DESENVOLUPAMENT DE JOCS 3D

Portal Game

“Portal consists primarily of a series of puzzles that must be solved by teleporting the player's character and simple objects using "the Aperture Science Handheld Portal Device", a device that can create inter-spatial portals between two flat planes. The player-character, Chell, is challenged and taunted by an artificial intelligence named GLaDOS (Genetic Lifeform and Disk Operating System) to complete each puzzle in the Aperture Science Enrichment Center using the portal gun with the promise of receiving cake when all the puzzles are completed. The game's unique physics allows kinetic energy to be retained through portals, requiring creative use of portals to maneuver through the test chambers.”

Wikipedia

Ejemplo Práctica 2 - https://youtu.be/MRsd0ptJxV8

Puzzle design

- Portal Gun (companion, resizing, paintable)
- Gravity Gun (companion, turrets)
- Companion cube
- Buttons & Doors
- Laser Turrets

GDC2016 Level Design Workshop: Solving Puzzle Design *[opcional]*

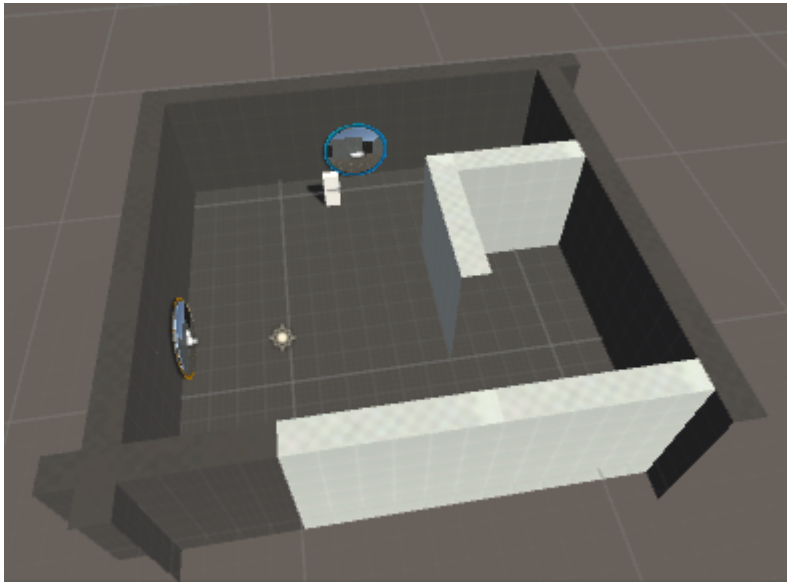
Creando nivel - prototyping

Recursos para el prototipaje

Prototyping Pack

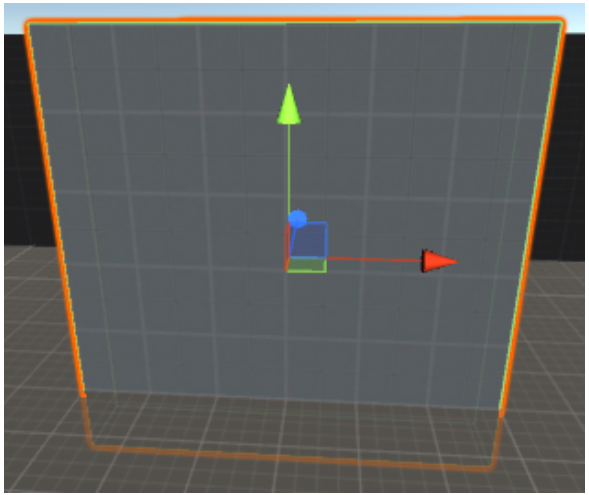
Herramientas para prototipaje

Unity Probuilder

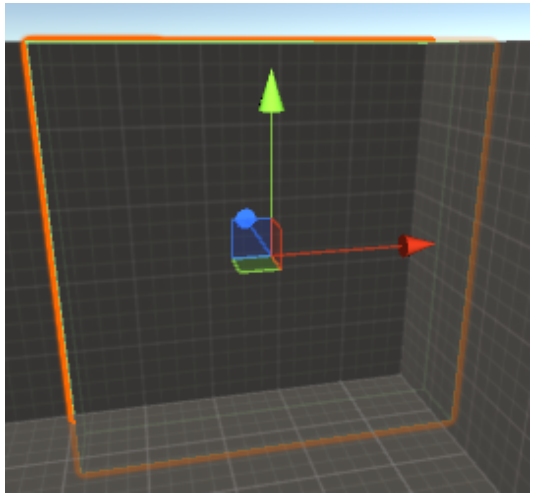


Creando nivel

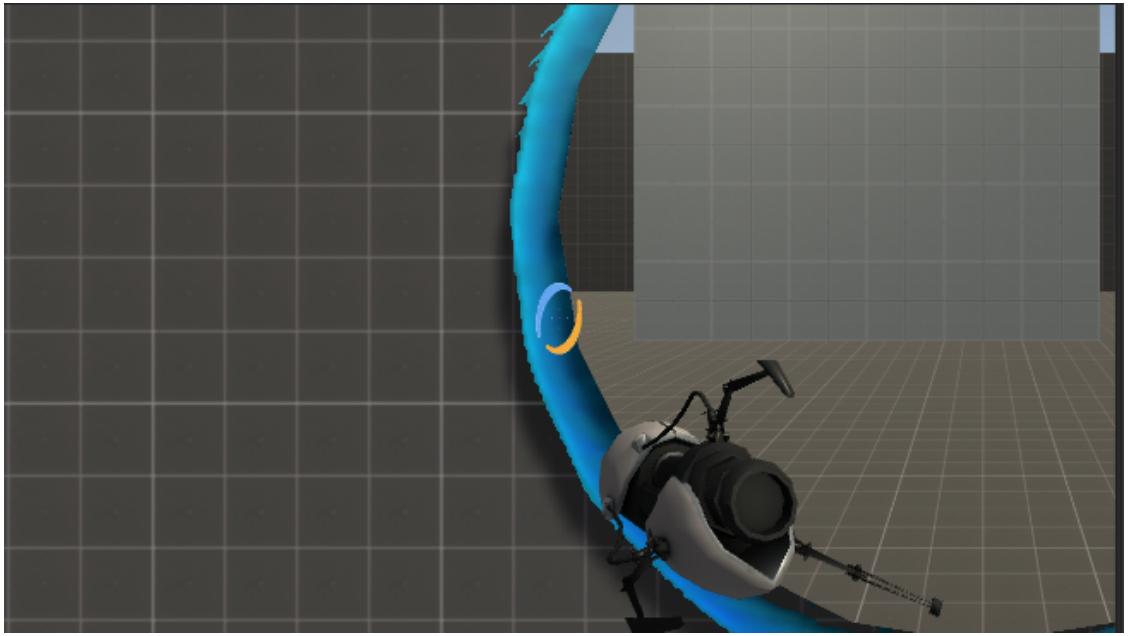
Pared pintable



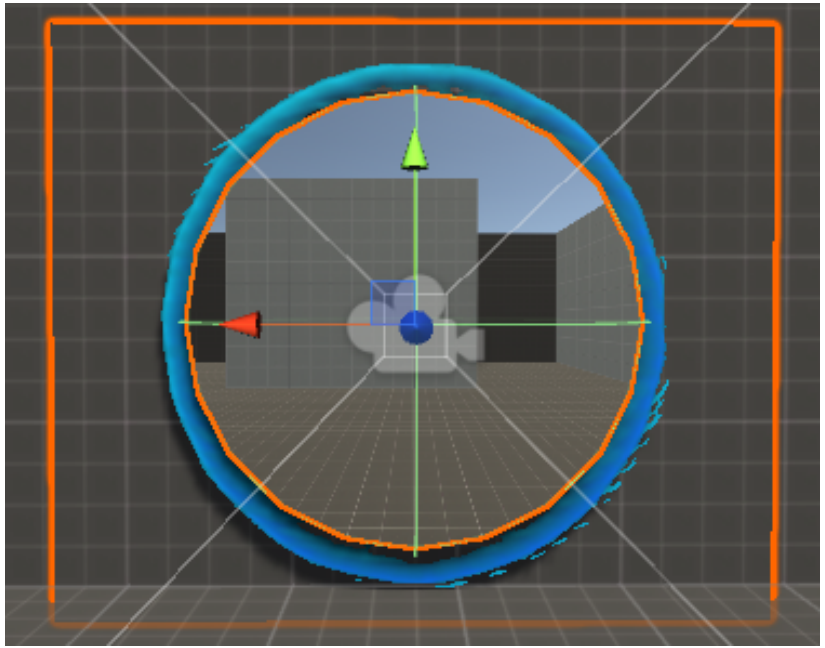
Pared no pintable

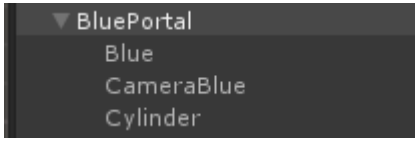


FPSPlayerController

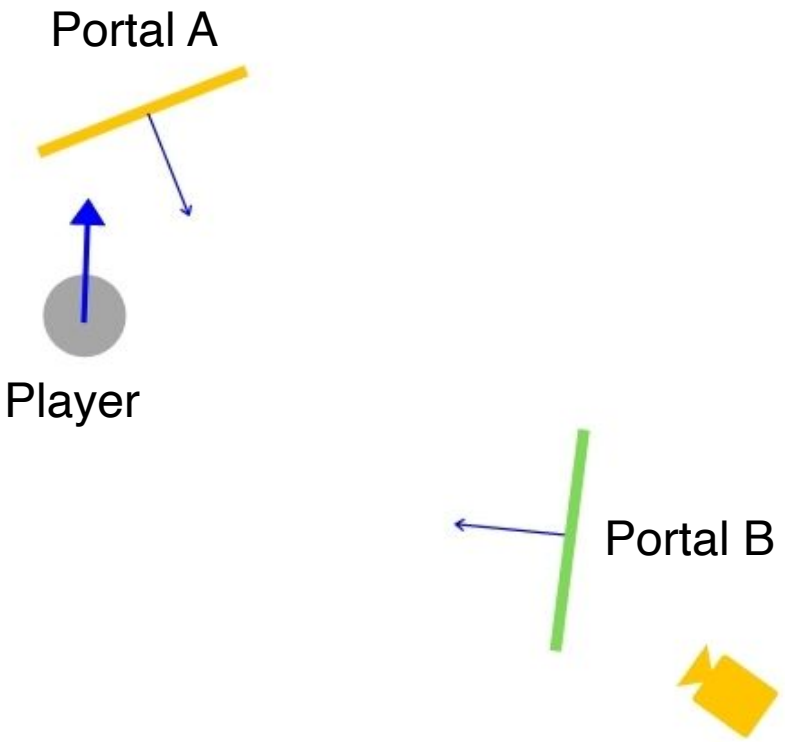


Portales

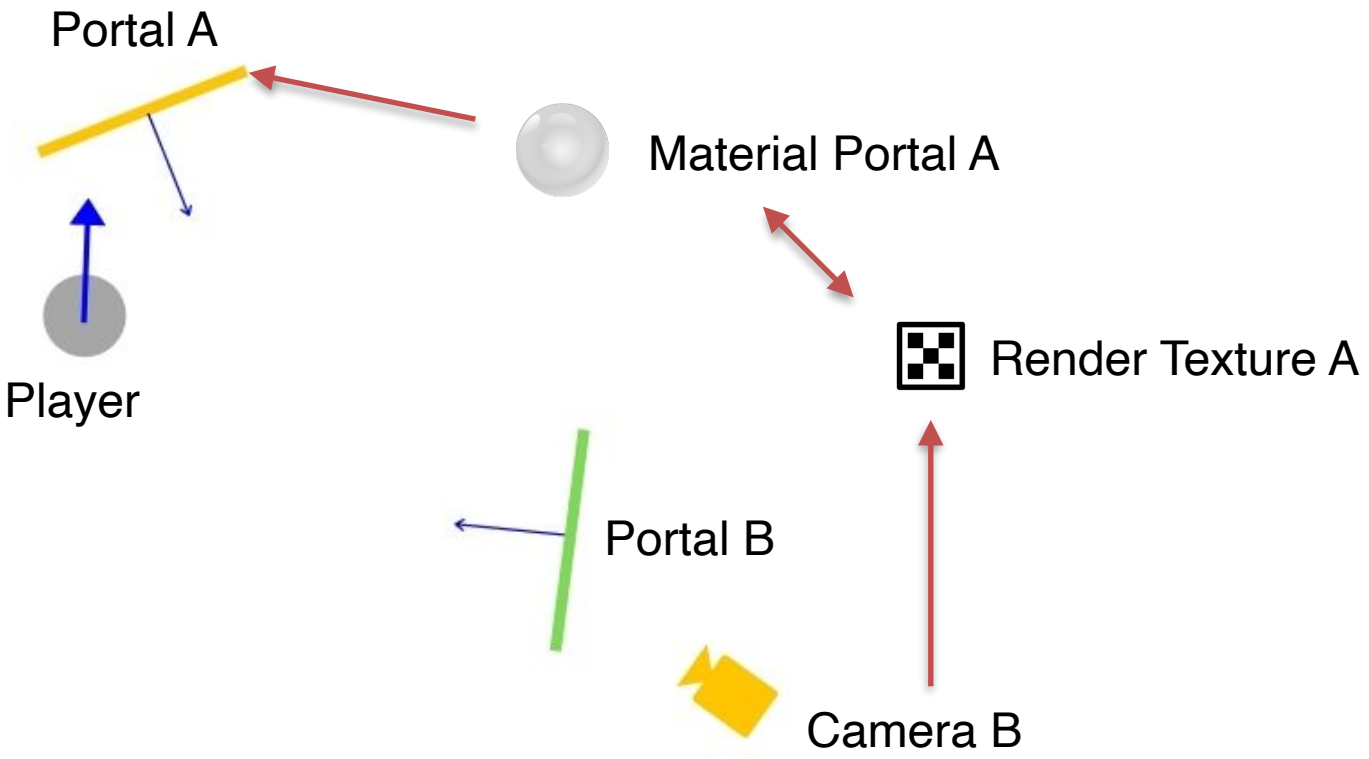




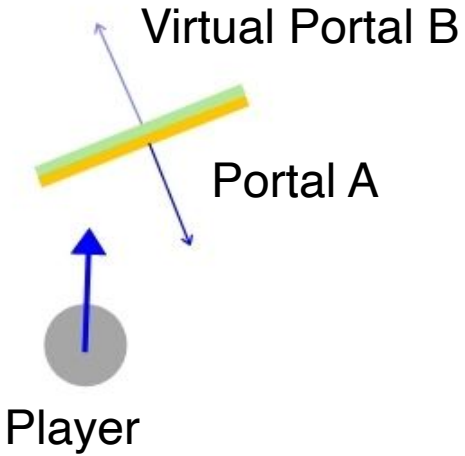
Portales

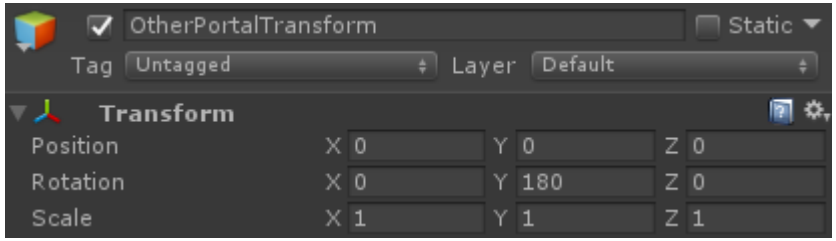


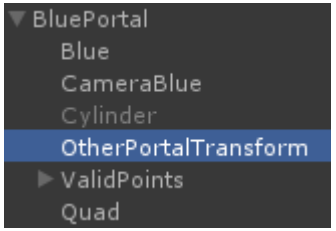
Portales

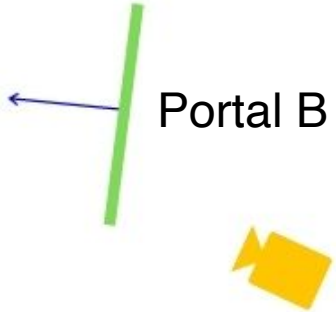

Portal A
Material Portal A
Render Texture A
Player
Portal B
Camera B

PORTALES

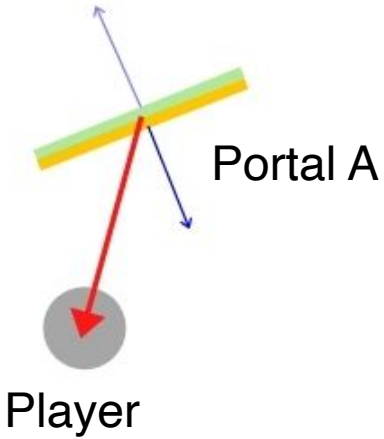

Virtual Portal B
Portal A
Player


OtherPortalTransform
Static
Tag
Untagged
Layer
Default
Transform
Position
X 0
Y 0
Z 0
Rotation
X 0
Y 180
Z 0
Scale
X 1
Y 1
Z 1


BluePortal
Blue
CameraBlue
Cylinder
OtherPortalTransform
ValidPoints
Quad

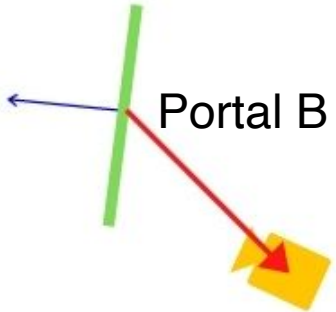

Portal B

Portales

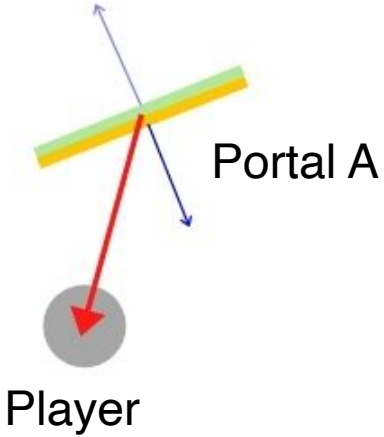


1. Calculamos la posición del player en coordenadas locales del portal virtual B.

```
Vector3 l_position =
    m_VirtualPortal.InverseTransformPoint(cameraPosition);
```

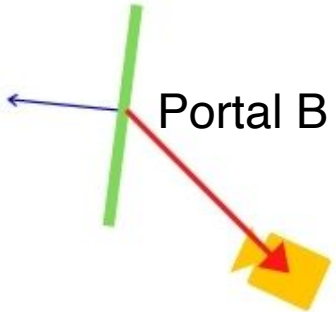


Portales

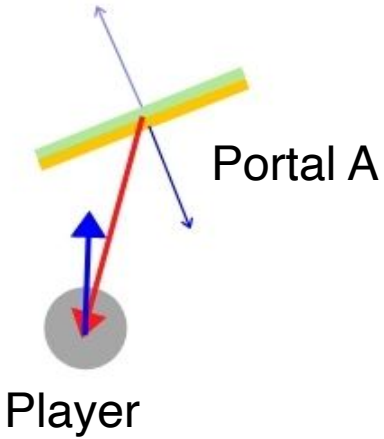


2. Transformamos esta posición, utilizando la transformación del Portal B, de coordenadas locales a coordenadas de mundo y colocamos la cámara B en esa posición.

```
m_OtherPortal.m_Camera.transform.position =
    m_OtherPortal.transform.TransformPoint(l_position);
```



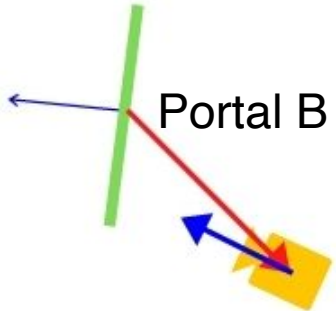
Portales



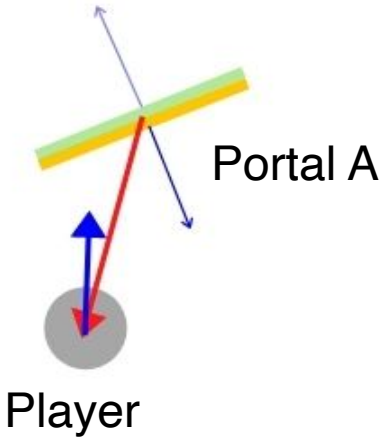
3. Hacemos la misma operación con la orientación del player, para conseguir el vector forward de la cámara, utilizando:

(…) = m_VirtualPortal.InverseTransformDirection(…);

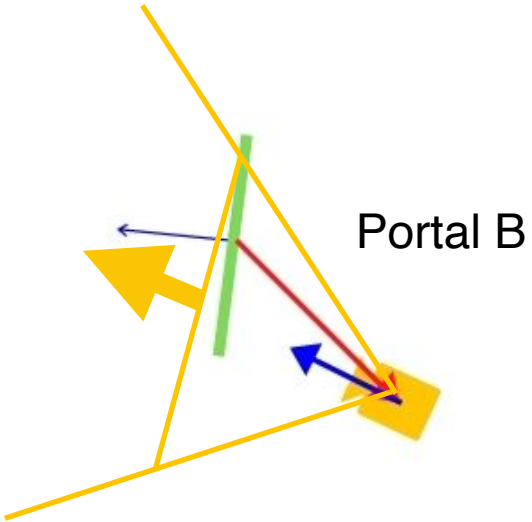
(…) = m_OtherPortal.transform.TransformDirection(…);



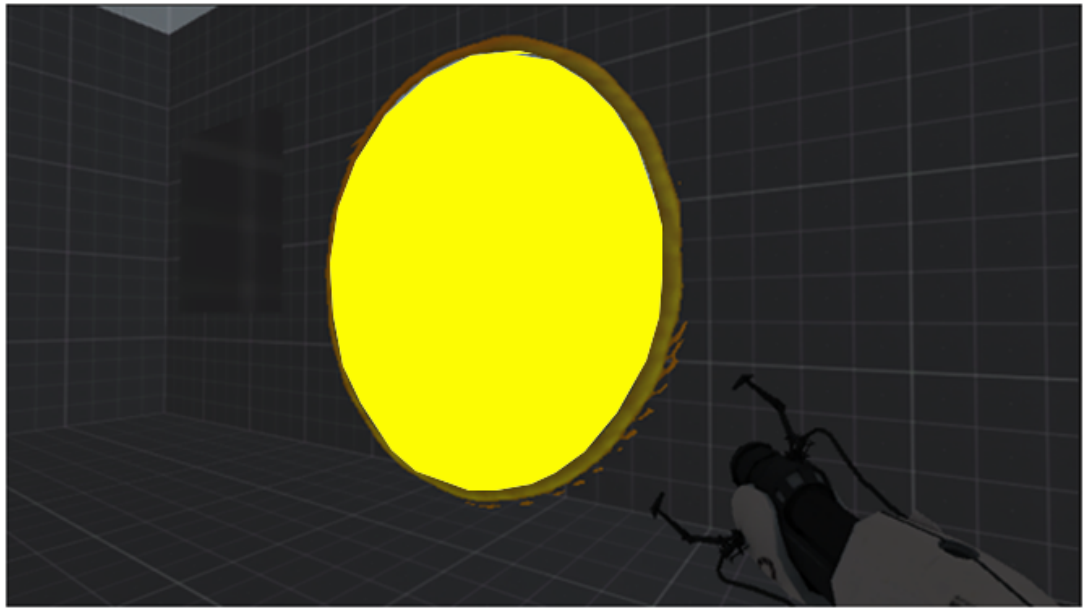
Portales



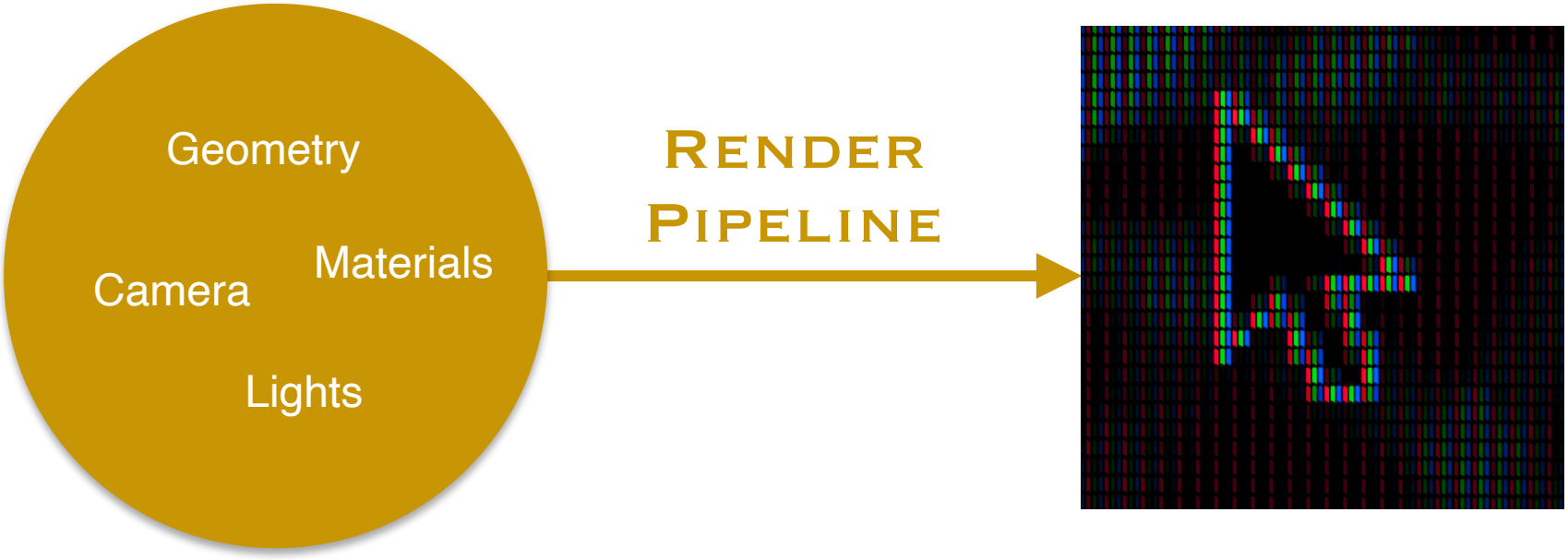
4. Para que la Camera B no renderice el portal ni las paredes adyacentes, definimos el nearClipPlane de la cámara como la distancia entre la cámara del player y el portal A, y le sumamos un offset definido como parámetro.



Portal - Shader



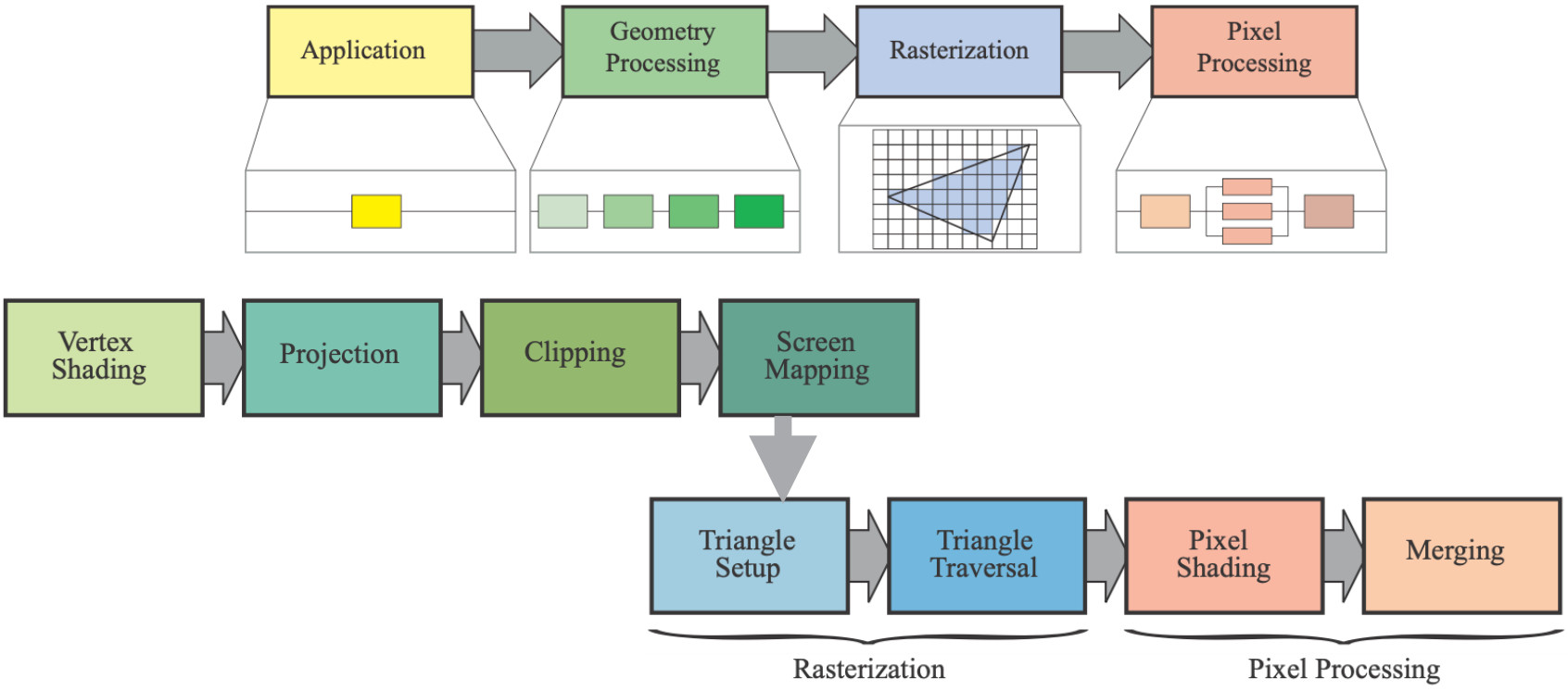
Render Pipeline



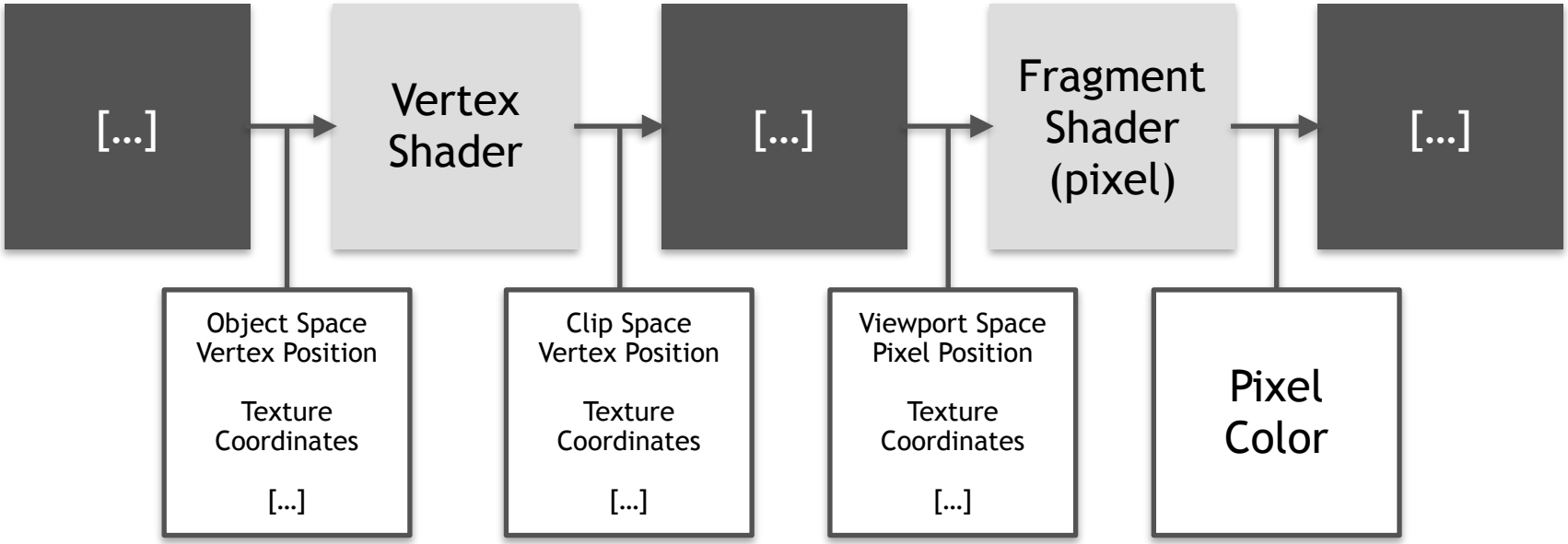
Unity Render Pipelines

Built-In Render Pipeline (Default)	Universal Render Pipeline (URP)	High-Definition Render Pipeline (HDRP)	Custom Scriptable Render Pipeline
The good old full-featured renderer for a wide range of platforms.	*A high-performance renderer focused on untethered devices.*	*The go-to renderer for AAA graphics on powerful & modern hardware.*	*Maximum freedom. Maximum optimization. Maximum programming.*
+ Full-featured & stable pipeline + Support for ALL platforms + Full Asset Store compatibility + Large knowledge base - More difficult to extend - Older renderer with less focus on optimum performance - No new features	+ Scalable graphics from mobile to PC + Performance-oriented + Unity Shader and VFX Graph support + Very easy to extend - Lack advanced real-time shading capabilities from HDRP * - Limited Asset Store compatibility *	+ Tailored for higher-end platforms (Compute-capable hardware only) + Performance-oriented + Unity Shader and VFX Graph support + Advanced real-time shading capabilities - Limited Asset Store compatibility *	+ Total customization - Graphics programming required - Limited Asset Store compatibility* - Complex support for other platforms than the ones already supported by URP or HDRP

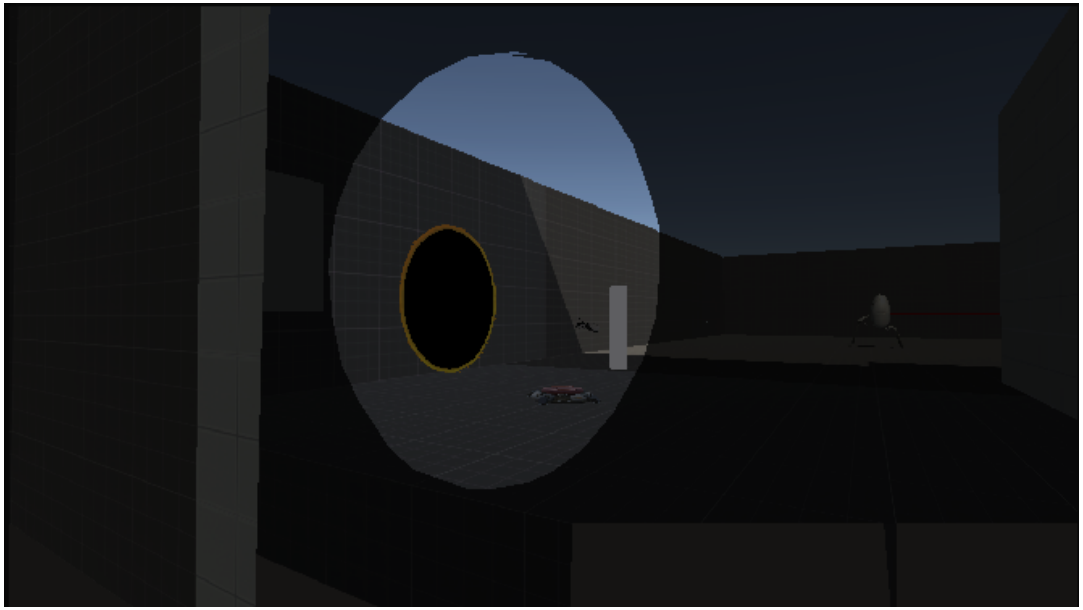
Render Pipeline



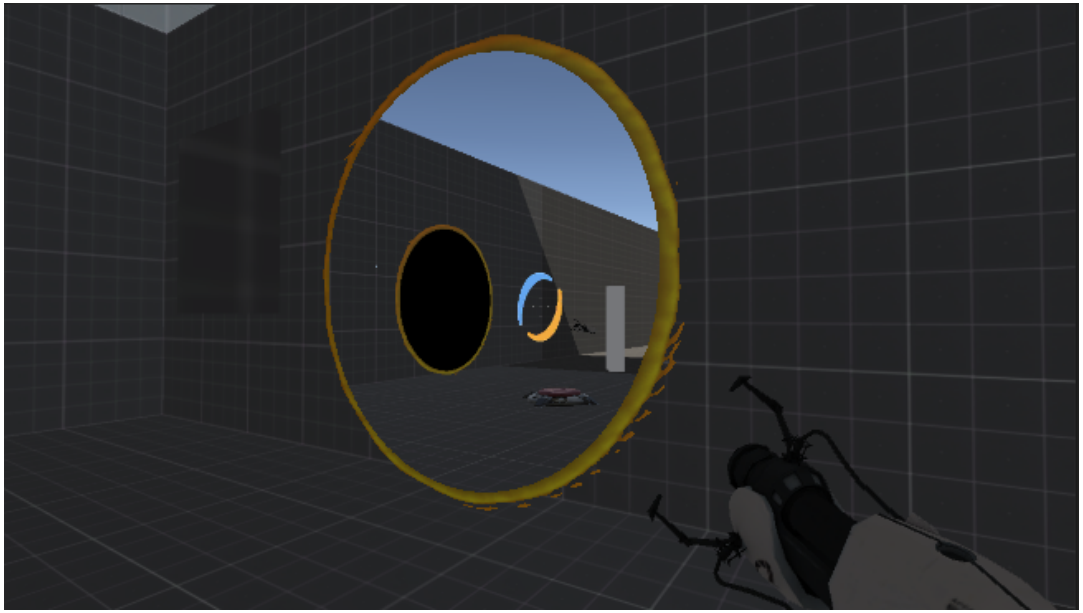
Programmable Pipeline



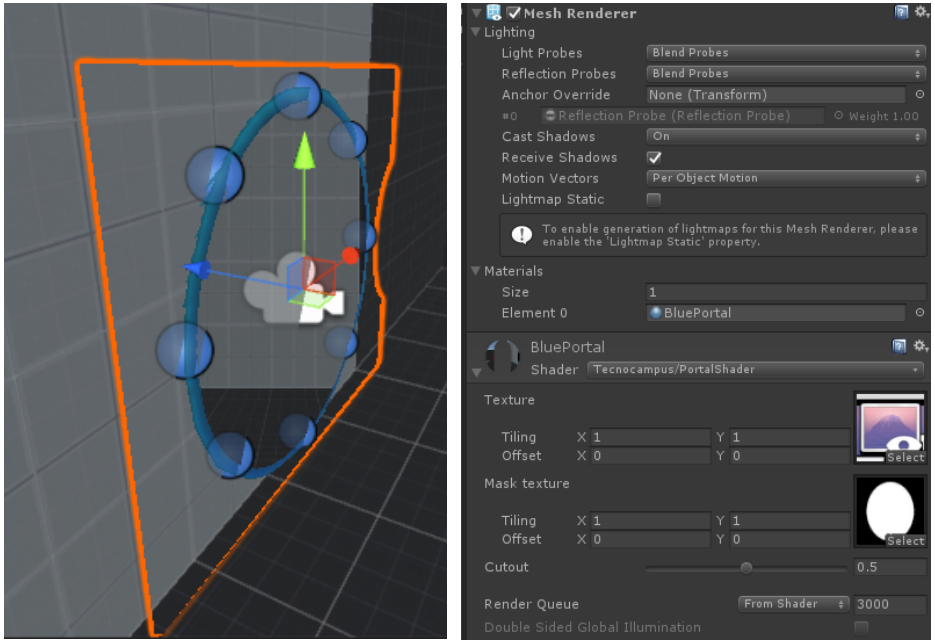
Portal – Shader RenderTarget



Portal - Resultado



Portal Shader - Implementación



Portal Shader – Implementación

```
Shader "Tecnocampus/PortalShader"
{
            Properties
            {
                        _MainTex ("Texture", 2D) = "white" {}
                        _MaskTex("Mask texture", 2D) = "white" {}
                        _Cutout("Cutout", Range(0.0, 1.0)) = 0.5
            }
            SubShader
            {
                        Tags{ "Queue" = "Geometry " "IgnoreProjector" = "True" "RenderType" = "Opaque" }
                        Lighting Off
                        Cull Back
                        ZWrite On
                        ZTest Less

                        Fog{ Mode Off }
```

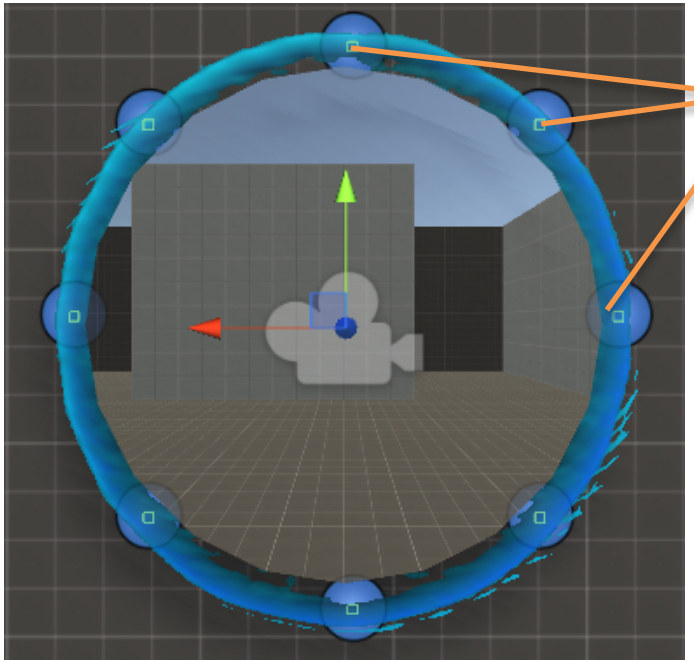
Portal Shader – Implementación

```
Pass
{
    CGPROGRAM
    #pragma vertex vert
    #pragma fragment frag

    #include "UnityCG.cginc"

    struct appdata
    {
        float4 vertex : POSITION;
        float2 uv : TEXCOORD0;
    };

    struct v2f
    {
        float4 vertex : SV_POSITION;
        float2 uv : TEXCOORD0;
        float4 screenPos : TEXCOORD1;
    };
```

Portal Shader – Implementación

```
            v2f vert (appdata v)
            {
                    v2f o;
                    o.vertex = UnityObjectToClipPos(v.vertex);
                    o.uv = v.uv;
                    o.screenPos = ComputeScreenPos(o.vertex);
                    return o;
            }

            sampler2D _MainTex;
            sampler2D _MaskTex;
            float _Cutout;

            fixed4 frag (v2f i) : SV_Target
            {
                i.screenPos /= i.screenPos.w;

                fixed4 l_MaskColor= tex2D(_MaskTex, i.uv);
                if (l_MaskColor.a < _Cutout)
                                clip(-1);
                fixed4 col = tex2D(_MainTex, float2(i.screenPos.x, i.screenPos.y));

                return col;
            }
            ENDCG
            }
}
}
```

Portales – IsValidPosition



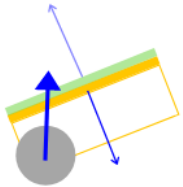
Portales – IsValidPosition – Implementación

Para decidir si un portal puede colocarse en una posición implementaremos *bool IsValidPosition()*:

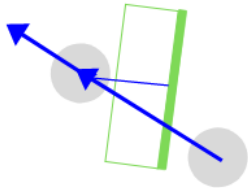
- Colocaremos a modo de previsualización el portal en la ubicación a evaluar y para cada uno de los ValidPoints del portal, lazaremos un rayo entre la cámara del player y el ValidPoint.
- Solo daremos por válida la posición si se cumplen todas las condiciones para todos los ValidPoints:
  A. Distancia entre el punto de RaycastHit y el ValidPoint es inferior a un umbral mínimo.
  B. El ángulo entre la normal del RaycastHit y el eje Z del ValidPoint es inferior a un umbral mínimo.
  C. El gameObject del RaycastHit es del tipo (tag) “pared pintable”.

Portales – Teleport

Para teletransportar al personaje utilizaremos un box trigger en el portal y cuando el player entre en el collider, llamaremos a la función Teleport(Portal portal) del player. Para implementar esta función utilizaremos la misma estrategia que para la colocación de la cámara.



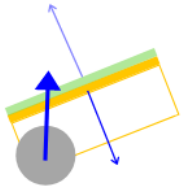
1. Calculamos la posición y dirección del player en coordenadas locales del portal virtual B.



```
Vector3 l_Position  =
    _Portal.m_VirtualPortal.transform.InverseTransformPoint(transform.position);
Vector3 l_Direction =
    _Portal.m_VirtualPortal.transform.InverseTransformDirection(-transform.forward);
```

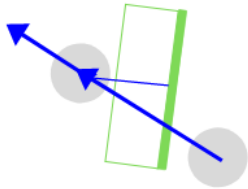
Portales – Teleport

2. Colocamos y orientamos al player en la misma posición relativa al portal B que la que tenía respecto al portal virtual.

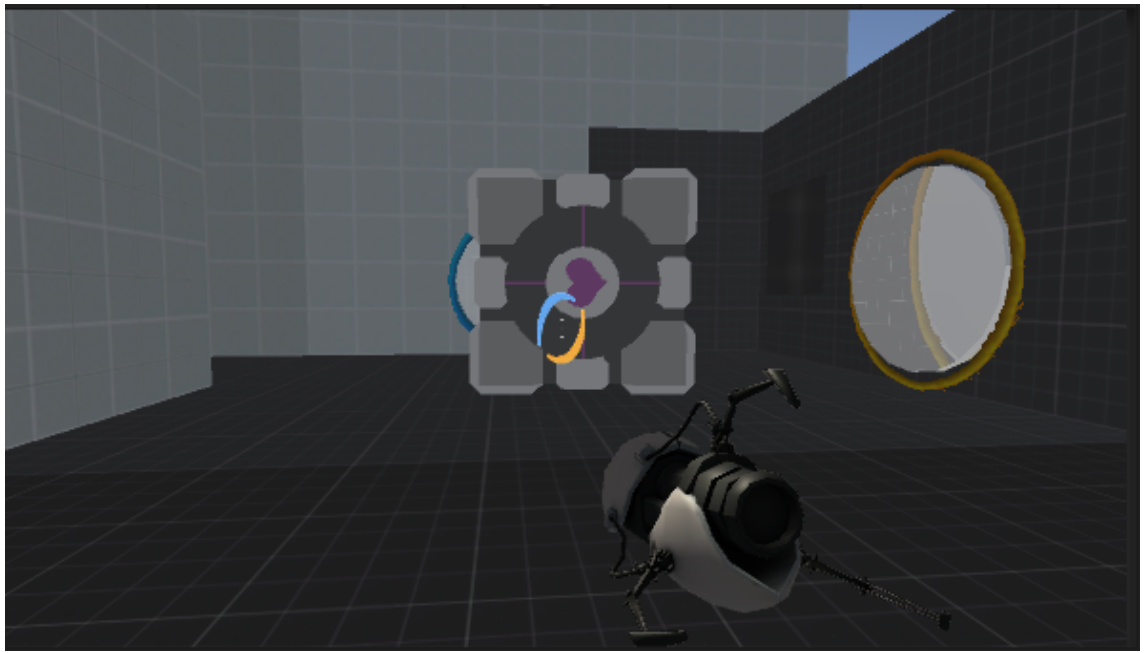


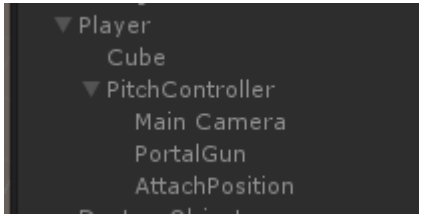
```
transform.position =
   _Portal.m_MirrorPortal.transform.TransformPoint(l_Position);
transform.forward =
   _Portal.m_MirrorPortal.transform.TransformDirection(l_Direction);
```

3. Hacemos avanzar al player en dirección a su forward un offset determinado para que “atraviese” el portal y salga del trigger del otro portal. Este desplazamiento debe ser pequeño ya que la profundidad del trigger será mucho menor al esquema mostrado. También tendremos que actualizar el mYaw del player para mantener el forward asignado en el paso anterior.



Portales – AttachObject





AttachObject – Implementación

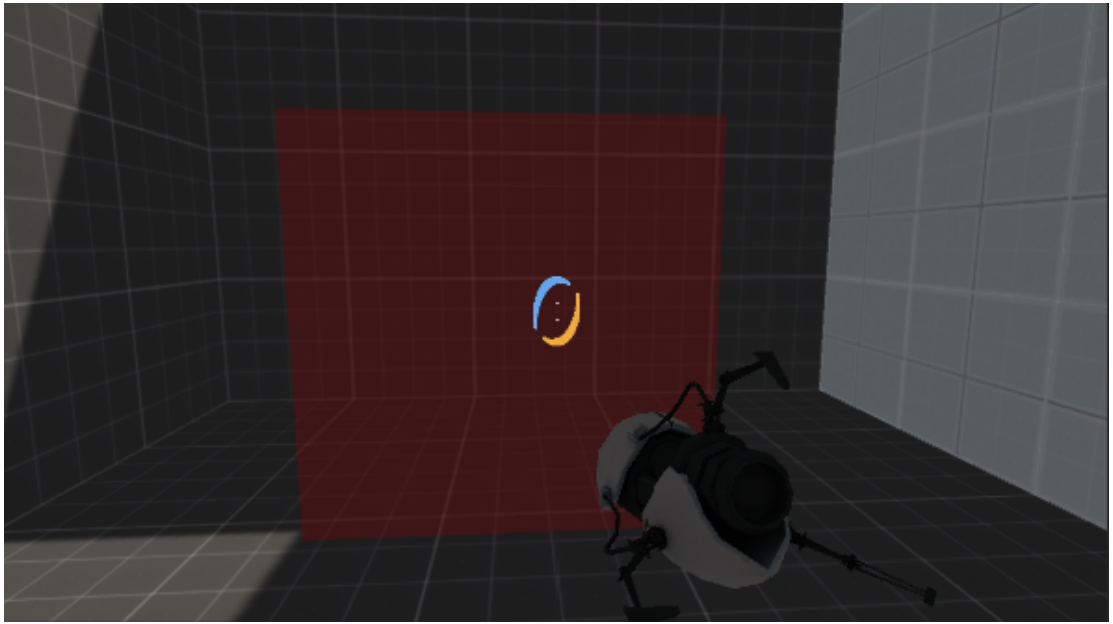
```
void UpdateAttachedObject()
{
	Vector3 l_EulerAngles=m_AttachingPosition.rotation.eulerAngles;
	if(!m_AttachedObject)
	{
		Vector3 l_Direction=m_AttachingPosition.transform.position-m_ObjectAttached.transform.position;
		float l_Distance=l_Direction.magnitude;
		float l_Movement=m_AttachingObjectSpeed*Time.deltaTime;
		if(l_Movement>=l_Distance)
		{
			m_AttachedObject=true;
			m_ObjectAttached.MovePosition(m_AttachingPosition.position);
			m_ObjectAttached.MoveRotation(Quaternion.Euler(0.0f, l_EulerAngles.y, l_EulerAngles.z));
		}
		else
		{
			l_Direction/=l_Distance;
			m_ObjectAttached.MovePosition(m_ObjectAttached.transform.position+l_Direction*l_Movement);
			m_ObjectAttached.MoveRotation(Quaternion.Lerp(m_AttachingObjectStartRotation, Quaternion.Euler(0.0f, l_EulerAngles.y,
	l_EulerAngles.z), 1.0f-Mathf.Min(l_Distance/1.5f, 1.0f)));
		}
	}
	else
	{
		m_ObjectAttached.MoveRotation(Quaternion.Euler(0.0f, l_EulerAngles.y, l_EulerAngles.z));
		m_ObjectAttached.MovePosition(m_AttachingPosition.position);
	}
}
```

Attach / Throw Object – Implementación

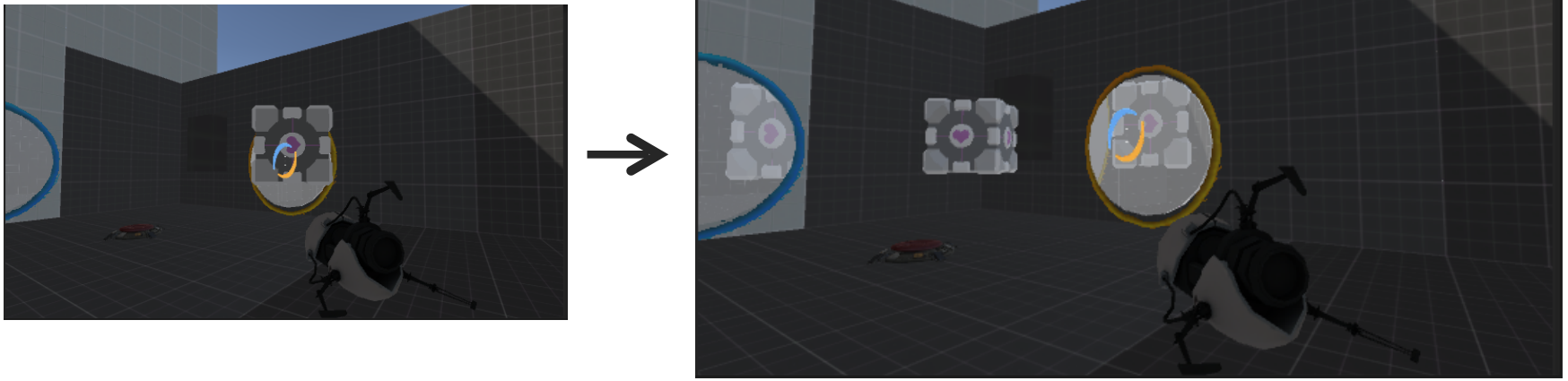
Crearemos un método en el Player para lanzar o dejar ir un objeto. Para reutilizar el método, pasaremos el parámetro *float Force* para poder llamarlo con un valor distinto de cero para lanzar e igual a cero, para dejar ir.

- Para dejar de controlar el objeto, tendremos que poner a falso los dos *booleanos* de control del método *UpdateAttachedObject*(): *m_AttachedObject* y *m_AttachingObject*;
- El Rigidbody del objeto pasará de modo Cinemático a modo Dinámico. (*Rigidbody.isKinematic*)
- Tendremos que informar al *script* del *Companion* que puede ser teletransportado por los portales. (*SetTeleportable(true)*)
- Le aplicaremos una fuerza al Rigidbody del objeto en dirección al transform que nos indica la posición en la que transportamos objetos (*m_AttachingPosition*) y de módulo *force* (parámetro).

CompanionSpawner



Companion – Teleport

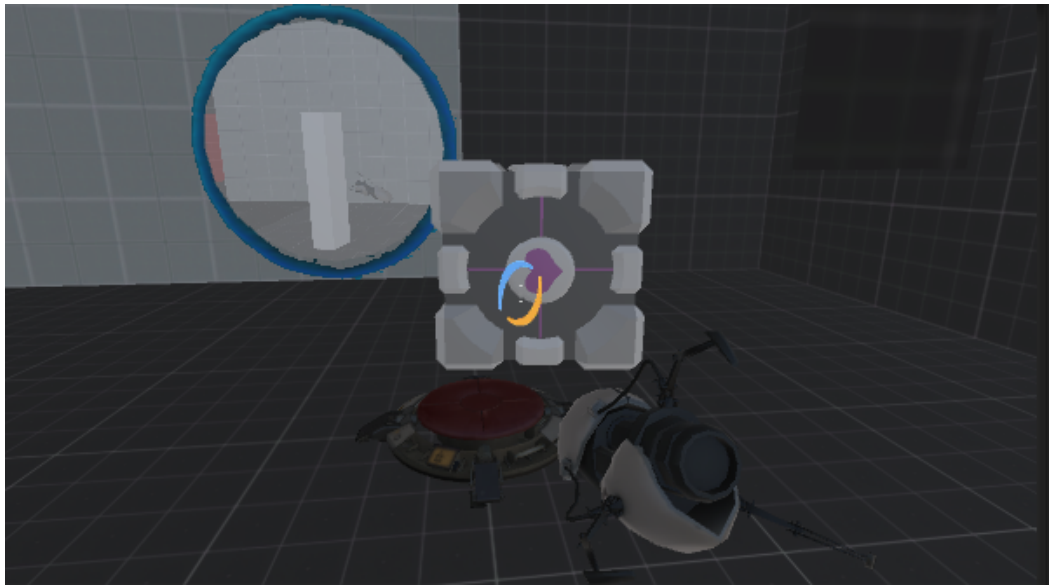


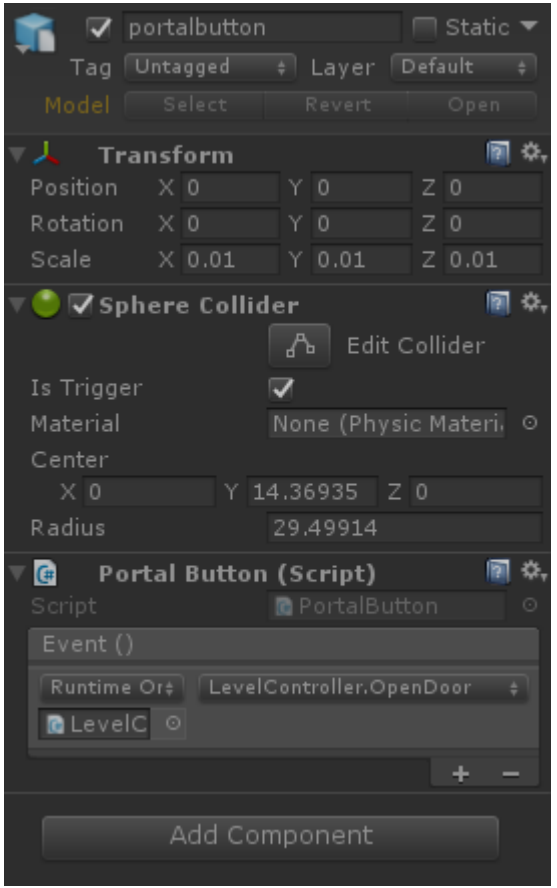
Companion Teleport– Implementación

Para que cuando lancemos un Companion, éste pueda atravesar el portal, implementaremos su función *Teleport*. Solo permitiermos que se teletransporte cuando no lo estemos llevando, por lo que tendremos que comprobar esa condición. Además de mover e orientar el objeto igual que al player, tendremos que modificarle la velocidad y la escala del objeto para orientarlo en la dirección del portal y para que le afecte el factor de escala entre los dos portales.

```
Vector3 l_Velocity=
      _Portal.m_VirtualPortal.transform.InverseTransformDirection(l_Rigidbody.velocity);
l_Rigidbody.velocity =
      _Portal.m_MirrorPortal.transform.TransformDirection(l_Velocity);
transform.localScale *=
      (_Portal.m_MirrorPortal.transform.localScale.x/_Portal.transform.localScale.x);
```

PortalButton

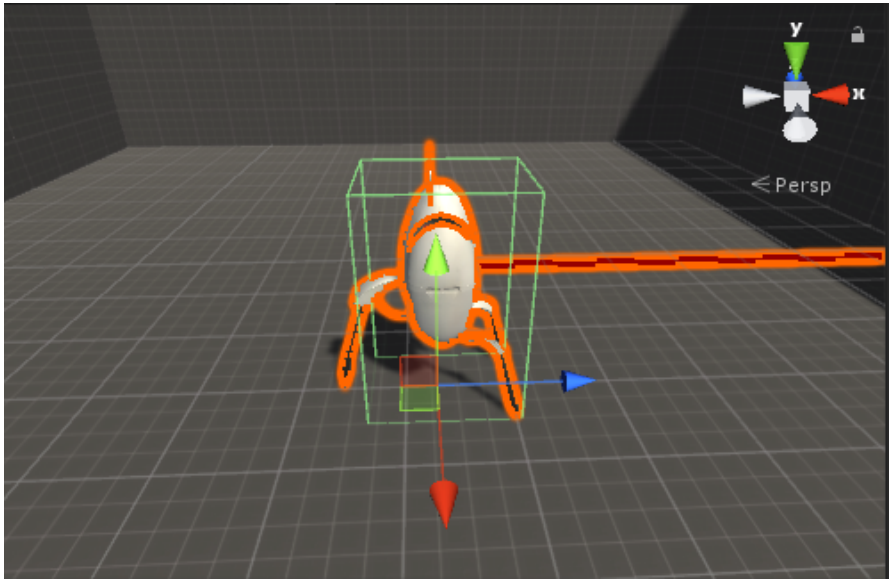


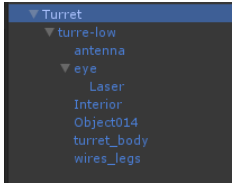


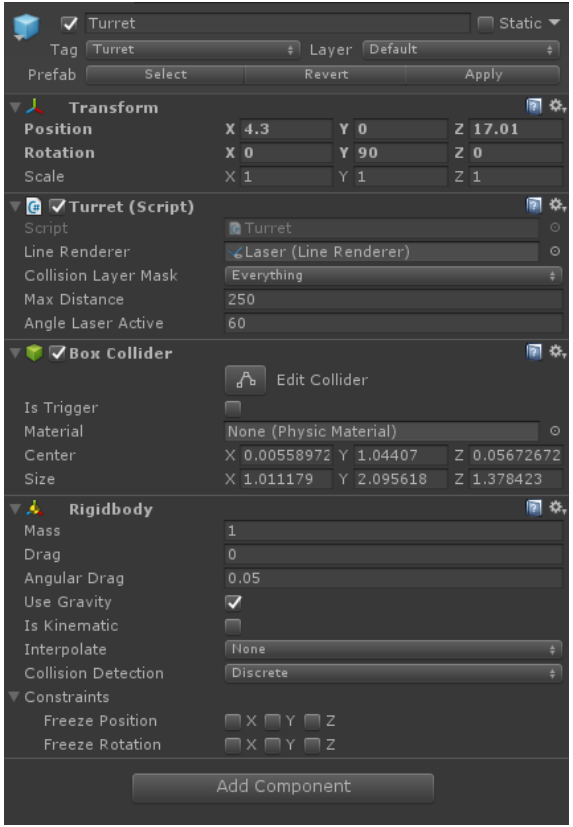
Unity Events - PortalButton

```
public UnityEvent m_Event;

void OnTriggerEnter(Collider _Collider)
{
    if(_Collider.tag==“ANY TAG")
        m_Event.Invoke();
}
```

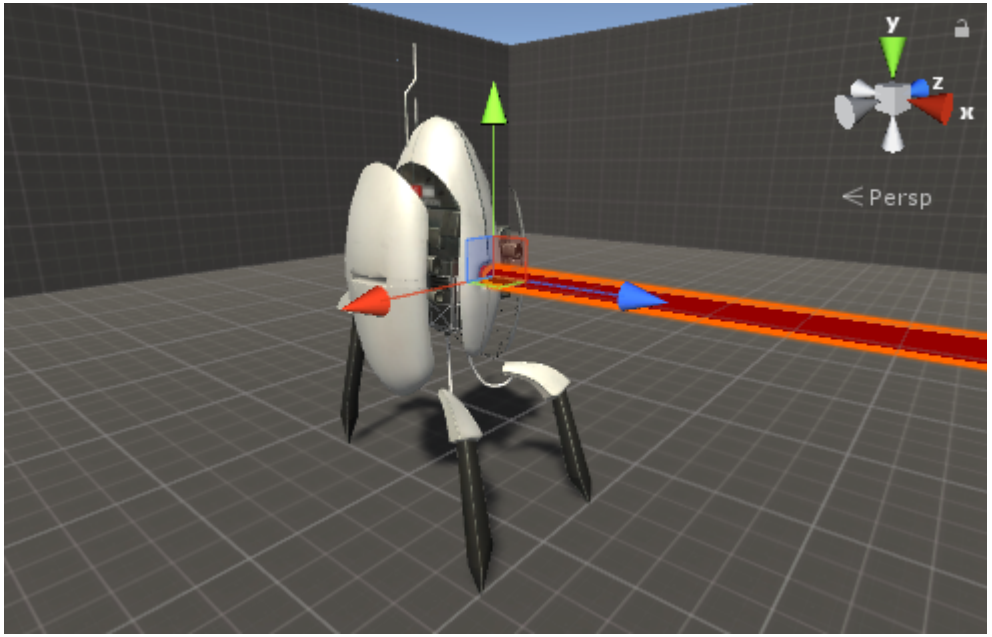
Turret

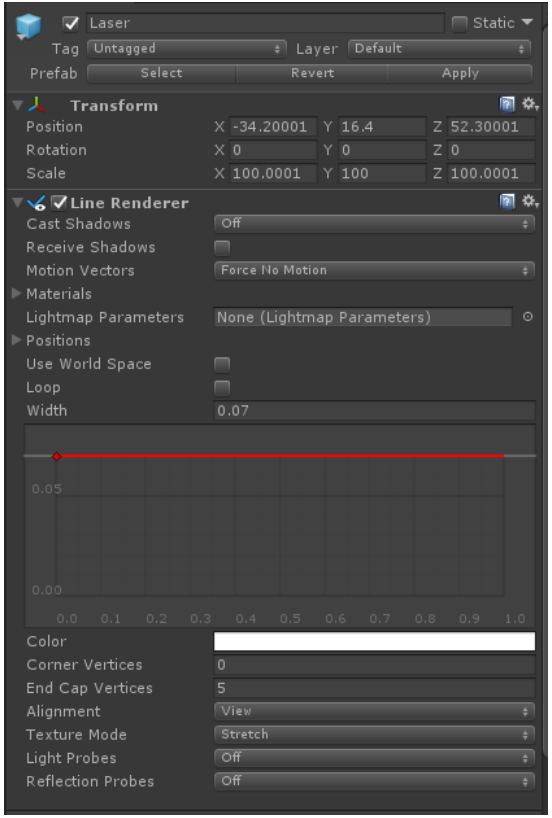

y
x
Persp


Turret
turre-low
antenna
eye
Laser
Interior
Object014
turret_body
wires_legs


Turret
Static
Tag Turret
Layer Default
Prefab Select Revert Apply
Transform
Position X 4.3 Y 0 Z 17.01
Rotation X 0 Y 90 Z 0
Scale X 1 Y 1 Z 1
Turret (Script)
Script Turret
Line Renderer Laser (Line Renderer)
Collision Layer Mask Everything
Max Distance 250
Angle Laser Active 60
Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0.00558972 Y 1.04407 Z 0.05672672
Size X 1.011179 Y 2.095618 Z 1.378423
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position X Y Z
Freeze Rotation X Y Z
Add Component

Turret - Laser





Turret – Limitar alcance del Laser

Declararemos un float m_MaxDistance para definir el máximo alcance del láser.
Lanzaremos un Raycast hacia el forward del LineRenderer limitado a la distancia máxima.

```
Physics.Raycast(new Ray(m_LineRenderer.transform.position, m_LineRenderer.transform.forward),
    out l_RaycastHit, m_MaxDistance, m_CollisionLayerMask.value))
```

Si impacta con algún objeto, este objeto limitará el alcance del láser, por lo que tendremos que asignar la segunda posición del LineRenderer a este punto.

```
l_EndRaycastPosition=Vector3.forward*l_RaycastHit.distance;
```

Si no impacta con nada, el segundo punto del LineRenderer será el máximo definido por el m_MaxDistance*forward.

```
l_EndRaycastPosition=Vector3.forward*m_MaxDistance;
```

```
m_LineRenderer.SetPosition(1, l_EndRaycastPosition);
```

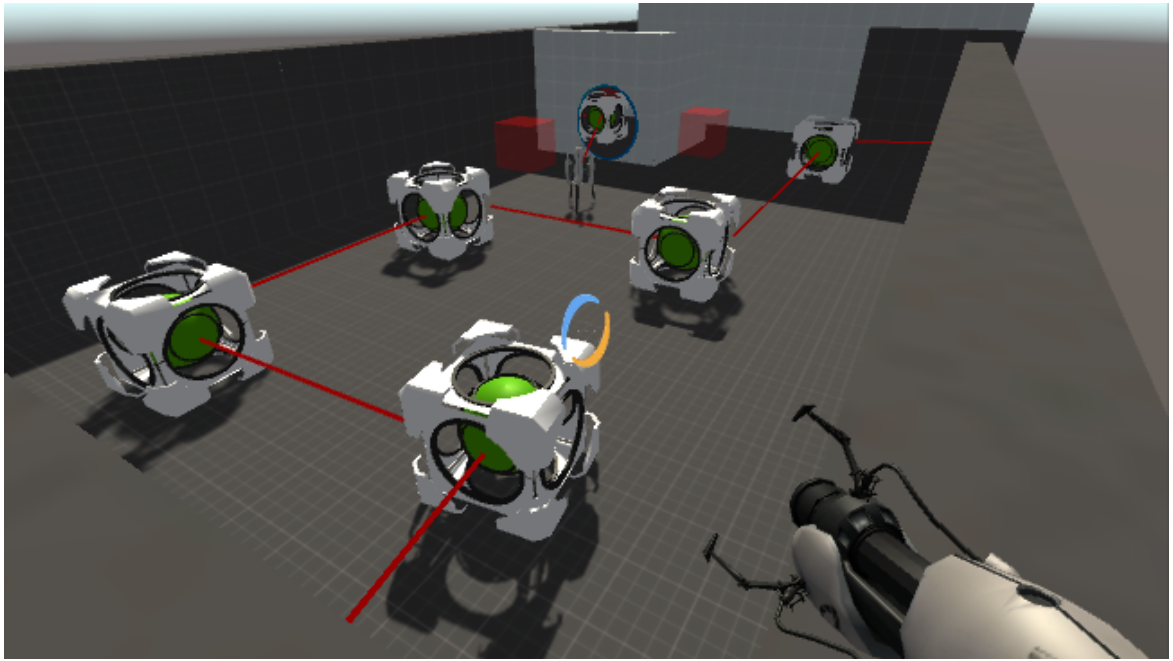
Turret – Laser activo Implementación

Definiremos un ángulo máximo en el que la Turret seguirá funcionando.

En caso que el ángulo entre el *up* del objeto y el *up* del ángulo sea mayor que el ángulo máximo definido, desactivaremos el láser.

Si es inferior, lo activaremos.

Refraction Cube



Refraction Cube – Implementación

```
void Update()
{
     m_LineRenderer.gameObject.SetActive(m_CreateRefraction);
     m_CreateRefraction=false;
}
public void CreateRefraction()
{
     m_CreateRefraction=true;
     Vector3 l_EndRaycastPosition=Vector3.forward*m_MaxDistance;
     RaycastHit l_RaycastHit;
     if(Physics.Raycast(new Ray(m_LineRenderer.transform.position, m_LineRenderer.transform.forward), out
     l_RaycastHit, m_MaxDistance, m_CollisionLayerMask.value))
     {
          l_EndRaycastPosition=Vector3.forward*l_RaycastHit.distance;
          if(l_RaycastHit.collider.tag=="RefractionCube")
          {
               //Reflect ray
               l_RaycastHit.collider.GetComponent<RefractionCube>().CreateRefraction();
          }
          //Other collisions
     }
     m_LineRenderer.SetPosition(1, l_EndRaycastPosition);
}
```

Refraction en Portal – Implementación

```
void Update()
{
     m_LineRenderer.gameObject.SetActive(m_CreateRefraction);
     m_CreateRefraction=false;
}
public void CreateRefraction()
{
     m_CreateRefraction=true;
     Vector3 l_EndRaycastPosition=Vector3.forward*m_MaxDistance;
     RaycastHit l_RaycastHit;
     if(Physics.Raycast(new Ray(m_LineRenderer.transform.position, m_LineRenderer.transform.forward), out
     l_RaycastHit, m_MaxDistance, m_CollisionLayerMask.value))
     {
          l_EndRaycastPosition=Vector3.forward*l_RaycastHit.distance;
          if(l_RaycastHit.collider.tag=="RefractionCube")
          {
               //Reflect ray
               l_RaycastHit.collider.GetComponent<RefractionCube>().CreateRefraction();
          }
          //Other collisions
     }
     m_LineRenderer.SetPosition(1, l_EndRaycastPosition);
}
```