Areas

Areas are oblong-shaped collision zones which can trigger callbacks when the player or other objects enter them; they are also often used as position markers for things spawned in code such as sounds or particle effects.

HPL3 ships with several different area types - here's a quick summary of them and of some of their properties.

Trigger

The most commonly used type of area, used for all sorts of purposes.

Important Properties

BlocksLineOfSight

If true, this area acts as a blocker for AI seeing the player.

CC_Entities, CC_Funcs

Collision callbacks - for example, if you set CC_Entities to `player` and CC_Funcs to `TriggerHitArea` then TriggerHitArea will be called in your map script whenever the player enters or leaves the area.

PlayerLookAtCallback

Called when the player has the area on-screen. Various other properties in the same section modify exactly when that happens.

PlayerInteractCallback

Makes the area interactable, and triggers this callback when the player interacts with it. You can also set `CustomInteractIcon` etc.

AmbientLight

The player, by default, casts a small point light around him so that you can see things even when the room is very dark. This area type changes that ambient light to fit the current area.

Important Properties

Type

Indoors / Outdoors / None

CameraAnimation

Used as a camera viewpoint in a camera animation. See Camera Animation.

Climb

This area becomes interactable; when you interact, you will climb up to a specified foot position. It's used for things like climbing into vent shafts or climbing up on to a nearby table or something; for higher climbs, you probably want a camera animation or a ladder instead.

Important Properties

FeetPosition

Name of an entity (usually a trigger area) specifying where the feet should land after the climb.

Crouch

Specify if the player should be automatically crouched after the climb (useful for vent shafts).

Crawl

The player will be crawling on their belly while in this area, and can't run or stand up. Useful for vent shafts.

Datamine

The player can interact with this area as if it were a datamine prop - see Datamine

DatamineAudioSource, DatamineAnimNode

Currently unused.

Description

Used with the Description module. Not used in SOMA.

Distortion

This area will apply a Distortion Effect to the player while they are in it.

DoorwayTrigger

This area is really useful for telling you when the player has walked into a particular part of the map. You set up a doorway with the Z-axis pointing in to the 'inside' and away from the 'outside'. Now when the player walks in through the door, the DoorwayCallback is called with an `alState` parameter of 1. When the player walks out through the door, the callback is called with an `alState` of -1.

You can also link sets of doorways, which is a great way to figure out if the player is inside an irregular space. If you specify the `DoorwayGroup` property, then you can use the function `Doorway_PlayerIsInGroup` to check if the player has crossed into the space delimited by these doorways. (Essentially, if you cross any of the doorways going 'inwards' so `alState ==1`, then you are in the group; crossing any going 'outwards' so `alState ==-1` means you are outside the group).

Important Properties

DoorwayEntity

This acts just like CC_Entities on an area - it just specifies which entity to check for collisions with. This is usually `player`.

DoorwayCallback

Called when the `DoorwayEntity` goes in or out of the doorway.

Hide

This marks of a specific area as somewhere that the player can hide i.e. they will no longer be visible

to AI. This is not used in SOMA, but can work regardless.

InteractAux

This is used to expand the interaction area of another entity such as a prop; so that you can, for example, give a small button a bigger interact area. For all purposes, interacting with the area is treated as if you're interacting with the prop itself. If you set interaction disabled on the prop, the area will deactivate too.

Important Properties

InteractParent

The entity to interact with.

Ladder

This is used to allow the player to climb ladders. Position the ladder area up the ladder prop. Note that in SOMA rungs have to be 0.5 units apart, and the ladder area itself has to be a multiple of 0.5 units tall and aligned to the rungs for the animation to work properly.

Important Properties

Exit_Top, Exit_Top_Crouching

Use these to specify an area to move to when you step off the top of the ladder (usually a Trigger area).

Exit_Bottom, Exit_Bottom_Crouching

Use these to specify an area to move to when you step off the bottom of the ladder (usually a Trigger area).

Liquid

Creates an area of liquid - useful for pools of water or for use in SOMA's airlocks. You can control waves, surface colors, fog under water etc. - properties should be reasonably self-explanatory.

MapTransfer

Map Transfer areas are used in map transitions; anything within a Map Transfer area is preserved as the next map is loaded. See Map Streaming.

PathNode

Pathnodes are points rather than areas; they are used in AI for the pathfinding system.

PlayerStart

Player Start areas are where the player will appear as the result of a map load (by default, the player appears at `PlayerStartArea_1`, although that can be changed) or as the result of a `Player_Teleport()` call.

Important Properties

Crouching

Specifies whether the player should be crouching when they appear at that spot.

PosNode

Not used in SOMA.

Rope

Not used in SOMA.

Sit

Sit areas specify an area that the player can interact with to sit down.

Soundscape

Soundscape areas are used to set up the ambient sound for particular areas. You'd generally have

one soundscape area for every room. See Soundscape Area for more details.

Sticky

Sticky areas are used to attach other entities to (normally `Prop_Grab`) - when the entities are pulled away, there'll be resistance, and if you don't pull hard enough they'll snap back. This is used to implement things like pulling power cables from sockets or attaching cables to sockets.

Important Properties

AttachableBodyName

Name or names of bodies that can be attached to this area. Accepts wildcards.

CanDetach

If true then the connected body can be detached.

RequiresInteraction

If true then bodies will only detach or attach if the player is actually interacting with them (as opposed to e.g. walking into an object to knock it free).

MoveBody, RotateBody

Whether to orient the body to the attach position.

AttachFunction

Called when an entity is attached to this area.

DetachFunction

Called when an entity is detached from this area.

SpringyActive, SpringyForceMul, SpringyDetachTime

Use these to give some resistance to the detachment or attachment - so that the attach area 'sucks in' the body.

Tool

If the player walks into a tool area and has the specified tool in their inventory, then it will automatically be equipped (i.e. brought out and shown on screen in their hands). This is what is used for showing the Omnitool next to an airlock control panel.

Important Properties

ToolsToEquip

A list of tools to bring out when in this area.

ConnectedToolAreas

Other tool areas that are part of the same group; this is to make it easy to set up irregularly shaped tool areas by combining multiple rectangular areas.

LookEntities

Specify a look entity, and the tool will only be brought out if the player is facing that entity. In most cases, it would be the object that they can interact with e.g. the airlock panel. This is used so that the tool isn't brought out if the player backs in to a tool area.

AgentRepel

This area type repels AI creatures who enter it.

Important Properties

ExitArea

Specifies a particular point (usually a TriggerArea) to head towards if the agent enters this area.

VisibilityArea

The visibility area is used to divide a map into areas that should not be visible at the same time. This helps the engine cull objects that are outside of the view and should not be visible. Adding viability areas to the map will increase performance since it will give the engine an easier time partitioning the objects. It is also possible to manually disable a visibility area, this will hide all the objects that are

inside it. An object will be part of all visibility areas that it is fully inside of.

Objects that doesn't fit inside any of the placed areas will get added to a main render container that has an infinite size.

Placement

Visibility areas should be placed around large rooms, floors or large outdoor zones. They should be placed so that when you are inside one of the areas the other areas are not visible because it is occluded by geometry (like floors or walls). If a map has multiple floors with stairways between them then it would be a good idea to place one area per floor.

A visibility area should be added to rooms that have a lot of objects. If the room is big but has few objects then placing an area is not needed. Most visibility areas should be large, covering a whole floor, but some can be small if the room has a lot of expensive objects. Only objects that are fully inside an area will be added to it.

Add Partial Intersecting Objects

If objects that are only partially intersecting the area should be added to it. The default behavior is that only objects that are fully inside the area will be added. Enabling this can lead to false positives and the objects being culled when they should be visible.

Min Objects Per Node

Small nodes will merge into larger ones if the number of objects per node is less than this value. This is based on the average and not enforced per node.

Max Objects Per Node

Large nodes will be split into smaller ones if the number of objects per node is larger than this value.

Scripting

Visibility_SetAreaActive(const tString &in asName, bool abActive)

Disable or enable an area and make all objects inside invisible.

Visibility_SetMainActive(bool abActive)

Disable or enable the main area and make all objects that are not part of a visibility area invisible. This can cause issues with large lights being disabled. Add Partial Intersecting Objects can be used with this to disable world the outside while inside a room.

Visibility_SetTerrainActive(bool abActive)

Hides terrain.

Notes

Visibility areas can be rotated to get a better fit. Rotating an area makes updating of dynamic objects a little more expensive.

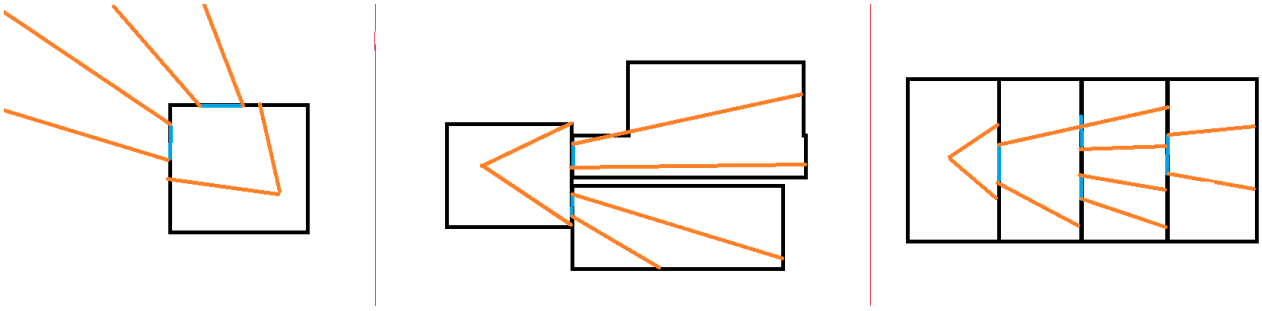
Play around with min/max objects per node to get the best result (F5 is supported). The goal is to reduce the number of draw calls and queries with draw calls being the main stat.

Placing visibility areas will never make objects on the map disapear when they should be in view.

Visibility Portal

A visibility portal is an area used to link together two or more visibility areas. A visibility area that is connected to a portal will only be visible if the portal is visible or if the camera is inside it. If the camera is inside of a visibility area then all the portals connected to it will be used as a portal to the outside.

A visibility portal will also clip the view frustum. This means that only objects that can be seen through the portal will be visible. Frustum clipping can also be used to disable other portals so that areas can be skipped directly.



The left image shows how a portal is used to cull the outside world.

The middle image show one visibility area connected to two others with the use of portals, only the objects inside of the clipped frustum will get rendered.

The last image shows that portals can be used to clip other portals. The current setup is limited so that only the first portal is used to cull other portals. If you look at the right most portal you can see that the whole portal is being used instead of clipping it with the portal before it, this might get fixed later.

Portals can also be combined with occlusion query based culling. This means that the GPU can check if a portal is blocked behind other rendered objects, and if that happens then the portal gets closed.

Portals should always be intersecting with all the visibility areas that it belongs too. If it is placed outside then the whole visibility area might get culled when it should be visible.

Important Properties

Connected Areas

List all the areas that are connected to this portal. Separate using ;

Notes

A good place to place portals are windows and doors. The portal should always be a little bigger than the window or door that it covers. If it is smaller then there might be objects that pop into view.

Portals can be used on outdoor levels as well. If there is no roof on the level then the portal should be scaled to a height that covers the whole wall.

Portals also work when retrieving shadow casters.

Be vary of how many portals and areas that you use on a map. Since one query is used per visible portal you might end up having more draw calls than you would without the portal.

Portals can be disabled in the debug menu.

Unlike visibility areas, adding portals incorrectly will make objects pop in and out when they should be visible.

Zoom

See also Readables.

A zoom area shows a magnifying glass icon when the player interacts with it; when they interact, the player's viewpoint is moved in towards the area, so that they're focused on the contents. You can also add optional text or GUI to pop up over the area.

This is used in SOMA for things like whiteboards, maps and pictures on the wall.

Important Properties

TextCategory, TextEntry

Specify the text to appear when you click interact for a second time. This is used for simple descriptive text.

OnGuiFunction

Instead of text appearing, if you specify an OnGuiFunction that will be called instead. This is useful for

drawing more complex text layouts such as the key locations on a map.

From:
https://wiki.frictionalgames.com/ - **Frictional Game Wiki**

Permanent link:
https://wiki.frictionalgames.com/hpl3/game/areas

Last update: **2020/02/11 19:16**