree is built in the editor, only the part that corresponds to currently loaded editor cells will be generated. Actors immediately surrounding the loaded cells will be automatically loaded by the build process to ensure the edge of the generated volume is consistent. These actors are then unloaded once the build is complete. **You must not load or unload editor cells while the build is running**.

It is therefore possible to generate one part of the octree, save that section, and then load a different set of editor cells, generate the octree for that region, and then have both sections of the octree seamlessly merge together at runtime. However, we strongly recommend having an overnight job that builds the nav octree using the commandlet (see below) in order to ensure the nav octree is consistent with all changes that have been made to the map.

If the Nav Octree is placed into any data layers, then when building in the editor only actors that share at least one data layer with the Nav Octree will be included for the purposes of generating the octree.

Runtime builds and rebuilds work as normal, but note that they will only build based on the geometry that is currently loaded in the specified volume.

Nav Octrees can also be built using the Mercuna world partition commandlet. This will build all the Mercuna nav graphs by iteratively loading the cells within the specified map. The commandlet can be run from the command line using:

```
UnrealEditor.exe ProjectName.uproject MapName
-run=WorldPartitionBuilderCommandlet
-Builder=MercunaWorldPartitionBuilder
```

Load Complete Events

When an octree is loaded and ready for use the **OnLoadComplete** event on the NavOctree triggers. In simple configurations where octrees aren't loaded from multiple sub-levels simultaneously and there is no level streaming, this event is triggered immediately after the octree is loaded. However, if the octree is merged with another octree when the level is loaded, then OnLoadComplete is not triggered until the merge is complete. Once the event fires, you know that pawns are able to navigate across the merged part of the octree.

When octrees are merged, first the octree data itself is merged and then the connectivity data used for hierarchical pathfinding is rebuilt. The second stage can take a few frames to complete, and is not required for short range pathfinding, so it can be useful to know when the first stage is complete.

Therefore, there is an **OnShortRangeLoadComplete** event that triggers once short range path finds are available on the merged octree, but before hierarchical pathfinding is available. If you don't need to pathfind long distances, then you can start navigating on the octree when this event fires. As with OnLoadComplete, if the octree is not merged with another on load, this event fires immediately after the octree is loaded.

Runtime Octree Building

It can sometimes be desirable to build the Nav Octree at runtime, particularly for procedurally generated levels. Since these Octrees will be generated at runtime it is recommended that they should have the **Never Save** property set on them. This prevents any data that might be generated in the Editor while testing from being unnecessarily saved.

Nav Octree Volumes should normally be placed or spawned, scaled and rotated, before the Octree is generated at runtime, however they can be spawned or moved after the initial build and the octree rebuilt in the relevant locations - see Changing Nav Octree Volumes at runtime. In order to ensure that the Nav Octree Volumes and Nav Octree have the same orientation (see Octree Transform section) either the **SetNavigationRotation** or **SetNavigationOrientation** methods must be used to modify the rotation. These are available on both the Mercuna Nav Octree actor and the Nav Octree Volume actor. Either function can be used and the orientation of the Octree and all its Nav and Modifier Volumes will be updated.

The build can be triggered by making a request on the octree actor (via Blueprint or C++) using the **Build** function. This clears the octree (if there was any previously built data), and then generates the octree within all linked Nav Octree Volumes

The generation happens in two phases - first a low level of detail version of the octree is built. This build completes quickly and allows agents to start navigating with short range

pathfinds within open spaces (but not close to geometry). The event **OnBuildLowResReady** is triggered once this low resolution octree is available to indicate that agents can start navigating. Once octree generation has fully completed, the event **OnBuildComplete** is triggered.

Runtime Octree Rebuild

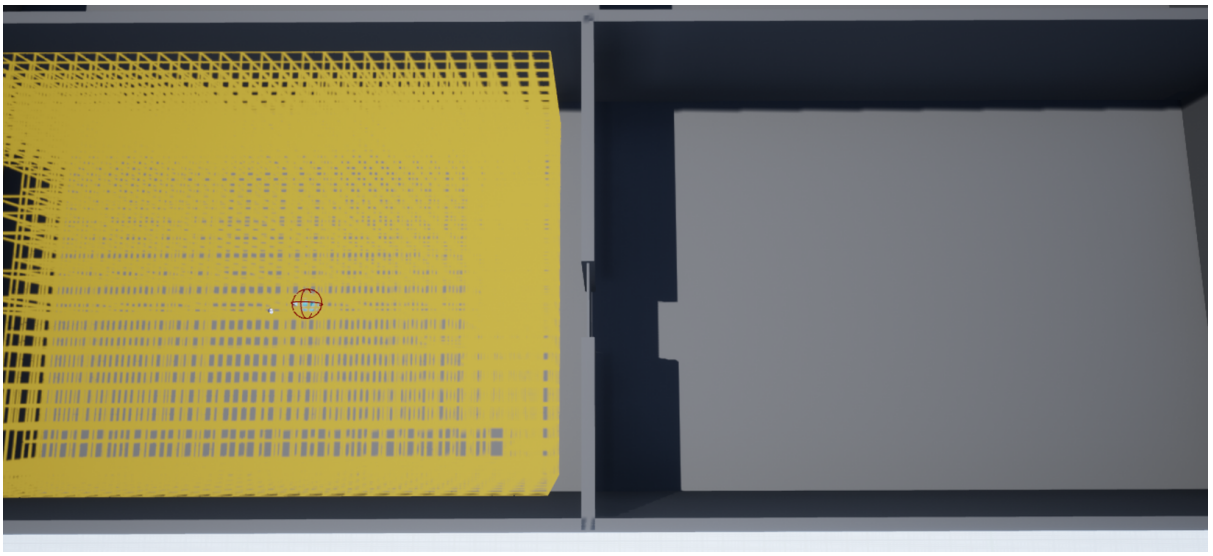
Specific volumes of the Octree can also be rebuilt while the game is running using the **Rebuild Volume** or **Rebuild Volumes** function on the Octree actor. Rebuild Volume is normally used to pick up runtime changes when just a section of the level changes, such as a door opening.

The two phase build, where a low resolution version of the octree volume is first built and then replaced by a full resolution version once it is ready, is optional when only part of the octree is rebuilt. Use the **Staged Build** flag on Rebuild Volume to specify whether or not to use it. When rebuilding only small volumes it is often more efficient to do a single phase build.

If Staged Build is used, the event **OnRebuildLowResReady** is triggered once the low resolution version of the rebuild volume is done. Either way, once the whole volume has been rebuilt at full resolution, the event **OnRebuildComplete** is triggered. This event includes the volume that was regenerated, to aid with scripting.

Pathfinds and reachability tests that go through the regenerated region may fail while the regeneration is in progress. Once regeneration is complete, all active paths are recomputed if they go through or close to the regenerated region, causing actors to path around new obstacles or through newly available shortcuts. If a pathfind fails during regeneration, you may want to retry it once the OnRebuildComplete event has been triggered.

Regenerating a region will not cause newly connected areas outside the regenerated region to become seeded. If a runtime regeneration might connect a volume that is not connected to any other nav seed when the navigation octree is built in the Editor, you must place a seed within that volume.



In the screenshot above, the yellow region on the left is navigable and seeded. The wall in the centre has a door in it, and when this opens Rebuild Volume is triggered through blueprint to regenerate the navigation data around the door.

However, navigation into the region on the right will still not be possible because that region was not seeded. This can be fixed by adding a Mercuna Nav Seed into the right hand region.

Changing Nav Octree Volumes at runtime

Nav Octree Volumes, as well as Exclusion Volumes, can be spawned, moved or deleted after the octree has been built. The octree is not changed immediately, instead **Rebuild Changes** must be called to trigger a rebuild. Nav Octree Volumes must have the same orientation as the octree.

Changing Modifier Volumes at runtime

Nav Modifier Volumes can be spawned, moved, update and deleted at runtime. The octree will automatically rebuild to reflect the changes.

Saving and reloading Octree changes

If players are able to make changes to the world that are built into the Octree using the rebuild functionality described above, and they should be persisted across the level being unloaded and reloaded, then it can be useful to save just the changes to the octree as a delta to be applied immediately after the base octree is loaded.

Mercuna supports this by tracking the volumes that are rebuilt and saving out deltas that contain just those parts of the octree. To enable the change tracking, switch on **Record Octree Deltas** in the Advanced section of the Mercuna Nav Octree configuration. *Note that delta tracking may not be used when octree merging is enabled.*

Deltas can be saved by calling **SaveDeltas()** on the Meruna Nav Octree object from C++, passing in an FArchive object - different Unreal Archive classes exist for saving to file or memory buffer.

On level load, deltas must be loaded with **LoadDeltas()** before any other changes are made to the octree. Loaded deltas are tracked and included in any future call to SaveDeltas(). You may only apply one set of deltas to an octree.

Saving and reloading the whole Octree

Mercuna also allows you to save and reload the entire octree using **SaveToArchive()** and **LoadFromArchive()**.

This can be useful when procedurally generating levels, for example you could cache the octree that is generated from a particular level seed without having to save the whole level.

Pathfinding

Finding paths through the Nav Octree can be done implicitly by making your pawn movement controlled by Mercuna, this method allows you to take advantage of the Mercuna steering and avoidance systems. Alternatively, path finding can be requested explicitly by making a request directly to the octree (via Blueprint or C++) and receiving a **MercunaPath** (or **MercunaSmoothPath**) object which can then be used as required.

As pathfinding is performed asynchronously, the returned MercunaPath or MercunaSmoothPath object is not immediately valid, but can take one or two frames to complete. You must either check each frame to see if it is ready yet or subscribe to its PathUpdated delegate.

For longer paths, Mercuna uses hierarchical pathfinding. An approximate path is found through a simplified representation of the level and then a detailed path find is performed guided by the approximate path. This allows much longer paths to be found than would be possible with simple A* pathfinding alone.

Mercuna also supports partial paths (enabled by default). A partial path is returned when a complete path can't be found to the specified destination, if it is disconnected from the start point, for example. Instead Mercuna returns a path to the closest point to the destination that is reachable.

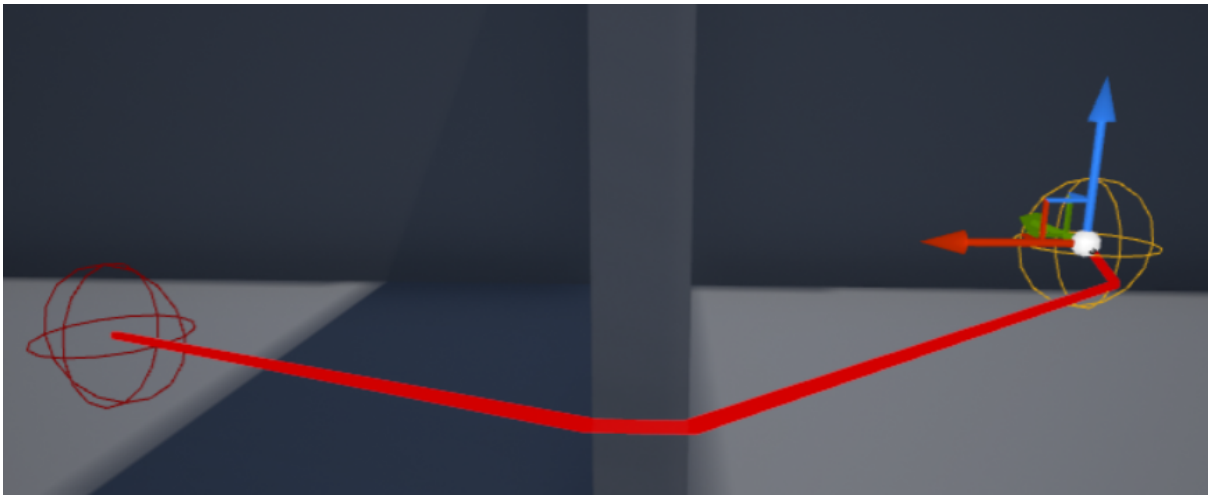
Path Testing Actor

In order to easily debug and understand pathfinding problems a pair of **Mercuna Nav Octree Testing Actors** can be used to generate test paths. Simply drag two testing actors into the level, and on one of the actors set the other one as the 'Other Actor' property. When you do this a test path will be drawn connecting the two actors. This path will update when

either actor is moved. A red path means a complete path could be found, while an orange path means that only a partial path could be generated.

The **Radius** property specifies how much clearance there should be around the test path - this allows you to check what path larger and smaller actors would take.

You can also set the **Height Change Penalty** to see how that affects the generated paths - when there is a height change penalty the pathfinder will prefer paths that don't go up and down as much.



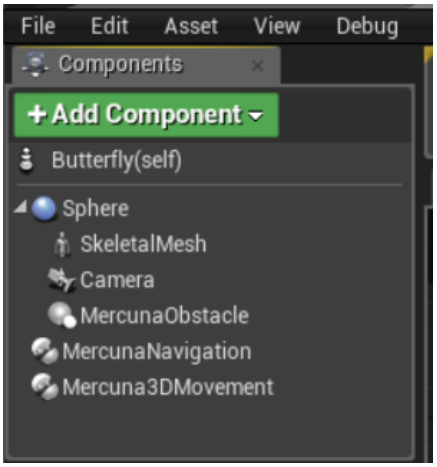
A test path between two Mercuna Nav Testing Actors

Creating a Mercuna navigated Pawn

In order to allow pawns to use Mercuna to navigate they need to have the following components:

- **Mercuna 3D Navigation** - this component provides the pawn with 3D navigation capabilities, accessible through Blueprint.
- **Mercuna Obstacle** - this marks the pawn as a dynamic obstacle for the purpose of 3D navigation. The obstacle component must be a child of the root scene component.
- A suitable **movement component**, e.g. the **Mercuna 3D Movement** component

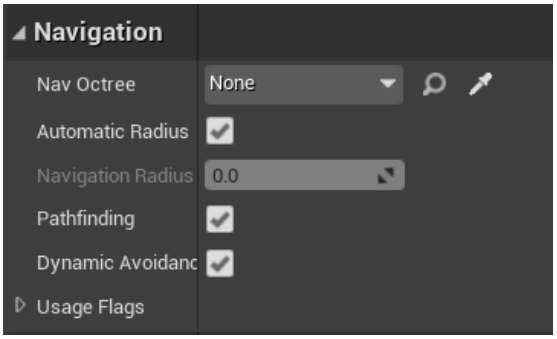
APawn::**GetMovementComponent()** should return the 3D movement component. If the pawn is derived from ACharacter, this may need to be overridden to return the correct movement component to prevent steering errors.



Pawn blueprint setup for Mercuna navigation

Navigation Component

The **Mercuna 3D Navigation** component offers both a C++ and Blueprint interface for making movement requests to the pawn.



Navigation component settings

When there are multiple octrees in a level, Mercuna will automatically try and choose the best one to use based on which nav volume the pawn is currently in and which octree best fits the pawn's size. The octree that a pawn uses can be overridden by explicitly setting the **Nav Octree** parameter or by calling the SetNavOctree function.

For the purpose of pathfinding and steering Mercuna treats pawns as spheres. When **Automatic Radius** is enabled (default) the Navigation Radius is derived from the bounding sphere of the collision components of the pawn. Alternatively, the **Navigation Radius** can be explicitly set. This can be desirable if your pawn is an usual shape, particularly if the width and height are much less than the length. However, using a smaller radius means the pawn is no longer guaranteed not to collide with geometry, if you do this you should test carefully to ensure it works correctly in your levels.

You can also turn off **Pathfinding** completely, in which case the pawn will not query any nav octree and will simply try to move directly toward any given goal. And finally, you can set an

Avoidance Mode to control how and whether the pawn attempts to avoid other actors with **Mercuna Obstacle** components.

During a move the maximum speed can be temporarily changed via a call to **OverrideSpeedMultiplier**.

Setting a Movement Style

How a pawn moves depends on how its movement style is configured. The movement style options are set on the Mercuna 3D Navigation component:

Movement style options

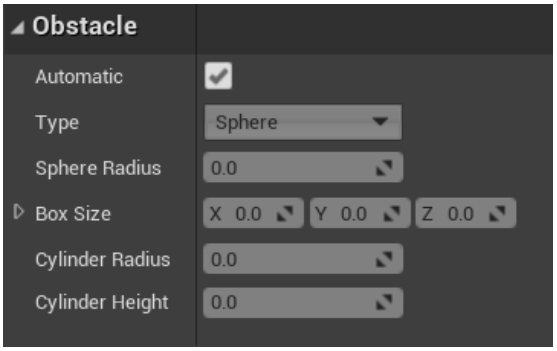
The options include:

- **Max Pitch** - The default maximum angle that the pawn can pitch up or down. This is ignored if Move In Forward Direction is set to Always.
- **Max Roll** - The maximum roll when turning, in degrees. Causes the pawn to bank when turning.
- **Move In Forward Direction** - Whether to restrict movement to forward direction. When set to **Always** the pawn will only move forwards making sweeping u-turns when changing direction. When set to **Prefer** the pawn will make u-turns when changing direction if already moving, but move directly backwards if starting from stationary. **Independent** means the pawn will always move directly along the path.
- **Stop at Destination** - Whether the pawn should slow down in order to stop at its destination.
- **Height Change Penalty** - How much the pawn should avoid unnecessary changes in height while moving.
- **Smooth Paths** - Whether to use spline based paths, or simple straight line segment paths. See Steering section for more detail.
- **Look ahead time** - When smooth paths are enabled, the pawn will orient itself so it is looking towards a point this far ahead along the path (unless another look target is set). When the Steering Debug Draw is enabled, a pink circle is displayed where the look ahead point is.

- **Roll anticipation time** - On smooth paths, the pawn will roll into turns depending on the yaw rate estimated at the specified time down the spline, i.e. 1 second roll anticipation time means the banking begins 1sec ahead of the turn. When the Steering Debug Draw is enabled, a green circle is displayed where the anticipation point is. If smooth paths are disabled, banking depends only on the current yaw rate and roll anticipation is ignored.

Obstacle Component

Any actor can be made a dynamic obstacle by adding a **Mercuna Obstacle** component. It is expected that Mercuna navigated pawns will usually have one, but other pawns such as the player, might also have an obstacle component so that the AIs avoid them.



Flight obstacle settings

By default Mercuna will automatically try to calculate the best shape of obstacle that encloses the pawn. Alternatively, the shape and size can be manually set. The available shapes are Sphere, Box, Cylinder. The shape only applies for avoidance.

The center of the obstacle component is used to define the position of the pawn in Mercuna. This means that if the obstacle component is offset to better define a bounding volume, the pawn will then turn about that point.

Actors with an Obstacle component should have “Can Ever Affect Navigation” switched off on their Collision settings on all their components so they aren’t baked into the nav octree.

Steering

In order to give smooth, natural looking movement Mercuna uses polynomial splines to interpolate between path points to generate smooth curves. These splines are constructed taking into account the available space and define a continuous velocity curve that can be followed precisely. Mercuna navigates the pawn down the spline path taking into account the maximum speed and acceleration configured on the pawn.

Whereas the pathfind is performed when a pawn is given a destination, the spline is generated on demand. If the pawn is pushed, or has to avoid other actors causing it to move

away from the spline, then the spline and, if necessary, the underlying path are automatically regenerated.

Spline based steering can be disabled to fall back to simple steering by turning off **Smooth Paths** in the movement style options. However, the simple steering method suffers from the problem that pawns frequently overshoot corners and fall off the path, resulting in collisions with level geometry and it is not recommended to be used unless necessary for backwards compatibility.

Avoidance

Mercuna offers dynamic obstacle avoidance to ensure that pawns don't collide while moving. Any actor with a **Mercuna Obstacle Component** (see above) is automatically considered as an obstacle that needs to be steered around for the purpose of avoidance. The avoidance algorithm used by Mercuna is a modified version of ORCA velocity obstacle method that additionally takes into account the fixed level geometry stored in the nav octree.

What type of avoidance to do is configured via the **Avoidance Mode** property on the Mercuna 3D Navigation component.

It is possible to specify particular actors to be excluded from avoidance on a per pawn basis using the **SetAvoidanceAgainst** function on Mercuna 3D Navigation Component. A common use case is to temporarily turn avoidance off against the target when executing an attack to avoid the attacking pawn veering off as it gets close to the target.

ORCA

Mercuna's implementation of Optimal Reciprocal Collision Avoidance (see e.g. https://gamma.cs.unc.edu/ORCA) is a modified velocity obstacle method that additionally takes into account the fixed level geometry stored in the octree. This can be enabled via setting the **Avoidance Mode** property to **ORCA**.

Context Steering

Contextual steering allows agents to perform path following whilst prefering to maintain pre-specified distances from other agents. Through combining the output of various steering contexts each agent attempts to choose a velocity that best satisfies all its contexts' constraints. Each context either scores a velocity, or marks as excluded (as it would cause a collision).

There are currently 5 contexts that are used:

- **Path following** - this is the simplest context that prefers velocities close to the path velocity.

- **Dynamic obstacle avoidance** - this context will add a high cost to any velocity that would make the agent collide with a dynamic obstacle.
- **Static geometry avoidance** - whereas the original path will be guaranteed to avoid geometry, other velocities chosen by context steering could result in collisions. This context therefore excludes velocities that will cause a collision with static geometry.
- **Repulsion** - this context provides a soft constraint to keep agents apart. If needed agents will get closer together but they will prefer to stay specified distance apart.
- **Cohesion** - this context prefers velocities that keep agents together in their group. Cohesion by default is zero, and should only be used for agents in the same group, moving together.

Agents use a gradient descent method to select the best velocity out of all available (non-excluded) velocities.

Configuration

Contextual steering can be enabled for an agent via setting the **Avoidance Mode** property on the Mercuna Navigation component to **Context Steering**.

Agent-Agent Parameters

Because how an agent reacts is different for each agent it encounters, for example it might want to stay close to allies and far away from enemies, context steering uses a different set of parameters for each agent pair. These parameters are obtained through a callback to game code the first time each pair of agents interact and then are cached.

Delegate

The repulsion and cohesion parameters for other agents are read into the Context Steering system from game code using the **OnGetContextualSteeringParams** delegate on the Navigation Component. This must be registered from C++, it is not exposed to Blueprint as performance of this delegate is critical.

The delegate should fill in an FMercunaContextualSteeringActorParameters struct which determines how this actor should react to the other actor (supplied as a parameter).

- **RepulsionWeight** (default 1.0) - How strong the repulsion force is, 0 indicates unused.
- **RepulsionDistance** (default 500.0) - The distance to the other agent at which the repulsion force starts being applied. The repulsion force gets stronger as the agents get closer together.
- **CohesionWeight** (default 0.0) - How strong the cohesion force is, 0 indicates unused.

- **CohesionDistance** (default 1000.0) - The distance from the other agent at which the cohesion force starts being applied. The cohesion force gets stronger as the agents get further apart. When both forces are used, this must be greater than the repulsionDistance.

Sensible ranges for the weights are 0.5 - 2.0, or 0 to disable.

There is a hard-coded maximum search multiple of 100 entity radii for considering neighbours.

Cache

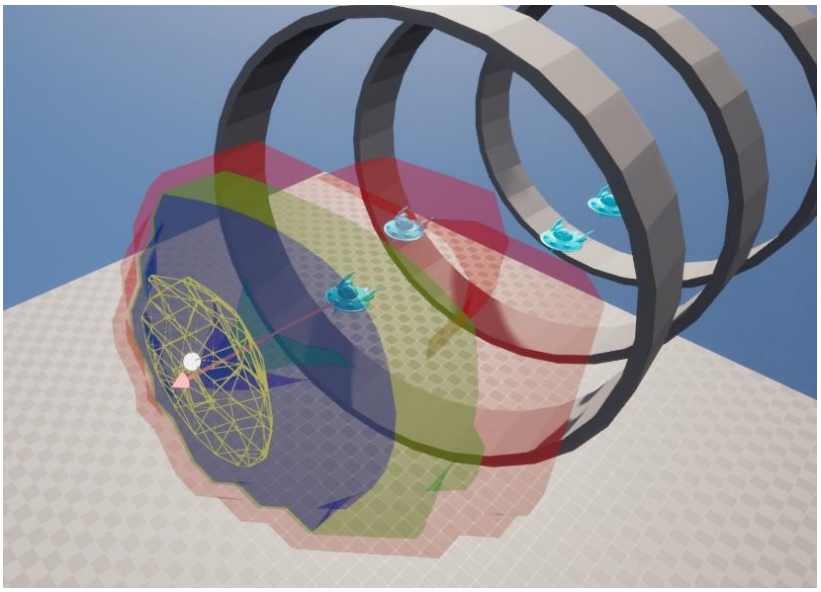
To avoid making excessive callbacks to the delegate, after the first encounter the parameters are cached on each agent. If you want to change the parameters, then you will need to invalidate the cache. The next time the agent's encounter each other they will call the delegate again. The invalidation can be done using the functions on the agent's 3D Navigation Component:

- **InvalidateContextualSteeringParams** - invalidate all cached parameters.
- **InvalidateContextualSteeringParamsAgainstActor** - only invalidate the parameters for the specified actor

Debug Draw

In order to understand how the agent's velocity is being chosen it is possible to draw the scored velocity field. In the main Mercuna menu, set a debug actor and turn on "Avoidance".

The values of a sampled velocity field are shown via coloured iso-surfaces, from coloured by value, from red (low) through green, blue and yellow-mesh (high). Excluded values (i.e those colliding with walls) are removed.



Drawing the velocity requires a regular grid of velocities to be sampled, compared with the usual method of determining the optimum velocity via gradient descent. This sampling of a very large number of velocities can have a notable impact on performance when the debug draw is active.

The pink-arrow indicates the velocity found by a gradient descent algorithm, while the white square is found by the sampling. The latter is only performed to generate this debug draw and as such may not always match the direction of the pink arrow.

Movement

In order for the Mercuna 3D Navigation component to drive the moment of a pawn, the pawn needs to have a suitable movement component. A simple default movement component is provided - the **Mercuna 3D Movement Component**. This is suitable for a variety of 3D movement styles.

Custom movement components can easily be implemented, but in order to be used by Mercuna they must provide the **IMercuna3DMovement** interface. The Mercuna Navigation component automatically detects and uses the first movement component it finds on the pawn that provides that interface.

By default Mercuna uses a Newtonian based flight model for pawn movement. For linear motion, Mercuna outputs a desired acceleration that is used to modify the pawn's velocity each frame. The acceleration may change discontinuously, but the velocity will vary continuously, giving smooth movement. The configured acceleration limits (see below) describe the capabilities of the pawn, these limits are combined by taking the strictest limit, e.g. if you are accelerating in the facing direction, but you are facing upwards, then you will be limited by the smallest of Forward and Upward limits.

One extra limitation compared to a pure Newtonian flight based model is the ability to set a maximum speed, as it is often desirable to prevent pawns moving too fast in a game.

Mercuna 3D Movement Component

The Mercuna 3D Movement component provides a Newtonian flight model for a pawn moving freely in space. It allows you to configure:

- **Max Speed** - The maximum speed the pawn may move at
- **Max Accelerations** - The maximum accelerations of the pawn (in the pawn's local coordinates).
    - **Forward** - maximum acceleration in the forward direction
    - **Backward** - maximum acceleration in the backward direction/deceleration in the forwards direction
    - **Sideways** - maximum acceleration in the left/right and up/down directions
- **Max Pitch Rate** - The maximum angular speed the pawn may pitch at, in radians/sec

- **Max Yaw Rate** - The maximum angular speed the pawn may yaw at, in radians/sec
- **Max Roll Rate** - The maximum angular speed the pawn may roll at, in radians/sec
- **Max Ang Accel** - The maximum angular acceleration allowed on each axis, in radians/sec

- **World Acceleration Limits** - advanced properties used to cap the maximum accelerations in the world frame. Disabled by default. These are useful, for example, to simulate the effect of gravity, where a pawn can accelerate faster downwards than it can upwards.
    - **Upward** - maximum acceleration in the vertical up direction
    - **Downward** - maximum acceleration in the vertical down direction

Modifier Volumes

Nav Modifier Volumes provide a mechanism for designers to influence and limit navigation within specific regions. For example, you can increase the cost of navigating through a volume, such that paths will prefer to go around it, or you can mark volumes as being restricted to particular pawn types, or pawns as restricted to staying within certain volumes.

Usage flags

In the project settings you can define up to 32 custom global usage types. These can then be assigned to individual Modifier Volumes to mark what the volume represents. Flags corresponding to each usage type can be set on the 3D Navigation Component of pawns to indicate which types of volumes it is allowed to enter, or that it is required to stay within.

For example, if you define a Fire usage type, you can mark Modifier Volumes as representing a region that is on Fire. Pawns can then be configured to be allowed to enter and move in Fire regions, or can be configured that they can only move inside of Fire regions. By default pawns will not be allowed to enter them.

Alternatively, you could define a usage type such as Shallow Water and create a modifier volume, with that flag set, that covers the region just below the surface of a sea. Pawns that have their Shallow Water usage flag set to Required would then be confined to pathfind and stay within that volume, and would not be able to travel down to deeper depths.

Costs

During a path search through the Nav Octree, Modifier Volumes allow designers to increase the cost of specific volumes and therefore discourage pawns from moving through them.

When a volume has a modified cost multiplier, the path distance is multiplied by the cost multiplier to calculate the expense of traversing a volume. The size of the additional cost of passing through a volume determines how far pathfinding will search for longer alternative routes that avoids it, before deciding to use it as part of the path.

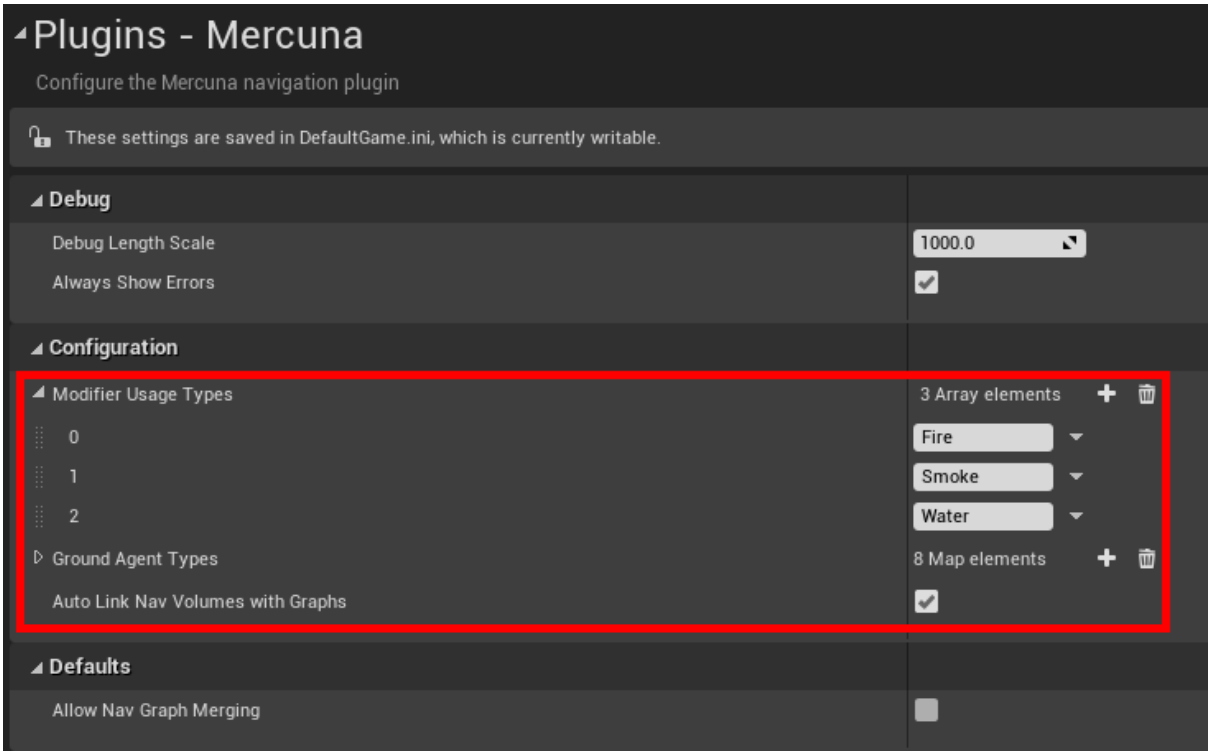
Due to the nature of the A* algorithm used for pathfinding, costs can only be increased and not decreased below 1.0x. Using a very high cost multiplier on a volume will mean that the pathfinder will search a long way for alternatives, and thus can considerably increase the computational cost of the path find. For this reason, the maximum cost multiplier that can be set on a volume is limited to 15.0x.

If multiple modifier volumes overlap, the maximum cost multiplier across the overlapping volumes is used.

Using Modifier Volumes

Defining usage types

Usage types are defined in the Mercuna page of the project settings. Up to 32 usage types may be defined.



Warning: removing usage types doesn't update the usage flags configured on existing modifier volumes or navigation components. Removing the Water usage type in the above example, means that all the usage flags on actors that previously corresponded to Water, will now be applied to the Glue usage type instead.

Creating modifier volumes

To create a modifier volume, add a **Mercuna Nav Modifier Volume** actor into the level, and size it appropriately. Only boxes aligned to the orientation of the Nav Octree are supported,

so you must not rotate the volume. To describe more complex boundaries, multiple Nav Modifier Volumes can be created.

Configuring modifier volumes

In the Mercuna section of the modifier volumes property panel, the cost and usage types for this volume can be configured.

Modifier volume properties

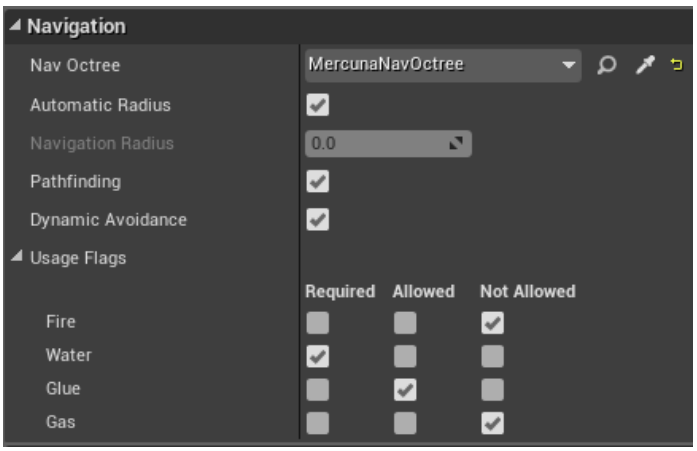
Whenever a volume is created, moved or resized in the Editor the navigation octree must be rebuilt for the change to be applied. Updating the usage types, cost multiplier or enabled setting does not require an octree rebuild.

Nav Modifier Volumes need to be associated with a nav graph. Similarly to Nav Octree Volumes, if the Mercuna project setting “Auto Link Nav Volumes with Graphs” is enabled (the default) new modifier volumes will be automatically linked to the nav graph if there is only one nav graph in the level when they are created. Otherwise you need to manually set the correct Octree in their properties.

When a Modifier Volume is associated with an Octree, the rotation of that octree will be applied to the volume.

Configuring navigation components

Costs are automatically taken into account during pathfinding and spatial searches, but by default agents will not enter modifier volumes that have any usage types set. To allow or require pathfinding through volumes of particular types, the corresponding usage flags can be set on the Mercuna 3D Navigation Component.



The default for each flag is Not Allowed. Required means the agent can **only** move in volumes with that usage type set. Allowed means the agent may, but does not have to, enter volumes with that usage type.

Runtime changes

At runtime, the settings on Modifier Volumes can be updated using the **SetEnabled**, **SetUsageFlags** and **SetCostMultiplier** functions on the Modifier Volume actor. These functions are exposed to blueprint.

Any paths through the changed modifier volume will be updated to reflect the change.

Changing these settings is relatively inexpensive, compared with the cost of a full octree rebuild for the modified volume.

Modifier volumes may be added, removed or moved at runtime, however this is nearly as computationally expensive as rebuilding the octree in the modified volume, so should not be done too frequently.

To add a modifier volume, spawn a Modifier Volume actor, set the usage types and cost multiplier, and then use the **AddToGraph** function to add the modifier volume to the octree.

To remove a modifier volume, simply destroy the actor, or alternatively call **RemoveModifierVolume** on the NavOctree.

Modifier Volumes can be moved and resized by changing their transform in the normal way. The orientation of Modifier Volumes must always match their Octree.

Restrictions

There is a limit to how many Modifier Volumes can be supported within each 64x64x64 voxel section of the Octree. The exact limit depends on the topology of the navigable space within the section of the Octree, but having up to 5 overlapping modifier volumes should not cause a problem.

If you hit an “Out of regions” error while building the octree then you should reduce the number of overlapping Modifier Volumes within the volume specified in the logged error.

Blueprint Functionality

The following functions are available on the **Mercuna 3D Navigation Component** and can be used to direct a pawn to move between goals:

- **MoveToLocation** - move to a position, stopping within end distance of the goal.
- **MoveToLocations** - move through a series of positions, visiting them in order.
- **AddDestinationLocation** - add an additional destination to the end of the current path. The agent must currently be moving to a position or series of positions.
- **MoveToActor** - move to within a given end distance of a destination actor, if the destination moves while the pawn is moving, the path will be updated to track the destination.
- **TrackActor** - get to and stay within a given distance of a target actor
- **Stop** - bring the pawn to a complete stop as quickly as possible.
- **CancelMovement** - immediately terminates the pawn’s current movement action
- **OnMoveCompleted** - a delegate that is triggered whenever a movement action is complete. Returns the result of the movement action as to whether it completed successfully, failed, was cancelled or was invalid (usually due to an invalid destination).

- **LookAt** - specify a target actor that the pawn will try to face as it moves. If no target is set then by default the pawn faces in the direction of movement.
- **CancelLookAt** - clear the LookAt actor.

- **IsReachable** (Latent action) - Test whether the pawn would be able to move to a given destination position from its current position.

- **SetEnabled** - Enable or disable navigation. This can be used when animation takes control of the actor, for example. When disabled any active move command is cancelled, and navigation is automatically re-enabled if a new move is requested.
- **PauseNavigation** - Pause navigation. Unlike disabling this does not clear the navigation command. This can be used when briefly handing control of the actor to an animation, for example.
- **ResumeNavigation** - Continue the move command that was previously running when PauseNavigation was called. Requesting a new move command automatically resumes navigation.
    - Before calling **ResumeNavigation** or **SetEnabled** after playing an animation you should ensure that the velocity of the actor is up to date in the movement component in order to ensure a smooth transition back from the animated motion. If using the Mercuna 3D Movement Component, use the **SetVelocity** function to do this.

The MercunaNavOctree actor offers the following Blueprint functions:

- **IsNavigable** - Does a point fall within navigable space
- **ClampToNavigable** - Clamp a position to the nearest point in navigable space
- **Raycast** - Perform a raycast through the navigable octree. If it fails, return the first point it hit
- **IsReachable** - Check whether there is a path from Start to End

- **FindPathToLocation** - Start an asynchronous path find from Start to End positions
- **FindPathToActor** - Start an asynchronous path find from Start position to End actor. Path will update as the destination actor moves
- **FindSplineToLocation** - Start an asynchronous path find from Start to End positions, and build a spline giving a smooth curve through navigable space along the resulting path.

- **Build** - Build all of a navigation octree at runtime. Normally used after procedural generation of a level is complete.
    - **OnBuildComplete** - a delegate that is triggered once a runtime Build has completed.
    - **OnBuildLowResReady** - a delegate that is triggered when a runtime Build has finished building the low resolution data, to indicate that short ranged navigation may now be possible.

- **RebuildVolume** - Rebuild the navigation octree within the bounds of the given actor.
    - **OnRebuildComplete** - a delegate that is triggered once a RebuildVolume has completed.
    - **OnRebuildLowResReady** - a delegate that is triggered when a staged RebuildVolume has finished building the low resolution data, to indicate that navigation may now be possible.

EQS

Mercuna currently offers three simple EQS tests. These tests are filters returning whether a point passes or fails, and do not score the points. The available **Mercuna 3D** tests are:

- **Navigable Volume** - test whether a point is within the seeded, navigable volume for a pawn of a given radius. Be aware that the points might be disconnected from the querying pawn, however this test is much cheaper than a reachability test.
- **Reachable** - test whether a point is reachable by a pawn of a given radius from its position within a specified path distance. If the path distance is set to 0.0, then a faster test is used that only filters out points with extremely long paths.
- **Raycast** - test whether there is a clear straight line path from the context to the positions.

Additionally, Mercuna offers one EQS test that modifies the positions of the test points:

- **Project** - test whether a point is in or close to navigable space. If the point is outside then clamp it back to the closest point within navigable space that is within a given search radius. The test fails and the point is not moved if the point is further than the search radius from navigable space.

Mercuna offers two EQS generators; they simply generate points without considering whether the resulting points are in navigable space or not. Add the Mercuna 3D: Navigable or Reachable EQS tests to filter out points in navigable regions or inside objects. The two available generators are:

- **Sphere** - generates points in concentric shells either uniformly or randomly distributed.
- **3D Ring** - generates points in rings in multiple vertical layers.

BT Nodes

Mercuna offers the following Unreal BT nodes:

- **Mercuna 3D Is Reachable** - Decorator - Test whether a given point is reachable by a particular actor
- **Mercuna 3D MoveTo** - Task - Move a pawn with a Mercuna navigation component to a specified location or actor read from the AI's blackboard. If the blackboard value is a location and the value changes while the node is running the path will attempt to update.

Debugging Problems

If you find that your pawn is not moving as expected, or at all, there are several debugging mechanisms available within Mercuna to help quickly identify problems.

Logging

Mercuna makes logs to the Unreal logging system to indicate progress and error conditions. By default, only Warning and Error logs are written. To enable progress information, add the following to DefaultEngine.ini:

```
[Core.Log]
LogMercuna=Log
```

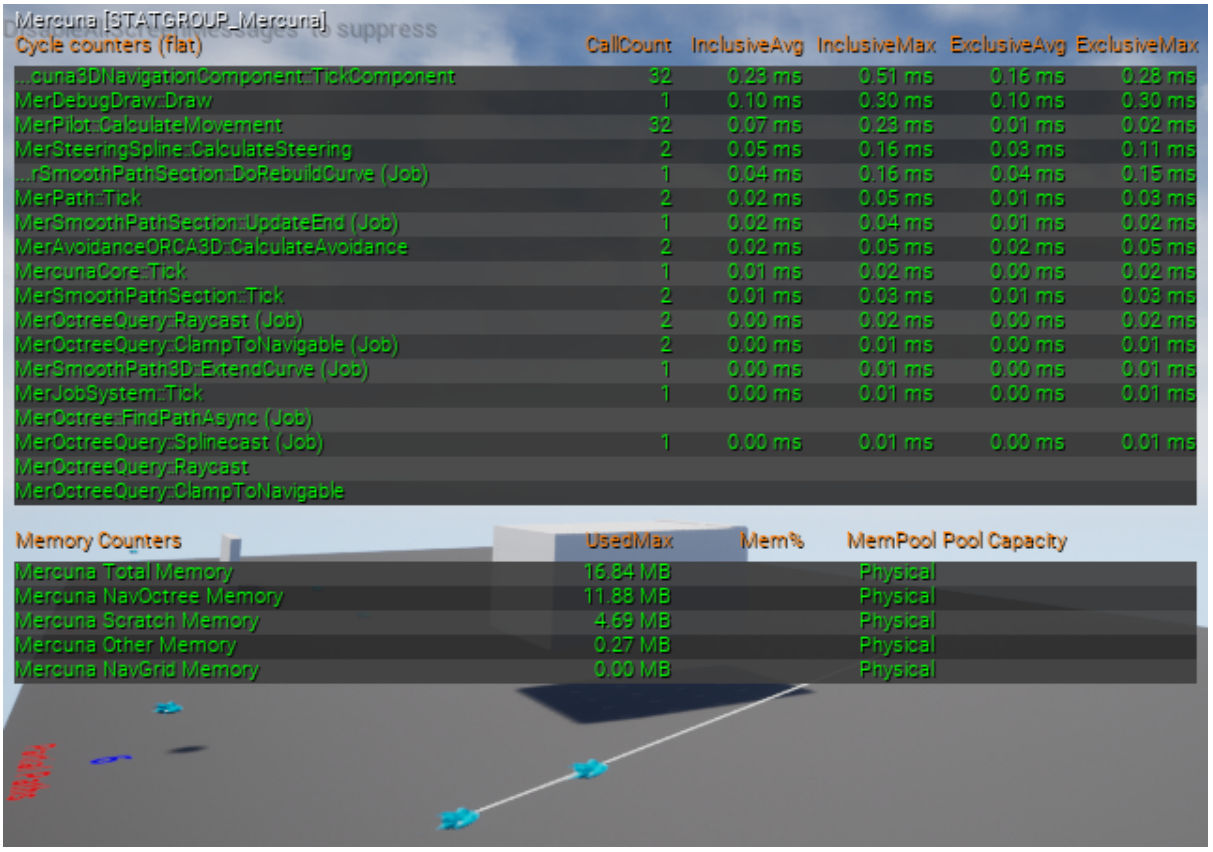
If more information is needed then Editor Preferences > Mercuna > **Enable Extra Logging**, can be enabled. This option persists between editor restarts. Extra Logging causes Mercuna to output all log messages, including additional debug messages, to a dedicated **Mercuna.log** file, located in the same directory as the standard Unreal logs.

To enable the **Mercuna.log** on standalone builds, add the following line to the Engine/Config/ConsoleVariables.ini:

```
mer.LogToFile=1
```

Profiling

Mercuna is integrated with Unreal Engine's inbuilt profiler. The current amount of CPU time Mercuna is using, as well as the current memory usage, can be seen by using the **stat Mercuna** console command.



Example Mercuna in-editor profile

If an entry in the profiling list has *(Job)* after it's name, then it is being run as an asynchronous job on a background thread and so will normally have no effect on the frame rate.

Mercuna is also integrated with Unreal Insights and a finer grained performance profile is available through Unreal Insights.

Memory Usage

All Mercuna memory allocations are tracked and the total memory used by different systems in Mercuna is displayed through **stat Mercuna**. Note that the Scratch memory usage is a fixed amount per concurrent thread using Mercuna.

Debug Actor

When trying to understand the actions of a particular pawn, it can be useful to set it as the Mercuna debug actor. This can be done by selecting the pawn and setting it as the Mercuna debug actor in the Mercuna toolbar menu. The same menu also allows the debug actor to be cleared.

On screen log messages will be displayed for the Mercuna debug actor and additional debug draw is available.

In Debug builds or if the define MER_DEBUGGING is set, then additional debug logs will also be recorded to Mercuna.log for the debug actor. If you report a movement problem to Mercuna support it is helpful to include these logs and a corresponding video capture of the problem.

Onscreen logging and information available about the debug actor

Debug Draw

In order to help understand the current movement of Mercuna controlled pawns the following debug draw is available from the Mercuna editor menu:

- **General**: Shows speed and velocity vector for all Mercuna actors
- **Obstacle bounds**: Shows blue spheres representing the dynamic obstacles registered with Mercuna
- **Paths**: Shows all paths currently being followed by Mercuna actors
- **Steering**: Shows the desired velocity vector for the current debug actor
- **Avoidance**: Shows various pieces of avoidance related debug draw for the current debug actor including the velocity obstacle cones and the ORCA planes, or the velocity field for context steering.

The actor debug draw can also be viewed through the Gameplay Debugger. This debug draw is replicated in multiplayer setups, allowing the Mercuna state from the server to be viewed by a client.

Navigation Octree

Debug Draw

In order to help understand Mercuna's representation of the geometry for navigation, you can draw the navigation octree, there are the following modes:

- **Unnavigable**: Draw the unnavigable part of the octree (in red).
- **Navigable**: Draw the navigable part of the octree (in green).
- **Cross section**: Draws a thin slice of the octree showing both navigable (in green) and unnavigable cells (in red).

If the Mercuna debug actor is set, then the navigation radius of that actor is used to determine which cells are treated as unnavigable.

There are also the following advanced visualisation options:

- **Pathfind**: This will give a visualization of the last pathfind through the octree made by the Mercuna debug actor, or by a Mercuna Nav Testing actor. Explored cells are shown in green, cells that are on the path are shown in cyan.
- **Reachability**: This shows the cells that were included in the last reachability query. Like the pathfind, it gives a visualization of how the test flooded through the octree from the test point.
- **Hierarchical Regions:** Shows how the octree is split up into different regions for the purpose of hierarchical pathfinding.

Troubleshooting

Once the octree has been built, the octree debug draw can be used to check the geometry has been built into the octree correctly. Switch on **Unnavigable** debug draw through the Mercuna menu, and you should see red boxes, representing unnavigable regions, around your geometry:

MERCUNA

Migration Issues

v2.4

It used to be possible to turn any actor into a Mercuna Nav Seed by adding a Mercuna Nav Seed component. We have retired the seed component, as this allowed some significant performance improvements. Now only Mercuna Nav Seed actors seed the nav octree.

Known Issues

Known issues in Mercuna 3D Navigation:

- Rotating the octree actor by any direction other than around the z axis can result in agents orientation getting confused