```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

//二叉树的结点类
public class TreeNode<T> where T:IComparable{

    public T data;//数据
    public TreeNode<T> leftChild;//左孩子
    public TreeNode<T> rightChild;//右孩子

    public TreeNode(T data,TreeNode<T> l,TreeNode<T> r)
    {
        this.data = data;
        this.leftChild = l;
        this.rightChild = r;
    }

    public TreeNode(T data)
    {
        this.data = data;
    }

    public int Compare(T t1, T t2)
    {
        if (t1.CompareTo(t2) > 0)
            return 1;
        return -1;
    }

    //插入函数 判断当前需要插入的数据 是否小于当前结点 如果小于加入到左孩子 大于加到右孩子
    public void Insert(TreeNode<T> other)
    {
        if (data.CompareTo(other.data) > 0)
        {
            if (leftChild == null)
                leftChild = other;
```

```
            else
                leftChild.Insert(other);
        }
        else
        {
            if (rightChild == null)
                rightChild = other;
            else
                rightChild.Insert(other);
        }
    }

    public void PreOrderNode()
    {
        Debug.Log(data);

        if (leftChild != null)
            leftChild.PreOrderNode();

        if (rightChild != null)
            rightChild.PreOrderNode();

    }

    public void InorderNode()
    {
        if (leftChild != null)
            leftChild.InorderNode();

        Debug.Log(data);

        if (rightChild != null)
            rightChild.InorderNode();
    }


    public void PostorderNode()
    {
        if (leftChild != null)
            leftChild.PostorderNode();

        if (rightChild != null)
```

```
            rightChild.PostorderNode();

        Debug.Log(data);
    }


}

```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;

public class BinaryTree<T> where T:IComparable{

    public TreeNode<T> head;//根结点

    public BinaryTree(TreeNode<T> head)
    {
        this.head = head;
    }
    public BinaryTree()
    {
        this.head = null;
    }

    public void Insert(T data)
    {
        if (head == null)
        {
            head = new TreeNode<T>(data);
            return;
        }

        TreeNode<T> temp=new TreeNode<T>(data);

        head.Insert(temp);
    }

```

```
    //前序遍历二叉树 先序 根左右
    public void PreorderNode()
    {
        if (head == null)
            return;
        head.PreOrderNode();
    }

    //中序遍历二叉树 左根右
    public void InorderNode()
    {
        if (head == null)
            return;
        head.InorderNode();
    }

    //后序    左右根
    public void PostorderNode()
    {
        if (head == null)
            return;
        head.PostorderNode();
    }

    //层次遍历
    public void LevelorderNode(TreeNode<int> head)
    {
        if (head == null)
            return;
        List<TreeNode<int>> datas = new List<TreeNode<int>>();
        datas.Add(head);

        while (datas.Count > 0)
        {
            Debug.Log(datas[0].data);

            if (datas[0].leftChild != null)
                datas.Add(datas[0].leftChild);
            if (datas[0].rightChild != null)
                datas.Add(datas[0].rightChild);
            datas.Remove(datas[0]);
        }
```

```
74      }
75  }
```

HTC Vive

Oculus rift

SteamVR的核心组件

1 StreamVR_Render

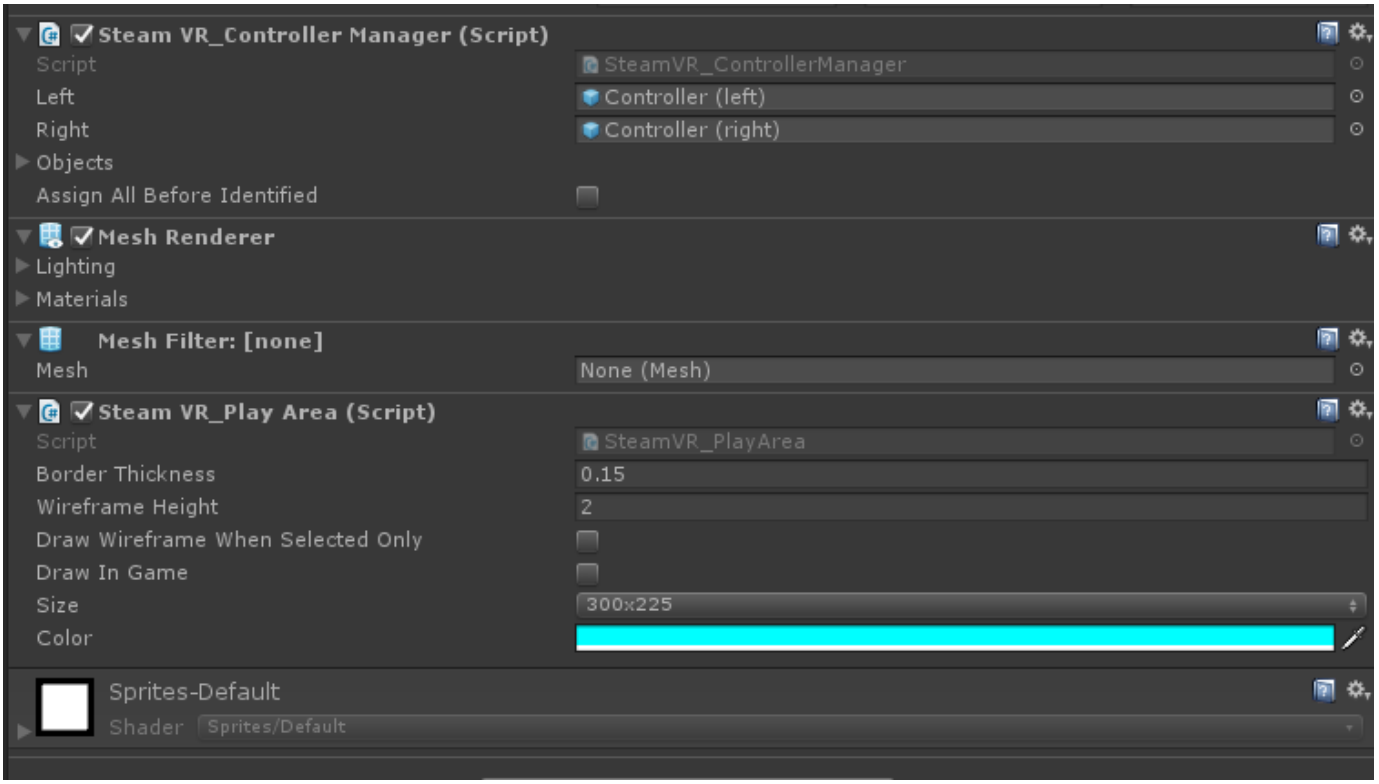
管理VR的渲染工作和基本设置

2 SteamVR_ControllerManager VR

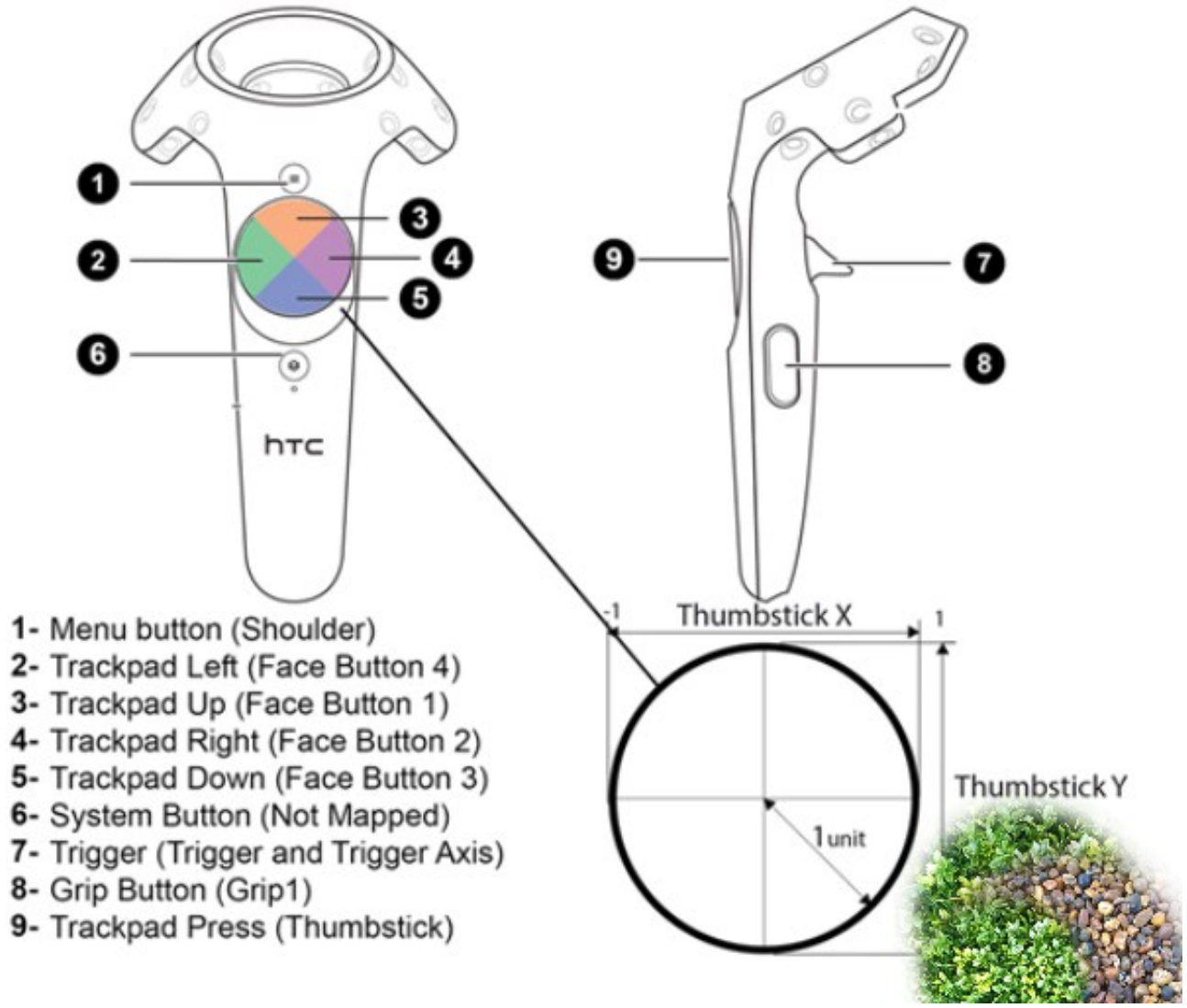
管理控制器的脚本 控制左右操作杆的事件更新

3 SteamVR_PlayerArea

游戏区域限制 宽高 周边限制框


Steam VR_Controller Manager (Script)
Script
SteamVR_ControllerManager
Left
Controller (left)
Right
Controller (right)
Objects
Assign All Before Identified
Mesh Renderer
Lighting
Materials
Mesh Filter: [none]
Mesh
None (Mesh)
Steam VR_Play Area (Script)
Script
SteamVR_PlayArea
Border Thickness
0.15
Wireframe Height
2
Draw Wireframe When Selected Only
Draw In Game
Size
300x225
Color
Sprites-Default
Shader
Sprites/Default


htc




```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Left : MonoBehaviour {

    SteamVR_TrackedObject trackedObject;
    SteamVR_Controller.Device device;
    void Start () {
        trackedObject = GetComponent<SteamVR_TrackedObject>();
    }


    void FixedUpdate() {
        device = SteamVR_Controller.Input((int)trackedObject.index);

```

```
        if (device.GetTouch(SteamVR_Controller.ButtonMask.Trigger))
        {
            Debug.Log("Touch Trigger");
        }
        if
(device.GetTouchDown(SteamVR_Controller.ButtonMask.Trigger))
        {
            Debug.Log("Touch Trigger");
        }
        if (device.GetTouchUp(SteamVR_Controller.ButtonMask.Trigger))
        {
            Debug.Log("Touch Trigger");
        }

        if (device.GetPressUp(SteamVR_Controller.ButtonMask.System))
        {

        }
    }

    void OnTriggerStay(Collider collider)
    {
        if (device.GetTouch(SteamVR_Controller.ButtonMask.Trigger))
        {
            collider.transform.parent = transform;
        }
        if (device.GetTouch(SteamVR_Controller.ButtonMask.Touchpad))
        {
            collider.transform.parent = null;
        }
    }
}

```