【游戏仿真实验】使用Unity仿真电视机光学三原色显示画面，我是要成为海贼王的男人

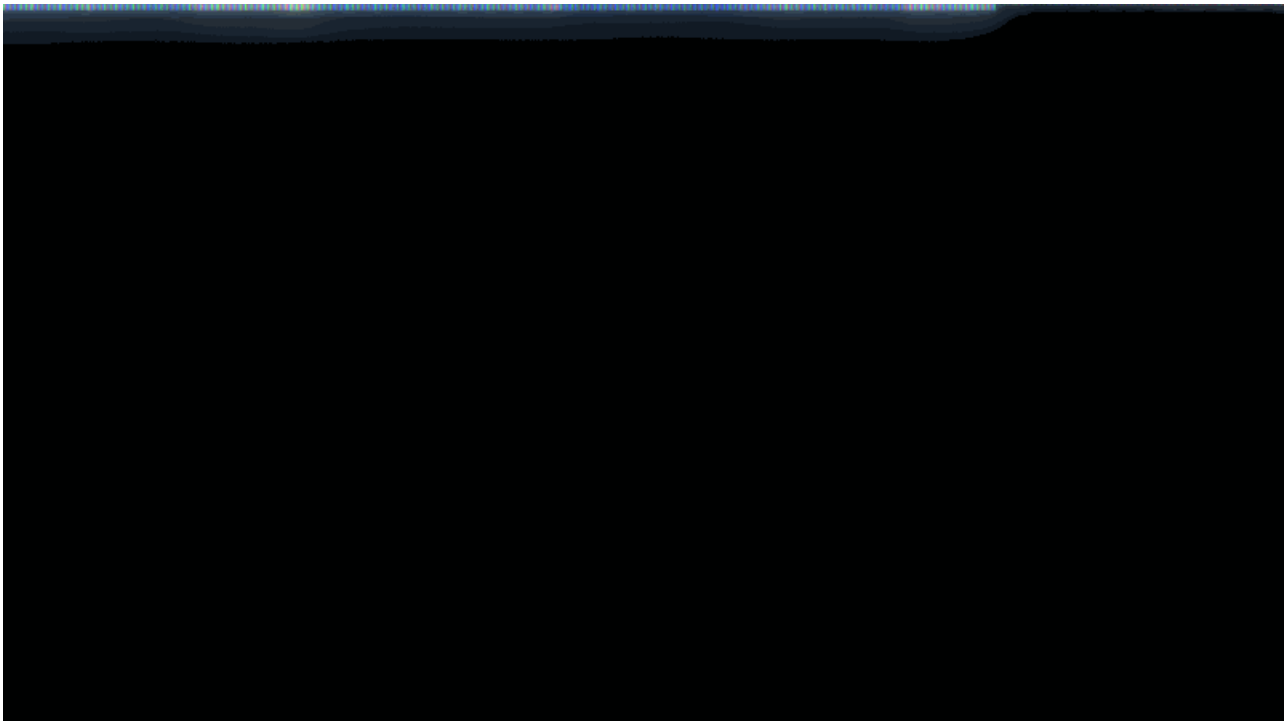
原创

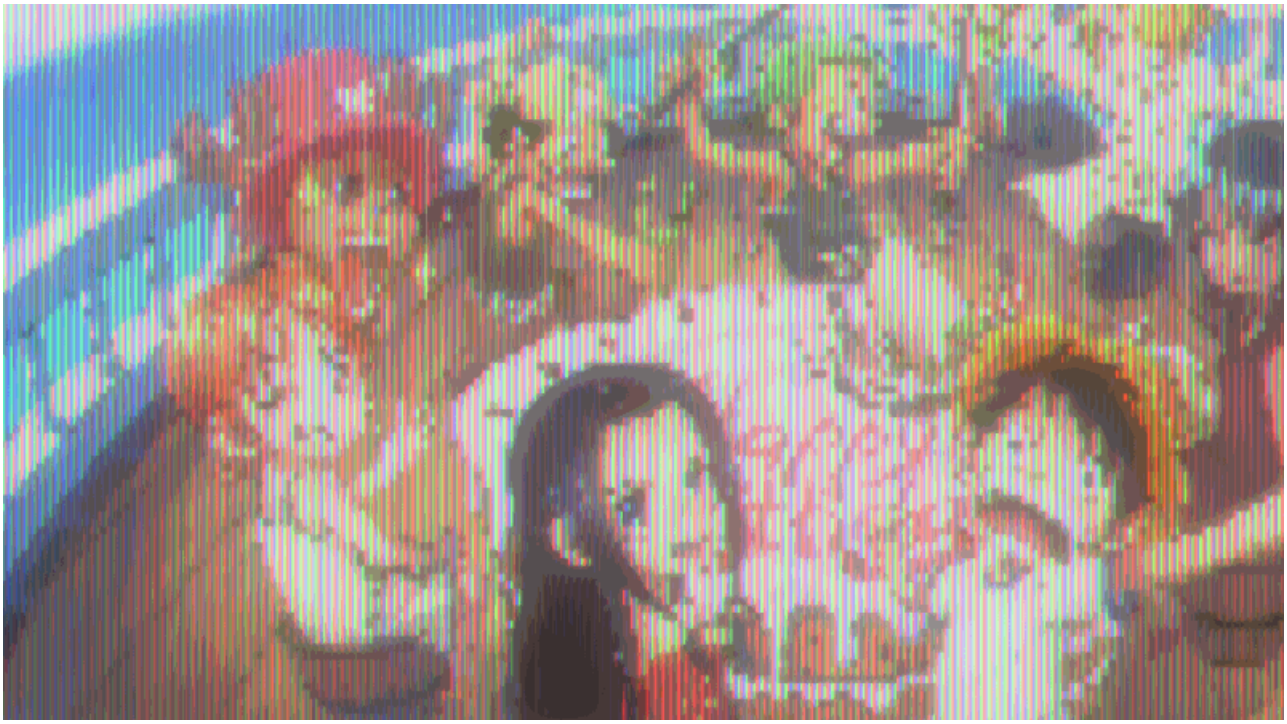
林新发 于 2021-08-22 08:08:48 发布 3581 收藏 43

分类专栏： Unity3D 文章标签： unity 三原色 电视机 光学 仿真

本文链接：https://blog.csdn.net/linxinfa/article/details/119845923

本文效果如下





文章目录

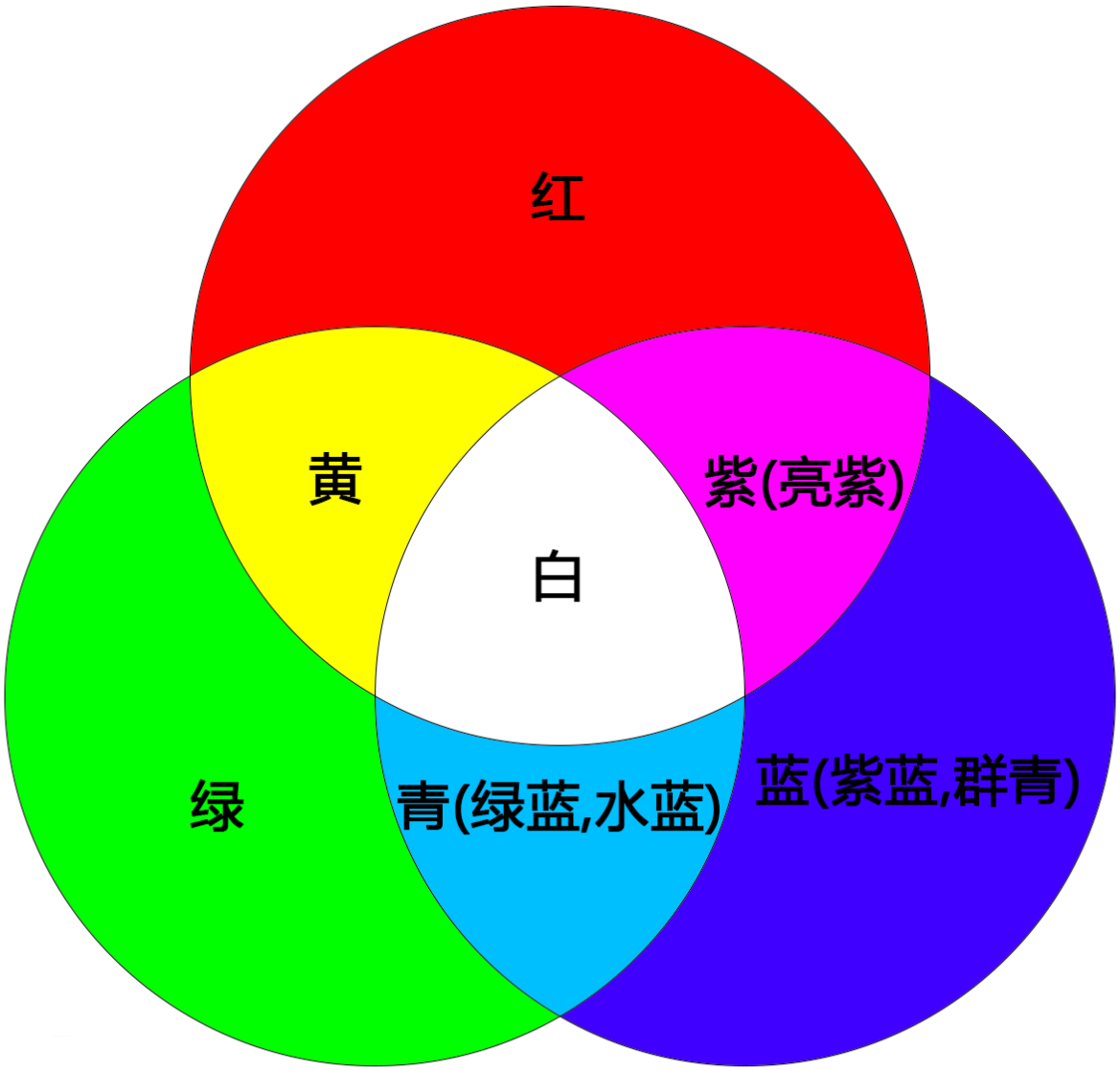
一、前言

嗨，大家好，我是新发。

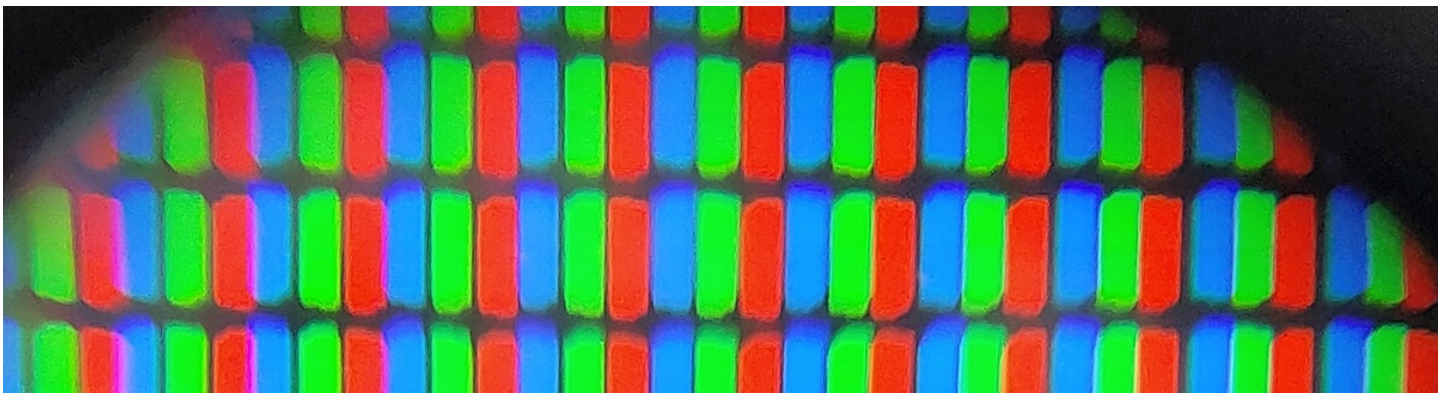
最近我在尝试用`Unity`制作非游戏的场景，做一些数学、物理的仿真实验，比如上一篇文章我做了一个《蒲丰投针仿真实验》（点击查看），今天，我又要做另一个实验，模拟电视机光学三原色的原理来显示画面，话不多说，我们开始吧~

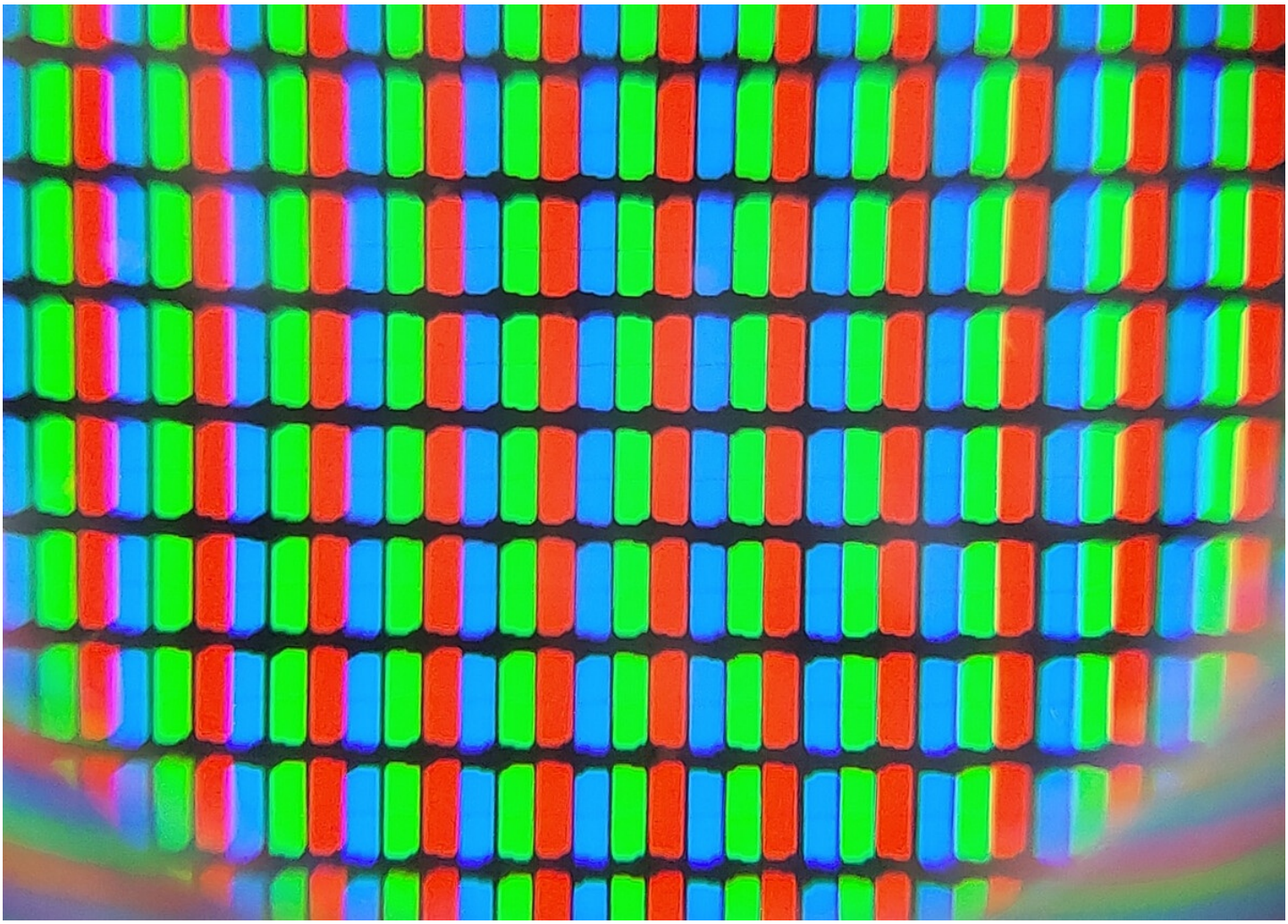
二、光学三原色

光学三原色分别为红、绿、蓝，这三个颜色通道色各分为`256`阶亮度，在`0`时最弱，而在`255`时最亮。也就是说，一共可以表现`256 x 256 x 256 = 16777216`种颜色。



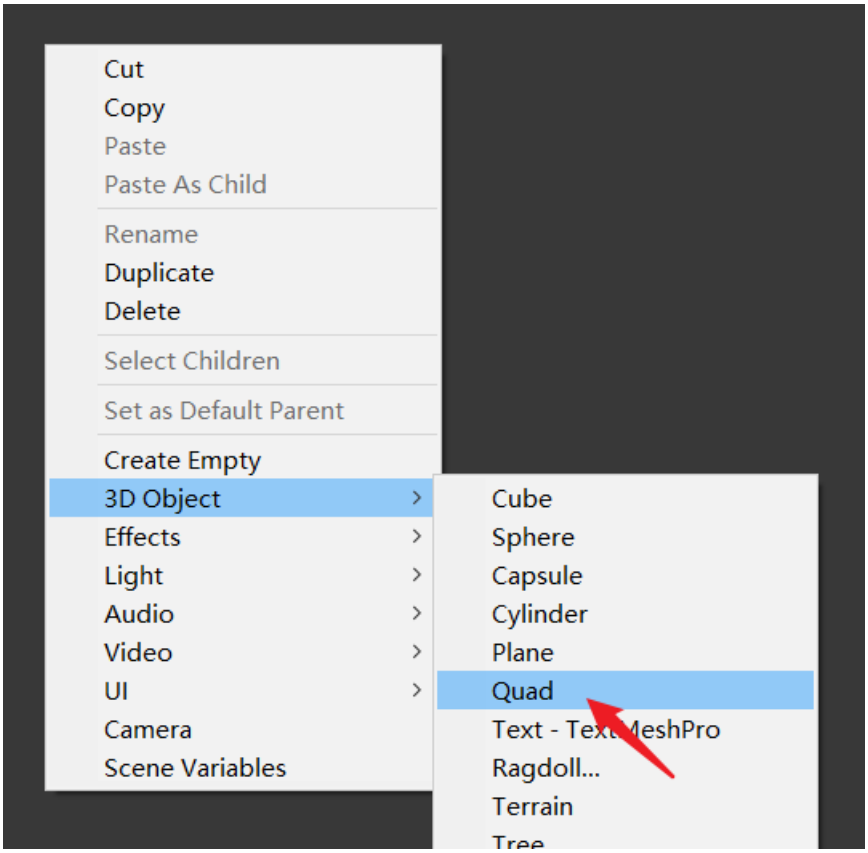
我们早期的彩色电视机显示器就是利用三原色的叠加来显示不同的色彩的，如果你用放大镜仔细观察电视机的屏幕，就会发现画面其实是由很多很多红绿蓝的小色块排列组合而成的，

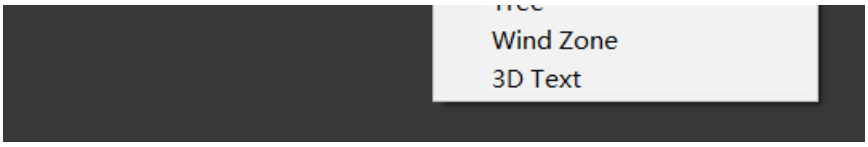




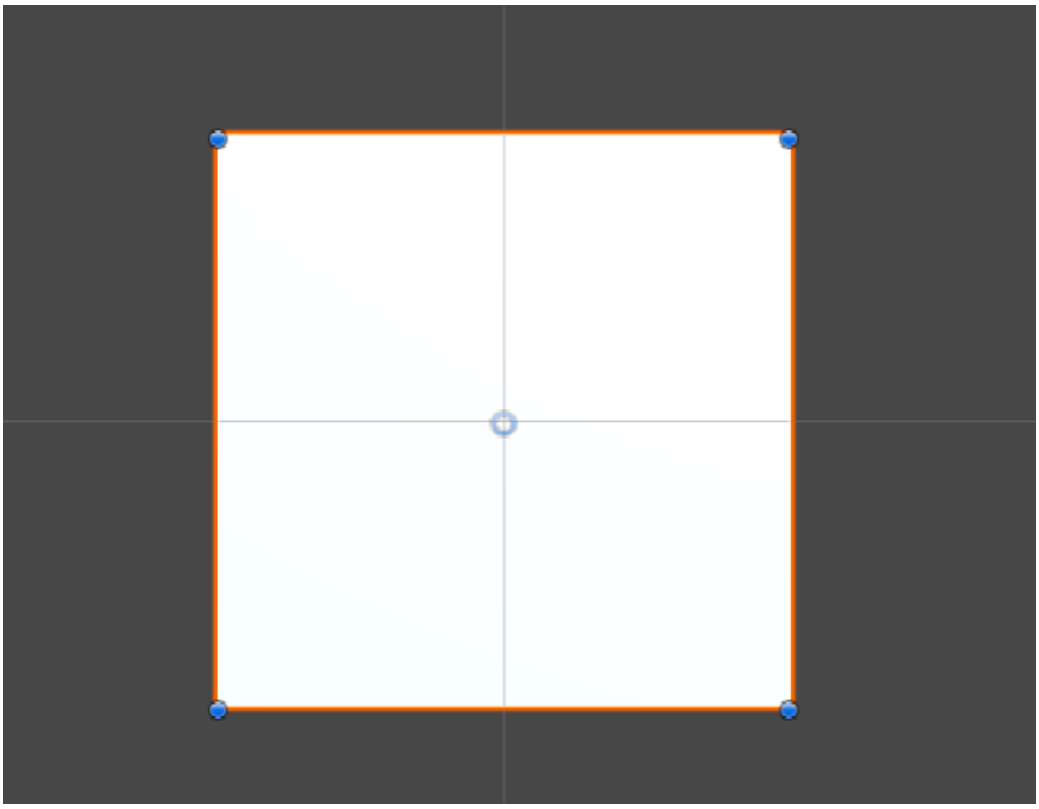
三、Unity制作红绿蓝色块

创建一个`Unity`工程，在`Hierarchy`面板空白处右键鼠标，点击菜单`3D Object / Quad`，创建一个四边形，

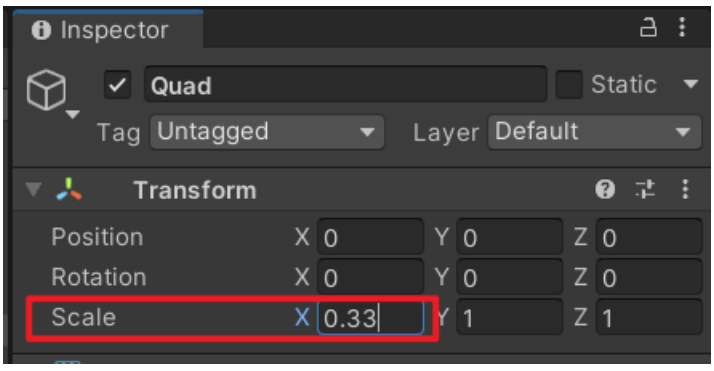




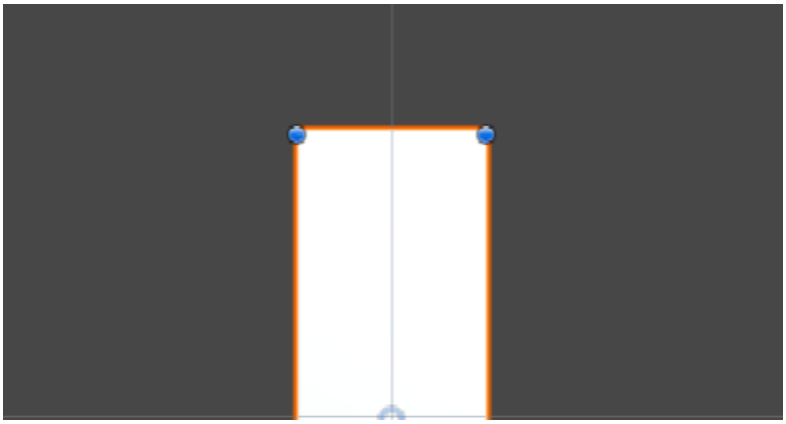
如下：

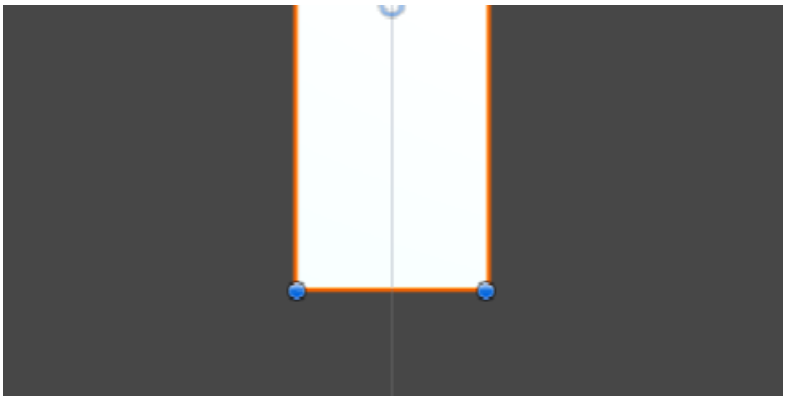


调整四边形的`Scale`的`x`缩放为`0.33`，

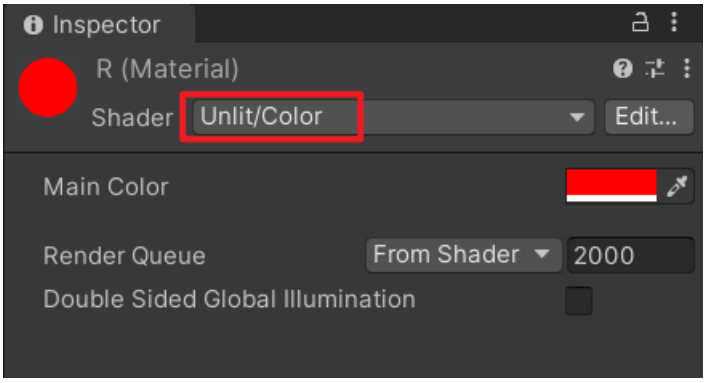


如下，

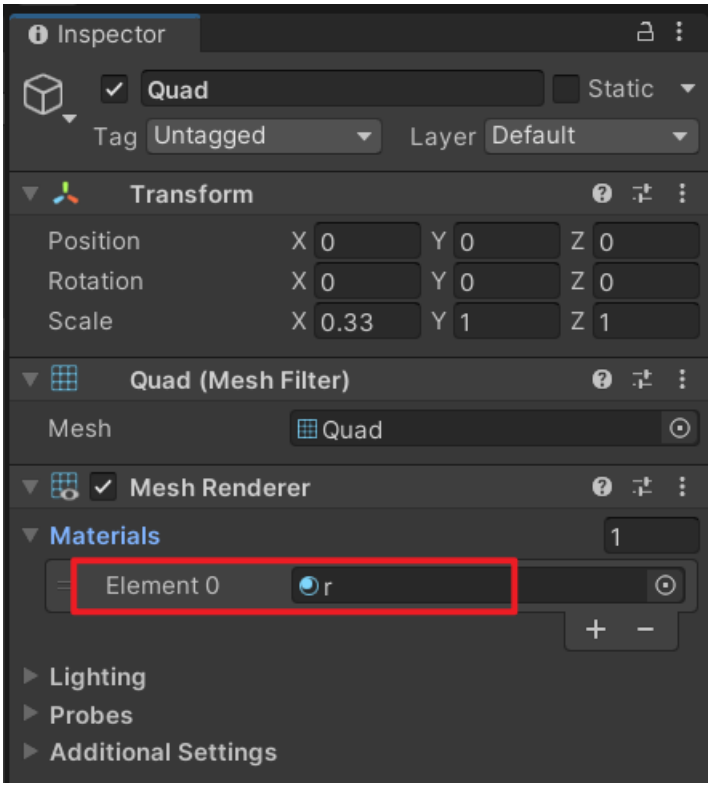




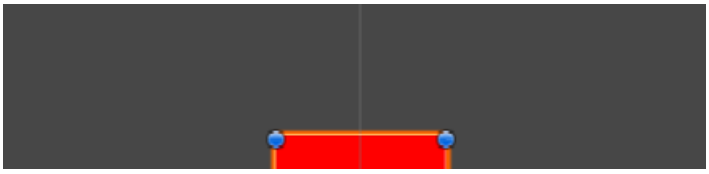
为它创建一个材质球，`shader`使用`Unlit/Color`，颜色设置为红色，

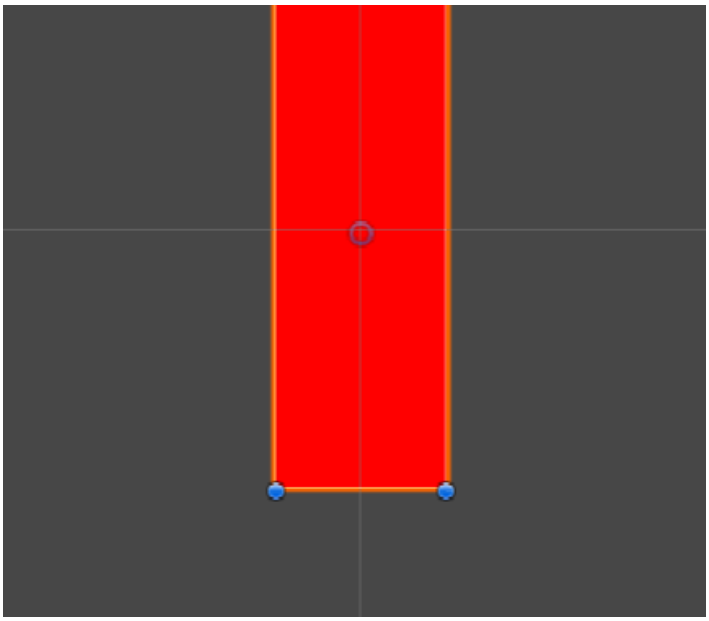


把材质球赋值给四边形，

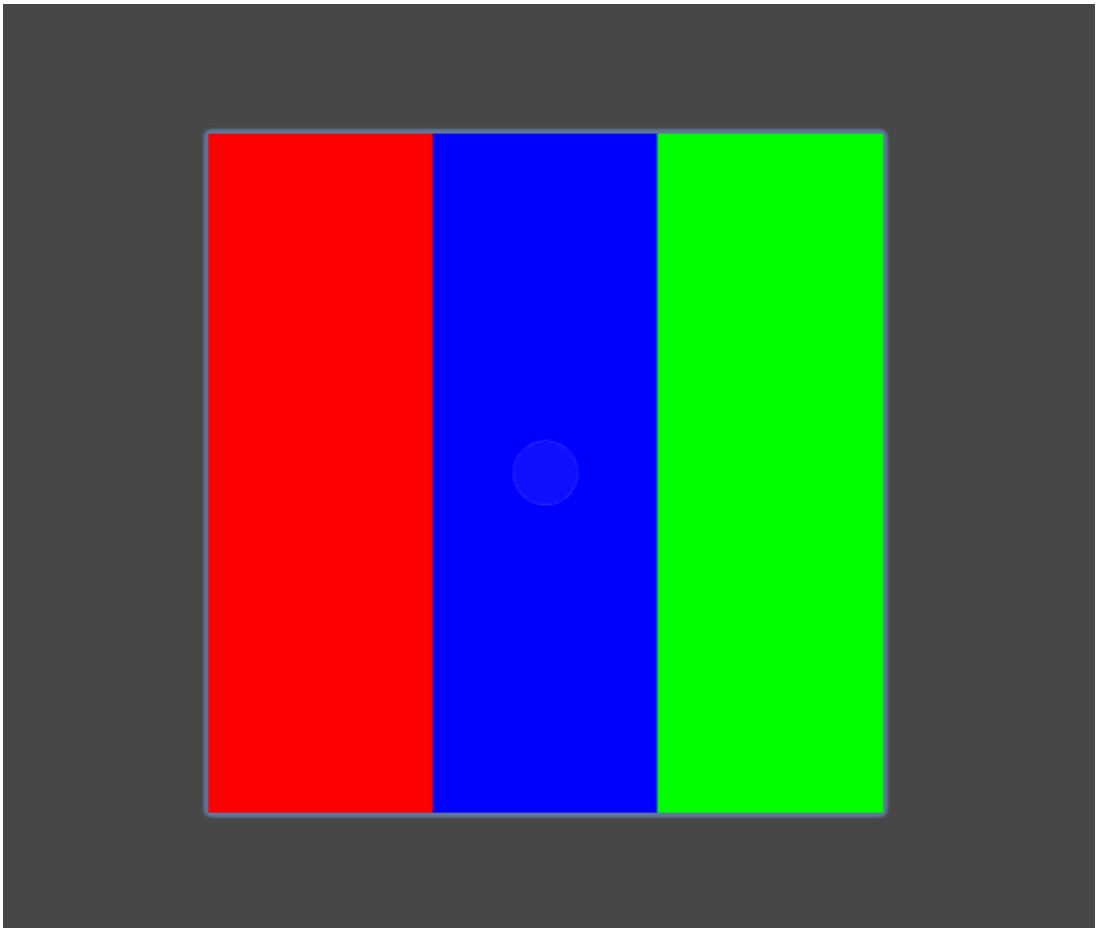


如下

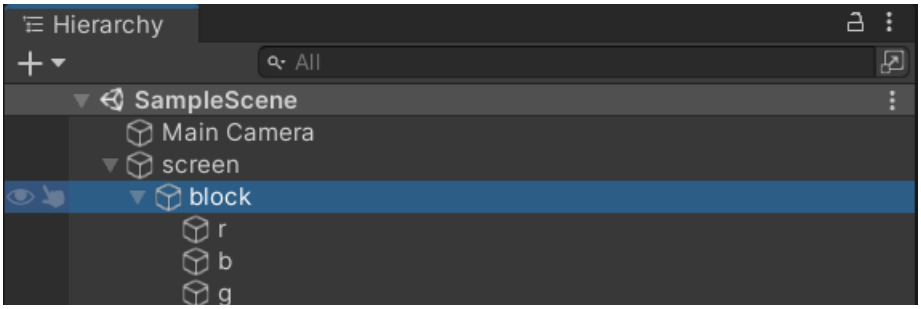




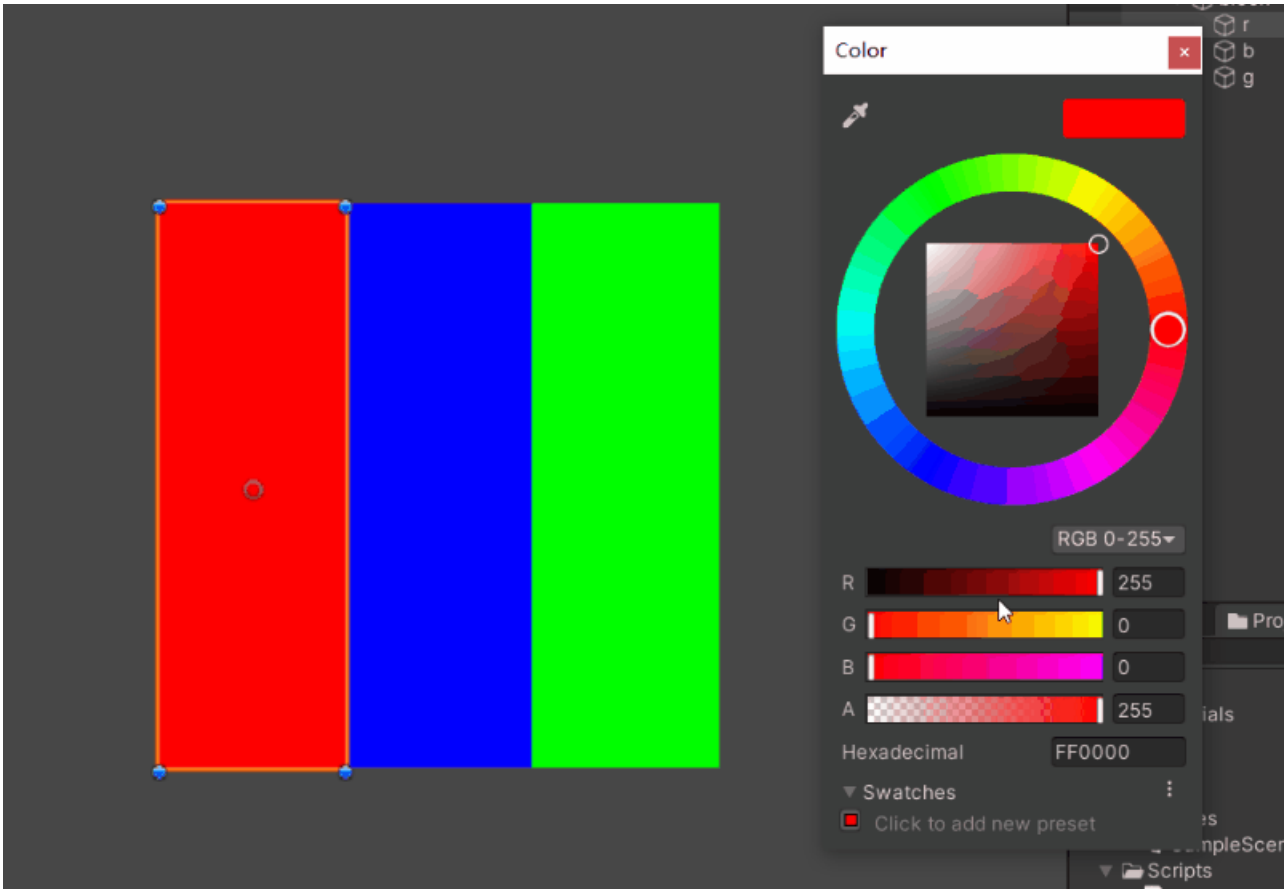
同理，制作蓝色和绿色，排在一起，如下：



层级结构如下，一个`block`就等价于一个像素，一个像素包括`r`、`g`、`b`三个三原色，



可以分别对`r`、`g`、`b`进行调整，



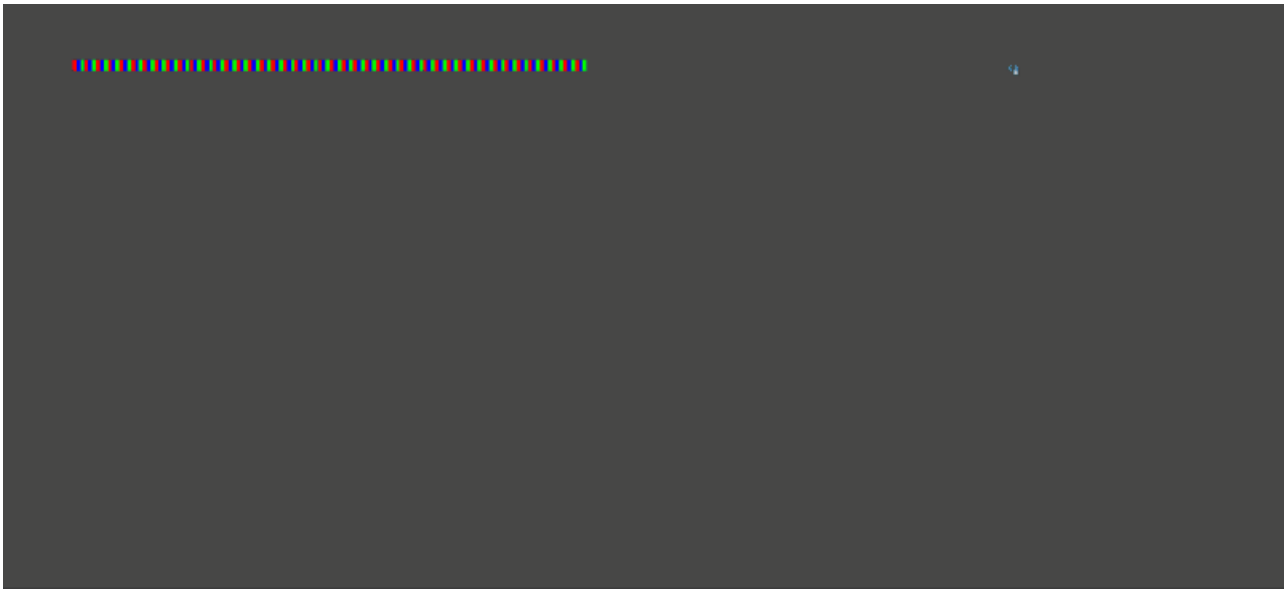
四、克隆像素块

接下来我们只需要进行克隆，排列好，理论上就可以做成一个光学三原色显示器了，

写个函数来执行：

```
// 创建N个像素块，WIDTH是宽，HEIGHT是高
private IEnumerator CreateBlocks()
{
    int counter = 0;
    for (int j = HEIGHT; j >= 0; --j)
    {
        for (int i = 0; i < WIDTH; ++i)
        {
            CreateBlockItem(i, j, Color.white);
            ++counter;
            if (counter > 100)
            {
                counter = 0;
                yield return new WaitForSeconds(0.3f);
            }
        }
    }
}

// 创建一个像素块
private void CreateBlockItem(int i, int j, Color color)
{
    var obj = Instantiate(block);
    var trans = obj.transform;
    trans.SetParent(blockRoot, false);
    trans.localPosition = new Vector3(i, j, 0);
    trans.Find("r").GetComponent<MeshRenderer>().material.color = new Color(color.r, 0, 0);
    trans.Find("g").GetComponent<MeshRenderer>().material.color = new Color(0, color.g, 0);
    trans.Find("b").GetComponent<MeshRenderer>().material.color = new Color(0, 0, color.b);
}
```

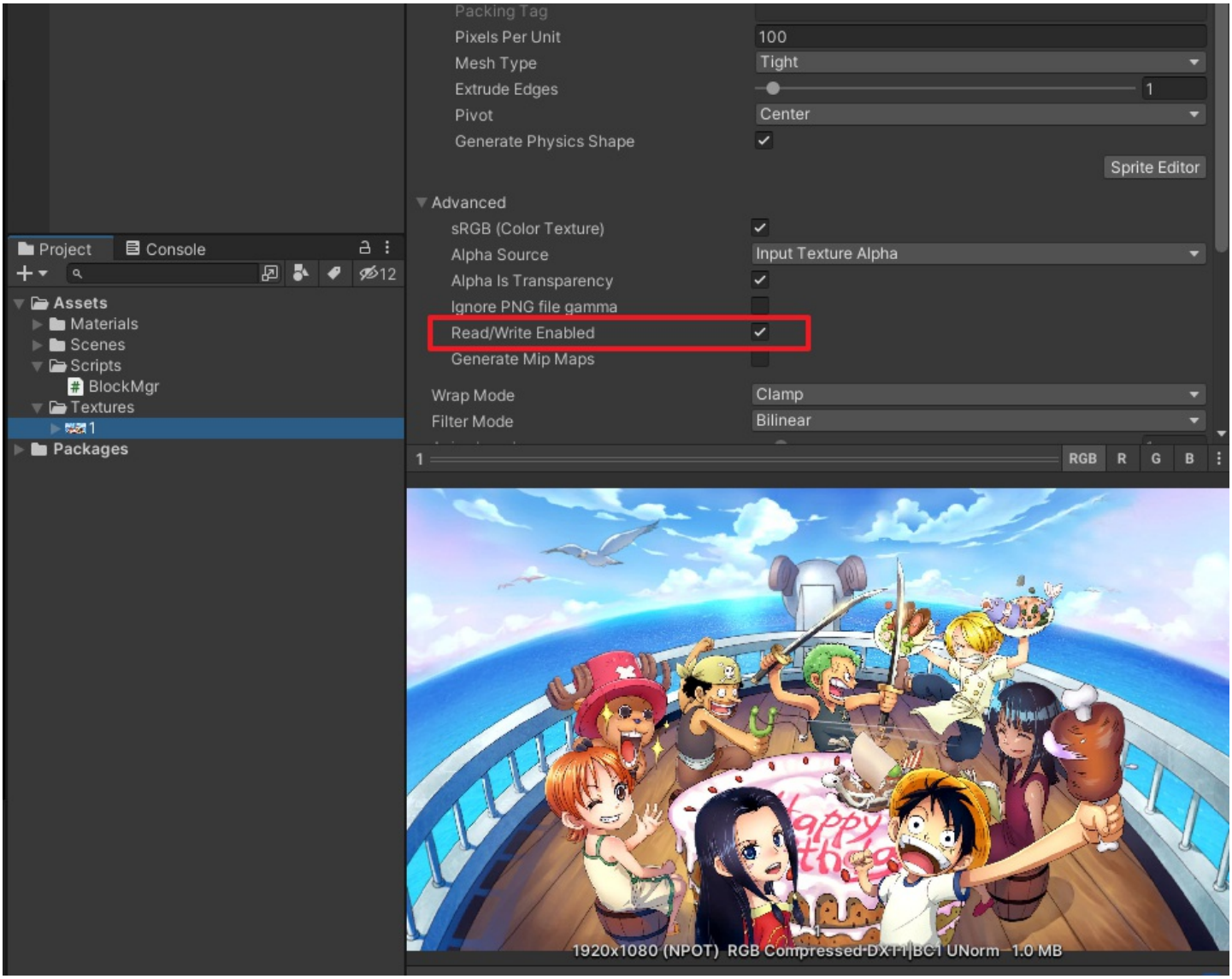
比如创建一个宽`100`，高`50`的屏幕，效果如下：



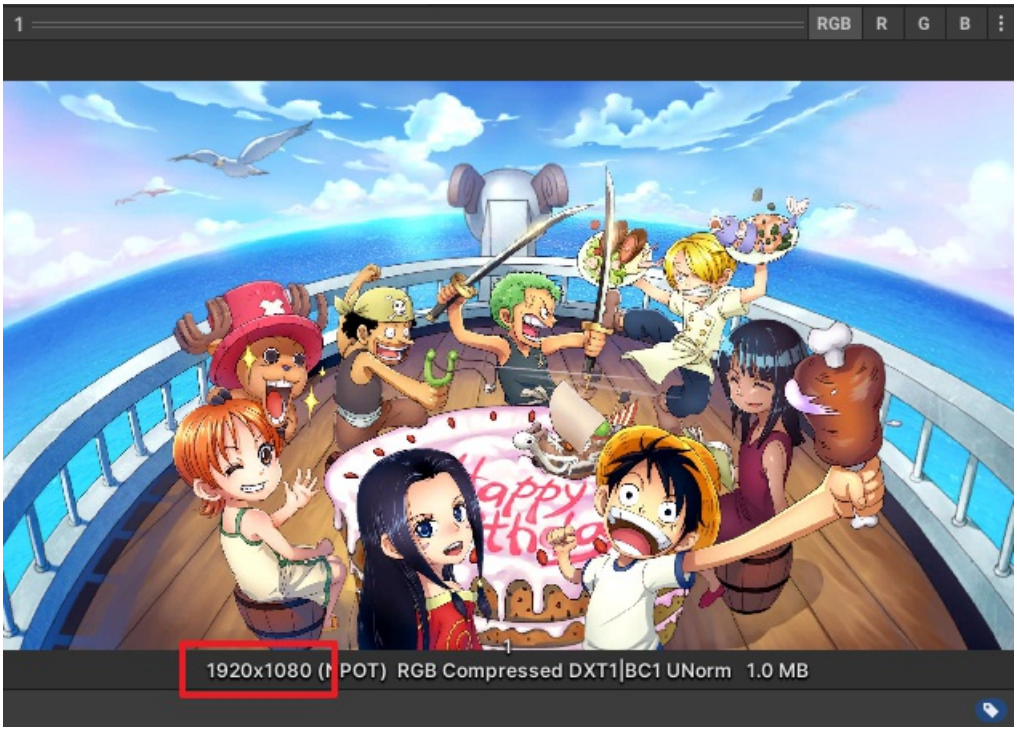
五、显示图像

我们找一张图片，比如海贼王的图片，

导入到`Unity`工程中，勾选图片可读，



我们可以看到图片的尺寸为`1920x1080`，



我们使用`240x135`的分辨率来显示，我们对图像进行采样，可以使用`Texture2D`的`GetPixel`接口，

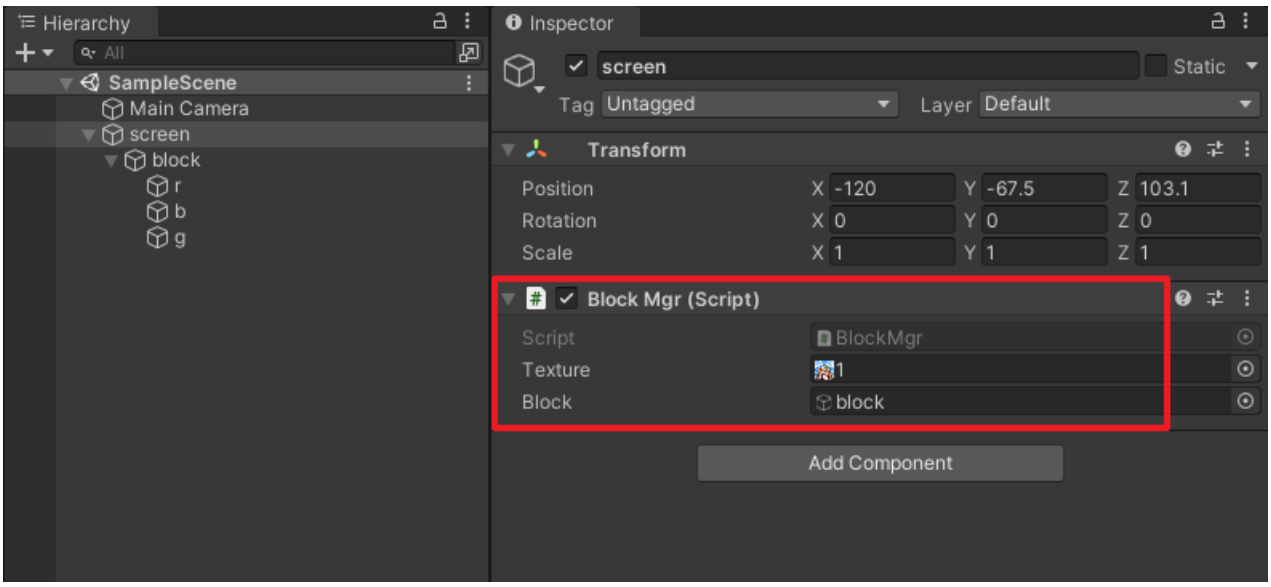
```
public Color GetPixel(int x, int y);
```

完整代码如下：

```
using System.Collections;
using UnityEngine;

public class BlockMgr : MonoBehaviour
{
    /// <summary>
    /// 原图像
    /// </summary>
    public Texture2D texture;
    /// <summary>
    /// 三原色像素块
    /// </summary>
    public GameObject block;
    /// <summary>
    /// 像素块父节点
    /// </summary>
    private Transform blockRoot;
    /// <summary>
    /// 宽
    /// </summary>
    private const int WIDTH = 240;
    /// <summary>
    /// 高
    /// </summary>
    private const int HEIGHT = 135;

    void Start()
    {
        blockRoot = block.transform.parent;
```

```
        StartCoroutine(CreateCreenBlocks());
    }

    private IEnumerator CreateCreenBlocks()
    {
        int counter = 0;
        var width = texture.width;
        var height = texture.height;
        for (int j = HEIGHT; j >= 0; --j)
        {
            for (int i = 0; i < WIDTH; ++i)
            {
                // 对原图像进行采样
                Color color = texture.GetPixel(width * i / WIDTH, height * j / HEIGHT);
                CreateBlockItem(i, j, color);
                ++counter;
                if (counter > 100)
                {
                    counter = 0;
                    yield return null;
                }
            }
        }
    }


    private void CreateBlockItem(int i, int j, Color color)
    {
        var obj = Instantiate(block);
        var trans = obj.transform;
        trans.SetParent(blockRoot, false);
        trans.localPosition = new Vector3(i, j, 0);
        trans.Find("r").GetComponent<MeshRenderer>().material.color = new Color(color.r, 0, 0);
        trans.Find("g").GetComponent<MeshRenderer>().material.color = new Color(0, color.g, 0);
        trans.Find("b").GetComponent<MeshRenderer>().material.color = new Color(0, 0, color.b);
    }
}
```

脚本挂在`screen`节点上，

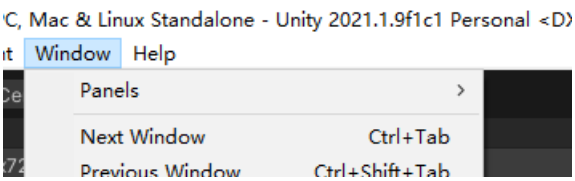


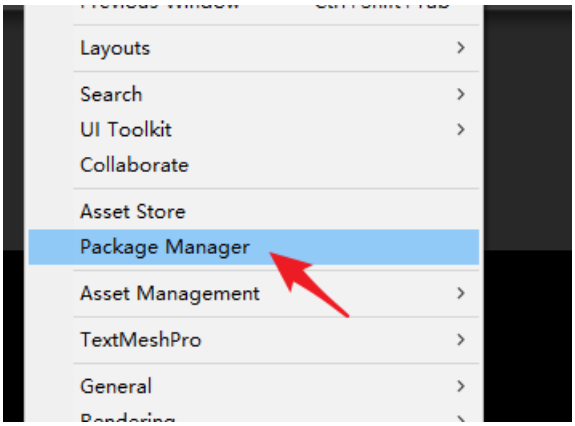
六、测试

运行`Unity`，执行效果如下：

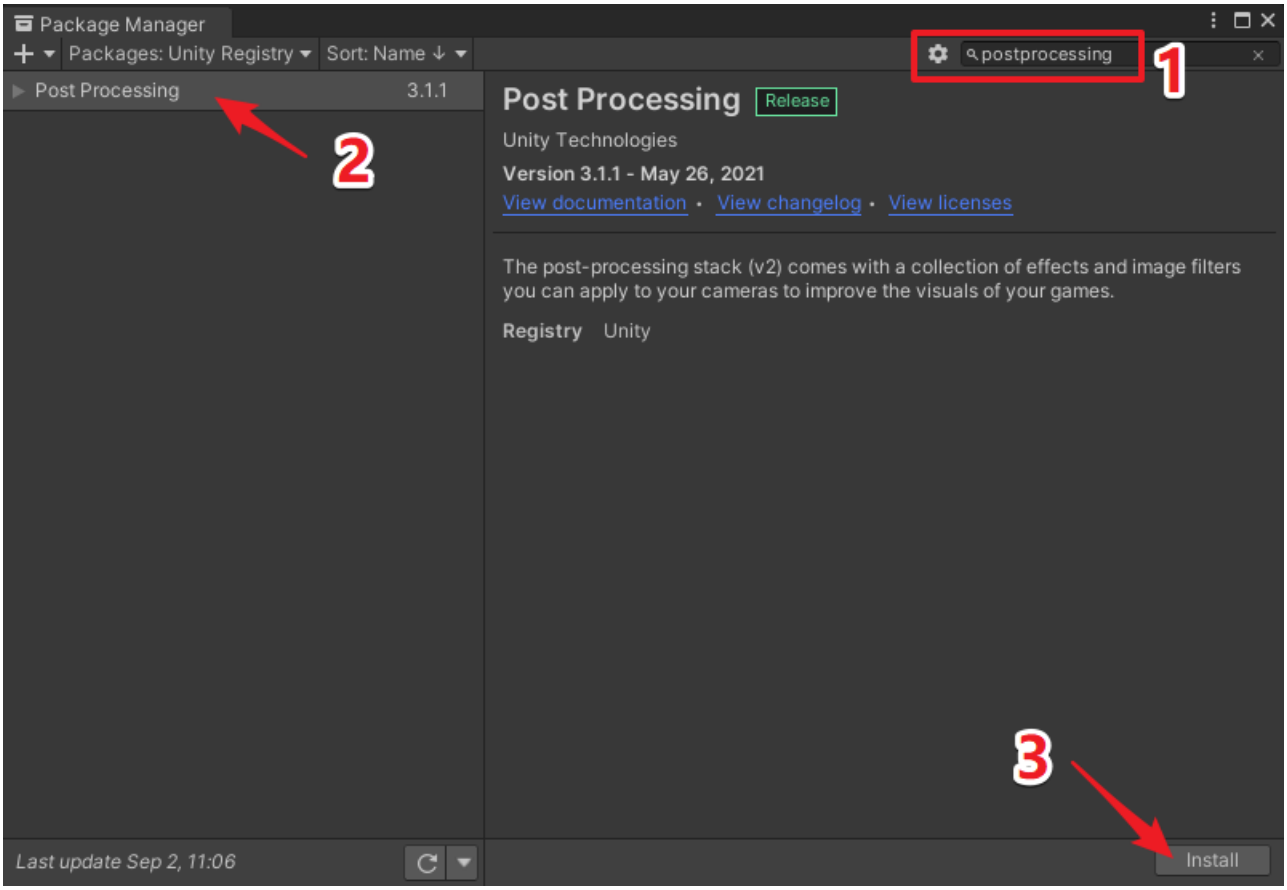
七、屏幕后处理

亮度有点暗，我们加上屏幕后处理，点击菜单`Window / Package Manager`，

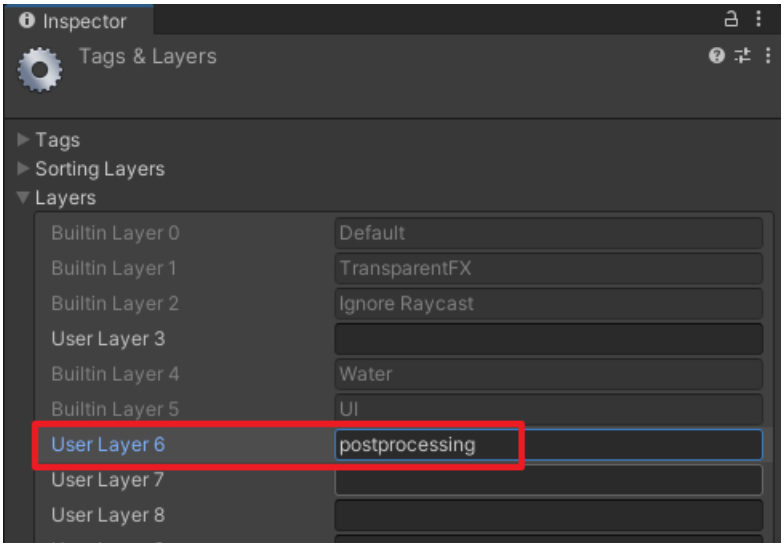




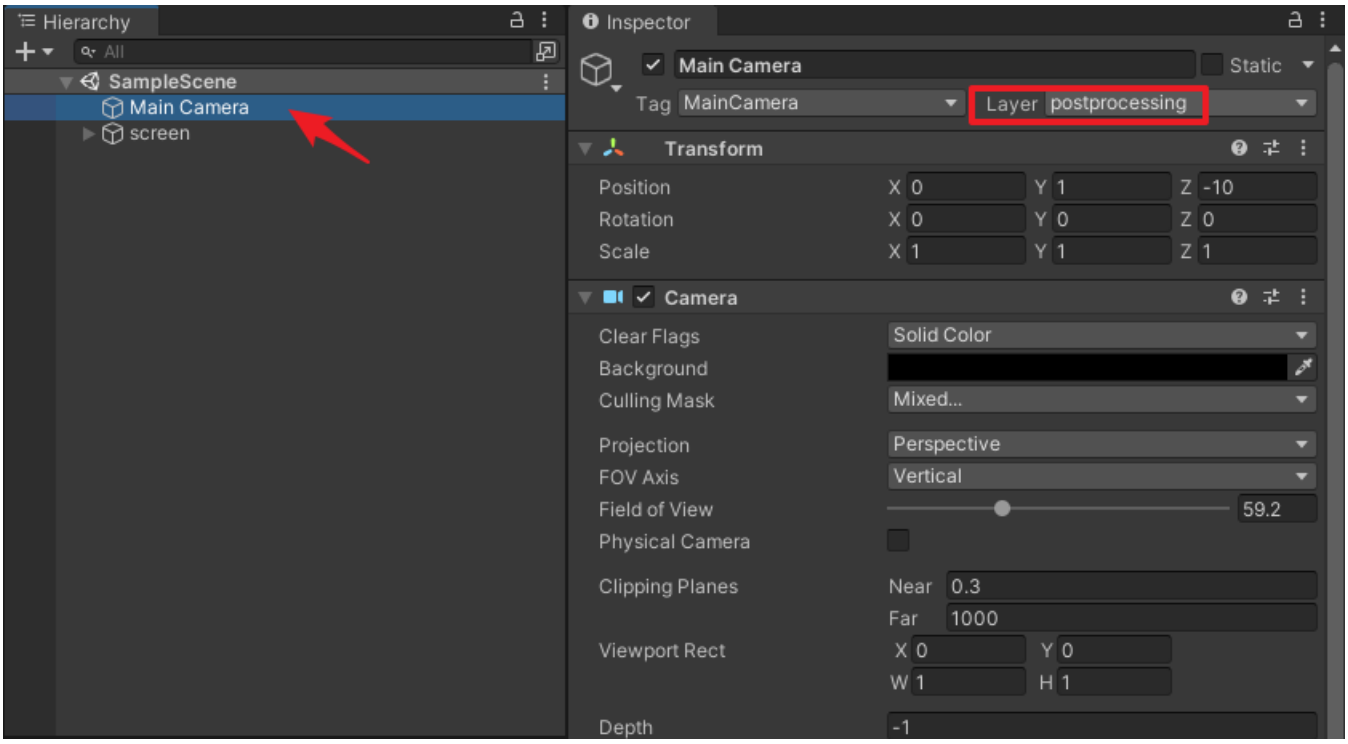
搜索`postprocessing`，选中`Post Processing`，点击`Install`，安装后处理插件。



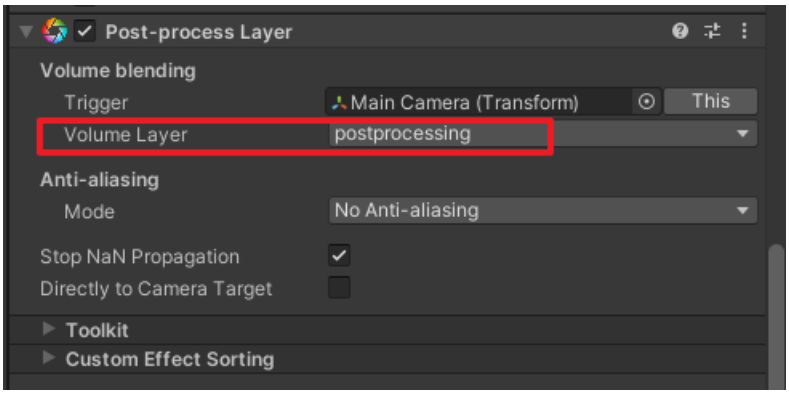
添加一个后处理层`postprocessing`，



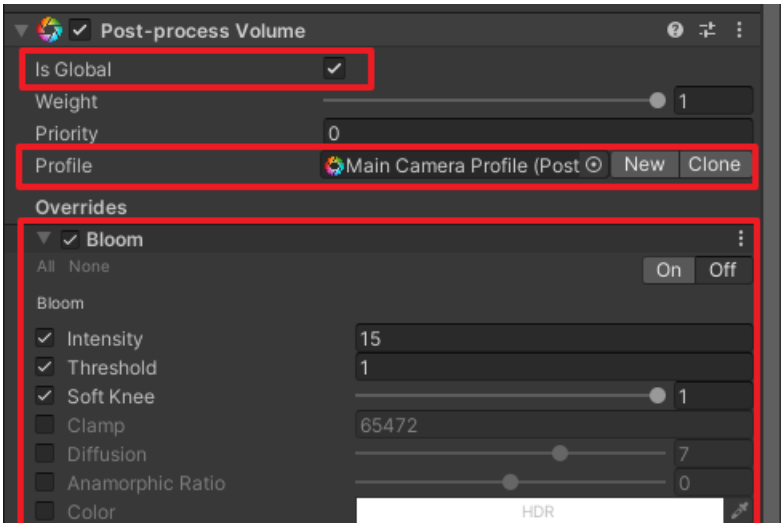
把`Main Camera`的`Layer`改为`postprocessing`，

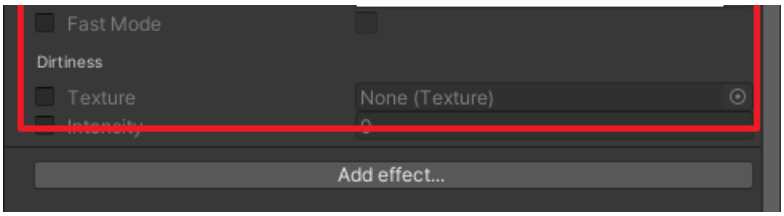


给`Main Camera`添加`Post-process Layer`组件，并设置`Volume Layer`为`postprocessing`层，

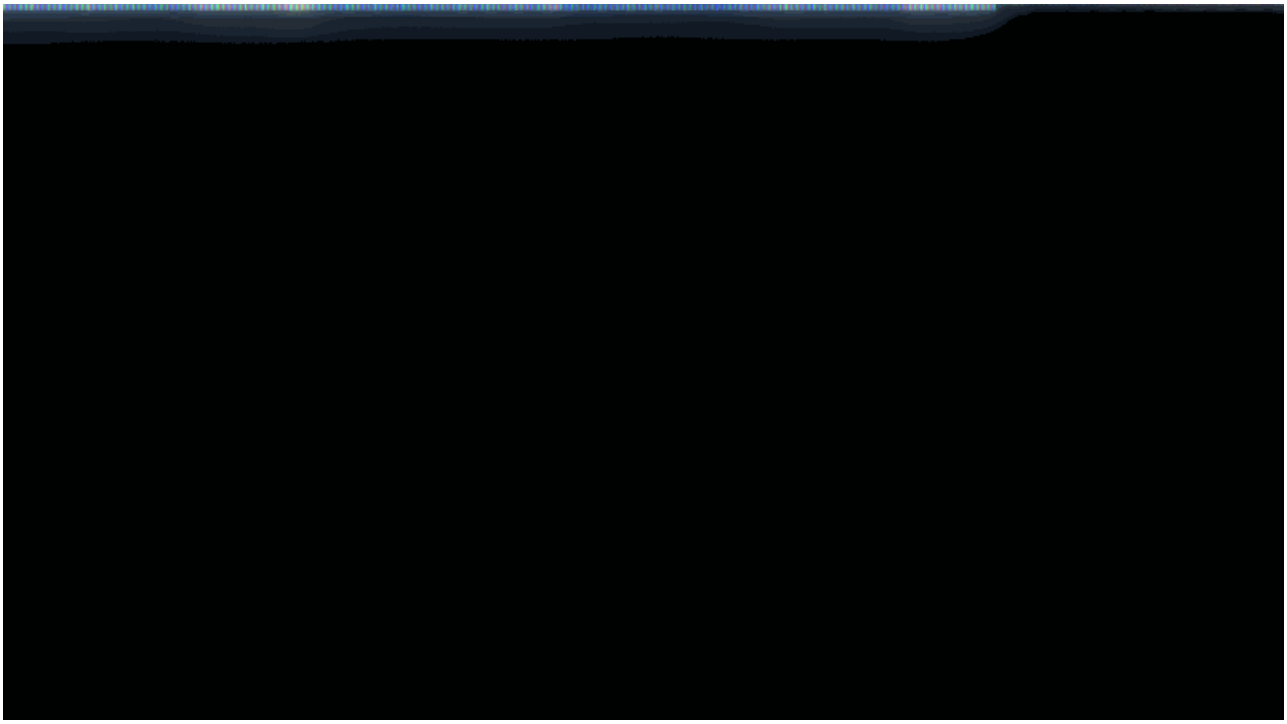


给`Main Camera`添加`Post-process Volume`组件，勾选`Is Global`，创建一个`Profile`配置文件，添加`Bloom`泛光效果，如下：





最终效果，



七、工程源码

本文工程已上传到`CODE CHINA`，刚兴趣的同学可自行下载学习，地址：https://codechina.csdn.net/linxinfa/UnityTVScreenSimulation

注：我使用的`Unity`版本为`Unity 2020.1.14f1c1 (64-bit)`

八、完毕

好了，今天就到这里吧。

我是林新发：https://blog.csdn.net/linxinfa

原创不易，若转载请注明出处，感谢大家~

喜欢我的可以点赞、关注、收藏，如果有什么技术上的疑问，欢迎留言或私信，我们下期见~