SECTR

By Procedural Worlds

SECTR is a toolkit to build structured spaces in Unity and provides solutions for Streaming, Audio Playback and Occlusion Culling.

Version 2019.0.1

Contents

Welcome!

Thank you for purchasing SECTR!

SECTR is a sophisticated toolkit covering a wide array of tasks, starting from structuring your scene into sectors, streaming terrains, playing back audio, or adding real-time occlusion culling.

A little bit of history: SECTR originally consisted of multiple products that were all sold as individual products on the asset store. Those products were:

SECTR Core – A free tool to organize your scene in connected spaces (Sectors) and a terrain splitting tool

SECTR Audio – Audio Playback for different Audio Environments, HDR hierarchical mixing, Audio Propagation & occlusion, etc.

SECTR Stream – Streaming Solution for dynamically loading in and out parts of your scene around the player to run large game worlds without running into performance issues.

SECTR Vis – Dynamic, baking free Occlusion solution that hides renderers that are currently not visible to the player.

While SECTR is a single product now, you still can find this separation when looking at the file structure inside the Scripts folder. This is also why there is multiple Quickstart documents for these different products.

Speaking of which: To get up and running quickly with the different SECTR tools we strongly recommend that you review and follow the separate Quick Start documents that came with SECTR in the Documentation Folder of your installation.

And finally, we have also created an awesome support network for you which you can access from the Window > Procedural Worlds menu in Unity.

PRO TIP:

Did you know that we also have a range of other products to enhance your environments in Unity? For example Gaia can help you to create entire terrains from scratch, or with Path Painter you can create texture-based paths on your terrain in minutes!

Check out our other products on the next page to learn more!

About Procedural Worlds

Powerful, simple, beautiful. Friendly tools, gorgeous games!

Procedural Worlds empowers artists and developers to bring their vision to life by making it easy to create beautiful worlds. Leverage the latest procedural generation techniques to take the pain out of creating stunning environments and focus on creating amazing games.

The only end to end environmental generation and delivery suite:

Ambient Sounds - Create interactive ambient soundscapes that adapt to your gameplay.

https://assetstore.unity.com/packages/tools/audio/ambient-sounds-interactive-soundscapes-142132

Gaia - A world generation system for creating, texturing, planting and populating scenes from low poly mobile, VR and through to high end desktop.

https://assetstore.unity.com/packages/tools/terrain/gaia-42618

CTS - Nominated by Unity of as one of the best assets in 2017, a PBR terrain shading system that significantly improves terrain look, performance and usability. The 2019 version comes with support for the Lightweight and High Definition Rendering Pipelines.

https://assetstore.unity.com/packages/tools/terrain/cts-2019-complete-terrain-shader-140806

GeNa 2 - A sophisticated localised level design tool that augments Gaia's broad-brush strokes, by working intuitively to give fine grained control.

https://assetstore.unity.com/packages/tools/terrain/gena-2-127636

Path Painter - A powerful path and river channel creation tool.

https://assetstore.unity.com/packages/tools/terrain/path-painter-127506

Pegasus - A cut scene and fly through creator that makes it easy to show off gorgeous environments and also drive characters through scenes with localised avoidance and mecanim animation support.

https://assetstore.unity.com/packages/tools/animation/pegasus-65397

Learn more at our website here: http://www.procedural-worlds.com/

Tutorials, Chat, Ticketed Support

For Tutorials for the individual SECTR tasks you can follow along with the Quick Start documents in the "Documentation" folder of your SECTR Installation.

Still Stuck? You can contact us on our discord server: https://discord.gg/rtKn8rw

Or lodge a Support Request: https://proceduralworlds.freshdesk.com/support/home

Installation

Installing SECTR will create this folder structure with the following contents:

Procedural Worlds – root folder for all Procedural Worlds Assets

- **SECTR** – root folder for the SECTR Asset
    - **Demos** – Various demo scenes that will make you familiar with the core concepts of SECTR. You can safely delete this folder if you don't need the demo scene and its contents in your project.
    - **Documentation** – SECTR documentation, including API and various Quickstart documents
    - **PlayMaker –** Contains extra packages for PlayMaker integration
    - **Localization** – Localized UI texts
    - **Scripts** – The core logic of SECTR
- **Framework** – shared functionality between Procedural Worlds assets

What is SECTR?

SECTR is a toolkit to created structured spaces in your scene. After these spaces or sectors and their connections are defined, you can play back Audio in them, load them in dynamically for streaming or hide them when they are occluded.

With SECTR you will:

- Organize the assets in your scenes into logical sectors
- Query this structure for your own gameplay logic
- Split up your large terrain into multiple smaller terrains (and sectorize it along the way)
- Play back dynamic Audio in these Sectors that includes features like propagation, occlusion and HDR mixing
- Mix your audio in a hierarchical structure
- Load and unload these sectors around a player or camera for world streaming
- Show and hide these sectors dynamically according to their visibility to the player to save performance.

SECTR Core Concepts

Most scenes in Unity can be broken down into spaces (Sectors) and the connections between them (Portals). This pattern works for indoor games, but also hybrid-indoor games, where exteriors are really just large rooms (i.e. Dead Space, Metroid Prime, Gears of War, etc). This principle can also be applied to open-world games where the large game world is usually separated into chunks which are connected with each other.
Over the years, developers have learned how to use this structure to create games that run faster, sound more realistic, and look better than they otherwise could.

SECTR also includes a suite of libraries for interacting with and extending this framework, including graph traversal and geometric operations. The building blocks of SECTR are Sectors (spaces), Portals (connections), and Members (objects in Sectors). When Sectors are connected through Portals, they form a graph, called the Sector/Portal Graph. Each of these key components are discussed in their own section below.

Sectors

A Sector represents a volume of space, and the objects within that space. In most games, Sectors will be rooms and hallways, but Sectors can represent anything from a section of an outdoor level to a bonus area in a side scroller. Sectors are connected by Portals (described below).

Static and Dynamic

Sectors can be marked as Static or Dynamic, but default to being Static (since most games do not have moving rooms). When marked as Static, the Sector will save some CPU time by not computing its bounds every frame. Generally speaking, any children of a Static Sector can either be Static themselves, have a Member component on, or otherwise be guaranteed to stay within the bounds of their Sector.

Children and Bounds

Sectors are defined by their Renderer components and those of their children (i.e. the objects parented underneath them). In the case that a Sector has a child that is a Member, the Sector will ignore that child and all of its children for the purposes of computing the Sector's. The basic idea is that if a Sector finds that one of its children is a Member, then it ignores it, and assumes that child Member can and will take care of itself. The base Member will still act as a parent in every other way, in keeping with Unity conventions. This behaviour is very useful if you want to have objects that are "part of" the Sector (like lights or particle systems) but which need to move within that sector, or change their bounds dynamically, or extend outside the bounds of that Sector. See the Optimization chapter for information about how to

reduce the CPU cost of bounds computation.

“Shared” Children

Because the bounds of a Sector are defined by the its Renderers, it’s possible for some children (like Lights) to extend beyond the bounds of their parent Sector and overlap other Sectors. This may or may not be a problem, depending on the specifics of your scene. To help highlight these items, if you select a Sector and with Sector Gizmos enabled, any “shared” children will be highlighted in red. You can ignore them, fix them yourself, or press the Fix Shared Children button in the inspector. If you press the Fix Shared Children button, each shared child will be given a SECTR Member component. In general, though, you don’t need to do anything unless you see a problem when testing your scene.

Membership

Sector membership is not exclusive. Members may be in multiple Sectors at once, so Sectors may overlap and even be nested in one another. Sectors should never be parented to one another. If this happens some assumptions will break, and strange things may begin to happen, especially when streaming or setting up occlusion culling with Vis. All of the SECTR editor UI will enforce this conventions, so just be careful when manually adding Sector components.

Portals

Portals represent the connections between Sectors. If a Sector is a room, then a Portal is like a doorway or window. Like a doorway, Portals have geometry that defines their shape, though in some applications the actual geometry is not strictly necessary. Because they connect Sectors, they should never be parented to a Sector, but be left at the same level of the scene hierarchy as the Sectors that they connect.

Connections

The most important thing that Portals do is connect two Sectors to one another. To do this, each Portal has two properties: a Front and a Back Sector. When both properties are filled out, the Portal forms a line between those two Sectors.

For many applications, it's important to know which side of the Portal the Sector is on, which is why the attributes are named as they are. The in-editor visualization will show you where the front and back sides are, but they are generally +Z for front and -Z for back, and the normal of the portal always points forward. When connecting Portals its important to get the sides correct, but if you get them backwards, you can simply press the "Swap Sectors" button. 7 Geometry The shape of a Portal is defined by a Mesh resource. This is to allow users to create Portals both within Unity (using the included Portal drawing tools) and in external programs like Max or Maya. Portals

are required to be planar (i.e. flat) and to be convex. They can, however, have as many sides as necessary. For most applications, Portal geometry does not need to perfectly fit the visual geometry of the level. Portals can almost always be left a simple shapes that extend a bit past the "real" opening that players see. As long as the Portal geometry is fairly close, everything will work fine. This is important, because some operations require more CPU the more sides a Portal has, so generally use the fewest number of sides necessary to accomplish the goal.

Flags

As the connections between Sectors, it's often useful for Portals to have some state associated with them, like being Open or Closed. SECTR provides some built in flags, but you are encouraged to add your own based on the needs of your game. All of the Sector/Graph algorithms will work with any flags you add, as well as the standard flags.

Standard Flags:

- Pass Through: Ignores the geometry when computing visibility.
- Closed: Treats the portal as inactive for the purposes of visibility calculations.
- Locked: Identifies the portal as logically locked. Currently unused.

Members

While Sectors and Portals define the spaces and their connections, most objects in a game are neither Sectors nor Portals. These game objects often want to know which Sector(s) they are currently in, where the nearest portals are, etc. The Member component is designed to meet this need, keeping track every frame of which Sectors it is a part of, and publishing that information to anyone who wants to know about it.

Useful Applications

Members serve many useful roles in the different SECTR modules, but the most basic idea is that if an object needs to know about Sectors but is not itself a Sector, then it should have a Member component added to it. Many components in other SECTR modules recognize this, and will add Member components for you automatically.

Children and Bounds

Like Sectors, Members can have children and their bounds are defined by the components in their children. Unlike Sectors, the bounds of Members are the union of all Renderer and all Light components, which ensures that any visual influence they have is fully represented by their bounds. In the case that a Member has a child that is also a Member, the base member will ignore that child and all of its children

for the purposes of computing its bounds, Sector memberships, etc. The basic idea is that if a Member finds that one of its children is also a Member, then it ignores it, and assumes that child Member can and will take care of itself. The base Member will still act as a parent in every other way, in keeping with Unity conventions. See the Optimization chapter for information about how to reduce the CPU cost of bounds computation.

Static and Dynamic

Like Sectors, members may be static or dynamic, and may or may not have children. If a Member is static, it will save some CPU time and do fewer calculations each frame. If a Member is static, all of its children should be static too. If a static Member has dynamic children, the Member may not give the right information about which Sectors it's in.

Membership

Sector membership is not exclusive. Members can handle being in multiple Sectors at once, and every system in SECTR that needs a Member is designed to work with multiple Sector membership. By default, Members are included in every Sector whose Bounds overlap with the Member Bounds. While simple and fast, this approach does not work well for some games with complex, interior geometry. For games with a significant amount of Sector overlap, nesting, or convexity, see the Portal Determined Membership section below.

Portal Determined Membership

As described above, by default Member components belong to all of the Sectors whose Bounds they overlap. For some games with complex scenes, this behavior may be undesirable, as Members may be part of too many Sectors. One simple example of this is a room nested inside another room. Some games may want the Member to only be part of the inner or the outer room, not both (which would be the case by default when the player is in the inner room).

SECTR provides a solution to this problem with the Portal Determined flag. This flag changes the membership computation so that it only changes when a Member passes through the Portal leading from one Sector to another. When the Member's transform position passes through the Portal geometry, the Member will leave the old Sector and enter the new one.

To support this feature, Member also includes a Force Start Sector feature, which allows you to specific a specific Sector to be used when the Member is first created/enabled. If not specified, the initial membership will be determined by the default bounds logic. Force Start Sector is useful for cases where the default behavior is undesirable, for example if the Member started in the inner room described above.

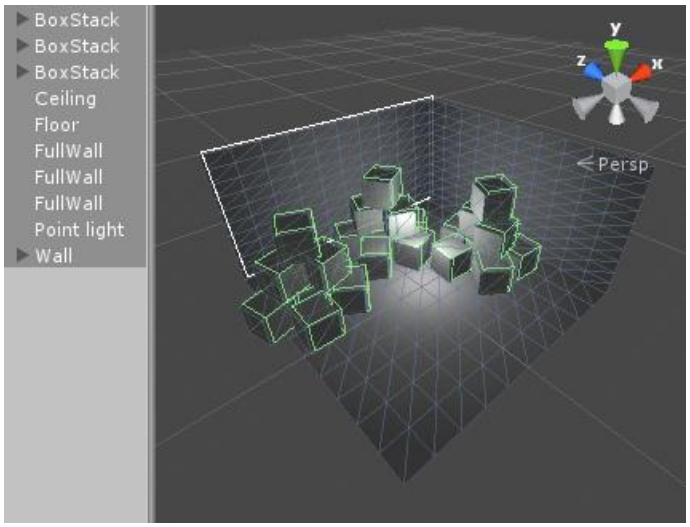
If using Portal Determined Membership, remember that the change happens when

the transform position passes through the Portal, not the Member bounds. If your game has Members whose object position is at the bottom of the object, make sure that all of your portals extend at least a little bit below the floor so that the Member transform does not accidentally "slip under" the portal due to numerical precision issues.

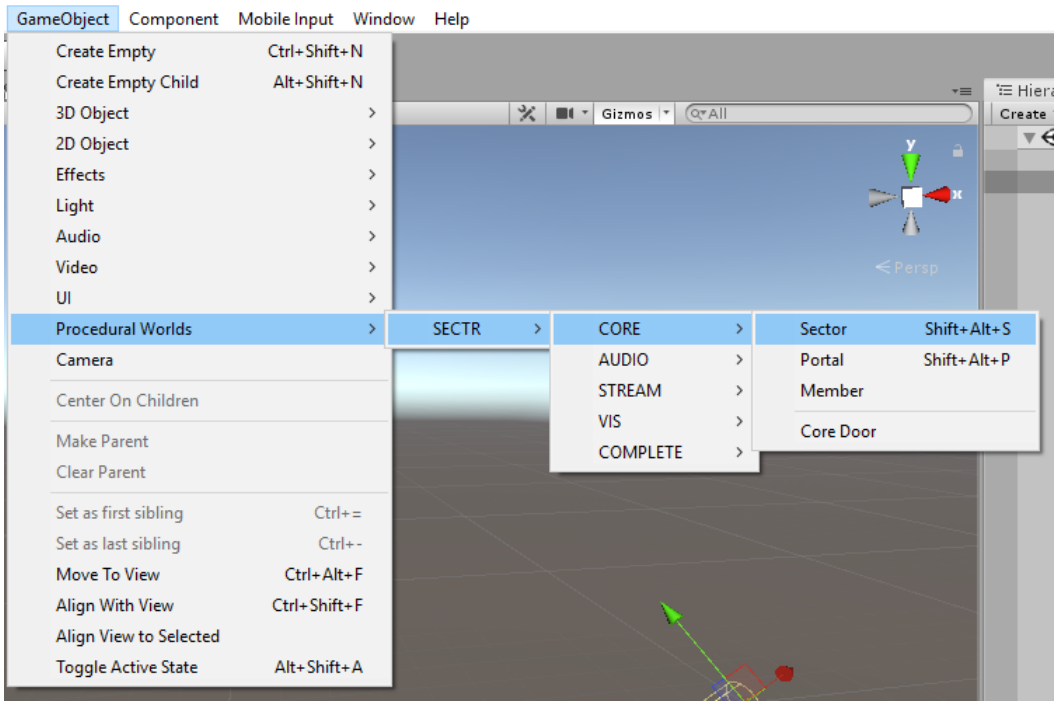
Creating your first Sectors and Portals

Creating Sectors

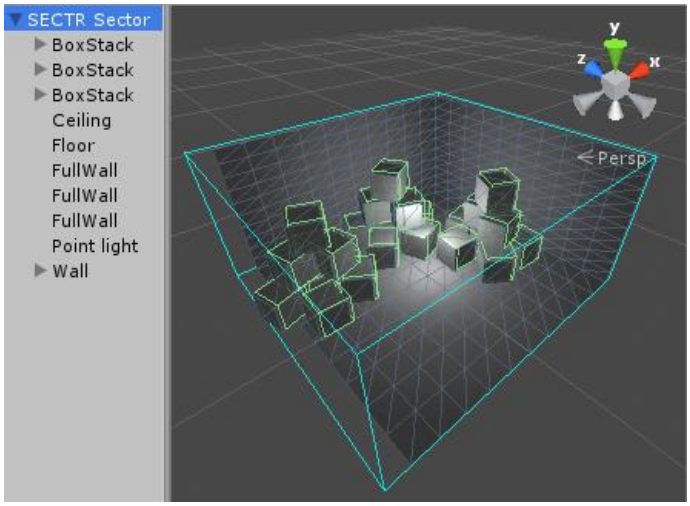
The first step in preparing a level for SECTR is creating a few Sectors. First, select the objects that you want to define the space of your Sector, like the walls, floor, ceiling, or anything else that you want.



Next, go to Game Object->Procedural Worlds->SECTR->Core and choose Sector or press Shift + Alt + S.



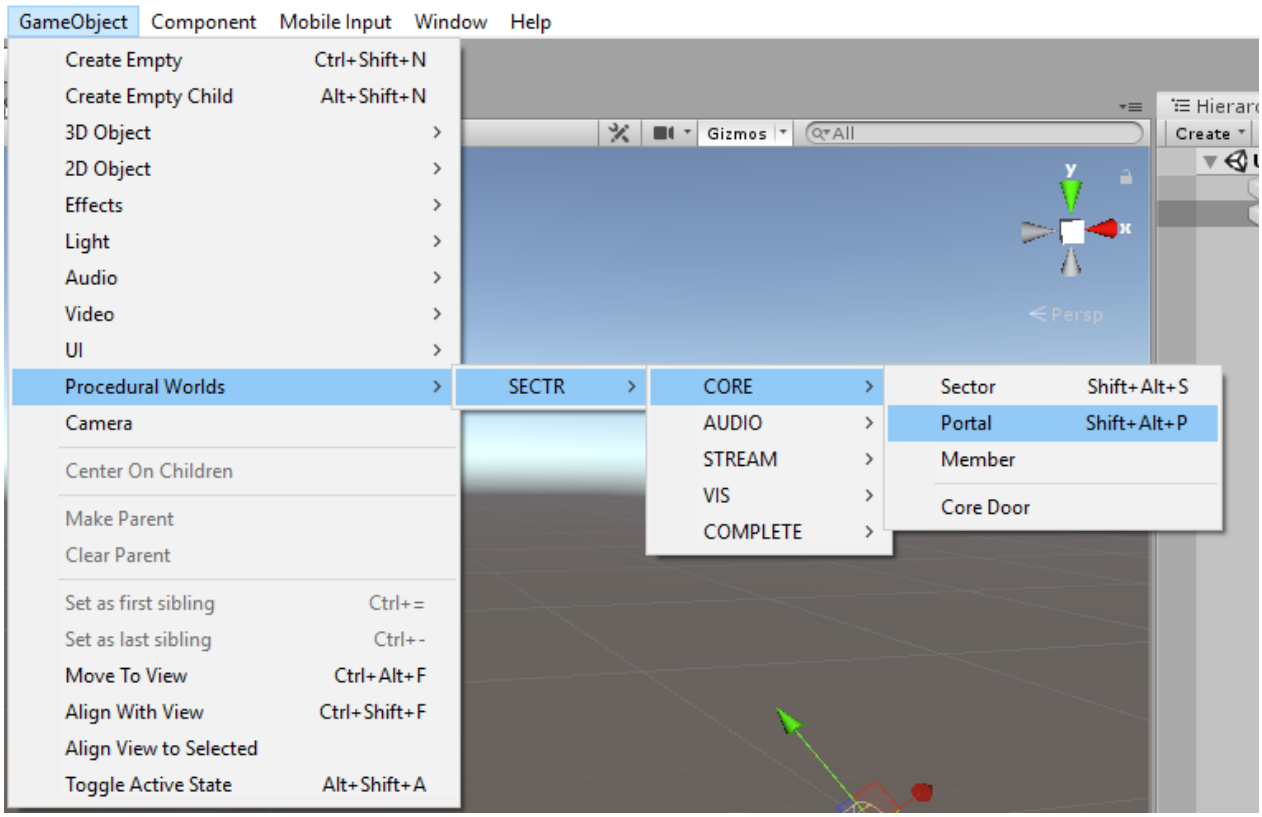
A new Sector is being created for the selected Game Objects. If the selected objects weren't already grouped under a single transform, they will be auto-grouped for you.



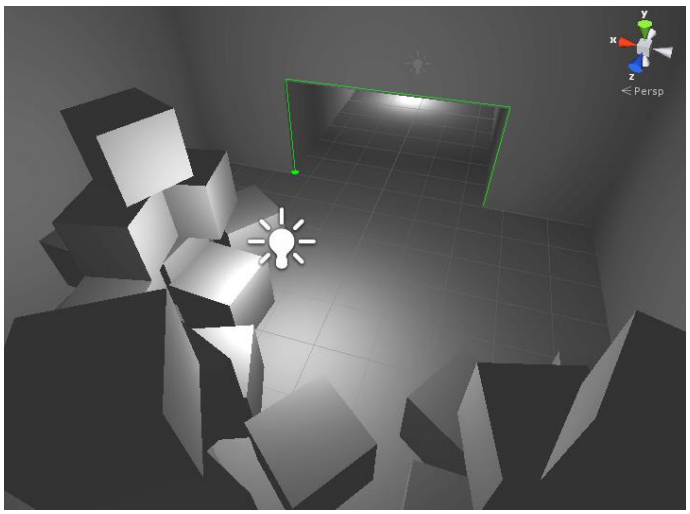
Adding Portals

Once you've created a couple of Sectors, it's time to connect them with a Portal.

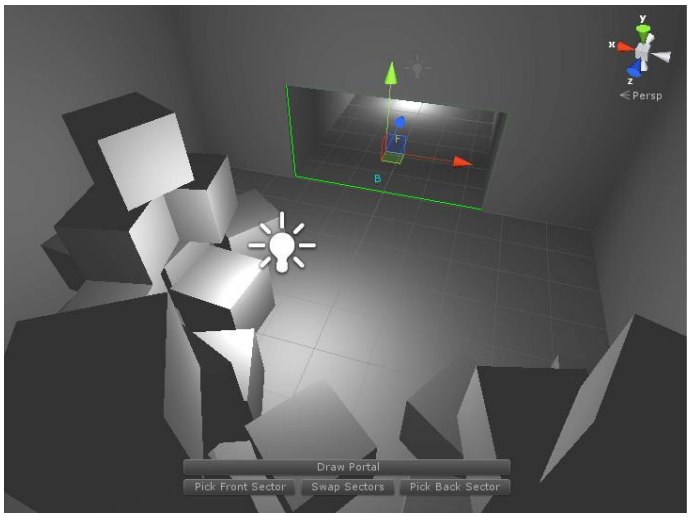
First, go to Game Object->Procedural Worlds->SECTR->Core and choose Portal or press Shift + Alt + P. This will activate the portal drawing tool.



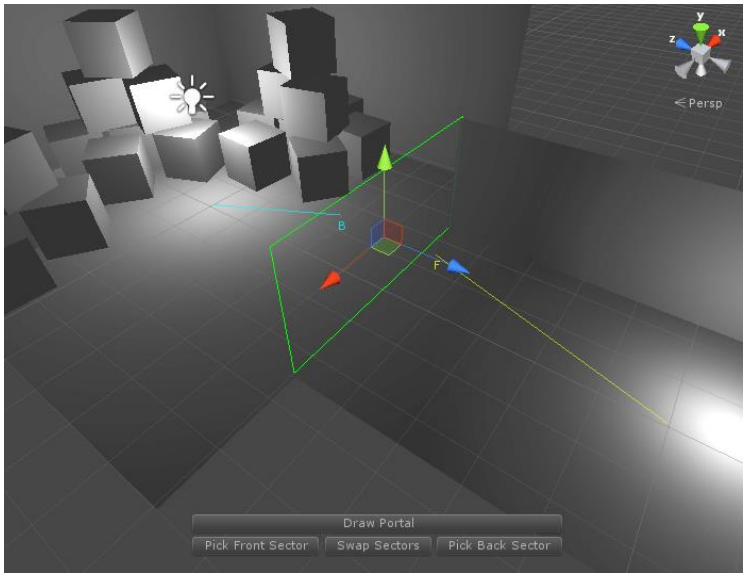
Simply click to place the verts of your Portal. Portals can have as many sides as you want, but they have to be planar and convex. The portal drawing tool will turn red if the current position would make an invalid portal.



When you're happy with the geometry of your portal, either press Enter or click on your first vert to complete the Portal.



Now that the Portal is created, the last step is to connect the Portal to the two Sectors on either side.



Simply click the Pick Front Sector button, and then click on any object in that Sector. Do the same thing for the Back Sector, and you're all done.

You'll know that you got it right if the B and F lines draw in the right direction. If you accidentally got them backwards, no problem, just click the Swap Sectors button.

Pro Tip: You don't have to draw out meshes for every portal. You can assign any mesh resource using the Inspector. Just make sure that the mesh is planar and convex.

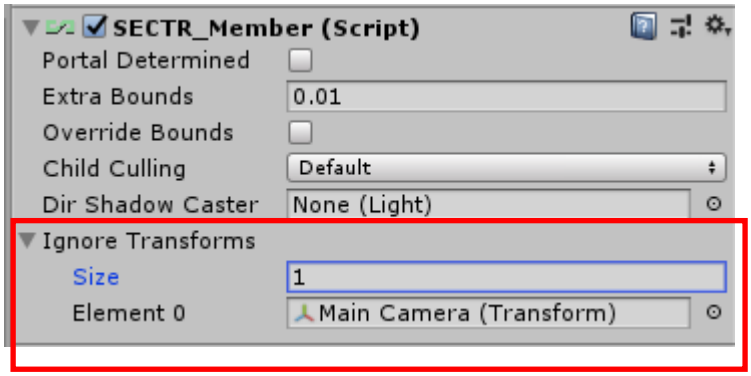
Adding Dynamic Objects

Sectors and Portals define and connect your space, but most of the interesting objects in the game (like the player, enemies, cameras, etc) are dynamic and need to be able to move from one Sector to the next. As these objects move around, it's important that they be able to keep track of what Sectors they are in.

The solution to this is very simple...

For any object that moves around and wants to know what Sector(s) it's in, simply add a Sector Member component to it. That's it!

Pro Tip: Member components will take care of itself and all of its children. You don't need to add a Member to each child. If you don't want to include a child into the member even though it is parented to a GameObject with a member component on it, there is an ignore list in the Member component as well:



Now with the basics covered you can have a look at the individual main functions of SECTR: Audio, Streaming and Occlusion culling. Please note that there is also a Quickstart Document available in the documentation folder for each of these main functionalities as well.

Playing back Audio (SECTR Audio)

Audio Playback in SECTR has three major aspects: creating sounds, playing them back, and mixing them together, each of which will be discussed briefly here, and at greater length below. Although part of the suite of SECTR products, playing back audio with SECTR does not require any Sector or Portals, except to enable a couple of advanced features.

Creating Sounds

SECTR introduces a new audio asset, the AudioCue. An AudioCue is the most basic unit of playable sound in SECTR. AudioCues contain many of the same properties as a regular Unity AudioSource, but include many features beyond those basics, including randomization, fading, and all of the properties for advanced features like occlusion, propagation, etc.

Playing Sounds

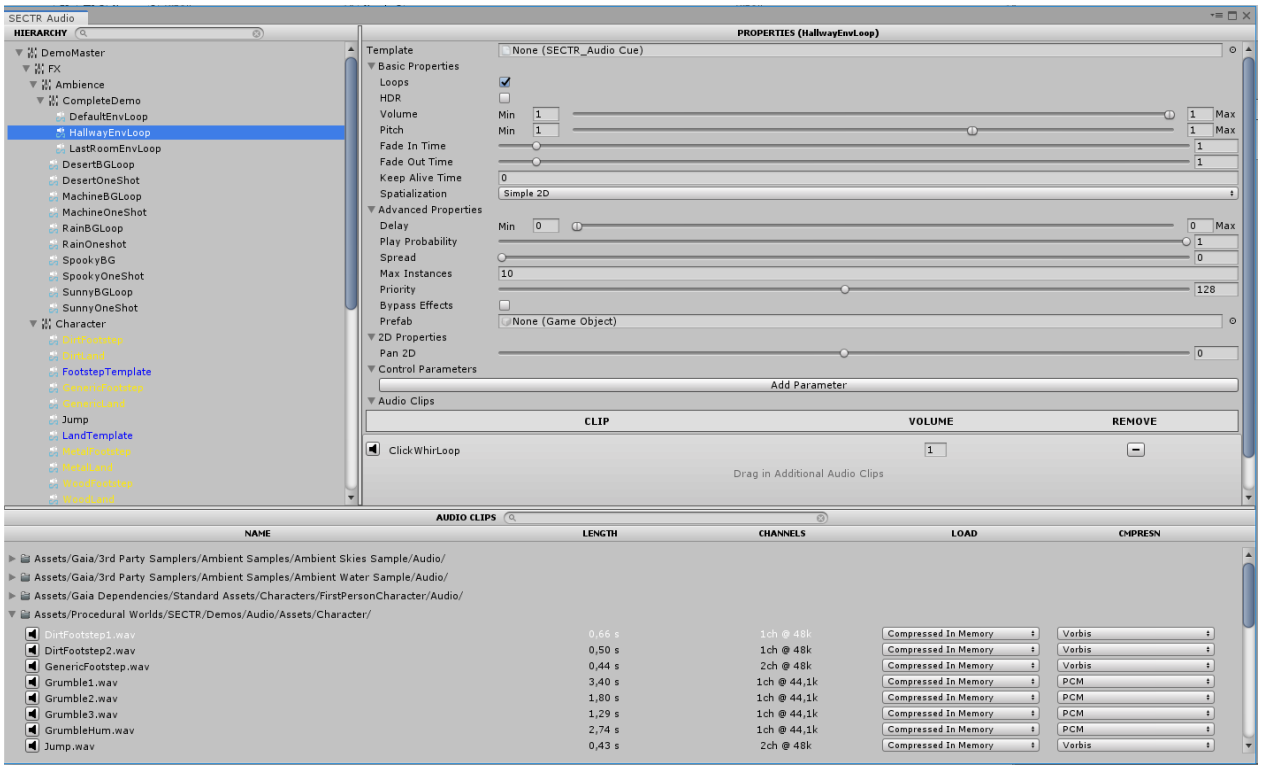
All sounds in SECTR route through the AudioSystem component. The AudioSystem provides a centralized interface for all sound playback and does all of the heavy lifting required to create new sounds, mix them, etc. Programmers can call this API directly, or use any of the included SECTR AUDIOSource components.

Mixing

SECTR includes a standardize hierarchical mixer that should be familiar to most sound designers. Each AudioCue in the game has a parent bus, and its final volume is determined by the volume of its parent and all of its parents' parents. This hierarchy makes it easy to make both very broad and very specific mixing changes easily. SECTR also includes an optional, high dynamic range (HDR) mixer. Individual Cues can be tagged as HDR. When tagged this way, they are given a physically accurate representation of volume - called loudness, and the final volume is determined on the fly based on all of the other HDR sounds that are currently playing.

SECTR Audio Window

One of the biggest challenges in making game audio is simply managing all of the assets involved. SECTR makes this task as easy as possible with the SECTR Audio window, which you can find under Window->Procedural Worlds -> SECTR->Audio Window.



The SECTR Audio window has three panes, each of which will be explained below. The SECTR Audio window supports drag and drop between panels. Dragging an AudioClip onto an AudioCue will add it to that AudioCue. Dragging it onto a bus will make a new AudioCue with that AudioClip in it. You can also drag AudioBuses and AudioCues around reparent them and otherwise move things around. The SECTR Audio window is also version control aware. Any objects that are not locally editable will be displayed in gray with their controls disabled. You can right-click on any asset to check it out, or you can check it out through the normal Unity GUIs.

Audio Clip Panel

Audio Clips are the basis of of all playback in Unity, and playing back audio with SECTR is no exception. Because a large game can have many AudioClips, the clip panel tries to make managing them as easy as possible. Clips are sorted by the folders they are contained in, and the list can be filtered using the search interface at the top. By default, only basic information is shown for each clip, but you can show all of the details by right clicking in the pane and choosing Show Full Details. Note that in very large projects the sheer number of AudioClips can overwhelm Unity's GUI system and the SECTR Audio window may become sluggish. If this happens, simply collapse the any folders you're not working in and, if you can, turn off full details mode.

Hierarchy Panel

The hierarchy panel shows all of the AudioBuses and AudioCues in the project. The tree structure shows you where individual assets fall in that hierarchy. Left clicking on

any Cue or bus will show you its information in the Properties Panel (see below). 5 Right clicking anywhere in the panel gives you options for modifying, creating, and deleting the assets displayed here.

Properties Panel

As its name suggests, the properties panel displays all of the information relevant to the object selected in the Hierarchy Panel. When an AudioCue is selected, a version of its inspector is drawn in the Properties Panel. You can edit things here exactly as you would in the regular inspector, with full support for Undo and the like. When an AudioBus is selected, the Properties Panel becomes a mixing view, displaying sliders for all of the AudioBuses in the project. If this is too many for you, you can use the search dialog at the top to filter the list down to what you are interested in.

Keyboard Shortcuts

The SECTR Audio Window also supports keyboard shortcuts for common functions. These shortcuts are:

- Arrow Keys: Navigate hierarchy and clip views.
- Enter/Return: Rename selected hierarchy item.
- Ctrl+D: Duplicate selected hierarchy item.

Creating Audio

While the SECTR AUDIO window makes it easy to create AudioCues, AudioCues themselves have a number of properties that you may want to understand before you get too far. This section describes those properties in detail, but you can also refer to the in-editor tool tips if you forget anything here.

Basic Properties

- **Template**: If set, this is another AudioCue that this AudioCue will take all of its properties from, except for the AudioClips and the Bus. Templates are a powerful way to manage a lot of sounds with just a few files.
- **Loops**: If this clip should loop on playback. Loops affects the culling behavior, as described below.
- **HDR**: Denotes if this AudioCue should be mixed in high dynamic range or low dynamic range, as described in the Mixing section.
- **Volume and Loudness**: LDR sounds will have Volume, while HDR sounds have Loudness. In both cases, you can specify a min and max Volume/Loudness. SECTR will choose a unique value from this range on every playback.
- **Pitch**: Like loudness, this allows you to supply a min and match pitch offset. This can be a very cheap way to create additional variation.

- **Fade In Time and Fade Out Time**: Specifies how long to fade in when played, and how long to fade out when stopped.
- **Keep Alive Time**: Specifies how long the Audio Source should be kept alive after playback has been finished. This setting can be used to make sure filters and effects (e.g. Reverb) have some extra seconds to finish playing before the source gets destroyed.
- **Spatialization**: Determines how this sound is positioned in the surround field.
    - Simple2D sounds are just that, 2D sounds with no position at all.
    - Linear3D sounds are your standard 3D sounds.
    - Occludable3D sounds are like Linear3D, but they also can be Occluded.
    - Infinite3D sounds have 3D position, but do not attenuate at all. 7 Advanced Properties
- **Delay**: Number of seconds to wait after Play() to actually start the audio.
- **Play Probability**: A chance that the cue will actually create a sound when Play() is called. Helps thin out the mix when set to be < 100%.
- **Spread**: Determines the range of speakers from which the sound is audible.
- **Max Instances**: The maximum number of instances of this Audio Cue that can exist at any one time.
- **Priority**: Internal FMOD priority, as described in the Unity docs.
- **Bypass Effects**: When true, Cue will ignore reverb zones or filters. (Pro Only)
- **Prefab**: The prefab that defines filters and other custom behavior for this Cue. 2D Properties
- **Pan2D**: Moves the sound around the speaker field. 3D Properties
- **Falloff:** Determines how this sound attenuates with distance. The options are linear and logarithmic, which behave the same as Unity AudioSource.
- **Min Distance:** The distance at which sounds start attenuating.
- **Max Distance:** The distance at which sounds are no longer audible.
- **Occlusion Scale:** The amount by which this Cue is affected by the global occlusion settings.
- **Doppler Level:** Scales the amount of doppler applied to this sound.
- **Proximity Limit:** The maximum number of instances that can be within Proximity Range of one another. Does nothing if set to 0.
- **Proximity Range**: The minimum distance between instances, if Proximity Limit is set to be greater than 0. See Mixing below for more information. 8 Audio Clips
- **Clip List:** Set of AudioClips to select from when playing this Cue.

- **Clip Volume:** Individual volume for each AudioClip. Allows for per-Clip mixing.
- **HDR Curve:** Displays the loudness envelope for Clips in HDR cues. Can be user edited for finer grained control over the HDR mix.
- **Playback Mode:** Determines the rules for how AudioClips are selected. The default behavior is random, but shuffle, loop, and ping pong are also options.

Playing Sounds

Once you've created some audio assets, it's time to start playing them. Before you can playback sounds with SECTR, you need to set up your scene.

Preparing for Playback

To play sounds in SECTR, all you need to do is add a SECTR AUDIOSystem component to your Listener. Once you've created an AudioSystem component, hook up an Audio Bus to the Master Bus attribute. And that's it! If you want to use PropagationSource or try out graph based (i.e. non-raycast) occlusion, you'll need to create some Sectors and Portals, too. For more information on how to do so, see the "Creating your first Sectors and Portals" section in this manual.

Audio Sources

SECTR includes a number of audio components designed to handle audio playback in common cases. These SECTR AUDIOSources are similar to Unity AudioSources, except they support a much wider range of features than just point source playback. The following explains what each one does:

- **Point Source**: Plays a sound at a 3D point. Very similar to the Unity Audio Source.
- **Region Source**: Plays a sound at a point on a collider nearest to the Listener. Great way to represent volumetric sounds.
- **Spline Source:** Plays a sound at a point on a 3D spline nearest to the Listener. Very useful for rivers and roads and the like.
- **Propagation Source:** This source models how audio bounces around through an environment. Sounds amazing, but requires Sectors and Portals to work.
- **Trigger Source:** Plays a sound when a player enters a Trigger. Good way to do simple sound scripting.
- **Impact Source:** Plays sounds when a rigid body collides with the world. Great way to tie sounds into the physics.

Audio Environments

Although Audio Sources are great for sounds that have a specific 3D position, games often want to have a base layer of "2D" or non-specific sound playing (sometimes called an ambience). To achieve this effect, SECTR AUDIO provides a set of AudioEnvironment objects that make creating ambient audio easy.

Each Audio Environment contains the settings for an AudioAmbience. An AudioAmbience simply contains a background loop, a list of one shots to play randomly, and a range of time between one shots. In general, the background loop should be Simple2D and the one shots should be Infinite 3D. During the game, there may be multiple Audio Environments active in the world, but only one of their AudioAmbiences will be audible at a time.

To achieve this, the AudioSystem has a stack of AudioAmbiences, the top of which is audible. When a new AudioEnvironment becomes active, its AudioAmbience is placed on top of the stack. When that AudioEnvironment becomes inactive, it removes its AudioAmbience from the stack, wherever it is. This approach allows AudioEnvironments to be overlapped, nested, etc and still sound correct.

Gameplay Events

For sounds triggered by gameplay events, like weapon impacts or UI sounds, programmers should call SECTR_AudioSystem.Play() directly, as this is the most efficient way to play sounds in SECTR. If this sound loops, or needs to be updated after it is played, the AudioSystem will return an AudioCueInstance that you can use to access the instance of this particular AudioCue.

Music

Music is an important component to any game audio, but SECTR AUDIO takes a simple approach to its music system. Music, in SECTR audio playback is simply a special way to play an AudioCue, and the AudioSystem guarantees that no more than one will play at once. While simple, this system is very effective for the vast majority of ingame music.

Mixing

Mixing is one of the most important, but generally time consuming aspects of creating interactive audio. SECTR includes several tools to both make for higher quality, and less time consuming mixing.

Bus Hierarchy

SECTR includes a standardize hierarchical mixer that should be familiar to most sound designers. Each AudioCue in the game has a parent bus, and its final volume is determined by the volume of its parent and all of its parents' parents. This hierarchy makes it easy to make both very broad and very specific mixing changes easily.

Instance Limits

SECTR also allows you to control the number of instances of an AudioCue, both globally and, for 3D sounds, within a region of space. Used together, these instance limits can help "thin out" the mix and prevent it from becoming cluttered. The first limit is called Max Instances. Max Instances determines the maximum number of times an AudioCue can be playing concurrently. Once this limit is reached, future requests will fail until one of the current instances stops playing.
The second limit is the Proximity Limit. This feature is for 3D sounds only, and, combined with Proximity Range, allows you to specify the number of instances of an Audio Cue that can play within a certain distance from one another. For example, a Proximity Limit of 2 and a Proximity Range of 1, means that no more than 2 instances of a sound can play within 1 meter of e from 0 to 1.

Loudness

Loudness is a logarithmic representation of the sound pressure level, called dB(SPL). Loudnesses can be set to any value that sounds good, but it's good to start with real world values, some examples of which can be found on Wikipedia.
While the Loudness property of the Cue establishes a baseline for the Loudness, real sounds are rarely uniformly loud. Real world sounds often have a mix of louder and quieter sections. To represent this variation, SECTR can compute HDR Keys, which are basically a compressed representation of each AudioClip's volume.

These HDR keys are not required to play HDR audio, but they will make the mix sound much better. HDR keys can be baked in the Cue Inspector and in the Audio Window. If HDR keys are not baked, SECTR will let you know. When HDR sounds are enabled, they still have to be converted to LDR sounds before playback. To do this conversion, SECTR uses a sliding "window", which functions like the exposure of a camera. The top of this window is the loudness of the loudest sound currently playing. The bottom of the window is determined by the HDR Window Size attribute in the Audio System. The final volume of the sound is determined by where its loudness falls within this window.

That means that a particular AudioCue will sound louder or softer based not just on its own loudness but on the loudness of all of the other active sounds. The dynamic nature of HDR audio creates a very realistic, impressive effect, but it takes time to learn if you're familiar with traditional, LDR mixing. If you try out HDR audio, make sure to play around with it for a while to get the hang of how this kind of mixing works. There are also a number of good articles on the internet about how to think about HDR audio mixing.

Spatial Effects

Interactive audio always takes place within a 3D (or 2D) context. The geometry of this environment influences how the sounds in it are perceived. Stock Unity has a few tools to help with this, specifically Audio Reverb and distance based Low Pass. SECTR adds several more powerful tools.

Near 2D Blend

The simplest spatial effect in SECTR is the ability to blend 3D sounds that are close to the AudioListener into 2D sounds, called Near 2D Blend. In the real world, we almost always hear sounds in both ears, especially for sounds close by. The ear opposite the sound source hears sound waves as they reflect off nearby objects and as they pass through the skull. This effect is called a head related transfer function (HRTF). Doing a true HRTF is complicated, but SECTR's Near 2D Blend is a very simple approximation. One of the benefits of this feature is that sounds that pass by the camera no longer make a harsh pop as they go from one side of the stereo field to the other.

Occlusion

In the real world, there are often objects in between things that emit sounds and the people that hear them. These obstructions "occlude" the source, and change the character of the sound, generally making the sound quieter while removing high frequency components of the sound. To enable audio occlusion in SECTR, two things must be done. First, any AudioCue that should be occludable needs to have its Spatialization set to Occludable3D. Occlusion requires additional CPU so it must be enabled individually.

Second, an occlusion mode must be set in the AudioSystem. SECTR has two mechanisms for computing audio occlusion. You can chose which one (or both) to be active at once. When ray casting is enabled, the AudioSystem will periodically do ray checks against the scene collision from each Occludable Source to itself. When Graph occlusion is enabled, the AudioSystem will use the Sector/Portal graph to determine if the sound is audible. This solution will take things like the closed flags on doors into account. In both cases, the time between tests is determined by the RetestInterval property of the AudioSystem. Making the RetestInterval shorter will increase the accuracy of the occlusion calculations, but will increase the CPU cost for all Occludable3D AudioCues.

Propagation

In complex interior environments, the sounds we hear may not even come directly from the source. Imagine standing in a room at the end of an L shaped hallway. At the other end of the L shaped hallway is another room, in which a stereo is blasting music. In most games, you would hear the music coming directly from the point of the stereo, but in the real world the sound would appear to be coming through the doorway that leads into the hall, because the sound of the reflections down the hall

are louder than the sound waves that have to travel through walls. This phenomenon of how reflected sound moves through an environment is called audio propagation.

SECTR has support for audio propagation thanks to the PropagationSource component. You can place one of these anywhere in your level, provided you've set up Sectors and Portals, and the sounds will appear to reflect properly through your rooms and hallways. SECTR uses the Sector/Portal graph to create a fast approximation of real sound propagation. Even this approximation is more expensive than playing a simple 3D sound, so PropagationSources should be used sparingly, only where they make a discernable impact to the player. Lastly, PropagationSource performs its own occlusion calculations, so you should only use Linear3D, not Occludable3D AudioCues.

Custom Filters And Effects

Unity includes Audio Filters, which allow developers to add realtime DSP effects like reverb and echo to their game audio. SECTR allows you to apply these filters (and other effects) to AudioCues using the Prefab attribute. The Prefab attribute allows you to tell SECTR to use a custom Prefab to configure the instances of an AudioCue, rather than the default behavior which uses very simple GameObjects (basically just a Transform and AudioSource). You can find the Prefab attribute in the the Advanced section of the Cue properties. Note that while the Prefab attribute is most useful as a way to add AudioFilters to an AudioCue, you can include whatever components you like. As an example, you could add a Particle System component to the Prefab for certain AudioCues to ensure that they play a visual effect every time the sound is triggered.

Realtime Parameter Control

RPC is an advanced audio technique intended for cases that require AudioCues to change directly based on game state. RPC allows the game programmers to pass in parameter values (simple floats) which the audio designer can then use to modify properties of the AudioCue and any other components that come from the Prefab (if specified, see Custom Filters and Effect chapter).

To add RPC to an AudioCue, simply go to the Control Parameters section and press the "Add Parameter" button. Next, enter the name of the parameter and choose what attribute you would like to effect. You can give the Parameter any name that you like, but there are two special names which SECTR will set for you. The first is distance, which is the number of units between listener and AudioCue instance, and the second is time, which is the number of seconds that have elapsed since the AudioCue started playing. If you want to modify the property from a Prefab, choose Attribute.

Attribute will automatically activate a new GUI which automatically detects the RPC compatible attributes on your custom prefab. If you see an attribute that you think should support RPC but does not, please contact support. Be sure to set a good Default value for the parameter. This is the value that will be set when the AudioCue is first played, and will be the parameter value until a programmer specifies a new value. Next, it's time to modify the actual parameter curve. To edit the curve, simply double click the curve in the inspector or Audio Window.

In SECTR, parameters multiply the base value of the attribute, which means that 1 means "do nothing", a value of 2 means "double the base value" and a value of 0.5 means "half the base value". The final step in using RPC is for your programmer to set a value for the parameter on a per-instance basis, which they can do simply by calling the SECTR_AudioInstance.SetParameter() function. Note that parameter names are case sensitive.

Culling and Optimization

In order to remain efficient, SECTR avoids doing work whenever possible, which mostly boils down to effective culling and virtualization.

One Shot Culling

One shot sounds are generally fairly short, and it's likely that if the sound was inaudible when played, then it will be inaudible when it finishes, too. As such, SECTR will pre-cull any non-looping sound if the sound is played beyond its Max Distance, or if it is an HDR sound but is too quiet to be audible.

Loop Virtualization

Looping sounds last forever (or until stopped) so SECTR always produces AudioCueInstances for looping sounds. However, SECTR will virtualize these sounds when they become inaudible. Virtual instances appear identical to the rest of the game code, but the require the bare minimum of resources to maintain. The AudioSystem will periodically test to see if a looping sound should become virtual or become real. The amount of time between tests is determined by the RetestTime property in the AudioSystem. Lower numbers mean less latency and more aural accuracy, but it also consumes more CPU time. SECTR does try to minimize the impact of these tests by generating random times within the range specified by RetestTime.

Programmer's Guide

For programmers who plan to integrate their gameplay and systems code for Audio playback, this chapter explains the best practices for using the included API.

The most important thing to know as a programmer is that your code should talk directly to the AudioSystem whenever possible. If you want to play a simple sound, call SECTR_AudioSystem.Play(), don't create a new GameObject and add a PointSource component to it. Using the AudioSystem API directly allows you to play sounds with the minimum possible overhead. The one exception to this rule is if playing propagation sources. If you want to play propagation sources, you should use a Propagation Source component and call its Play() and Stop() functions.

When you play sounds directly through the AudioSystem, you will be given an AudioCueInstance, which acts like a handle to the currently playing sound. You are not required to use this handle, but if you want to modify the sound after playback, you will need to hold on to the instance handle. Note that because SECTR culls soun ds that are guaranteed to be inaudible, you may receive a null AudioCueInstance even when playing a valid AudioCue.

Because AudioCueInstances are handles, they may stop playing and/or become null at any time. Always use the bool operator (i.e. if(someInstance) …) before modifying an instance. Instance handles are pooled and recycled, so be sure to clear your handles when you are done with them (if the provided API does not do it for you). In particular, handles to oneshot sounds can be dangerous to hold on to in the current implementation of the handle system.

If you want to play sounds when a level is beginning, put your code in Start, not in OnEnable. The AudioSystem initializes itself in OnEnable and there is no guarantee that your component will be enabled before or after the AudioSystem.

Streaming your Scenes (SECTR Stream)

SECTR makes it easy to save memory, increase performance, and decrease load times by splitting your scene into multiple chunks and streaming them in and out in realtime. Streaming is ideal for titles targeting memory limited devices like tablets and smartphones, or games on any platform that want to have large, seamless worlds free of loading screens.
SECTR includes all of the editor tools and runtime components needed to split your scenes up and stream them at runtime. Games that already have already setup Sectors and Portals can be streaming in seconds, and it only takes a few minutes to setup a scene for streaming from scratch.
Note that if you want to stream large terrains you don't need to create the Sectors and the terrain chunks for streaming yourself – SECTR comes with a set of tools that automates splitting the terrains and putting the terrain splits into the right sectors. Streaming a terrain only takes a few clicks, check the "Terrain Streaming Quick Start Guide" in the documentation folder for a tutorial.
SECTR even handles streaming "global" objects like lightmaps, nav meshes, and light probes.

Overview

Streaming with SECTR works by taking each Sector in the scene and exporting it (along with all of its children) into an individual scene file (called a Chunk). During export, some additional data is added to both the exported scene and the main scene to ensure that the Chunk is loaded correctly. SECTR provides a convenient GUI for managing this export process, but the code is structured to allow command line exporting as well.

At runtime, each Chunk is managed by a Chunk component. These components are reference counted, which allows Chunk loads to be requested by many different components while guaranteeing that no memory is leaked and nothing is loaded unnecessarily.

While any script object may request that a Chunk be loaded, SECTR comes with a set of Loader components, designed to make it easy to load Chunks based on common patterns, like entering a trigger volume, opening a door, etc. Users are encouraged to use these components, extend them, or add your own.

One of the tricky aspects of chunking and streaming a Unity scene is handling the "global" level data like lightmaps, light probes, and nav meshes. SECTR takes a different approach for each of these components, as described in the Global Data section.

SECTR also includes a solution to another common problem with streaming, namely what to do with global objects that need to be in the scene, but which are in areas that have been unloaded. For more about this, see the Hibernators section.

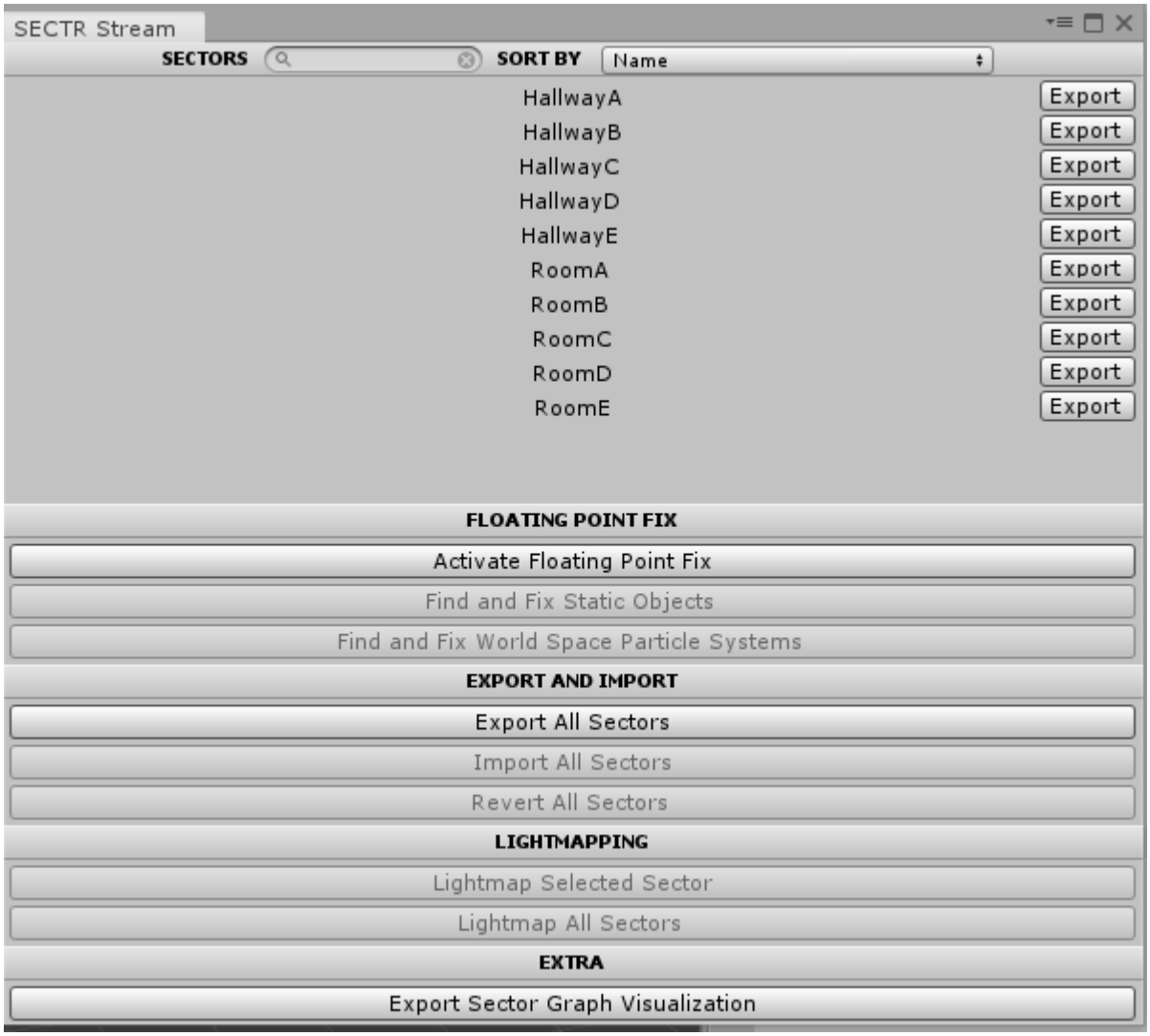
Setup and Exporting

Preparing a scene for export is easy. It simply requires the creation of one or more Sectors in the scene. For new users, please see the "Creating your first Sectors and Portals" section for more detail on creating Sectors.

Note that for streaming with SECTR, you do not actually have to create Portals unless you want to use some specific Loaders, and even then the Portals do not have to have geometry.

Once you've created some Sectors in your level, you should add some Loaders. Which Loaders are right for you depends entirely on the game you're trying to make, but adding a NeighborLoader to your main player GameObject is a good place to start.

With Sectors and Loaders created, it's time to split your scene into Chunks. The easiest way to do this is through the SECTR Stream Window, which you can find under Window->Procedural Worlds->SECTR->Stream Window.



Simply press the “Export All Sectors” button, follow the prompts, and wait until it completes. You will also be asked if you want to create a backup of your scene file before exporting. When the export is finished, you'll see that all of your scene geometry is now gone, and each Sector component now has a Chunk component, too. If you look in your Project window, you'll also see that there is a folder with the same name as your scene, and in that folder is a sub-folder called Chunks. That sub-folder contains all of the exported Chunks as individual Unity scene files. You should also see a folder called “Backups” if you chose the Backup option, containing a backup of your scene. Future backups will be collected in this folder as well.

The individual scene files are added to your build settings automatically as well so they can be loaded during runtime. It is important to have all the scenes that you want to stream included in the build settings in the final build. You can remove scenes partially for faster build times (E.g. if you are only testing parts of your game) but for the final build you need to have all scenes included or else they can’t be loaded by streaming.

It's important to point out that the export process will only export static Sectors and their children. This behavior is described further in the Global Objects section

Global Objects

While the bulk of the data in the scene can be exported into Chunks, Unity has some scene data that is considered global to the entire level. Also, as described previously, dynamic Sectors and any GameObjects that are not a child of a Sector will also not be exported. Together, these objects are considered "global" and require some special discussion here. A lot of interesting information about these global objects can also be found in the Unity Manual Page for “Multi Scene Editing”: https://docs.unity3d.com/Manual/MultiSceneEditing.html

Global GameObjects

As described above, dynamic (i.e. non-static) Sectors and any GameObjects not parented to a Sector will not be exported into a Chunk. This is necessary because an object can only be exported into a single Chunk, but Members may be in multiple Sectors at once. Non-static sectors are assumed to change at runtime, and are also unsafe to export. Once the game is running, however, both dynamic Sectors and global GameObjects they will still behave with the exported data just as they did before export. This behavior can actually be used as a design tool. Often, there are some objects in the level that should always be instantiated (i.e. the player, puzzle objects, etc.). As long as these objects do not try to hold direct references to exported objects, then they will function just as they did before the export.

Lightmaps

For games that use them, lightmaps can consume much of the game's total texture memory budget. Unity only has a single, global list of lightmaps, and normally all of them are loaded when the level is loaded. Fortunately, SECTR includes logic to get around this and actually stream lightmaps. The lightmap streaming works by computing (at export time) which lightmaps are used by a given Sector. When the Sector is exported, references to those lightmaps are included in the exported data. When the export is complete, any lightmaps that are no longer referenced are removed from the main scene. At runtime, individual lightmap textures are loaded along with the rest of their Chunk, after which SECTR fixes up the global lightmaps list. The opposite happens when Chunks are unloaded. The end result is that lightmap textures stream just like everything else. The only very important limitation to lightmap streaming is that lightmaps must be baked properly. For users of Unity 4.x, you must import all Sectors in the main scene and then bake using Unity's Lightmapping UI. In Unity 5.x, you must have all Sectors exported and then use the Bake Lightmaps button in the SECTR Stream UI. SECTR tries to ensure that the game does not crash if lightmaps are baked incorrectly, happens, but it may look very strange indeed until the data is re-baked and everything re-exported.

Nav Meshes

In Unity, there is only a single nav mesh per scene, which can make navigation across sectors difficult: While you could create a navmesh for each scene individually and connect those with off mesh links, you would run into problems when you have characters that need to navigate across sectors that are currently not loaded yet.
SECTR's solution to this is very simple - generate the NavMesh while all Sectors are in the scene, and then leave it alone. It will load along with the main scene and work fine. Generally, NavMesh data is not a significant amount of total memory, but if this is a concern for you, please contact support.

Light Probes

Baked light probe data is also stored globally in Unity. While it is possible to split this data up in a manner similar to lightmaps, SECTR currently does not do so. Like light meshes, simply bake light probe data while everything is in the main scene, and then leave it alone. Generally, light probes are not a significant amount of total memory, even with a large number of light probes.

Unity Occlusion Culling

Unitys Occlusion data is stored separately for each scene. However all other scenes that are open at the the time of the occlusion culling baking is performed will receive a reference to this occlusion culling data as well. This means if you bake the occlusion culling data with all sectors imported in, occlusion culling should work as if it was a single scene.

If you need a dynamic occlusion system that works with SECTR, note that SECTR comes with its own occlusion culling system, which is fully compatible with Streaming.

Loaders

While exporting Chunks is a critical first step towards making a streaming scene, they are not much use until someone actually requests that they be loaded. While any script may request a Chunk load, SECTR includes a number of different Loader components which implement common loading patterns. This section describes the functioning of each of those components. Note that Loaders do not need to be exclusive. You can have many independent requests to load the same Chunk and SECTR will do the right thing, loading the Chunk on first request and only unloading when all requests have been released.

Start Loader

This is the simplest of all Loaders. It simply loads whatever Sectors it is in at Start, and then removes itself from the game. Start Loader is meant to be used in combination with Loading Door, to balance out the door's reference counts.

Trigger Loader

This component lets you load a list of Sectors whenever a particular Unity Trigger is activated. These same Sectors will be unloaded whenever the trigger is deactivated (provided no one else is trying to load them).

Neighbor Loader

This component works by loading the current Sector(s) as well as connected (i.e. neighboring) Sectors. The exact definition of what to consider a neighbor Sector is determined by the Max Depth property (for programmers, this is the max depth of a breadth first traversal of the Sector/Portal graph). In practice, a Max Depth of 0 means only load the Sectors that the Neighbor Loader is currently in. A MaxDepth of 1 means also add any Sector connected to a current Sector. A Max Depth of 2 means also load any Sector connected to one of those Sectors. And so on. Note that a MaxDepth greater than 0 requires that your Sectors be connected by Portals, though the exact geometry of the Portals does not matter.

Loading Door

This component is perhaps the most clever of the Loaders. It's designed for games that are laid out like Metroid Prime or Dead Space, where rooms are separated by doors that do not open until the room on the opposite side is loaded, and where that load is triggered by getting close to the door. Specifically, Loading Door works by combining a Trigger and a reference to a Portal. Whenever the Trigger is activated, the Loading Door loads the Sector on the opposite side of the door's

Portal. When the player exits the Trigger, the door unloads a Sector, but which Sector is unloaded depends on whether the player passed through the door or not. The Loading Door also waits for the door to close before unloading anything.

Group Loader

There are occasions where a section of the scene needs to be split into multiple Sectors (perhaps for occlusion culling or game logic) but they need to be loaded as if they were part of a single Sectors. Group Loader takes care of this, by automatically incrementing and decrementing reference counts whenever one of the Sectors in the list is loaded or unloaded.

Region Loader

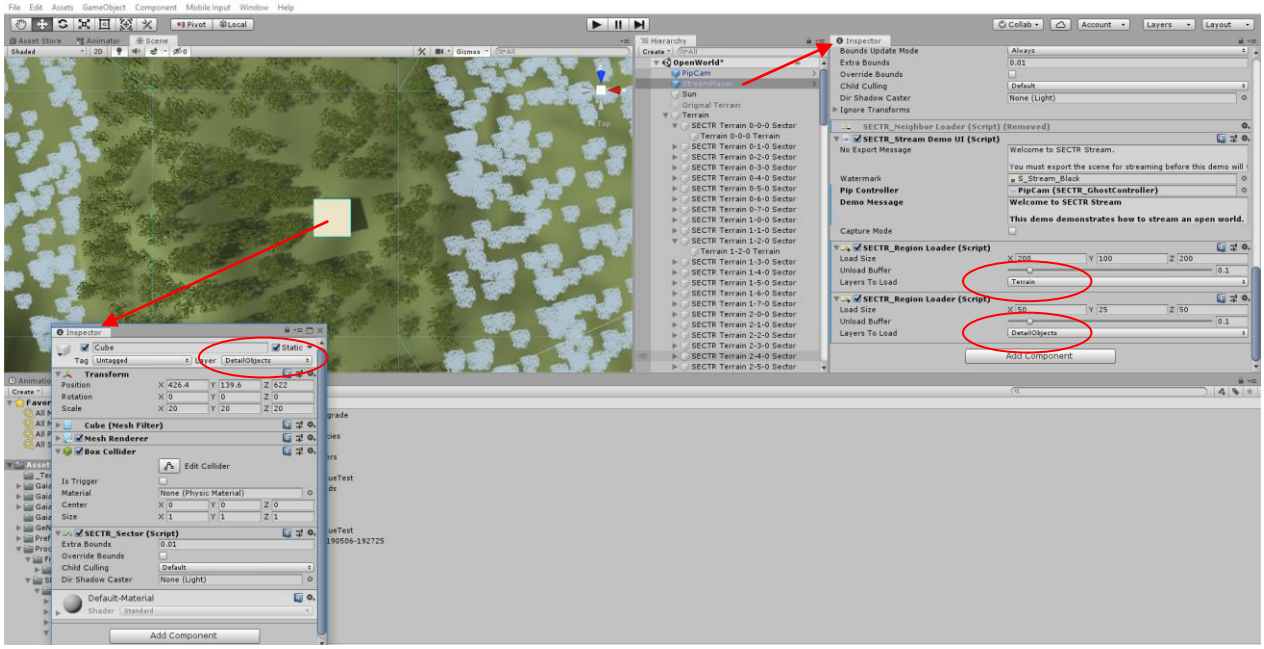
If you just want to load all of the Sectors within a volume, Region Loader is for you. Region Loader is most commonly used for terrains, where you want to load all terrains grid elements around the player. Region Loader supports a Layer mask which you can use to filter which Sectors are considered by the Region Loader. This is useful if you want to load terrain with the Region Loader, but load other Sectors, like the interior of a building, using other techniques (like a Door Loader).

DIY Loader

While you can use any of the above Loaders, you are not required to use a Loader at all. You are free to directly request Chunks to load and unload in your own script code, PlayMaker routines, or anything else. Just be careful that you always match calls to SECTR_Chunk.AddReference() and SECTR_Chunk.RemoveReference()).

Streaming in Layers / Nested Sectors

While it might be enough most of the time to stream Sectors that are nicely aligned with each other, it can also be useful to have Sectors inside of other Sectors. A common use case for this is to separate larger GameObjects from smaller details to load them in different distances. For example you can load terrain chunks in a larger Grid structure, and then have smaller sectors inside them with detail GameObjects that you only load when the player is really close. To do so, simply create further Sectors inside other Sectors for your detail objects.
To load them at a different distance than your terrain pieces, put them on another layer than the terrain pieces, and add a region loader for this layer with a different loading distance than the original loader for the terrain. The following screenshot shows the general setup:



The cube on the terrain is its own Sector, set up on the DetailObjects Layer. The Player Object selected in the scene hierarchy possesses two Region Loaders, one operating on the terrain layer with a larger loading distance (200), another operating on the DetailObjects Layer with a smaller distance (50).
When these Sectors are exported and the scene is run, the Terrain chunks will be loaded in on 200 units around the player, while the cube will appear when the player crosses the loading distance of 50 units for the DetailObjects.

Hibernators

As described earlier, GameObjects can be left as part of the main scene so that they are never unloaded. However, while these global GameObjects are never unloaded, the Sectors around them may very well be. When this happens, it's often desirable for the global object to "hibernate" - to stop updating - until its part of the scene is loaded again. The Hibernator component is designed to do exactly this.

Specifically, the Hibernator component can disable other components whenever its Sector is unloaded, and re-enable them when the Sector is loaded again. The Hibernator has properties that control which kinds of Components should be affected by hibernation, and whether children of the Hibernator should be affected or not.

For programmers, Hibernator also provides a set of Events that you can subscribe to. These Events will tell you whenever the Hibernator sleeps, wakes up, or is updated while hibernated.

Chunk Proxies

By default, when a Sector is unloaded, it is simply invisible, appearing when loaded and disappearing when unloaded. However, some games need to have a visual representation of the Sector when the Sector itself is unloaded. This "proxy" representation is usually a simplified version of the Sector itself, one that appears when the Sector is unloaded and disappears when the Sector is loaded. SECTR includes tools and logic within the Chunk component to make creating Proxies simple and easy.

Creating Proxy Meshes

Each Chunk component can only have one Proxy Mesh, though the mesh can have multiple sub-Meshes. This mesh can be any Unity mesh resource, including meshes created by hand. As a convenience, the Chunk component also includes a tool that can create Proxy Meshes for you automatically. Simply expand the Create Proxy Mesh foldout, select the child meshes that you wish to include, and press the Make Proxy Mesh button. This will create a new mesh resource that is a combination of all selected meshes, including combining subsets with identical materials.

Proxy Mesh Memory

It's important to note that while the Proxy Mesh disappears when the Sector is loaded, it is simply hidden, not unloaded. This means that the Proxy Mesh and all textures and materials referenced by it will be in memory as long as the level is loaded. In other words, Proxy Meshes improve visual fidelity but require more memory to do so.

Collaboration

As teams get larger, they often run into an issue where multiple people want to edit the same scene file at the same time, which Unity normally does not allow. With SECTR, however, it is possible for multiple people to share a (split) scene by editing the exported Sector scenes, provided that they follow a few rules:

- When editing a Chunk file, care should be taken not to delete or otherwise modify the root node. This node contains important information necessary to stream that Chunk, information which users should not modify.
- Most newly created objects should be parented to the root node or one of its children. This will ensure that they remain part of the Sector during loads and unloads.
- New global objects can be created in the Chunk scenes. However, they will not function correctly until imported into the main scene and re-exported.

- Global Unity data like lightmaps, Nav Meshes, Light Probes, should never be generated within a Chunk scene. Doing so may cause problems at runtime.

Fix for floating point imprecision

Unity (& many other game engines) suffer from the problem that graphic and physics calculation can become imprecise after a certain distance from the world origin. When your scene exceeds roughly 5000 units into any direction it can happen that you see symptoms like:

- Shadows flickering
- Shaky physics
- Issues in animations

SECTR contains a "floating point fix" component combats this by shifting all GameObjects back to the origin after a certain threshold is reached. If you set the threshold at 100 units for example, your camera will simply put back to the 0,0,0 position after it has moved more than 100 units from the origin, and all other gameobjects currently in the scene will be shifted as such as well.
While this is barely noticeable for the player, it prevents the floating point imprecision issues as mentioned above as the world never exceeds 5000 units because it is shifted back before things like these can happen.

Please note that the floating point fix is considered to be more like a giant workaround for a fundamental problem in the game engine, not a "cool feature" that you should enable just because it exists. Enabling the floating point fix comes with the following trade-offs that you need to consider:

- Shifting all GameObjects in a scene around can be a costly operation, so it might lead to slight hitches / lags when it is performed. The good news is you usually don't need to perform this shift very often, so it should be a rather rare occurance.
- All objects that need to be shifted can't be marked as static anymore, else they can't shift.
- Some graphical effects such as Motion Blur or TAA can additionally exaggerate the shifting effect, making it more prominent when it happens.
- As soon as you enable the fix, you need to take the world shifting into consideration when programming your gameplay logic. If you want to calculate a distance how far the player has traveled for example, you can't simply take the transform.position value anymore, as it might just shift around during gameplay anytime. There is a static function `SECTR_FloatingPointFix_ConvertToOriginalSpace()` that you can call anywhere to get the "game world" position of a vector 3, just as if the shifting never happened.

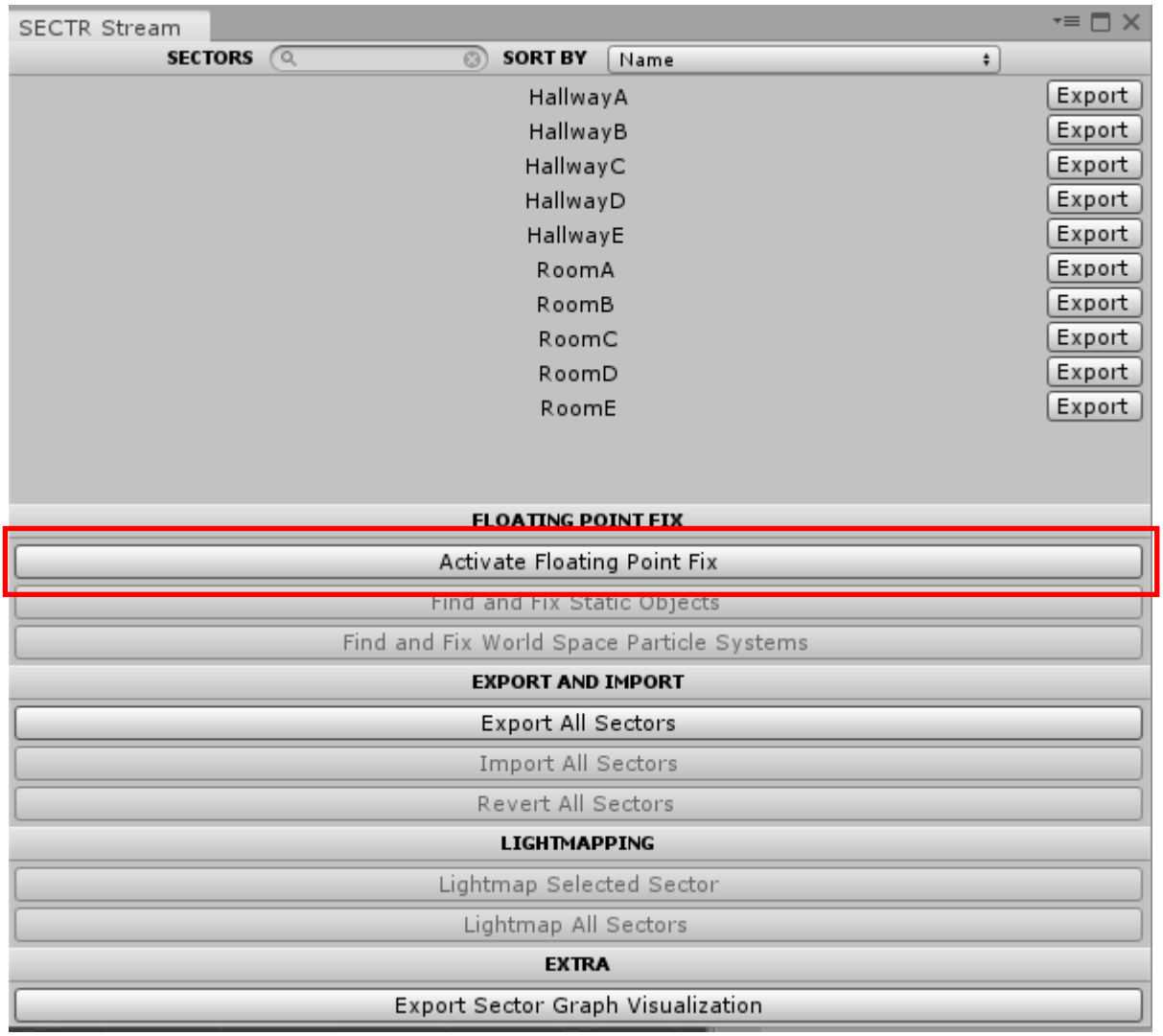
- GameObjects / 3rd Party Assets that are dependent on tracking points in world space will most likely not function properly when the shift happens or need to be adapted for it.
- Due to the nature of the issue and the fix, the fix can only be applied to a single object that determines the position that the world shifts around accordingly, usually the camera or the player GameObject.

As you can see from these points, the floating point fix is something you really should consider only then when you are actually running into the issues as mentioned above, otherwise it will create more problems for you than it can solve.

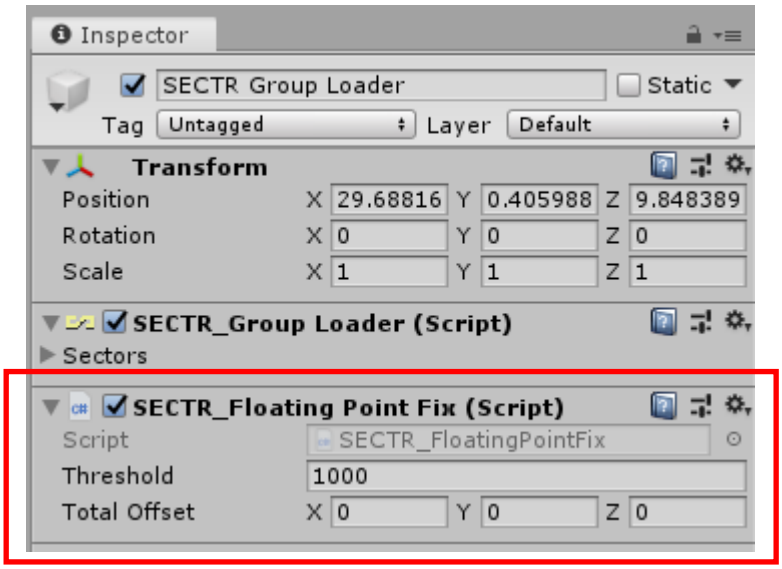
Activating the Fix

The fix can be activated from the stream window. (Technically the fix does not have anything to do with streaming, but usually you are looking into streaming already anyways due to your large scene size when you are experiencing the floating point issues.

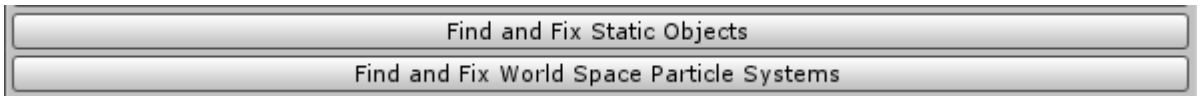
To activate the fix, click the "Activate Floating Point Fix" button in the Stream window:



Acivating the fix via this button will attach a floating point fix component to the first loader that is found in the scene. You can also activate the fix manually by just adding the component to the desired loader in your scene:



Adding the Floating Point Fix to the scene either way will unlock new entries in the streaming window:



"Find and Fix Static Objects" will go through the currently loaded sectors and remove the static flag from these objects so they can be moved around later during runtime.

"Find and Fix World Space Particle Systems" will go through the currently loaded sectors and will add an additional fix to particle systems that simulate their particles in world space.

Since those two checks only operate on the currently loaded sectors, it is recommended to load in all sectors one time after adding the fix component so you can perform those checks. After you have run those two checks and there are no static objects / unfixed particle systems remaining, you can Export all sectors again.

Fixing global or non-streamed objects

When you run the game now, the fix will automatically be applied to all sectors that are subject to streaming and their contents. But what about global objects or sectors that are not streamed at all? You can simply mark them for the fix being applied to them by adding the SECTR_FloatingPointFixMember script to them. This will automatically mark them for the fix, and they will be affected by the world shifting as well during runtime.

Dynamic Occlusion Culling (SECTR Vis)

Overview

SECTR also contains is a high performance, low memory, occlusion culling solution for Unity, with support for the complete set of Unity rendering primitives including lights, shadows, particles, meshes, and terrain. SECTR occlusion culling is fully dynamic and requires no tedious baking process. Heavily optimized, SECTR occlusion culling is a great solution for all platforms, especially tablets, smart phones, portables, and "last gen" consoles.

SECTR occlusion culling works with any Unity scene, but is designed primarily for indoor or hybrid-indoor (think most popular first and third person games) games. Only truly outdoor, fully open world games are not a good fit for SECTR occlusion culling.

The occlusion culling by SECTR is an implementation of recursive, portal based occlusion culling. The culling starts when a SECTR CullingCamera component is added to one of the Cameras in a scene with Sectors and Portals. Before the Camera renders, the CullingCamera does its work.

The following is a basic outline of the culling algorithm:

1. Starting in the current sector, the CullingCamera tests its frustum against all of the renderable objects in that Sector, hiding any that are not visible.
2. Next, the CullingCamera tests its frustum against all of the Portals leading out of that room. If any portals are visible, a new frustum is generated by performing a geometric intersection of the camera frustum and that Portal, dramatically winnowing down the size of the frustum.
3. This new, clipped frustum is then passed into the Sector on the other side of the portal, and the objects in that Sector are tested as in Step 1, and the Portals in Step 2.
4. This process continues until there are no visible portals. Usually, this process terminates quickly due to the natural layout of game spaces.

Because all of this logic is implemented during PreCull (and cleaned up in PostRender), it's completely safe to have multiple CullingCameras active in a scene at once (i.e. for split-screen multiplayer or PiP gameplay).

SECTR also supports Occluders, which can be thought of as anti-Portals. Where a Portal allows visibility to pass, Occluders prevent visibility from passing, hiding any objects behind them. Occluders are discussed in greater detail in their own section below.

The entire SECTR occlusion culling module is designed for maximum simplicity and controllability, with as little "automagic" as possible. This means that it takes more to enable culling than the simple press of a button, but it also means that you can control the balance memory, CPU, and GPU that is right for your game, on the platforms you care about.

Setting Up

Preparing a scene for SECTR occlusion culling is easy. First, you need to have a scene with some Sectors and Portals in it. For new users, please see the “Creating your first Sectors and Portals” section for more detail on how to do that.

Once your scene has Sectors and Portals in it, enabling culling is as easy as adding a CullingCamera component to the main camera of your scene. If you have multiple game cameras, each one can have a CullingCamera component. They will not interfere with one another.

If you want to see the effects of the culling in the Unity Editor, select the GameObject with the CullingCamera component on it and make sure that the Gizmos are drawing for CullingCamera. As you move the camera around the scene, you'll see the debug rendering of the frustums as they propagate through the Portals. If you run the game in the editor and then select the CullingCamera, you'll actually see objects appear and disappear as they are culled.

One common question when setting up a scene for occlusion culling is which objects should be children of Sectors and which should have their own SECTR Members. SECTR occlusion culling was designed to make this process as simple as possible, provided you follow two simple rules:

- Objects that move between sectors should have a Sector Member component.
- Objects that extend beyond bounds of their Sector should have a Sector Member component.

Following these two rules will ensure that the visibility calculations are always correct, even in tricky situations like objects outside a Sector casting shadows into it.

Occluders

While Portals do most of the culling work, there are cases where they are not completely effective on their own. In particular, large, open spaces (like huge rooms our outdoor sections of hybrid-indoor games) may have a large number of objects, but few natural windows or doorways. Fortunately, these scenes often have large, solid objects in them, and that's where Occluders come in.

Where Portals allow visibility to pass through them, Occluders block visibility, and hide all of the objects behind them. This makes them a good fit for large, solid objects in big spaces. These large objects hide many objects and can be used for culling. To do so, simply add a SECTR Occluder component to the object in question and draw a mesh for it.

Like Portals, Occluders in SECTR are required to be convex and planar (for performance reasons). This requirement works well for some occluders (like a big wall) but may not work well for others (like a cylindrical column). To handles cases like columns, Occluders can be set to auto-orient to the culling camera, and users can specify which axes should auto-orient. This makes it possible to efficiently represent a much larger range of occluded shapes. For example, to represent a column, make a rectangular mesh for the Occluder and set it to auto-orient about its Y axis.

Shadows and Occlusion

SECTR occlusion culling is designed to accurately and efficiently properly cull all shadow casting lights. The one tricky case for SECTR is handling shadows cast by directional lights, because they cast very long shadows and have infinite bounds. SECTR supports shadow casting directional lights, but in order to keep performance high, requires one additional piece of human intervention.

Each Sector and Member has an attribute for a Directional Shadow Caster. If the object that this Member is part should cast shadows from a directional light, simply assign that object to the Directional Shadow Caster attribute. This works because Unity will only cast shadows from a single directional light at a time anyway.

You may also wish to tweak the Directional Shadow Distance attribute in the Shadow Culler. This value determines how much to the object's render bounds will be extended during culling. The value must be high enough that it matches the maximum distance of the cast shadow. However, excessively large values will waste CPU time.

You can change the Direction Shadow Caster at any time, just make sure you use the light that's actually casting the directional shadows.

Level of Detail

SECTR occlusion culling includes support for dynamic LOD (level of detail). LODs are most commonly used to replace high poly geometry with lower poly substitutes when an object is far from the camera, but SECTR VIS LODs can be used to control any game object, including ones with lights, particles, collision, etc.

Creating LODs

Creating LODs is as simple as adding a SECTR LOD component to any game object. Once added, use the Inspector to add LODs and specify which children of the node should be included in each LOD. Children will be activated when their LOD is in range, and deactivated when it is out of range. Any child that is not part of any LOD will be unaffected by the LOD system.

Once your LOD components are created, make sure to add a SECTR Culling Camera component to the scene camera. The Culling Camera is responsible for actually updating LODs every frame. Note that SECTR LOD components are compatible with the SECTR occlusion culling but do not require Sectors or Portals to work.

LOD Selection

The current LOD is selected based on the screen space size of the object. LOD thresholds are expressed as a percentage of screen space size. For example, a threshold of 50% for LOD0 means that LOD0 will be active whenever the object is 50% of the screen or larger.

The LOD bounds are computed based on the union of all renderable objects in all of the LODs. The LOD bounds are separate from the Member bounds in order to ensure that they are stable and conservative, both of which are required for high quality LODs.

Optimization

SECTR occlusion culling tries to be as efficient as possible "out of the box" but there are some additional steps that users can take to squeeze more performance out of the system.

Basics

If any Sector, Portal, or Occluder will not move during gameplay, you can mark it as Static and get some additional CPU savings. Just make sure that this object really will be Static.

SECTR occlusion culling works with Portals with any number of sides. However, the more sides the Portal has the more expensive it is to cull. Portal geometry usually does not need to exactly match the visible geometry, so you can save CPU time by simplifying the silhouette of your Portal.

Controlling the Hierarchy

SECTR will cull any object that has a Sector or a Member component on it, as well as all of the children of that object. Sector and Member have an attribute called Child Culling that determines how that group of objects will be called. The value you assign to Child Culling determines if each child will be culled based on its individual bounding box or if children will be culled based on their aggregate bounds. Individual culling is more accurate, but requires more CPU. In some cases, the marginal benefit of more accurate culling is not worth the extra CPU cost of the detailed culling. The default behavior is to cull individual children for Sectors and to cull the group as a whole for Member. You can override these behaviors and even add new Member components to control the granularity of the culling.

One example of this might be on lower power platforms (like smartphones and tablets). On these platforms, it may be desirable to mark the Cullers of each Sector to not cull individually. That means Sectors will only be culled if they are completely invisible, but that kind of gross rejection may be worth the savings of detailed culling.

Another example is culling a pile of objects, like a stack of boxes. Even on high end machines, it may not be worth the CPU time to cull each box individually. Instead, it would be sufficient if they render or cull together. To implement this, simply create a new empty transform for the pile of objects, add a Culler component to that new object, and set Cull Individually to "false" for the new Culler.

MultiThreading

SECTR can use multiple threads to perform occlusion culling in parallel. On games with complex scenes and multiple CPUs, this can yield a significant performance increase. To enable multi-threaded culling, simple set the "Num Worker Threads" variable to a value greater than 0 in the Culling Camera component.

Some care should be taken when enabling multi-threaded culling. The Unity scheduler is only one of many systems that use threads, and it is possible for threads created by SECTR to be halted while other critical work happens. If a SECTR thread is halted for too long, it can cause a spike in the amount of time spent culling which will feel like a hitch to the player. Usually it's best to start testing with 1 or 2 threads, and to pick the number of threads based on the number of player CPUs for production games.

Multiple Active Culling Cameras

If you want to have multiple cameras that need independent occlusion culling (like a picture-in-picture security cam), you have a few options, each of which work around an issue in Unity's multi-camera culling:

- Go to SECTR/Scripts/Vis/SECTR_CullingCamera.cs and uncomment LAYER_CULL_RENDERERS and LAYER_CULL_TERRAIN.
- Create a new project layer called InvisibleCulling.
- Select your culling cameras and set the Invisible Layer property to the InvisibleCulling layer.
- If you have lights and renderers on the same GameObject, make sure that the bounds of the light is the same size or smaller than the renderer. If you don't, you may see visual glitches when one of them culls unexpectedly.

Note that for games that use multiple cameras for UI, FPS weapons, etc. you can ignore this section as long as only your main camera needs scene culling.

Comparison with Unity Occlusion Culling

No discussion of SECTR occlusion culling would be complete without an objective comparison with Umbra, the built in Occlusion system for Unity. The following are some of the major differences between the products:

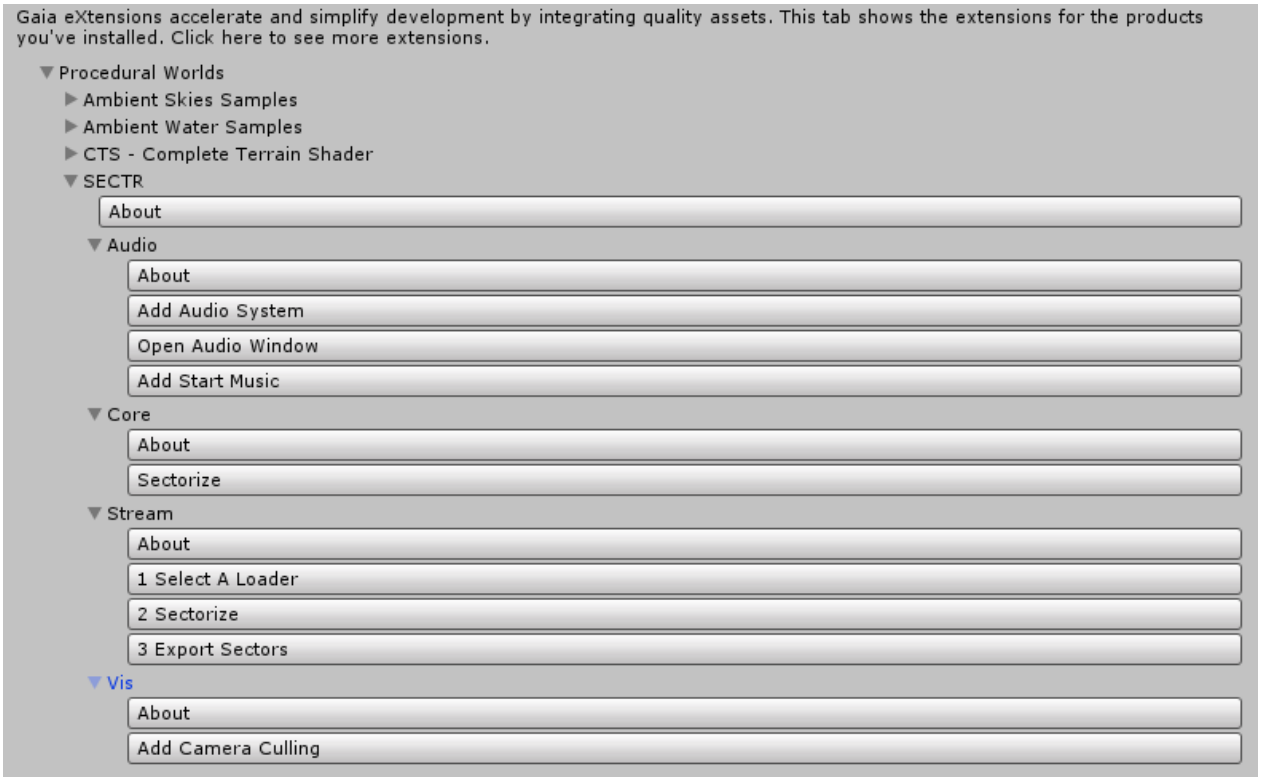
- SECTR occlusion culling is designed for indoor and hybrid-outdoor games, where Umbra works for fully outdoor and fully indoor games. Umbra is especially effective where SECTR occlusion culling is weakest - fully outdoor environments of the kind you see in open world games.
- Umbra requires a baking step. While this has gotten much better since 4.3, it can still take time on large scenes. SECTR occlusion culling requires no baking at all.
- Umbra has some dynamic elements (if you use their helper components), but they are not well explained and they have dependencies on some of the static, baked data. SECTR occlusion culling is completely dynamic, so if you can create it, you can change it during the game.
- Umbra can be difficult to set up and maintain for scene streaming: Every time you make changes to the layout in one of the sectors you would need to load them in afterwards to rebake the occlusion information, and then export them again. With the dynamic occlusion culling of SECTR this is not required.
- Umbra is implemented in C/C++ and has access to native Unity code. SECTR occlusion culling is implemented in C#, and does not have super low level access to engine or hardware. While Umbra and SECTR occlusion culling perform comparably in many test cases, Umbra's baking step and low level access mean it will likely always have the raw speed advantage.
- Umbra is closed source to most Unity users, while SECTR occlusion culling includes all source as part of its license.
- Umbra is designed to be an auto-magic solution. It requires nothing more than a button press to get working, but it can be difficult to control if that auto-magic solution does not produce the results you want. SECTR occlusion culling requires some human work to setup, but once set up does exactly what you tell it to. The products make different tradeoffs here, and which is right for you, only you can say.

Integration with other Procedural Worlds Products

Since SECTR is part of the Procedural Worlds product family, there are several integrations with other PW products available.

Gaia

When using Gaia, you can find additional entries for the integration with SECTR in Gaias "GX" panel. These integrations will allow you to set up common SECTR elements quickly on your terrain.



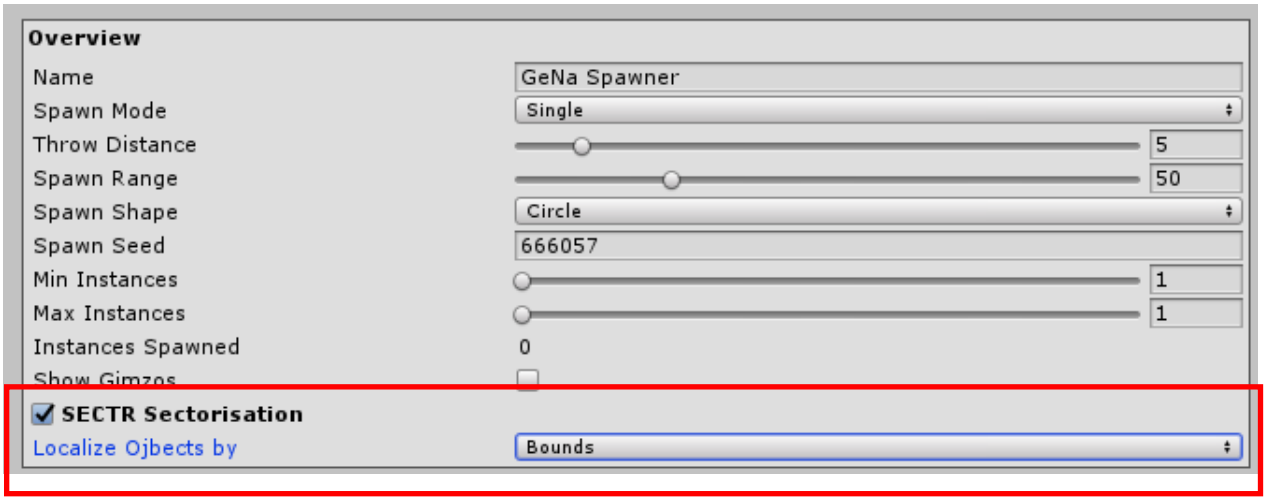
Especially notice the three steps outlined in the „Stream" integration. If you follow these three steps after creating a terrain with Gaia, you can set up your terrain for streaming mostly automatically.

CTS

When you are using CTS on a terrain that you split up for sectorization, the same CTS profile will automatically be applied to all the terrain split chunks that are being created in the process. This means your created chunks will just look the same if used in streaming or occlusion culling with SECTR, no extra steps needed.
If you are using CTS in a streaming scenario, it is recommended to look into the CTS option to "Disconnect the profile" from the terrain. This will create a persistent material for your terrain which has less overhead when the terrain is being loaded in for streaming, which is beneficial for streaming performance.

GeNa 2

When using GeNa 2 and SECTR together in a project, additional options will appear in the spawner to allow you to place your spawning result directly in the fitting sector which can be a huge timesaver when working on an already sectorized scene:



Advanced Topics

Core Libraries

In addition to the components and tools described above, SECTR includes libraries that perform basic functions related to Sectors and Portals. These routines are foundational to many SECTR algorithms, and are used in CORE and the other modules.

Sector/Portal Graph

When Sectors and connected by Portals, they form a graph, called the Sector/Portal Graph. In SECTR we think of the Sectors as nodes of the graph, and Portals as the edges between them. The Graph library includes many useful functions for easily exploring the graph including:

- **Relationships**: This includes information on how the Sectors and Portals are connected, like which Sectors are the neighbors of the current sector, or how many Portals there are between one sector and another.
- **Traversal**: The Graph can be traversed in a variety of ways. The Sector/Portal graph is cyclic, so some care must be taken to not-revisit nodes unless desired, which the included traversal routines demonstrate.

- **Pathfinding**: Should you want to find the shortest path between two points, the Graph includes pathfinding routines, which are based on a well optimized A* based pathfinding routine that should be sufficient for most needs.

Geometry Routines

SECTR is, at its core, a spatial library. Much of its value comes from doing interesting things in 3D (or 2D) space. Many of these routines require access to a common suite of geometric functions, which SECTR provides through the SECTR_Geometry library including:

- **Spatial Queries:** Determine which Sectors contain a particular point or volume.
- **Bounds:** Compute the extends of different kinds of objects (like lights) and evaluate their intersections.
- **Geometry:** Evaluate meshes for planarity, convexity, and the like.

Optimization

SECTR is designed to be easy and correct by default, while also being as efficient as possible. As you become more familiar with SECTR and how you use it in your game, you can use the following techniques to further reduce the CPU cost of SECTR components.

Marking Objects as Static

If you know that an object like a Sector, Portal, or Member will not move during gameplay, you should mark it as Static. When marked as Static, SECTR objects will precompute as much data as possible on Start, saving CPU time during regular gameplay. SECTR components on static objects can be safely enabled, disabled, created, and destroyed during gameplay.

Using Member Bounds Modes

In order to work correctly, Sectors and Members need to compute information about their children and their bounds. For games with a large number of non-static Sectors or Members, this CPU time can add up. However, you can significantly reduce the amount of time spent in SECTR_Member.LateUpdate by using the Bounds Update Mode attribute.

- Static: Makes the object behave as if its Game Object was marked static. Use this when marking the entire object as static causes problems.
- Always: Updates all children and bounding information every frame. This will always give the correct results for dynamic objects, but it's the most

expensive, and is often overkill.

- Movement: Updates all children and bounding information when the object moves, but does nothing while the object is stationary. Much cheaper than Always for objects that move around periodically, but are mostly stationary.
- Start: A hybrid of Static and Movement, this computes the children only at Start and updates the bounds when that object moves. This is the fastest Bounds Update Mode, and is ideal for objects that are always on the move, like NPCs, but whose children don't change significantly during their lifetime.

Advanced Bounds Controls

Much of the behavior Sector and Member are determined by their Bounds. SECTR does everything it can to efficiently compute the correct, most useful Bounds by default, but sometimes you simply need more control. SECTR provides the following controls for refining or overriding the default Bounds logic.

Extra Bounds

Extra bounds allows you to specify a certain amount of extra "padding" to the Bounds of a Sector or a Member. SECTR uses a small amount of padding by default to work around numerical precision issues, but you can use more (or less) if you want to quickly grow or shrink the default computed bounds.

Bounds Override

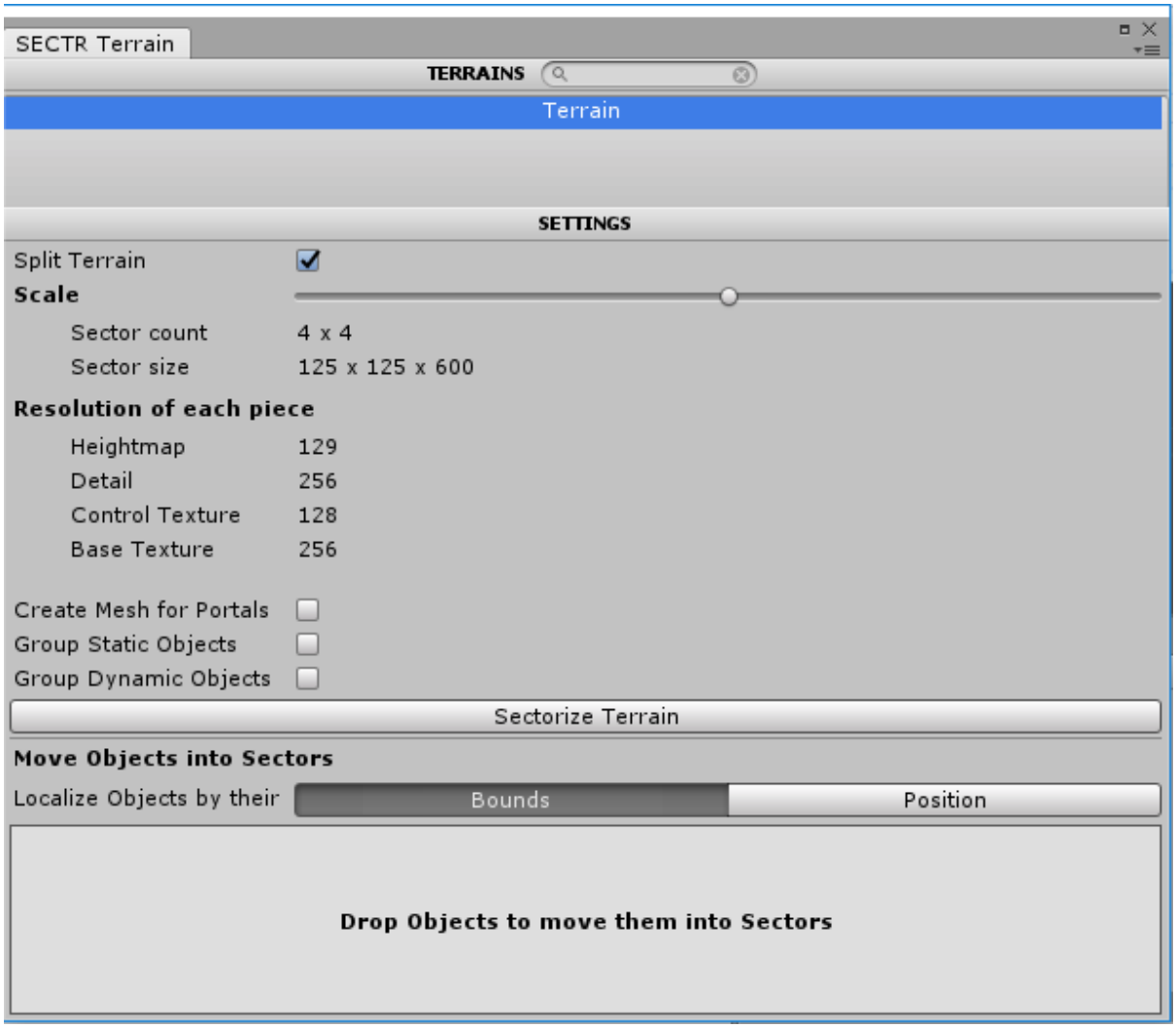
There are occasions where the default computed bounds are simply insufficient and need to be overridden. In these cases, you can specify your own bounds by ticking the Bounds Override check box and entering in your own bounding volume. Note that the bounding volume is in world space and will not update if the Sector transform is moved around.

Bounds Update Mode

Member includes a number of algorithms for how and when the recompute the Bounds, algorithms that trade accuracy for efficiency. These options are described in greater detail in the Optimization section.

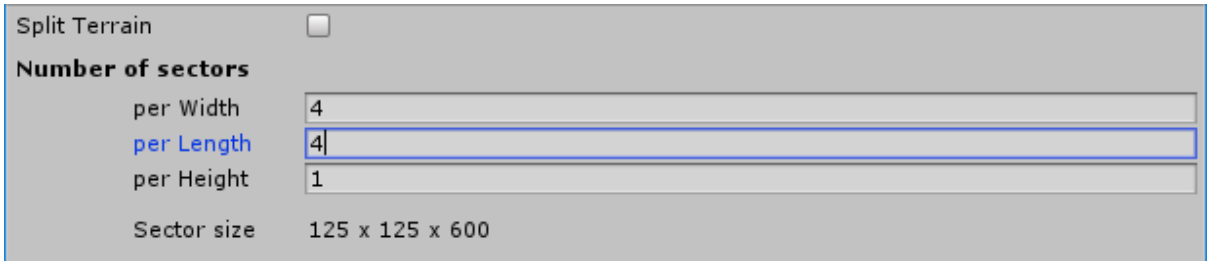
Working With Terrain

For games that are primarily built out of Unity Terrain (like open world or RTS games), SECTR includes a custom Terrain window. This tool makes it easy to automatically create Sectors and Portals for Unity Terrain, and also includes support for terrains created with Gaia. You can access the Terrain Window via
Window -> Procedural Worlds -> SECTR -> Terrain Window



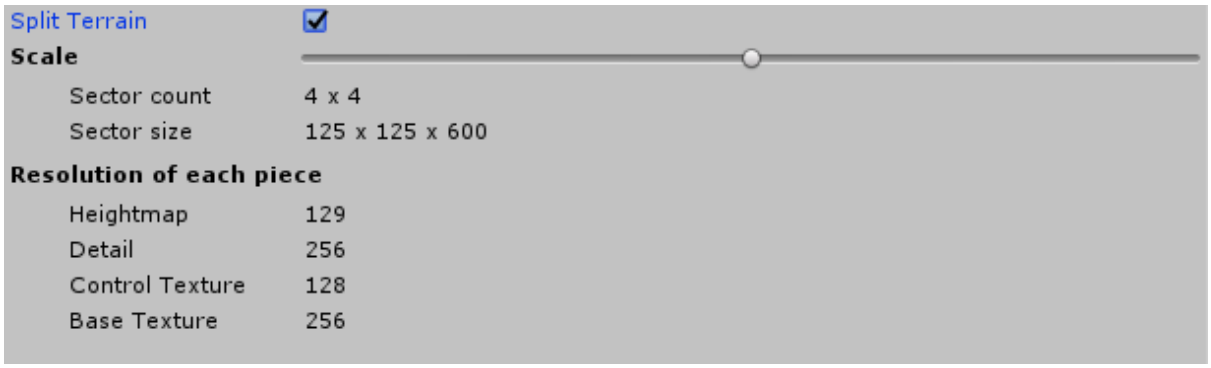
Single Terrain Sectorization

The simplest way to build Sectors and Portals on a Terrain is to Sectorize a single terrain. This mode can create any number of Sectors and Portals in length, width, and height. However, the terrain itself will not be in any Sector, because it is too large. This mode works well for games that want the objects on the Terrain to be in Sectors, but for the Terrain itself to be "global". To choose this mode, uncheck the box for "Split Terrain" in the Terrain Window. This changes the options to query how many sectors you want to create across the terrain:



Terrain Split Sectorization

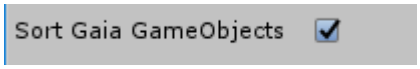
In cases where you want a single Terrain to be contained within individual Sectors, the Terrain Window provides a split option which will split a single terrain into multiple, smaller terrains while preserving the geometry, splats, trees, and other important information. Because this split mode will create multiple Unity terrains, the options are more limited. Specifically, the number of Sectors in width and length must match, must be powers of two, and must not create any terrains smaller than 32x32. The Terrain GUI will automatically disable Sectorization if these criteria are not met. To choose this mode, check the box for "Split Terrain" in the Terrain Window:



Automatically Including Objects

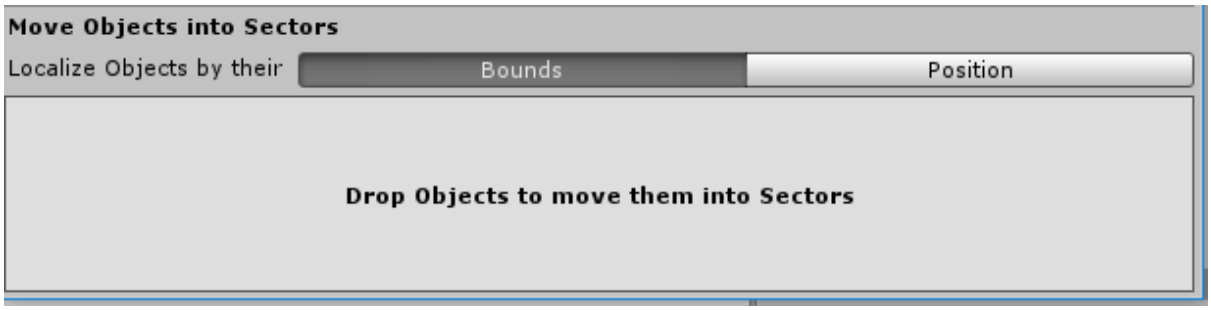
Games often include many objects placed on top of the Terrain. The Terrain window can automatically include static and/or dynamic objects during the Sectorization process. Note that only objects whose bounds are entirely included by a Sector will be parented. Any objects that overlap more than one Sector will be left in the global set.

If you are using Gaia, a special checkbox for sorting the Gaia spawned objects into Sectors will appear. Activating this checkbox will take the GameObjects out of the Gaia Spawners and put them into the correct Sectors during terrain splitting / sectorization.



Inactive Gaia Spawners will **NOT** be processed, this is intentional to give you an option to opt-out from automated sorting for certain spawners.

If you find yourself in the situation that you already split / sectorized the terrain, but now also got a heap of unsorted GameObjects in the scene that need to be assigned to the correct Sector, the Terrain Window has a solution for you as well: Just drag and drop the GameObjects on the "Drop Objects to move them into Sectors" box at the bottom of the Window:



The GameObjects will then be sorted away in the fitting Sector, depending on the objects position or its bounds (switch between the options with the corresponding buttons above the drop area box).

If the GameObject could not be moved into a sector there will be a warning popup that notifies you. A GameObject can not be assigned to a sector in the following two cases:

A) You selected bounds mode, and the bounds of the GameObject go over multiple Sector borders or exceed a single Sector.

B) You selected "Position" and the position of the GameObject does not fall into the bounds of any Sector at all.

For both cases you will need to review the GameObject in question and make the appropriate changes so that it can be sorted automatically, or you can just assign it to a Sector by hand by making it a child of this Sector in the hierarchy.

SECTR API

SECTR comes with a powerful API that lets you control many aspects of SECTR'S behaviour and code your own complex solutions in the SECTR framework. The API is documentation can be found in a separate document in the "Documentation" folder.