MERCUNA

Mercuna 3D Navigation
Unity User Guide

v2.0

MERCUNA

Contents

Installing

Mercuna Singleton
- Automatic Setup

Configuring the Navigation Octree
- Manual Setup
- Setting up the Nav Octree
- Nav Volumes
- Nav Seeds
- Building the octree
- Memory usage and Performance

Pathfinding
- Path Testing Object

Creating a Mercuna navigated Agent

Setting a Flight Style
- Steering

Movement

Control Interface

Example Scripts

Advanced Features
- Modifier Volumes
  - Usage flags
  - Costs
  - Creating modifier volumes
  - Configuring modifier volumes
  - Configuring navigation components
  - Restrictions

Debugging Problems
- Logging
- Debug Object
- Debug Draw

Installing

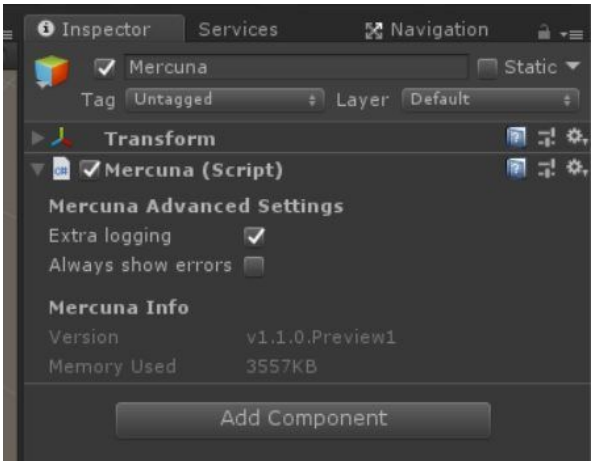
The Mercuna plugin requires Unity 2019.4 or newer, if you're using an older version you must upgrade your project first.

To install simply unzip the archive and import the Mercuna package into your game project through the **Assets/Import Package/Custom Package** menu in the Unity Editor. The plugin will now be loaded automatically when you load your game project.

Mercuna Singleton

Any scene in which you wish to use Mercuna must contain an instance of a Mercuna singleton object. The singleton is used by Mercuna to internally load and access the Mercuna native libraries.

In addition it contains various advanced Mercuna project settings and information about Mercuna, including memory usage.



Settings and information available on the Mercuna singleton

Automatic Setup

The simplest way to get started is to use the **Tools/Mercuna/Add Mercuna To Scene** menu option. This will automatically add a Mercuna Singleton object, Nav Octree, Nav Volume and Nav Seed to your level.

You should check that the Nav Volume encompasses the entire volume that you wish your 3D agents to navigate within, increasing or reducing the size if necessary.

Also move the Nav Seed so that it is in free space that you want your agents to be able to navigate through.

See below for more details on setting up the Nav Octree, Nav Volumes and Nav Seeds.

Configuring the Navigation Octree

The Mercuna Nav Octree is used to find paths for agents through 3D space, much like a 2D Nav Mesh is used to navigate characters moving over terrain.

Nav Octree is generated in the level everywhere that is within a **Mercuna Nav Volume**. Multiple nav volumes can be present in one level, and if they overlap then agents can navigate seamlessly between them. All space outside of the Nav Volumes is treated as unnavigable.

Manual Setup

There can only be a single Nav Octree in a scene. It can be created by adding a Mercuna Nav Octree component to an empty game object.

Setting up the Nav Octree

The octree generation parameters to be used in this level are configured on the Mercuna Nav Octree game object. Upon creation, the parameters are set to default values.

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 agents 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 0.3, a minimum radius of 2 and a maximum radius of 5, entities of radius between 0.6 and 1.5 can be accurately navigated through the level. Agents that are smaller than the minimum radius will navigate successfully, but might not take paths through small gaps they could fit through. Agents that are larger than the maximum radius can't be navigated properly – paths might make them collide with geometry.

Nav Volumes

To set up a nav volume, add a Mercuna Nav Volume component to an empty game object, and position and scale it to the correct size. Only axis aligned volumes are supported, so you must not rotate the volume.

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.

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

Nav Seeds

In order to identify which regions should be considered navigable Mercuna requires you to place objects with the **Mercuna Nav Seed** component into your scenes. 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 agent 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 agents 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 agents to move, a seed must be placed in each separate area.

Building the octree

The octree must be built after it is first configured or after the level geometry has changed. Do this by selecting Build octree from the Tools/Mercuna menu. An on screen notification displays the progress of the octree construction.

If full Mercuna logging is enabled (see 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.

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 agent radius:** Larger maximum agent radiuses take longer to generate and slightly increase memory usage.

Pathfinding

Finding paths through the nav octree can be done implicitly by making your agent 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 the MercunaNavOctree C# script) 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.

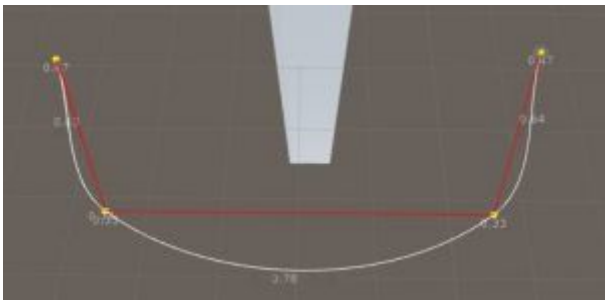
For longer paths Mercuna uses hierarchical pathfinding. An approximate path is found through a simplified representation of the level and then a detailed path find is performed guided by the approximate path. This allows much longer paths to be found than would be possible with simple A* pathfinding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

In order to easily debug and understand pathfinding problems a **Mercuna Nav Testing** object can be used to generate test paths. Simply add a Mercuna Nav Testing component to an object, and set any other object in the scene as the **End**. When you do this a test path will be drawn connecting the two objects. This path will update when either object 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 agents would take.



A test path from a Mercuna Nav Test object to target

Creating a Mercuna navigated Agent

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

- **Mercuna Obstacle** – this marks the agent as a dynamic obstacle for the purpose of 3D navigation.
- **Mercuna 3D Navigation** – this component provides the agent with navigation capabilities.
- **Mercuna Move Controller** – used to actually move the agent. Can be modified to fit your game's specific movement model.

Setting a Flight Style

How an agent moves depends on how its flight style is configured. The flight style options are set on the **Mercuna 3D Navigation** component. The options include:

- **Move Only in Look Direction** – Whether to restrict movement to facing direction. This stops agents from flying backwards when turning around.
- **Stop at Destination** – Whether should slow down in order to stop at its destination.
- **Dynamic Avoidance** – Whether to do avoidance against Mercuna obstacles.
- **Smooth Paths** – Whether to use spline based paths, or simple straight line segment paths. See Steering section for more detail.
- **Height Change Penalty** - How much the pawn should avoid unnecessary changes in height while moving.
- **Usage Flags** - Restrict or allow pathfinding through usage types applied to modifier volumes.

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 available space and define a continuous velocity curve that can be followed precisely. Mercuna navigates the agent down the spline path taking into account the maximum speed and acceleration configured on the pawn.

Whereas the pathfind is performed when an agent is given a destination, the spline is generated on demand. If the agent is pushed, or has to avoid other objects causing it to move away from the spline, then the spline and the underlying path are automatically regenerated.

Spline based steering can be disabled to fall back to previous simple steering by turning off **Smooth Paths** in the flight style options. However, the simple steering method suffers from the problem that agents 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.

Mercuna offers dynamic obstacle avoidance to ensure that agents don't collide while moving. Any game object with a **Mercuna Obstacle Component** 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.

Movement

In order for the Mercuna navigation component to drive the moment of an agent, the agent needs to have a suitable movement controller. A simple movement controller is provided – the **Mercuna Move Controller**. This is suitable for a variety of 3D flight styles.

Custom movement can easily be implemented by customising the controller. By default the Mercuna Move Controller provides a Newtonian flight model for a pawn moving freely in space. It allows you to configure:

- **Max Speed** – The maximum speed the agent may move at.
- **Max Forward Accel** – The maximum acceleration of the agent in the z-direction (in the agent's local coordinates).
- **Max Backward Accel** – The maximum acceleration of the agent in the -z direction (in the agent's local coordinates).
- **Max Sideways Accel** – The maximum acceleration of the agent in +/-x direction (left/right) and +/-y direction (up/down) (in the agent's local coordinates).
- **Max Ang Speed** – The maximum angular speed the agent may rotate at, in degrees/sec.
- **Max Ang Accel** – The maximum angular acceleration allowed on each axis, in degrees/sec^2.

Control Interface

The following functions are available on the Mercuna Navigation component and can be used to direct an agent to move between goals:

- **NavigateToLocation** – move to a position, stopping within the end distance of the goal.
- **NavigateToLocations** – move to a sequence of positions
- **NavigateToObject** – move to within a given end distance of a destination object, if the destination moves while the agent is moving, the path will be updated to track the destination.
- **Track** – get to and stay within a given distance of a target object.
- **Cancel** – immediately terminates the agent's current movement action.
- **NavigateToAdditionalLocation** – Add an additional location after the current moves have completed

All the functions (except Cancel) take an optional MoveCompleteDelegate delegate that is triggered whenever a movement action is complete. The delegate receives a boolean that indicates whether the move command was completed successfully or whether it failed.

The Mercuna Nav Octree actor offers the following query C# 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 between two points.

As well as the following pathfinding functions:

- **FindPathToLocation** – Start an asynchronous path find from start to end positions.
- **FindPathToObject** – Start an asynchronous path find from start position to goal object. Path will update as the destination object moves.
- **FindSmoothPathToLocation** – Start an asynchronous path find from start to end positions, and build a smooth curve through navigable space along the resulting path.

Example Scripts

Mercuna comes with a set of example scripts that demonstrate the different ways that Mercuna can be used. These can either be used directly or can be used as the basis of your own movement behaviors.

- **NavigateToLocation** – sends an agent to a destination location.
- **NavigateToObject** – sends an agent to a destination object.
- **MoveBetweenObjects** – will move an agent around an array of destination objects. Can set to do so either sequentially or randomly.
- **RandomWander** – causes an agent to wander randomly around the navigable octree.
- **CircleAround** – circle around a target object. This is a more advanced behavior that queries the octree for potential positions and then scores them to find the best next position to move to.

Advanced Features

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

Usage Types allow you to mark what a modifier 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 consider UsageType0 as Fire, you can mark Modifier Volumes as representing a region that is on Fire. By default pawns will not be allowed to enter the region. 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.

Alternatively, you could use a usage type to represent Shallow Water and then create a modifier volume, with that flag set, that covers the region just below the surface of a sea. Pawns that have their corresponding 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.

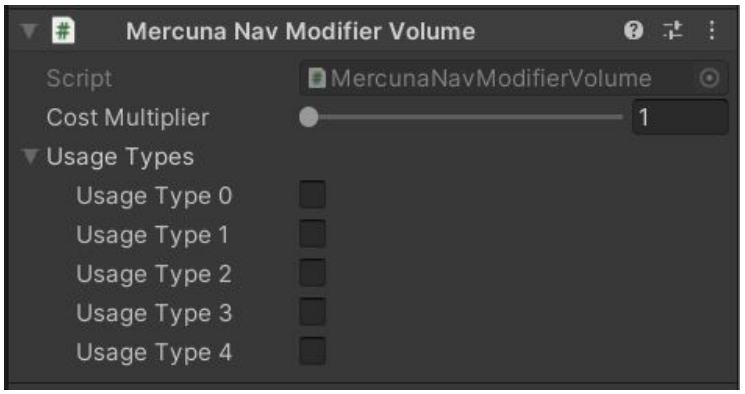
When a volume has a modified cost multiplier, the path distance is multiplied by the cost multiplier to calculate the expense of traversing a volume. The size of the additional cost of passing through a volume determines how far pathfinding will search for longer alternative routes that avoids it, before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

Creating modifier volumes

To create a modifier volume, add a Mercuna Nav Modifier Volume into the level, and scale it to the required size. 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

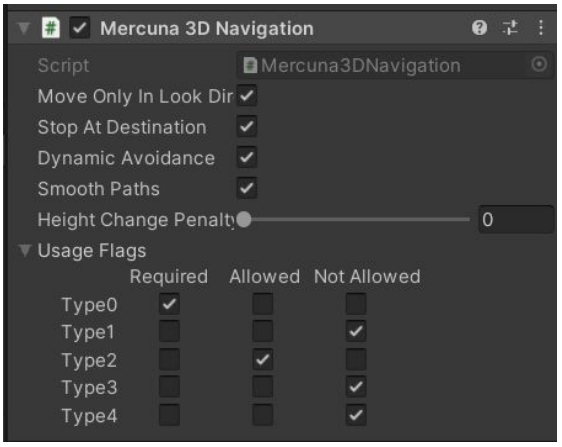


Modifier volume properties

Whenever a volume is created, moved or resized in the Editor, or a property is changed the navigation octree must be rebuilt for the change to be applied.

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.

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.

Debugging Problems

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

Logging

By default Mercuna makes logs to the Unity logging system to indicate warnings and error conditions. If more information is needed then the advanced **Enable Extra Logging** option on the Mercuna singleton, 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 Unity logs.

Debug Object

When trying to understand the actions of a particular agent, it can be useful to set it as the Mercuna debug object. This can be done by selecting the agent and setting it as the Mercuna debug agent in the Mercuna toolbar menu. Only objects with a Mercuna Navigation component can be set as the debug object.

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

Onscreen logging and information available about the debug object

Debug Draw

In order to help understand the current movement of Mercuna controlled agents, Mercuna offers various debug draw options, both for the agents and for visualising the octree.

For agents the following debug is available:

- **General** – Draw the name and speed of any Mercuna controlled agent, as well as an (green) arrow showing the current velocity.
- **Obstacles** – Draw a sphere showing all objects that are currently being considered as dynamic obstacles by Mercuna.
- **Paths** – Draw the paths currently being followed by agents.

If the debug object is set then just for that agent there is extra debug draw:

- **Steering** – Draw the desired velocity (red arrow) and desired acceleration (yellow arrow).
- **Avoidance** – Draw nearby obstacles, ORCA velocity cones and planes.

The octree can be visualised in different ways:

- **Unnavigable**: Draw the unnavigable part of the octree (in red).
- **Navigable**: Draw the navigable part of the octree (in green).
- **Navigable** and unnavigable: Draw both.

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

There are also options to draw particular cells in the octree:

- **Pathfind**: This will give a visualisation of the last pathfind through the octree made by the Mercuna debug object, or by a Mercuna Nav Testing object. Explored cells are shown in blue, cells that are on the path are shown in green.
- **Reachability**: This shows the cells that were included in the last reachability query. Like the pathfind, it gives a visualisation of how the test flooded through the octree from the test point.