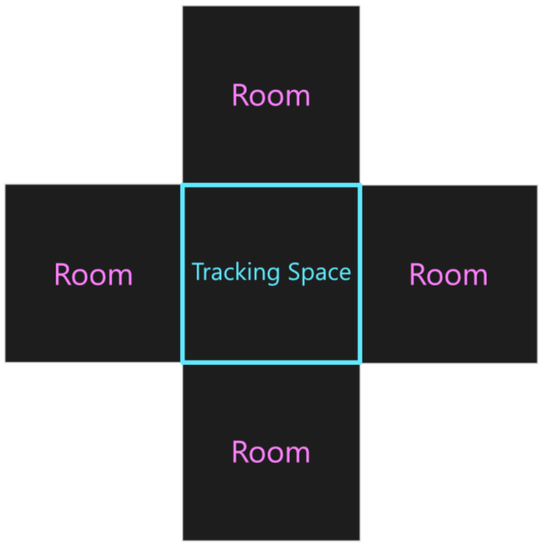
INTRODUCTION

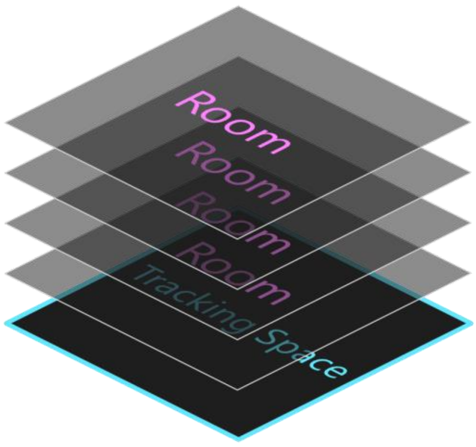
QuantumVR is a tool providing the technology and editors to easily compress virtual space. With this tool, you can create environments that contain potentially infinite amounts of space inside a finite space.

The goal of QuantumVR is to enable a new kind of room-scale VR experience, where players can explore vast environments just by using real walking.

Imagine a big, multi room virtual escape room for example. With QuantumVR, all rooms of the scenario fit inside the tracking space of the player, enabling him to walk freely through all rooms with no need for teleportation or other kinds of conservative locomotion approaches.



Classical multi room setup. The environment expands past the edges of the tracking space, which means these areas cannot be reached by real walking.



Compressed multi room setup. The complete environment is contained inside the tracking space, and thus accessible by real walking.

INSTALLATION AND DEPENDENCIES

QuantumVR can be used with Unity 2019.2 or higher with the core renderpipeline. To install it, just import the QuantumVR package and follow the instructions the tool provides during import to run the project setup.

CONCEPTS

If you want to use QuantumVR, it is important to understand a few concepts first.

With the current versions of QuantumVR and Unity it is advised that you always save your scene before entering play mode. This will trigger a scene validation for all QuantumVR objects, which is necessary because due to a bug in Unity some editor callbacks are not called consistently. If you encounter any weird behavior in play mode concerning QuantumVR, try exiting play mode and saving the scene.

SPATIAL CELLS

A Spatial Cell is a unit of normal, uncompressed space. A QuantumVR level consists of multiple Spatial Cells that overlap each other and are connected by portals; they are compressed.

A Spatial Cell is represented in the scene by a Game Object with the *Spatial Cell* component on it. Creation and deletion of Spatial Cells happens exclusively via the *Spatial Cell Map Editor*.

Everything that should be part of a Spatial Cell has to be a child of the Game Object of that Spatial Cell.

At runtime, there is always one cell that is considered the Active Cell. This is the cell the player, or more precisely the main camera, is in. If the player goes to another cell by passing through a Portal, that other cell is the new Active Cell.

SPATIAL LAYERS

A spatial layer is an abstract concept used for rendering and managing the compressed space environment at runtime. Spatial Layers can be imagined like parallel universes.

There is a maximum of 9 Spatial Layers; however less can be used by changing the settings. At runtime Spatial Cells will be assigned to them. A Spatial Layer has at max one Spatial Cell assigned to it. Which cell is assigned to which layer is dependent on which cell is the Active Cell and changes every time the player enters another cell.

Note that one cell can be assigned to multiple Spatial Layers if your scene is set up in a circular fashion where a path of Portals leads back to the original cell.

The Spatial Layers are what is rendered through portals and they define collision behavior. Each Spatial Layer owns a Unity Layer that is set up so it does not collide with any other layer. That ensures that objects on one Spatial Layer do not collide with objects on another Spatial Layer.

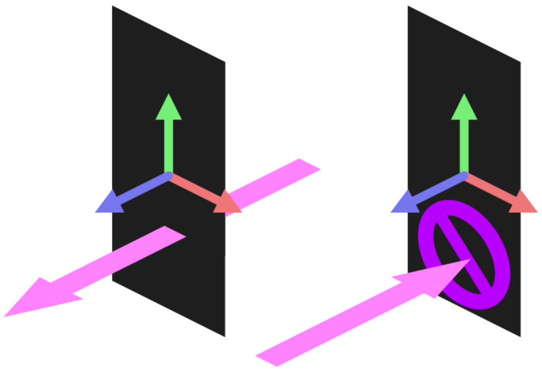
The Spatial Layers are rendered in a predetermined order.

PORTALS

Portals connect Spatial Cells with each other. One Portal between two cells consists of two Game Objects in the scene, one as part of each of the two Spatial Cells that are connected by it.

These two Game Objects are constrained together so that they always have the same position, size and orientation in world space.

Each Portal object is only visible and traversable in one direction, in its forward direction. This means it will be visible coming from its negative Z direction, but not from its positive Z direction. The two Game Objects of a Portal always have their forward vectors face 180 degrees away from each other so that one half can lead back to where the other half comes from.



Portals can be scaled in their local X and Y directions to make them the preferred dimensions. Scaling in Z direction is not advised.

If the main camera goes through a Portal, the Active Cell is changed.

SPATIAL CELL MAP

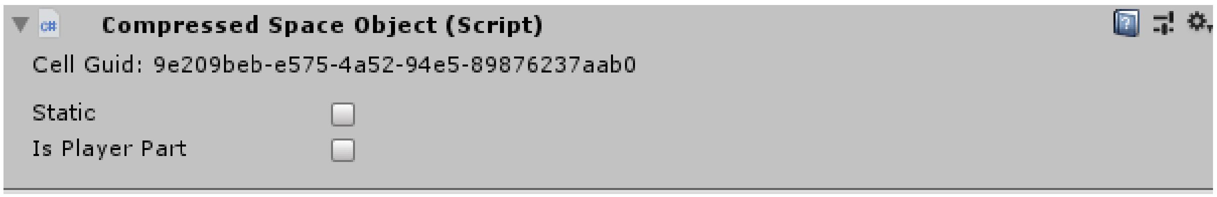
Each Scene that is set up for QuantumVR has a Spatial Cell Map asset assigned to it. The Spatial Cell Map can be edited via the *Spatial Cell Map Editor* and contains all information about which Spatial Cells exist, how they are connected to each other and which Spatial Cell will be the Start Cell, the cell that will be the Active Cell once gameplay starts (when entering play mode or loading the scene).

COMPRESSED SPACE ENTITIES

Compressed Space Entities are objects with the *Compressed Space Object* or the *Compressed Space Light* component. Every Game Object that should interact with the compressed space of the environment via rendering or physics needs to have one of those components.

The rendering and collision of Compressed Space Entities are managed by QuantumVR. Compressed Space Entities are also able to pass through Portals if they have a collider.

QuantumVR will automatically add the proper component to Game Objects that need to be a Compressed Space Entity in most cases.


Compressed Space Object (Script)
Cell Guid: 9e209beb-e575-4a52-94e5-89876237aab0
Static ☐
Is Player Part ☐

A Compressed Space Entity can be made QuantumVR static by checking the *Static* checkbox in the component Inspector. This means that it will not be tracked by Portals, making it unable to pass through Portals. It is advised to check this if you know an object never has to pass through a Portal, as leaving it unchecked might have a significant impact on performance in some cases. If you make an object Unity static it will be made QuantumVR static automatically.

Compressed Space Entities always belong to a Spatial Cell and have to be a child of that cell, with one exception: The Compressed Space Entity can be defined to be a part of the Player, for example a hand that is controlled by a tracked VR controller. In that case, *Is Player Part* needs to be checked and the Compressed Space Entity can live outside the Spatial Cell hierarchy because it does not belong to one cell anymore.

Note that only objects that are attached to the Player from the start should have *Is Player Part* checked. Objects that the player can pick up and carry around for example likely belong to a Spatial Cell before they are picked up and need to be set as a player part via code only for the duration the player carries them.

QUANTUMVR RENDERING BASICS

This paragraph is not important if you are not a technical person and not involved with shaders and rendering in your project.

The portal rendering of QuantumVR is achieved via the stencil buffer. The *Number of rendered Portals* setting directly corresponds to the number of stencil buffer bits used by QuantumVR (counting starts from the least significant bit).

All QuantumVR rendering happens inside the *Render Queue Range* specified in the Settings. If you want to use the stencil buffer for custom rendering before that, make sure to clear it before QuantumVR rendering happens. After that range you can use it as you please, but note that it will have a bunch of values in it.

There are two stages to QuantumVR rendering:

First all Portals will render the stencil values of the Spatial Layer they should render to the stencil buffer. Some depth information will be written during this stage, but cleared before the next stage.

In the second stage the Spatial Layers will be rendered. They will be rendered from farthest from the camera to closest to the camera. The distance in this case however is not physical distance, but is defined by how many Portals are between the camera and the Spatial Layer. Each Spatial Layer has a render queue range to which all queues of the materials of the objects in that layer will be remapped to. This means that in one cell the relative render queues are kept intact.

After all Spatial Layers that have at least one Portal between them and the camera are rendered, the Active Spatial Cell is rendered via normal Unity rendering, with the queue values that are set on the materials.

PROJECT SETUP

A Unity project using QuantumVR needs some setup. QuantumVR checks whether it needs to run the setup by checking whether the QuantumVR settings file in the StreamingAssets folder exists. If it doesn't find it, it knows that the setup has to be run.

When you import the Package, it will ask you automatically if you want to run the setup. If you choose *No*, it will ask you again once you open any QuantumVR editor window.

The automatic setup will generate the QuantumVR settings file and the Unity layers needed for the Spatial Layers along with their collision matrix. It will not overwrite existing layers; only layers that have no name set yet are used (Layer 31 is not used because it is used by Unity internally). If your project does not have enough unused layers, the setup will inform you how many layers it could use and will reduce the *Number of rendered Portals* accordingly.

SCENE SETUP

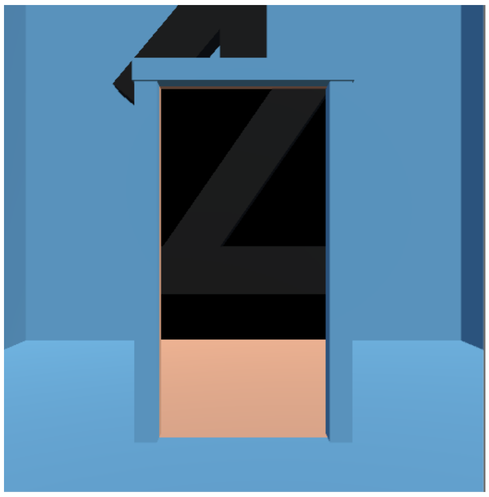
Every scene that you want to use with QuantumVR needs to be set up correctly. QuantumVR can do this automatically for you. You can run the scene setup for the currently opened scene either by clicking *Tools/QuantumVR/Set up Scene* or by clicking the *Run Scene Setup* button in the Spatial Cell Map Editor.

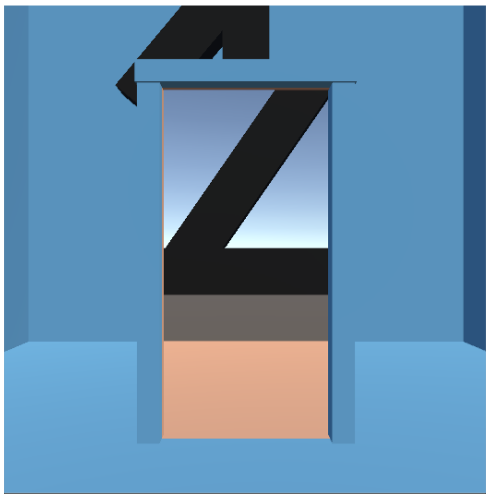
The scene setup creates a Spatial Cell Manager along with a Spatial Cell Map asset for the Scene. Where this asset is created is specified in the settings.

It furthermore converts the main camera of the scene to a QuantumVR camera (if there is no camera tagged as main camera it will search for another camera and make it the main camera, if there is no other camera it will create one).

The setup will ask you whether you want to create a skybox camera. If you know that your skybox will be visible to the player, you should click *Yes*. Without it, the skybox will not be visible through Portals, instead areas where the skybox should be visible through a Portal will appear black.

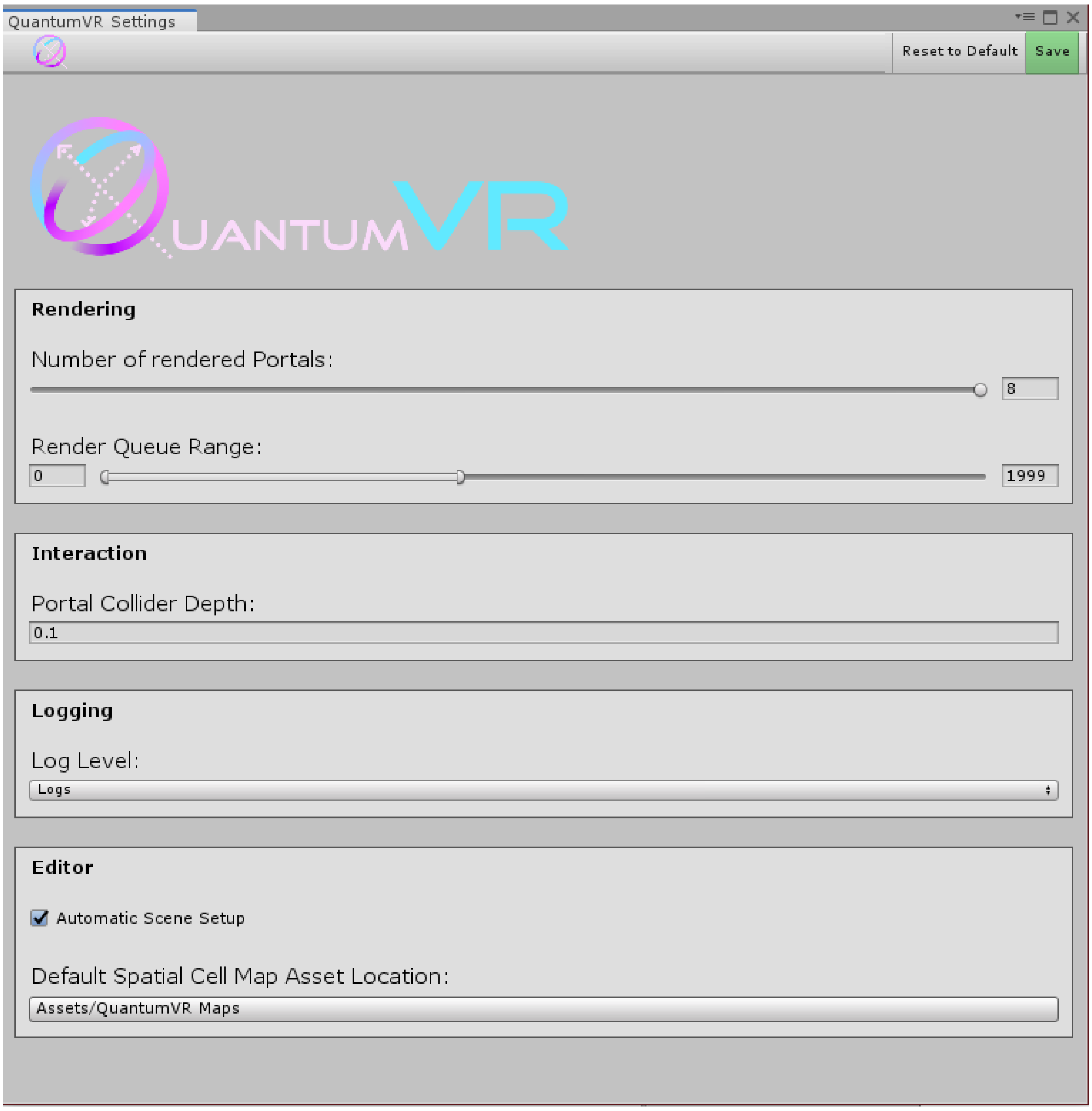
The skybox camera will render before the main camera and culls out everything but the skybox. The main camera then draws over the rendered skybox. Because sky areas behind portals are actually rendered as fully transparent pixels, the skybox image will be visible at those areas.


No Skybox Camera


Skybox Camera enabled

Depending on how your main camera is controlled, you may need to manually make sure your skybox camera always has the same orientation of your main camera (you can make it a child of the main camera). OpenVR for example will track more than one camera automatically, so there is no need to manually handle the skybox camera.

SETTINGS



Rendering

Number of rendered Portals: 8

Render Queue Range: 0 – 1999

Interaction

Portal Collider Depth: 0.1

Logging

Log Level: Logs

Editor

☑ Automatic Scene Setup

Default Spatial Cell Map Asset Location: Assets/QuantumVR Maps

RENDERING

Name	Effect
Number of rendered Portals	The amount of Portals that will be rendered. This value+1 is the amount of Spatial Layers rendered (the Active Cell is always rendered and not rendered through a Portal). Due to the nature of the stencil buffer, 8 is the maximum value. If you have performance issues, try to reduce this value.
Render Queue Range	The part of the render queue that will be used for QuantumVR rendering. Usually normal rendering starts at 2000, so the default QuantumVR range is between 0 and 1999. The bigger this range, the less likely it is you encounter artefacts, but if you need queue values before 2000 for normal rendering you might need to change this.

INTERACTION

Name	Effect
Portal Collider Depth	Portals use colliders to determine whether an object is passing through them. If you find that fast moving objects do not go through portals properly, try to increase this value, if your environment is really small you might have to reduce this value.

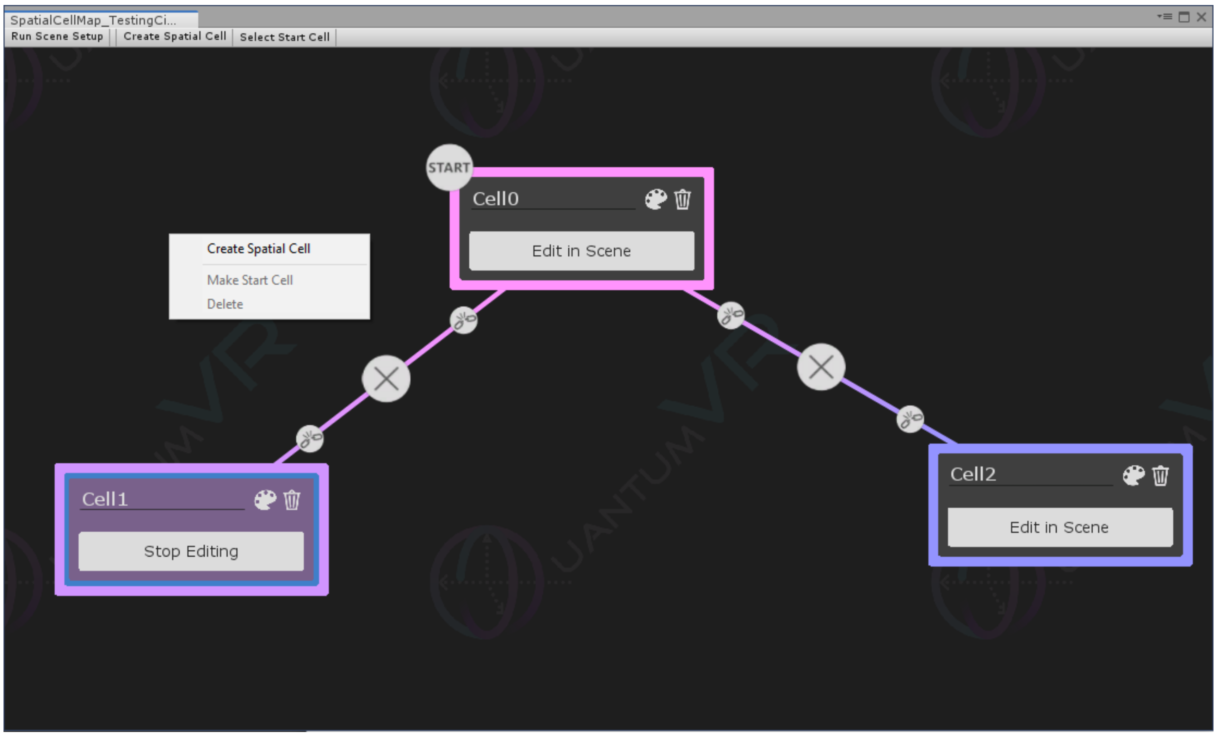
LOGGING

Name	Effect
Log Level	Allows you to customize the information QuantumVR will log to the Console. Verbose – Logs a lot of information that you will most likely never need. Could be helpful for debugging. Logs – The default log level logs warnings, errors, exceptions and some other useful information. Warnings – Only logs warnings, errors and exceptions. Errors – Only logs errors and exceptions Fatal – Only logs exceptions.

EDITOR

Name	Effect
Automatic Scene Setup	If this box is checked, QuantumVR will detect scenes that have QuantumVR functionality and components but are not set up correctly when they are opened and run the setup automatically.
Default Spatial Cell Map Asset Location	The folder in which Spatial Cell Map assets for scenes will be created. You can move the assets anywhere once they are created, but this is where QuantumVR will put them when creating them.

SPATIAL CELL MAP EDITOR



The Spatial Cell Map Editor lets you edit the Spatial Cell Map by creating and deleting Spatial Cells and Portals as well as determining the Start Cell.

The *Edit in Scene* button will isolate the Spatial Cell in the Scene View using Unity's Scene Visibility feature. That means no Game Objects are disabled and the isolation affects only the Scene View to make it easier to work in overlapping Spatial Cells. Furthermore all Portals leading out of the cell will be highlighted by the color of the Spatial Cell node they lead to.

The small buttons at the ends of connections let you retarget a Portal. By doing this, the position, scale and orientation of the original Portal will be retained while it would be lost if you deleted the connection completely and made a new one.

KEYBINDS

Key	Action
Escape	- Exit Start Cell selection mode - Stop connection creation
Delete	- Delete selected Spatial Cells - Exit Start Cell selection mode - Stop connection creation
Backspace	- Delete selected Spatial Cells - Exit Start Cell selection mode - Stop connection creation

SHADERS

The QuantumVR Portal rendering requires all objects taking part in it to have a compatible shader. At the moment, there are 3 QuantumVR shaders.

STANDARD

The QuantumVR Standard shader has all functionality of the Unity Standard shader with the added QuantumVR functionality. If QuantumVR detects a Compressed Space Entity that does not have a QuantumVR shader, it will ask you whether it should replace the shader of the material. This is the shader it will set if you confirm.

UNLIT

A simple unlit texture shader that works with QuantumVR rendering.

UNLIT TRANSPARENT

A simple transparent unlit texture shader that works with QuantumVR rendering.

WRITING CUSTOM QUANTUMVR SHADERS

If you want to write your own QuantumVR compatible shaders, you can access all functionality from the .cginc files included in the package.

Every QuantumVR shader needs these two properties that are set by QuantumVR at runtime:

```
// portal rendering
[HideInInspector] _MaskingBit("Masking Bit", Range(0, 128)) = 0
[HideInInspector] _DoPortalClipping("Do Portal Clipping", Float) = 0
```

To make the shader render through portals properly, it needs this simple stencil operation:

```
Stencil
{
    Ref 255
    ReadMask [_MaskingBit]
    Comp equal
}
```

Finally the shader needs to be clipped at portal surfaces properly.

Include *Assets/FK/QuantumVR/Shaders/Includes/QuantumVRObjects.cginc* in your shader to get access to the functionality.

The clipping happens in the fragment or surface function of your pass. You need to know the world position of the fragment in order to perform this operation.

In the fragment shader, call *performPortalClipping* and pass it your fragment world position. This function may discard the current fragment, so it can be useful to make this the first thing that happens in the fragment shader to prevent unnecessary calculations.