NaviPro 1.0 for Unity Documentation

Seagull Entertainment
Contact: info@seagullentertainment.com

Contents

1 Introduction

NaviPro is a professional path finding and navigation solution for Unity. It is designed for games and similar real-time applications. Using the latest research in navigation, it features natural agent behaviour and very fast path finding, even in large and complex environments.

Its main features include:

- Fast agent simulation running multi-threaded native code, supporting hundreds of agents
- Natural and customizable agent movement, including local avoidance
- Crowd flow for dense crowds
- Fast and precise construction of the navigation mesh using custom meshes or integration with the Unity NavMesh API
- Fast modification of the navigation mesh using obstacles and platforms, allowing precise and large scale changes at a low cost
- An automated multi-layer system for the navigation mesh, allowing overlapping areas such as bridges and even multiple floors without any additional setup
- Free transformation of the navigation mesh and its agents, including translation, rotation and scaling
- Sensible and intuitive settings without any required tweaking for performance or quality
- Varying agent sizes and other agent properties
- Optional direct control of agents
- Manual path finding queries

2 Installation

To start using NaviPro, simply download it from the Asset Store inside Unity as usual. This will create a folder called NaviPro in your project Assets folder. To view the demo scenes, you will need Unity's Standard Assets. These are also available on the Asset Store by Unity Technologies.

Although it is not strictly required, you may want to move the Plugins folder contained in NaviPro to the Assets folder of your project to comply with the Special Folder names.

NaviPro requires Unity version 2019.1 or newer, since it relies on the Jobs system for its multithreaded simulation.

3 Quick Start Guide

The first step in setting up a working navigation system is creating a surface that can be used for path finding and that can be traversed by Agents. This is commonly referred to as a Navigation Mesh. In NaviPro, such a surface is represented by a Sector. There are two ways to create a Sector:

- From a custom Walkable Mesh: this is the most accurate version, ensuring complete control over the outcome.
- From Unity's native NavMesh API: this is the easiest way to quickly compose a surface from multiple objects.

The default creation method uses a custom Walkable Mesh. For more detail on the creation methods, see section 4.1.

Create an empty game object and give it a NaviProSector component. This object will represent the Sector that your Agents can traverse, so name it accordingly. For this guide, change the Creation Method of the NaviProSector to 'From Unity NavMesh'. This can be done by selecting the new object and choosing the entry from the drop down menu in the Inspector window. You can leave the other settings at their default values.

The child objects of the new Sector object will contain the parts that make up the Navigation Mesh. Since the Sector uses the 'From Unity NavMesh' creation method, you can add any combination of meshes and further modify these using boxes. To start, right-click on your Sector object and from the '3D Object' category choose 'Plane'. This will add a planar mesh as a child object. To register that this plane should be part of the Sector, give the new object a NaviProUnitySurface component. You can scale the object to make the plane larger or add more planes or other geometry to create a more interesting scene. The NaviProUnitySurface will add any geometry from the object it is placed on and its child objects. Any other geometry placed in your Sector is ignored, so be sure to add a NaviProUnitySurface when required.

For a lot of projects, it is not required to manually create a Sector object. If you do not need multiple Sectors and do not need to move or rotate your Sector, you can simply skip the above step and use the Base Sector instead. See section 4.1 for more information.

With the Sector set up, it is time to add an Agent. Create an object to use as your Agent and add a NaviProAgent component. In the hierarchy, Place the Agent object as a child of the Sector. This will indicate the Agent can move in that Sector.

To make the Agent move, it needs a goal. For this guide, add the NaviProExampleAgentGoal component from the supplied example scripts to your Agent object. This script calls the SetGoal function described in section 5.1. Create a new GameObject as your goal position somewhere in your scene and set it as the Goal in the NaviProExampleAgentGoal component. When you press play, the Agent will begin moving towards the assigned goal.

4 Components

This section describes the components that drive the navigation system. These components should be added to Game Objects as normal.

No separate component is needed to run the simulation.

4.1 Sectors

A NaviPro Sector represents a Navigation Mesh, the surface upon which Agents can move. Components such as Agents, Obstacles and Platforms can be part of a Sector by creating them as child Game Objects in the hierarchy. Each Sector is treated separately, with Agents from different Sectors not interacting in any way.

Sectors can be translated, rotated and scaled individually by adjusting the transform of its Game Object. This allows the creation of moving Navigation Meshes and Navigation Meshes with a different orientation, providing support for 2D scenes and even allowing the Agents to move around upside down. The Agents, which are child objects of the Sector, will be transformed along with the Sector.

Different Sectors are allowed to overlap. As mentioned above, Agents from different Sectors do not interact in any way, which means they can pass through each other unhindered. Agents can be moved to another Sector by changing their parent Sector in the hierarchy. This combination of features supports, for example, Agents as flying units that can land and continue moving on the ground, where the sky and ground are overlapping Sectors with their own Navigation Mesh.

Base Sector

In addition to any Sector components, all scenes may also contain a Base Sector. Any components that are not a child of a Sector component will be included in the Base Sector. The Base Sector Settings (see section 4.4) are used for its creation. If the scene only needs one Sector and it does not need to move, scale or rotate, it is usually easier to leave out Sector components entirely and simply add a NaviPro Settings component to control how the Base Sector is created.

Creation Methods

A Sector can be created in two ways, either by a custom Walkable Mesh or by converting geometry using Unity's native NavMesh API. These are listed as the Creation Method property of the NaviPro Sector component. Each has its advantages and disadvantages as is discussed in the following sections. Once a Sector has been created, it can only be modified using Obstacles and Platforms. Adding a mesh to a Sector or removing a mesh from a Sector during runtime will have no effect.

Walkable Mesh

Creating a Sector using a custom Walkable Mesh is the simplest method of creating a Sector. It uses a separate mesh, requires no additional settings and produces a reliable result.

To do this, add a Game Object with a mesh, with the Game Object as a child of the Sector, and give it a NaviPro Walkable Mesh component. Each mesh becomes a separate, disconnected part of the Navigation Mesh. The mesh can be imported from an external file or otherwise created as an asset in Unity. Note that the NaviPro Walkable Mesh component relies on the presence of a MeshFilter component to find the mesh, where other common components for meshes, such as a MeshRenderer or Mesh Collider are not required. Without a MeshRenderer, the mesh can remain hidden from view while still being used for navigation.

The advantage of this creation method is that it requires no extra settings and produces exactly the Navigation Mesh that is provided by the mesh. The downside is that it requires a custom mesh, which will need to be updated separately to reflect changes in a scene.

Unity Surface and Unity Box

The alternative to using a custom Walkable Mesh is the creation of a Navigation Mesh through Unity's NavMesh API. This method converts the supplied geometry into an intermediate voxel representation from which a Navigation Mesh is extracted.

A combination of mesh or terrain geometry can be used as input, alongside modifier boxes to add or remove sections of the Navigation Mesh. Meshes and terrain can be specified as input by adding a NaviPro Unity Surface component, which collects all Terrain and MeshFilter components from the game object and all its children. The modifier boxes can be created by adding a NaviPro Unity Box component, which requires no other components. Note that other common components for meshes, such as a MeshRenderer or Mesh Collider are not required. Without a MeshRenderer, meshes can remain hidden from view while still being used for navigation.

The advantage of this creation method is that it allows scene creation inside Unity without being restricted to a predefined mesh. The disadvantage is that it takes longer to create the Sector, since the creation of a voxel representation and conversion to a mesh requires additional computation. It also results in a less precise Navigation Mesh, due to the inaccuracy of the voxel to mesh conversion.

Since Unity does not expose a fully accurate version of the resulting mesh via its API, some height difference between the generated Navigation Mesh and the supplied meshes may occur on sloped surfaces. To ensure a correct height placement of Agents, a separate query is performed for each Agent to place them on the more accurate internal mesh. The height difference may also have an effect on overlapping Platforms or Obstacles and goal placement. If any height difference should occur, the incorrect height is visible in the boundary view of the NaviPro Debug component. See the Unity documentation on NavMesh.CalculateTriangulation for more information.

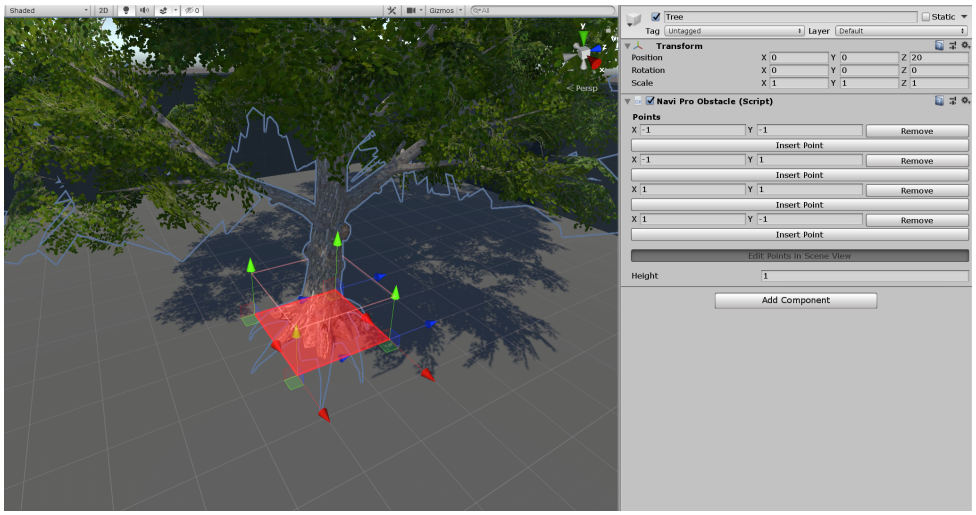
The settings provided in the inspector for the NaviPro Sector component provide some control over the generation of the voxels and the resultant Navigation Mesh. These settings are similar to the settings provided by Unity for the Agents in the NavMesh API.

4.2 Obstacles and Platforms

A Sector can be modified using Obstacles and Platforms. Obstacles mark an area as impassable and Platforms can be used to extend the Sector. In the case of overlap, Obstacles will always block Platforms. Obstacles and Platforms can be created, disabled, re-enabled and removed at any time. While these are useful for dynamic objects, they can also be used for static changes. After creation, they do not create any additional performance cost compared to a Sector with the changes baked in.

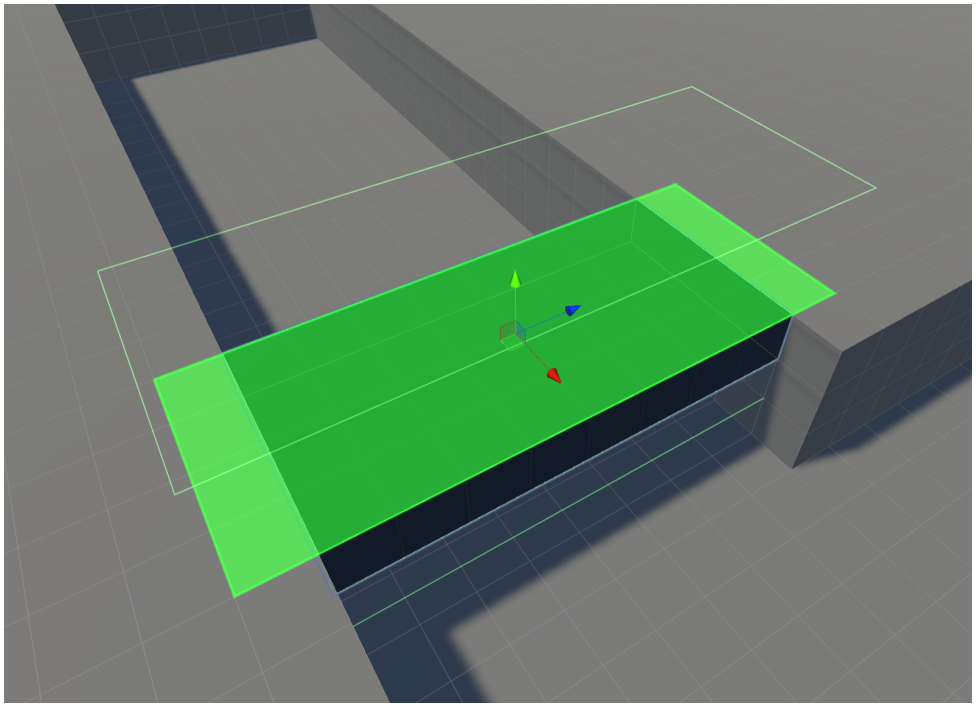
An Obstacle or Platform can only modify one Sector and should be a child object of that Sector. When creating one using the Editor or through scripting, always check whether the Obstacle or Platform has its parent set to its Sector.

Both Obstacles and Platforms are created by adding a NaviPro Obstacle or NaviPro Platform component to a Game Object. This component allows you to modify a shape using points. This shape does not have to be convex, but its edges should not intersect or touch each other. The shape can have any arbitrary size. The object containing the Obstacle or Platform can be rotated to fit the shape to sloped surfaces. A height setting is provided to control what parts of the Sector are affected by the shape. This height is represented in the scene view as an offset outline of shape, both above and below the shape. It should be adjusted to contain the entire surface you want to be affected, but no additional layers.



Modifying the points for an Obstacle representing a tree

Please remember that the edges of a Sector are very strict. If there is a small gap between two parts of a Sector, that gap is not traversable. When creating platforms, it is good practice to avoid this by overlapping the shape of the platform with the Sector, as shown in the figure below.



Platforms should overlap the Sector a bit to avoid any potential gaps

Once created, Obstacles and Platforms cannot be moved or modified. Changing the position or shape will have no effect until either the component is disabled and re-enabled or the Apply() function is called on the component. Doing so will simply destroy it and immediately create it again. Agents standing on such a Platform will be disabled.

An Obstacle or Platform may never modify multiple layers of the Sector at once. They can only be applied to parts of the Sector that are locally flat (see section 6.1).

Obstacles and Platforms are allowed to overlap each other. They will simply be applied in the order

they are created. Removing a previously created Obstacle or Platform returns the Sector to the state it was in before, without leaving any changes behind.

Creation and removal is usually quite fast, but it is important to remember that it is not entirely free. The cost is strongly dependent on the local surface complexity and the size and complexity of the given shape. Any modification should only be done when necessary. Also note that removal of an Obstacle or Platform may require that overlapping Obstacles or Platforms be re-evaluated. This is done automatically, but may take some additional computation time. If an Obstacle or Platform is planned for later removal, it is recommended to keep overlapping and touching Obstacles and Platforms to a minimum.

4.3 Agents

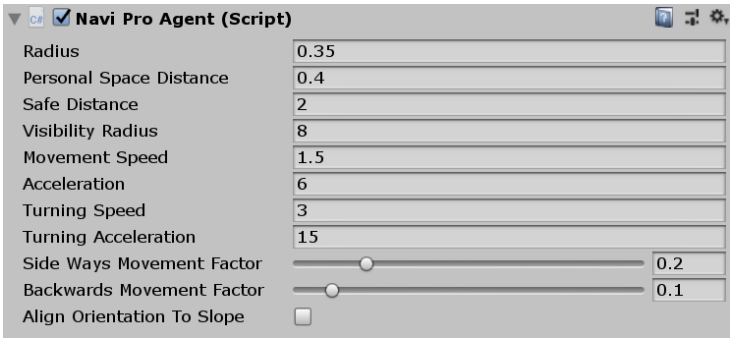
Agents are autonomous moving entities in a Sector. These can be pedestrians, robots, vehicles or anything else that can move. Agents are capable of finding a path to a given goal, moving and steering across the Sector and evading walls and other Agents while doing so. They are modelled using a circular shape in 2D for collision.

An Agent can be created by adding a NaviPro Agent component to a Game Object. An Agent is restricted to moving in one Sector and should be a child object of that Sector. It is possible to move an Agent to a different Sector by changing its parent.

Agents can be given a goal to move towards. They will find the fastest path to this goal and proceed to move along this path. When an Agent's path to a goal is blocked by an Obstacle or when a faster path is opened by a Platform, the Agent will recompute its path automatically. For more details on Agent behaviour towards goals and how to set them, see section 5.1. When a path changes or a goal is reached, a notification is provided through the API (see section 5.4).

Agents accelerate and steer towards their intended direction of motion and will attempt to stop at their goal. They will steer away from walls and other Agents. In a dense crowd an Agent will opt to follow the crowd more, rather than steering towards their personal intended direction.

Agents have various properties to control their size and behaviour. These properties can be adjusted at any time for each Agent individually.



The NaviPro Agent component and its properties at default settings.

The adjustable properties are:

- Radius: the physical radius of the Agent for determining collision.
- Personal Space Distance: the range within which an Agent prefers to have no other Agents. Less weight will be given to this distance when the Agent is in a dense crowd.
- Safe Distance: the preferred distance to walls or other boundaries of the Sector in an open area. This setting does not limit the ability of Agents to enter narrow corridors and does not

influence their choice of a path. It is mostly represented in a generated path, but changes to the Safe Distance do not cause the path to be updated. Thus, changing it while following a path may have less result than expected.

- Visibility Radius: the range within which the Agent can see and react to other Agents. Higher values allow the Agent to respond better to avoid collisions, but will be more costly. Also note that Agents will only react to other Agents when a line of sight can be drawn.
- Movement Speed: the maximum movement speed in the forward direction. An Agent will accelerate towards this speed.
- Acceleration: the maximum acceleration to increase the current Movement Speed.
- Turn Speed: the maximum turning speed of the Agent. Agents use this to change their orientation.
- Turn Acceleration: the maximum acceleration to increase the current Turn Speed.
- Sideways Movement Factor: this determines the degree in which an Agent can move sideways without turning first. This is set as a fraction of the Movement Speed.
- Backwards Movement Factor: this determines the degree in which an Agent can move backwards without turning first. This is set as a fraction of the Movement Speed.

4.4 Settings

The NaviPro Settings component can be created to adjust the behaviour of NaviPro. Only a single instance of NaviPro Settings should be present in a scene. If it is not present, the default settings are used.

Base Sector Settings

The Base Sector Settings are the same as the settings of a Sector. It is possible to choose between generating the Base Sector by a custom Walkable Mesh or by converting geometry using Unity's native NavMesh API. When using Unity's NavMesh API, the settings provided in the inspector for the NaviPro Sector component provide some control over the generation of the voxels and the resultant Navigation Mesh. These settings are similar to the settings provided by Unity for the Agents in the NavMesh API.

Update Settings

The Update Settings are used to control when the simulation will run and whether the resulting movement should be interpolated. More detail is provided in section 6.6.

4.5 Debug

The NaviPro Debug component is provided as a tool to visualize Sectors. Simply add a NaviPro Debug component to your scene to show the Boundary or outline of the Navigation Mesh of all Sectors in the scene. In the inspector, you can also choose to show the Medial Axis. These are the middle lines of all areas of the Navigation Mesh.

When enabled, the NaviPro Debug component will draw the boundary during play mode. Through the inspector, it is also possible to show the same information outside of play mode, which can be a useful tool for scene construction.

5 API Reference

While most basic functionality of NaviPro is available through its components, some parts can only be accessed through its API. This functionality is detailed below.

5.1 Setting Goals

A goal can be set using the SetGoal function in NaviPro Agent. Setting a goal using SetGoal will override any previous goals. You can also add additional goals to be followed after the first, using AddGoal. Goals can also be cleared using ClearGoals.

SetGoal and AddGoal are both called using a 3D position. This position is snapped to the nearest point on the Sector when possible, but can also be placed outside of it. Agents will make a best effort to reach the goal and will otherwise choose to move to the nearest possible position.

Wherever a goal is placed, an Agent will try its best to reach it. Only when the goal is reached will the Agent become idle again or will it seek out its next assigned goal. When an Agent is unable to reach its current goal, it will move to a nearby point it can reach, based on the distance it needs to travel and the distance to the goal. Reaching this nearby point does not count as reaching the goal, where a separate notification is provided for the event (see section 5.4). If a change in the Sector allows the Agent to get closer to the goal or reach it, it will move to do so automatically.

When multiple Agents are assigned to move to the same goal position, they are likely to get in each other's way. There is no automated system to prevent this. It is therefore preferable to assign a different goal to each Agent, such as a random point in an area.

Placing a goal at a point outside the Sector is a lot more computationally expensive. Pathfinding to find a point on the Sector near the goal is a more complex problem than regular path finding. Updating such a path upon changes in the Sector also requires more frequent re-evaluation, since it is difficult to rule out such a change does not lead to a new path that can reach the goal.

5.2 Moving an Agent

During simulation, the transform of a NaviPro Agent is driven entirely by the system. Manual changes to the transform are ignored and overwritten at the next timestep. To move an Agent, you should instead call the MoveTo function in NaviPro Agent. This will move the Agent at the start of the next timestep. Note that moving an Agent will invalidate its current path, but an Agent will retain its current goals. A new path to the current goal will be automatically recomputed.

The MoveTo function is called using a 3D position and rotation parameter. This position is snapped to the nearest point on the Sector when possible. If the Agent is moved off the Sector, it will be disabled (see section 6.3). The rotation may be adjusted to keep the Agent upright.

An Agent can also be moved to a different Sector. This can be done by changing its parent, available in scripting through the Transform component, so that the Agent becomes a child of the other Sector object.

5.3 Local Speed

Aside from adjusting the position and rotation of each Agent, the simulation also provides a local speed. This represents the forward and sideways components of the speed, relative to the current orientation of the Agent, given as a 2D vector. This speed can be very useful for animation. It is interpolated between timesteps, much like the Agent transform.

The local speed can be obtained by calling the GetLocalSpeed function in NaviPro Agent.

5.4 Agent Notifications

While moving to a Goal, an Agent keeps a state with regard to its current goal and path. For instance, when an Agent is first created its goal state will be 'No Goal Set' and its path state will be 'Not Following Path'. When a goal is given and the Agent begins to move towards that goal, these will change to 'MovingTowardsGoal' and 'FollowingPath' respectively. When an Agent has reached its goal, the goal status will briefly turn to 'Goal Reached' before changing 'No Goal Set' if no further goals are available.

For any significant changes of state or other event, a callback function is available as a notification that something has changed. The following notifications are available:

- Nearing goal, slowing down: the Agent is very close to the goal and is slowing down to come to a stop at the goal position.
- Nearing goal, moving to next: the Agent is very close to the goal and, just before slowing down, starts moving to the next goal.
- Goal reached: the Agent has reached the goal position and has no further goals assigned. Usually this also means the Agent has come to a full stop, but it may still be moving a little bit.
- Nearing point near goal, slowing down: this is similar to 'Nearing goal, slowing down', but the Agent could not reach the exact goal.
- Point near goal reached: this is similar to 'Goal reached', but the Agent could not reach the exact goal. It will continue to try moving to the goal as soon as a new path is found. This is indicated by the 'Improved path found and updated' notification.
- Path obstructed and updated: the path the Agent was following was changed by an Obstacle or Platform. A new path to the goal has been computed. This does not always mean the path ahead of the Agent was blocked or that the Agent needs to take a detour. Because of the way paths are computed and stored, this notification may also appear when the Sector near the path changes.
- Path lost and updated: the Agent is no longer able to see the closest point on its path, meaning it has deviated too far. A new path to the goal has been computed.
- Faster path found and updated: due to a change in the Sector, a faster path has been found to the goal. A new path to the goal has been computed.
- Improved path found and updated: due to a change in the Sector, the Agent has found a path that leads it closer to the goal or that allows it to reach the goal. This notification is only given when the Agent could not reach its goal before. A new path to the goal has been computed.
- Moved Agent, path updated: the Agent has been moved by the user, invalidating its old path. A new path to the goal has been computed.
- Agent disabled: the Agent could not be placed on the Sector and has been disabled.

5.5 Custom Agent Control

Normally, Agent behaviour is handled entirely by NaviPro, but it is also possible to control an Agent manually. This can be done by calling the SetCustomControlCallback function in NaviPro Agent. The callback function supplied to this function must return a Vector3 and accept a parameter of type NaviPro.AgentControlData. This data structure contains the same information used by NaviPro to guide Agent behaviour, including information on nearby Agents, the nearest obstacle and more. The return value is the desired movement speed vector in world space. The Agent will steer and accelerate to match this speed vector as normal. A manually controlled Agent will not avoid other

Agents, but other non-controlled Agents will avoid this Agent as normal.

While this method of control still uses many of the systems and optimizations that NaviPro provides, the callback cannot be run multithreaded and results in some additional overhead. The control callback function is only used to determine the desired speed of the controlled Agent.

An example script is provided as a simple demonstration on how an Agent can be controlled directly.

5.6 Point Queries

A point query can be performed on a Sector to check whether the point lies on the Sector. This check tries to map the point to the Sector and compares any height difference to the maximum snapping height of that Sector. This can be done by calling NaviProCore.SectorContainsPoint for any Sector or by calling NaviProCore.BaseSectorContainsPoint for the Base Sector.

5.7 Nearest Agent Queries

From a given point in a Sector and a given radius, the nearest agent position can be found using NaviProCore.FindNearestAgentPositionInSector or, for the Base Sector, using the equivalent NaviProCore.FindNearestAgentPositionInBaseSector. These functions can be useful to determine whether a part of the Sector is occupied by an Agent.

5.8 Path Queries

Aside from finding paths for Agents, NaviPro can also find paths directly. This can be done by calling NaviProCore.GetPathInSector or NaviProCore.GetPathInBaseSector. The path finding is similar to path finding for Agents and uses the same Agent radius and safe distance parameters.

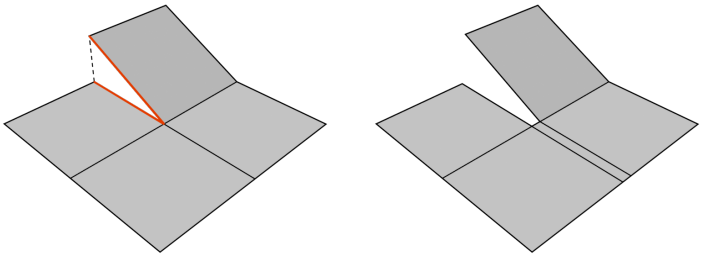
Please note that, since path finding is performed in 2D, the returned path is also 2D. No height information is provided.

6 Technical Details

6.1 Navigation meshes

The supplied mesh for creating a Sector using a custom Walkable Mesh should only consist of the surface upon which Agents can move. In NaviPro, Agents can only move upright, so they cannot cross vertical planes. Thus, the mesh should consist of flat or sloped, non-vertical polygons and it should not contain any sections that are not meant to be traversable. Also note that all polygons that are not connected in the mesh will have an untraversable gap between them. For example, stairs should not be modelled as blocks or flat planes, but rather as a single sloped polygon. Walls or cliffs should not be part of the supplied mesh, but should be left out.

Furthermore, while NaviPro supports multi-layered geometry, all surfaces should be locally flat. This means that, when viewed in 2D from above, a section of the mesh should not overlap itself, which includes the overlap of exterior edges. Instead, locally connected areas of the mesh should always have some space between them, much like walls dividing adjacent rooms. This requirement is not be a limitation towards what can be represented. It is possible to model a spiral staircase, but the centre of the stair case should be cut out.



The left mesh has two overlapping edges, which should be avoided. The right mesh is correct, with some space between the edges.

6.2 Precision and Size limits

The geometry in Sectors is limited to a precision of 0.1 mm, assuming a unit distance of 1 meter. The maximum size of a Sector is limited by a distance of a little over 200 km from the origin of the Sector. These limits are mainly enforced by the use of integer precision in Boost's voronoi implementation. For most users this should not be a limitation, but it can be extended by modifying the source code.

6.3 Disabled Agents

When an Agent is moved off a Sector or when the part of a Sector an Agent was on is removed, eg. by removing a Platform, the Agent is disabled. When an Agent is disabled, its NaviPro Agent component is disabled as well. A disabled Agent is no longer being simulated and loses its assigned goals. To re-enable an Agent, move it to a valid position on the Sector by adjusting its transform and re-enable the NaviPro Agent component.

6.4 Agent Pivots

After simulation, all Agents are assigned a position on the Sector or in other words, on the ground. To ensure the position of the Agent is visually correct, for instance with the feet on the ground for human Agents, the pivot of the Game Object containing the Agent component should be located at its feet.

The forward direction used for Agents is along the positive Z-axis, following the default Unity convention.

An Agent has its pivot located where the ground should be, at the feet. The Z-axis, shown in blue, points to the forward direction.

6.5 Native Library

NaviPro is largely written in C++. This code is compiled into a native library for Windows, Android and iOS/OS X. To support other other platforms and for your reference, the source code is also provided.

The source code relies on some parts of the boost library. The modules used are all header only files and are provided with the source code for easy compilation.

6.6 Simulation

By default, the simulation and all related computation is performed during FixedUpdate. The transform of Agents is then interpolated between simulation updates during Update, before rendering. This ensures consistent behaviour for varying framerates, while showing smooth movement for any framerate. This behaviour can be changed in the Settings (see section 4.4). It can be useful to run the simulation only in FixedUpdate and disable interpolation. This will allow the Agents to interact predictably with physics simulation. The user is then responsible for interpolating the movement. If predictable behaviour for varying framerates is not a concern, the simulation can also be run in Update instead.

Most of the computational work in NaviPro is performed by mulithreading through the Jobs system. In a computationally heavy scene with a large number of Agents, the Profiler can be used to gain insight into the performance. Depending on the settings detailed above, NaviProCore.FixedUpdate and NaviProCore.Update may use a significant portion of CPU time.

The creation and removal of internal objects used by NaviPro for eg. Sectors and Agents is performed just before its simulation update. This time is also used to move Agents (see section 5.2).