Table of Contents

Labyrinth/Portal Demo Walkthrough/Instructions

Controls

Movement:

W :- Forwards

A :- Left

S :- Backwards

D :- Right

Mouse :- Look

Menu

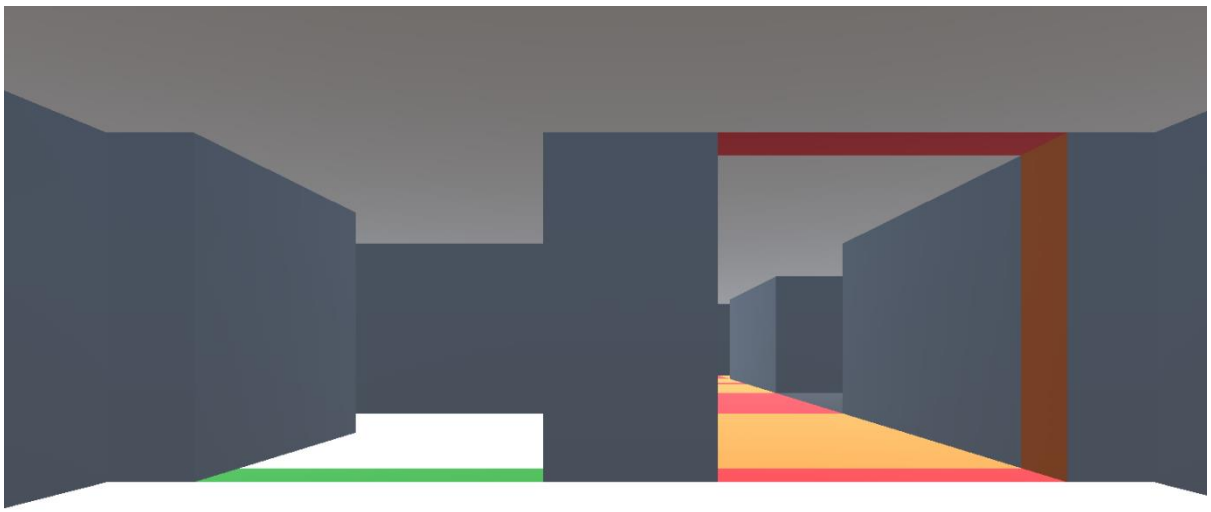
Escape :- Pause Game/Open Menu & Resume Game/Close Menu

Mouse :- move cursor

Mouse 1 :- Select Option

Start Room

Upon application start-up a player will find themselves in a room with two portals. The left portal leads to the Labyrinth portion of the application and is marked with green, whilst the right portal leads to two demo rooms and is marked with orange and red.

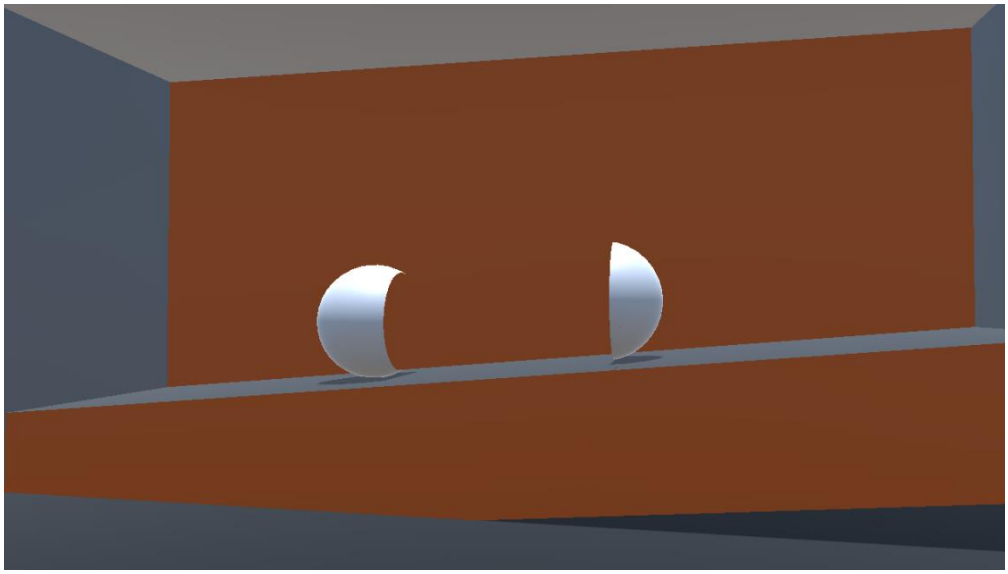


Demo Rooms

Slicing Demo

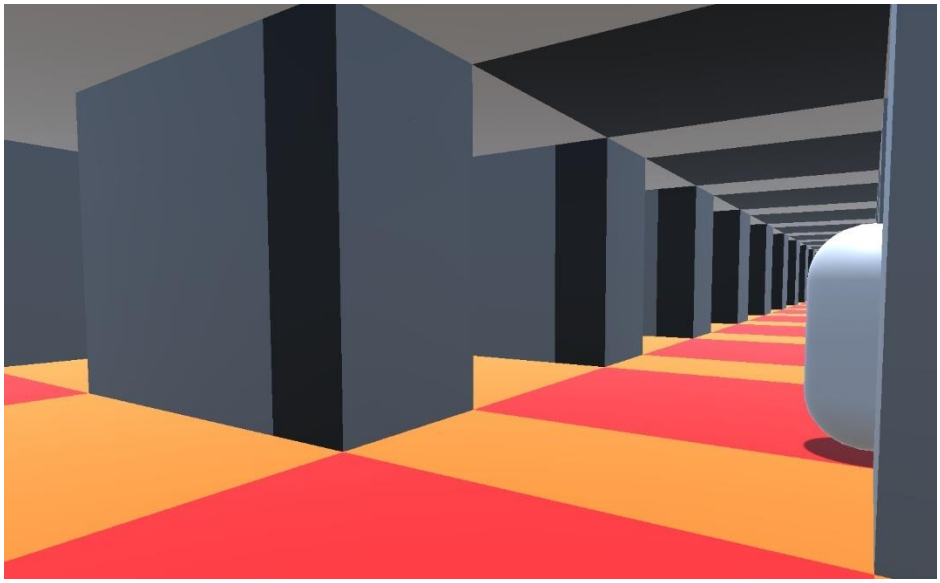
When entering the first room in the demo path a player may notice a ramp with a ball on their right. What may not be entirely clear is the two portals on the ramp and the invisible wall which stops the player from reaching the ramp. Approach the ramp to reset the ball and see it roll down through the portals. The player should see the balls “slice”/vanish as they roll through into one portal and out of

the other. Note that the “slice” is very slightly offset from the portals surface such that a small sliver of the model could be seen clipping through the back of the portal (this is very difficult to see as the offset is so small and the portal itself is invisible, but it is there). This is intentional as without this small offset the shadow of the object/clone would cut off just before the portal and ruin the illusion due to how unity’s shadows work. As the player should never see the rear of a portal unless they have just walked through it (such as with the Directional Walk-through portal) in which case it should be impossible to see the offset.



Recursion Demo

Moving past the ramp and into the next demo room the player will find three portals in a “junction”. Two of these portals form an infinite corridor with each other whilst the third links back to the portal through which you entered the room (in the demo corridor). This room exists to show off portal recursion through multiple portals. Whilst this effect can be seen in the labyrinth section it is intentionally limited as the recursion process can become very computationally expensive as more portals become involved. Note that the portal marked with the red and black stripes can be seen through the infinite corridor portals, or rather that it can’t be seen. This demonstrates that recursion works with more portals than the two portals which are linked together. This is not possible with a more traditional portal recursion technique. To exit the room walk around the L bend nor marked by the red and black stripes.

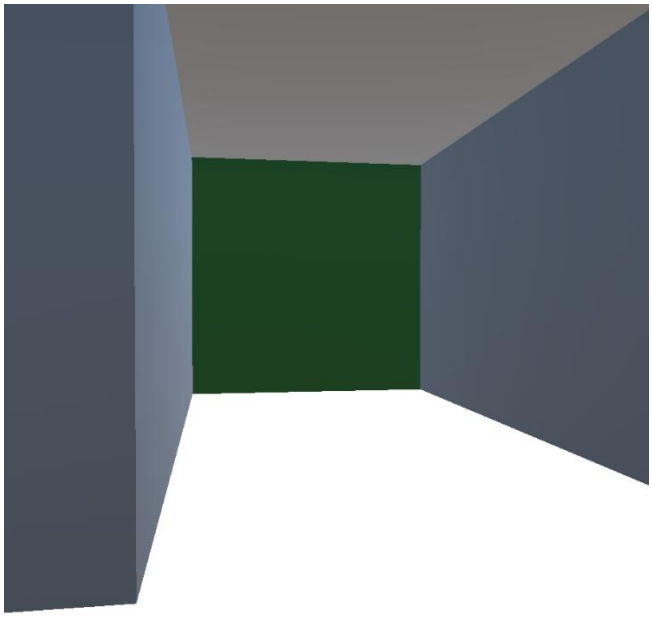


Labyrinth

Returning to the start room either by walking back through the demo corridor or by hitting restart in the pause menu, take the left corridor marked with the green stripe to enter the labyrinth. Note that green stripes etc will denote the way to progress, though sometimes how to progress may not be entirely clear.

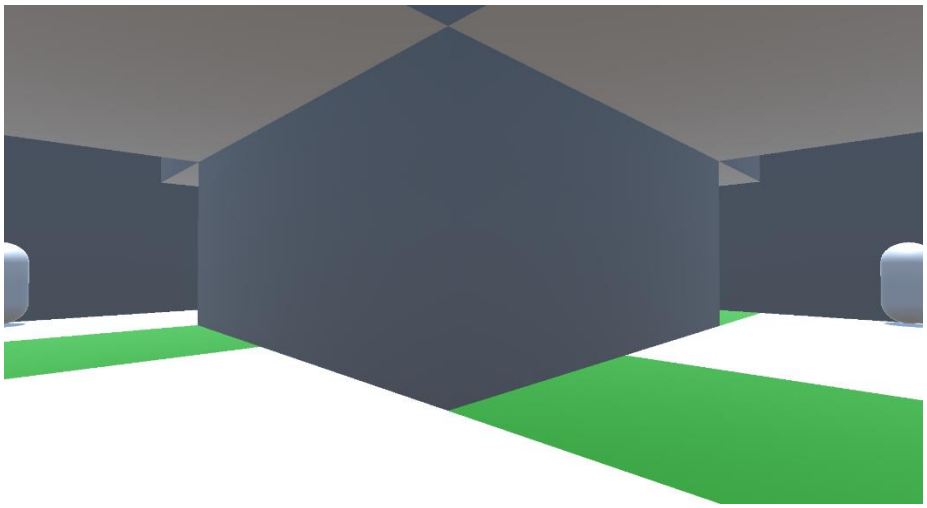
Switch Corner

Immediately upon entering the Labyrinth, the player will be met with a left turn to a dead-end with a green wall. Before walking into this dead-end the user can return to the start room, however once they walk far enough towards the green wall this will become impossible without using the pause menu or completing the labyrinth as when turning around and heading back the way they came the player will find a drop instead of the start room.



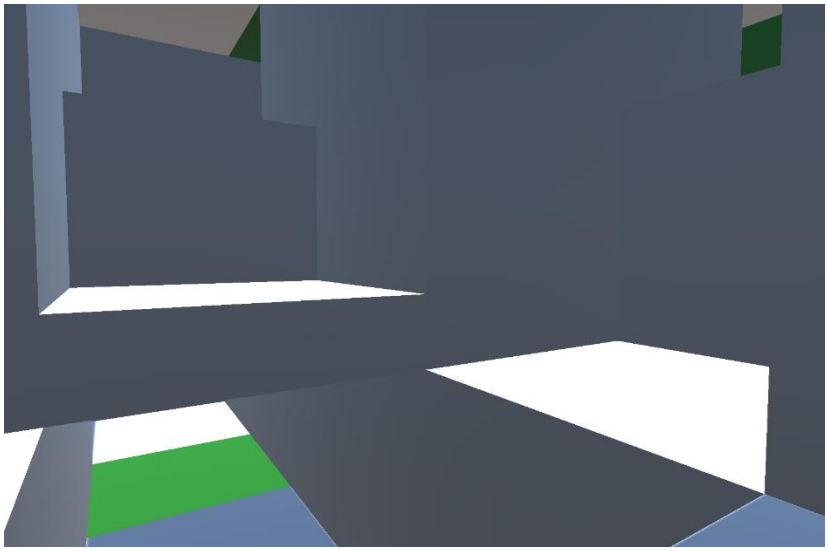
Reverse Infinite Corner

When the player drops down, they will find themselves in the corner of an L bend. The player will see themself at the end of both corridors available to them. Note the green floor next to where the player dropped down. Running along either corridor, the player will find they are now trapped in an infinite loop. To escape the loop and progress, the player must walk backwards along the corridor marked with the green floor. Upon doing so the player will find they are in a new corner with a green floor. The path leading away from the infinite corner is again marked with a green floor and the player will once more find that they are facing a drop.



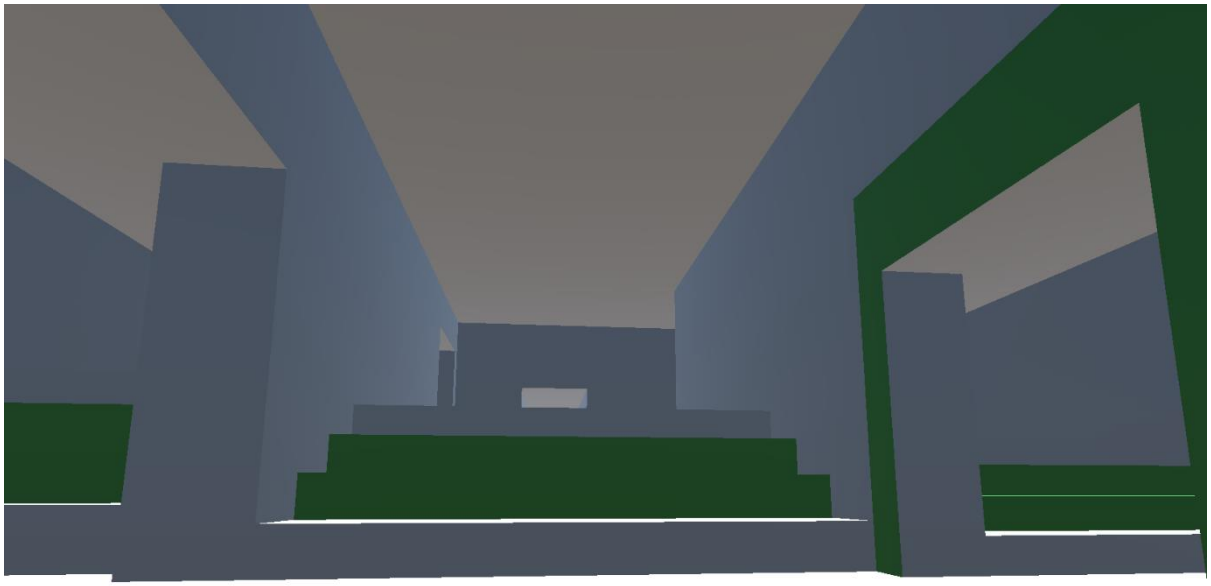
Directional Switch Room

Upon dropping down the player will see that they are in a room with four raised passageways which are currently inaccessible to them. Additionally, it they are not already in the hole the player will see a hole leading to a lower passageway marked with a green floor. This lower passageway will take the player into one of the four raised passageways, this one marked with a green floor. For now, this is another infinite loop. In order to escape this loop the player must walk backwards through the lower passageway. Upon doing so the player will find themselves in another of the raised passageways, this one marked with a green wall on their left. Backing through the lower passageway again will take the player to yet another of the raised passageways, this one marked with a green ceiling. Backing through the lower passageway once more they will find they are in the final raised passageway, this one marked with a green wall on their right. Finally, backing through the passageway one last time the player will find themselves in a blank corridor.



Scaling Room

The blank corridor ends in a turn into a large room. The room features a large set of stairs with green steps as well as two large exits of different sizes, one marked in green, which form an infinite corridor. The steps are too large for the player to climb for now. Walking through the green marked exit the player will find themselves exiting the unmarked exit, larger than they were before. If the player keeps walking through the green marked exit, they will eventually become large enough to climb the green steps. Up the steps the player will find another infinite corridor with different sized exits, one marked with a green floor, and a smaller corridor leading to a green marked corner. Entering the green marked exit until they are small enough to enter the passage, the player will turn the green marked corner and find another drop. This drop will return the player to the start room.



How Things Work

Tutorials Used

When looking to build the basic portal framework credit must be given to “Ignis Incendio” and their tutorial “Multiple Recursive Portals and AI in Unity”. This tutorial was used to learn how to build much of the basic framework including portal rendering, most of the basic portal teleportation function, portal recursion, and basic portal cloning, as well as a handful of optimisations such as the portal occlusion volumes. The tutorial’s framework has been expanded, modified, and improved in various ways such as adding functionality for changing velocity through portals, as the tutorial’s framework was built only for players to move through portals and did nothing to change or preserve velocity. Additionally, various parts of the code such as the teleportation has been reformatted and some parts of the code needed updating to the version of unity being used. Credit must also be given to Sebastian Lague and his video “Coding Adventure: Portals” which was used to learn object slicing. In particular, the slicing shader was especially useful.

https://github.com/limdingwen/Portal-Tutorial-Repository

https://github.com/SebLague/Portals/tree/master

Teleportation

Each Portal consists of four main components: the normalVisible plane, the normalInvisible plane, the portal script, and a collider. Each portal stores an array of GameObjects called “objectsInPortal”. In addition to the portal itself, each object which can move through a portal must have a collider and a script called “PortableObject”. Whenever the portal’s collider is entered by another collider it checks if that collider has a PortableObject component and if it does it adds it to objectsInPortal. Once an object is in objectsInPortal the portal checks each frame to see if the object has crossed its threshold, using the dot product between the portal’s normalVisible and the direction vector from the portal to the object’s current position. If this is zero or negative, then the object is teleported. The teleportation itself is a simple matter of getting the local position, rotation, and velocity of the object relative to the entrance portal and then translating those values to world space relative to the exit portal.

Portal Camera View

To make the portal look like continuous space, the entrance portal’s render texture must show what the player would be seeing from their position relative to the entrance portal, relative to the exit portal. To do this a camera is used which, similar to how teleportation works, takes the local space of the main camera relative to the entrance portal and translates it to world space. The portal creates a render texture of what it can see from this position and takes the screen coordinates of the edges of the portal. The shader discards any pixel not within the screen coordinates it is given and then renders what remains as a screen-space texture. This means that regardless of the geometry of the object being rendered, the texture will look the same. This render texture is then displayed on the entrance portal, making the portal look like a continuous space. An additional important step in this process is the calculation of an oblique projection matrix which aligns its near-clip plane with the plane of the exit portal. This oblique projection matrix is applied to the portal camera before rendering to ensure that nothing intercepts the portal camera’s view of the exit portal. This rendering method is expanded for recursive rendering.

Portal Recursion

Portal recursion uses the same basic steps described in the Portal Camera View section except with several added layers of complexity. To do portal recursion each portal stores a list of all portals which could be seen through the portal. Additionally, each portal stores a maxRecursionsOverride variable so that different portals can have different depths. Another script called "PortalRenderer" was created to handle all portal rendering and recursion. This script stores a maxRecursions variable which determines the default maximum number of recursions which will be performed. Just before the rendering step, this script calls every relevant portal's RenderViewThroughRecursive function and passes in several variables, notably: the main camera's position and rotation, the portal camera (of which there is only one in the whole scene), and the default max recursions. Each portal calculates the appropriate portal camera position, rotation, and oblique projection matrix and then checks if it overrides the max recursions value. If the max recursion value or its override has not yet been reached, the portal then loops through every portal in its stored list of visible portals and calls their RenderViewThroughRecursive function, passing in the calculated position and rotation for the portal camera appropriate to the current portal, the portal camera, and the max recursions value. By passing in the calculated position and rotation of the portal camera rather than passing in the same values for the main camera, the correct portal perspective for portals past the first portal is maintained. If the max recursion value has been reached, then all visible portals are set to show as a default texture. In both cases, the textures etc are added to a list of portal resources stored as structs. Finally, the portal creates and renders its own resources before returning the render texture and original texture. The render textures created are stored in a pool of render textures and are released immediately after the render step.

Cloning

Cloning is somewhat of a messy process as it is currently implemented, but it follows some relatively simple basics. The PortableObjectClone script handles the logic for all clones of a single object. It stores a list of all clones of that object and when the script is loaded it subscribes to several events from that object. Most of what the script itself does is tell the clones to update, activate, and deactivate at certain times alongside switching touching portals upon teleport. The script also detects which portal is closest to the PortableObject and only clones through that portal. The clones are typically disabled but are enabled whenever their linked PortableObject is within a portal. The script CloningBounds is used to detect which portals are close to the PortableOject using a sphere trigger collider so that the PortableObjectClone script doesn't need to check every portal in the scene to detect which is closest. Finally, whenever the clone is active, every fixed update it is moved to the appropriate spot relative to the portal destination portal.

Optimisations

The basic framework includes various optimisations. The main two to note concern the portal rendering steps. The first optimisation sees portals placed into occlusion volumes. Before the PortalRenderer script starts rendering any portals it finds the occlusion volume which the main camera is currently in. The script then only renders the portals contained within that occlusion volume. The second optimisation is a check on whether a portal is currently visible to the camera before it renders, i.e. the portal lies within the camera's view frustrum. An optimisation that isn't in the code is the layout of rooms in the scene as rooms linked only by portals are separated out far enough to never fall within the camera's frustrum whilst the camera is in another room, thus ensuring only portals in the same room as the camera are being rendered.

Scaling

To scale the portal views between two portals which are of different sizes the portal camera's position is scaled to be closer or further away from the portal based on relative portal size. This forces the camera's perspective to match the appropriate scale, assuming the portals are proportional. This is performed within the RenderViewThroughRecursive function. The PortableObject's size is scaled in warp to be proportional to the destination portal. To keep the PortableObject proportional to its original scale its x, y, and z are all scaled by the y scale value. For clones, every time they update, they calculate and apply the appropriate scale. Additionally, to appropriately match the scaled clone with the portal's view the clone's position is scaled in a similar manner to the camera.

Slicing

Object slicing is handled by a custom shader. The shader takes in a position and a direction. It then calculates the vector from the position to the pixel and uses the dot product of the direction and the calculated vector to determine whether it renders the pixel. Additionally, the shader takes in an offset value and adjusts the position using it before the dot product is calculated. This offset value is necessary due to unity's lighting system. Without this offset being applied, the shadow of the sliced object will have a clear divide as it enters the portal. As the implemented portals are not two-way and the player should never see their back (except for Directional Walk-through portals, where this offset will not be visible) this solution is acceptable. The slice parameters are updated for the PortableObjects every time CheckForPortalCrossing is called and then again just after the PortableObject has been teleported. Additionally, whenever the PortableObjects exit the portal's bounds the slice parameters are reset to disable the slicing. The clones update slice parameters every time they update.

Directional Walk-through Portal

Directional Walk-through Portals are actually just normal portals with non-standard parameters as the code for Directional Walk-through Portals was implemented into all portals. When it is detected that a PortableObject needs to be teleported the dot product between the direction the PortableObject is facing and the NormalVisible of the portal is calculated. If the PortableObject is not a player then it is teleported normally. If the PortableObject is a player and the dot product is within the min2DAngle and max2DAngle then the player is teleported. If the dot product is not within the range then the PortableObject is removed from the objectsInPortal list and is allowed to pass through.

Portal Switch Trigger

Portal switching was implemented into all portals as the ChangeTargetPortal function which takes in a Portal and then changes the targetPortal variable to the passed-in portal. The trigger itself is a simple script which stores the portal-to-be-switched and the portal-to-switch-to and then calls ChangeTargetPortal on the switchPortal, passing in the newTargetPortal.

Directional Switch Portal

Directional Switch Portals are a child class of Portal. They store a list of destination portals and a counter variable. CheckForPortalCrossing is overridden such that when the player is facing the appropriate direction, the destination portal is switched to the next portal in destination portals before the player is teleported. Additionally, unlike with standard portals the player is still teleported even if they are not facing a direction outside of the set range, the portal just doesn't change. The

Directional Switch Portal is compatible with Portal Switch Triggers as ChangeTargetPortal is overridden to update the Directional Switch Portal's count to match with the new target portal if the new target portal is included in the list of destination portals.

Evaluation

Strengths

The produced portal framework supports smooth portal teleportation and slicing which means that the portals can reliably be used discreetly for puzzles etc and will not break the player's immersion in their environment. Additionally, both the Portal Switch Trigger and the Directional Switch Portal allow for a wide range of possibilities, such as creating a sort of "portal nexus/hub" room with only one portal which is used to enter the room and is then switched to the desired destination, or closing a passageway behind a player without them noticing. The optimisations included allow for many portals to be included in every level. Portal recursion is possible through multiple sets of portals and portals of different sizes. Slicing Should allow for large and/or oddly shaped objects to move through portals without clipping through walls etc.

Weaknesses

When being teleported through scaled portals there is a slight 'jump' in visuals which is not always very noticeable but happens consistently. Some sections of the code could be combined and/or reformatted to make it easier to understand, set-up, and use with the main offender being the cloning scripts. There are currently around 5 scripts handling cloning, but with some reformatting and adjustment this number could probably be lowered to one, two, or three. Additionally, rather than needing to set-up the clones beforehand, it should be possible to create the necessary clone objects for each PortableObject on startup. Currently the portal cloning and slicing only works with one portal at time. Whilst this is not an issue for the demo and currently serves as an optimisation measure, this is something that should be changed in future. Infinite corridors currently struggle with the recursion limit as there is currently no way to stop the deepest portal from being visible. This means that players will often see the "ends" of the "infinite" corridor, which is undesirable.

Critical Evaluation

When comparing the finished product to the aims, it would not be hard to state that all had been met. Comparing the work to that of both Ignis Incendio (Lim Ding Wen) and Sebastian Lague, there are improvements which can be noted over the work of both. For example, Ignis Incendio's portal framework did not translate a PortableObject's velocity upon teleporting which has since been implemented. Ignis' framework also did not account for portals of different sizes being linked, which has since been implemented. Lague's work did not account for more than two portals, or portal size, nor did it include optimisations such as occlusion volumes. Overall I would say that the project's aims were met, however further work could be done to refine the portals and general code of the project such as reworking cloning.