Unity3D 小实验：太阳系模拟

原创

七月简语 于 2018-04-04 08:44:53 发布 4859 收藏 18

分类专栏： Unity3D C# Unity3D学习笔记 文章标签： Unity3D C# 脚本

本文链接：https://blog.csdn.net/ShenDW818/article/details/79811994


Unity3D 同时被 3 个专栏收录

11 篇文章 0 订阅

订阅专栏


C#

6 篇文章 0 订阅

订阅专栏

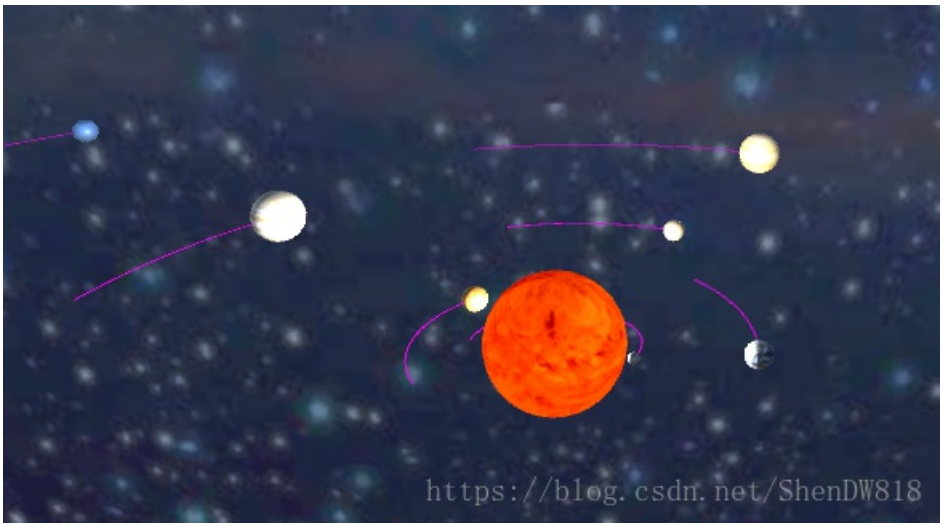

Unity3D学习笔记

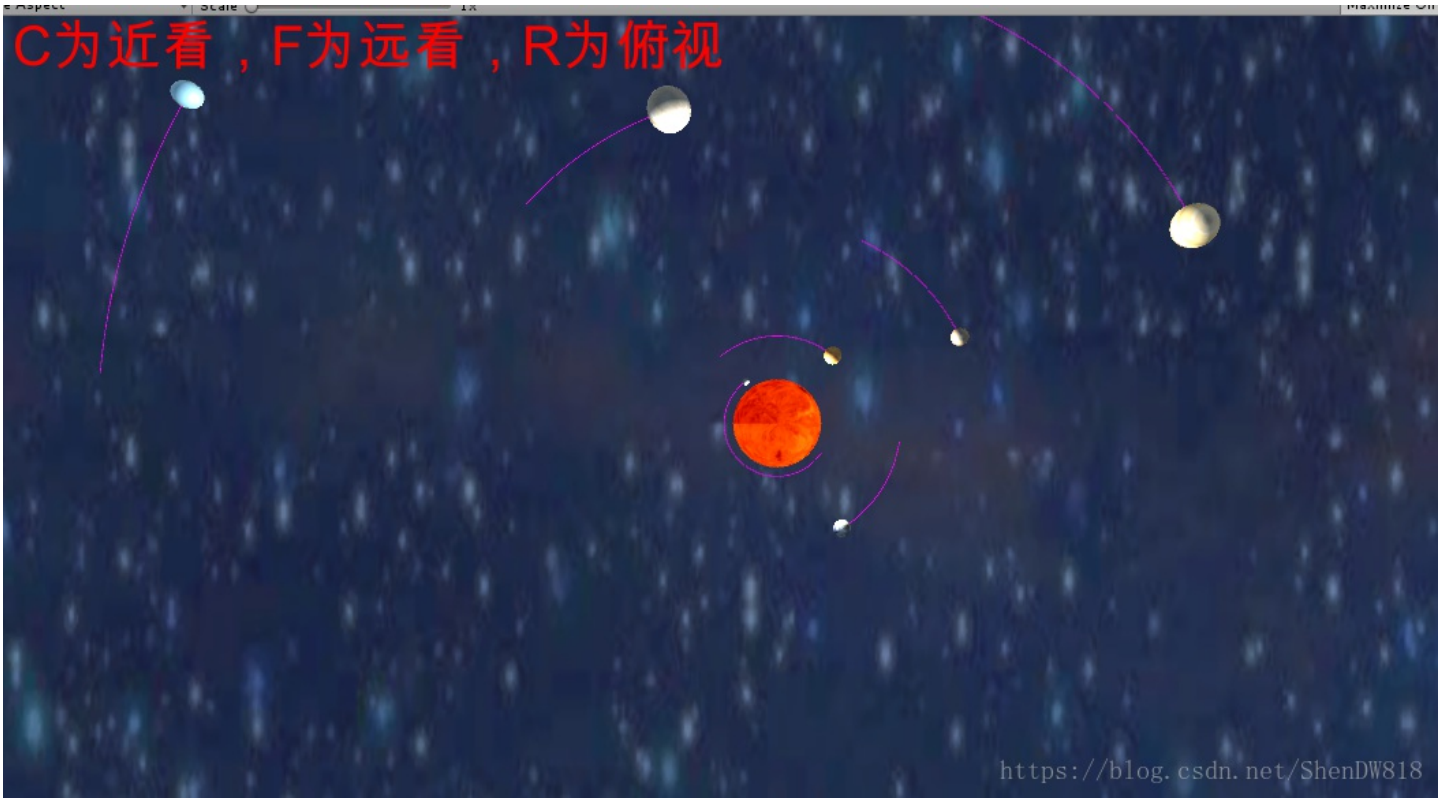
11 篇文章 0 订阅

订阅专栏

太阳系大概是什么样子，大家都知道的，这里就不多说了。太阳系模拟，本来应该是按太阳系中太阳和行星的真实比例来实现的，后来，发现用真实比例，呈现出来的模拟结果很难看（太阳确实太大了，另外，大的行星和小的行星之间的比例相差也太大），所以，这次实验就没有按照真实的数据比例来进行模拟啦。

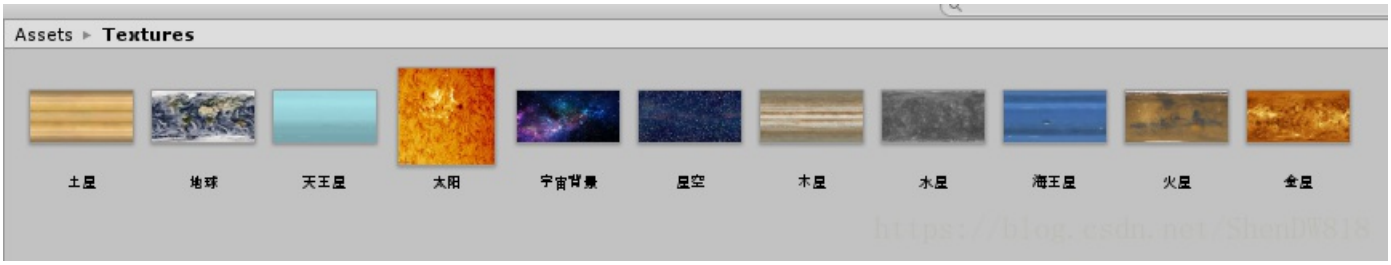
先来看看成品图：


https://blog.csdn.net/ShenDW818

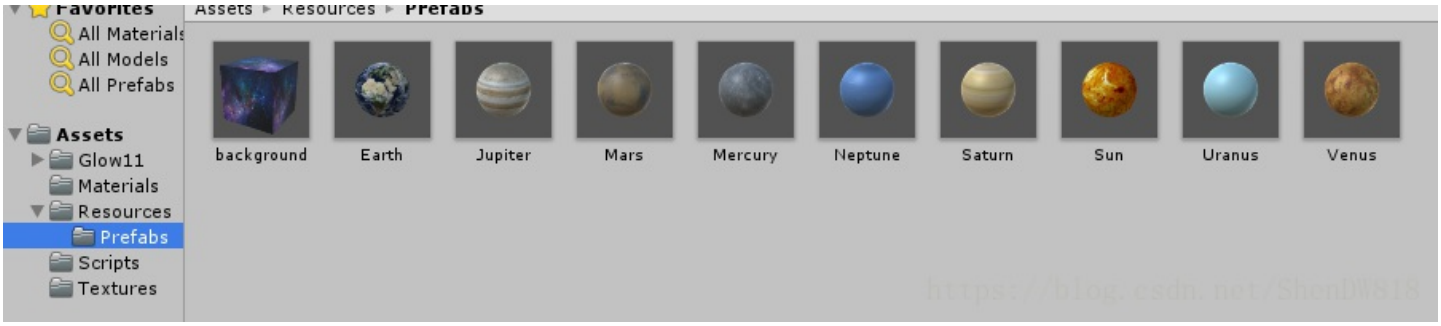

C为近看，F为远看，R为俯视
https://blog.csdn.net/ShenDW818

下面，就看一下怎么弄吧

首先，就先下载好太阳和各行星的贴图：



并且用这些贴图来弄好太阳和各行星的prefabs：



接下来，就是代码实现了。

首先，就是导演类（单例模式，一个游戏只有一个导演）和场景控制接口，这两个就没什么可说的啦。

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

public class SSDirector : System.Object {
    //singlton instance
    private static SSDirector _instance;

    public IScenesController currentScenceController { get; set; }

    public static SSDirector getInstance()
    {
        if(_instance == null)
        {
            _instance = new SSDirector();
        }
        return _instance;
    }

    public int getFPS()
    {
        return Application.targetFrameRate;
    }

    public void setFPS(int fps)
    {
        Application.targetFrameRate = fps;
    }
}
```

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

public interface IScenesController
{
    void LoadResources();
}
```

然后，就是实现整个场景的控制器了。这里就不多说什么，看代码和注释胜过很多语言表达。

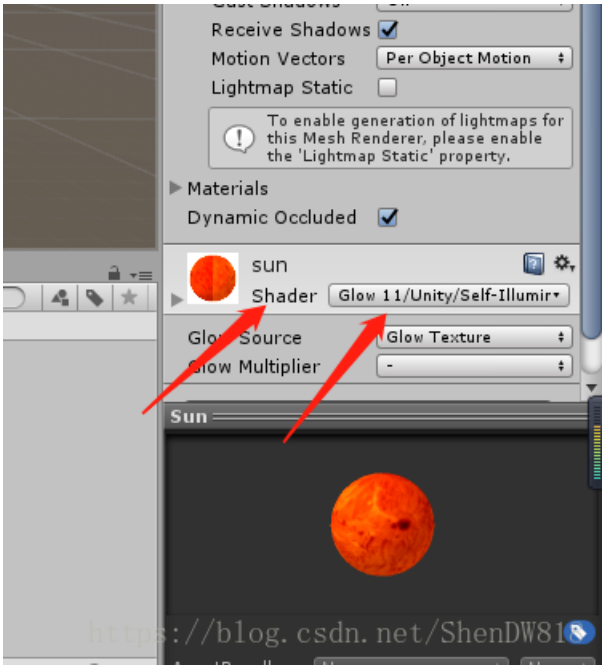
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Controller : MonoBehaviour, IScenesController {
    private Transform[] planets = new Transform[8];
    //行星名字，用于辅助后面资源导入
    private string[] planets_name = { "Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus
    //三个摄影机，用来处理三个不同的拍摄视角
    private GameObject ca_0;
    private GameObject ca_1;
    private GameObject ca_2;

    //行星的初始位置，其实也可以看作是公转半轻比例啦
    Vector3[] position =
    {
        new Vector3(6, 0, 0),
        new Vector3(10, 0, 0),
        new Vector3(14, 0, 0),
        new Vector3(23, 0, 0),
        new Vector3(38, 0, 0),
        new Vector3(53, 0, 0),
        new Vector3(77, 0, 0),
        new Vector3(90, 0, 0)
    };
    //各行星的公转法平面倾角
    Vector3[] includedAngle = {
        new Vector3 (0, 1, Mathf.Tan(Mathf.PI/180*7)),
        new Vector3 (0, 1, Mathf.Tan(Mathf.PI/180*3.4f)),
        new Vector3 (0, 1, 0),
        new Vector3 (0, 1, Mathf.Tan(Mathf.PI/180*1.9f)),
        new Vector3 (0, 1, Mathf.Tan(Mathf.PI/180*1.3f)),
        new Vector3 (0, 1, Mathf.Tan(Mathf.PI/180*2.5f)),
        new Vector3 (0, 1, Mathf.Tan(Mathf.PI/180*0.8f)),
        new Vector3 (0, 1, Mathf.Tan(Mathf.PI/180*1.8f)),
    };
    //公转速率和自转速率，公转的胡乱编的，自转倒是比较真实
    float[] gongZhuan = { 40.0f, 16.0f, 10.0f, 8.0f, 6.0f, 8.0f, 5.0f, 6.0f };
    float[] ziZhuan = { 6.1f, -1.5f, 360f, 350f, 878f, 844.5f, -500.5f, 540.5f };

    Transform Sun;//太阳组件，作为所有行星的父亲
    //两个背景，至于为什么两个，就为了多个视角，然后，技术不够，只能用这种方法来实现背景
    Transform background1;
    Transform background2;
    //导演的设定
    void Awake()
    {
        SSDirector director = SSDirector.getInstance();
```

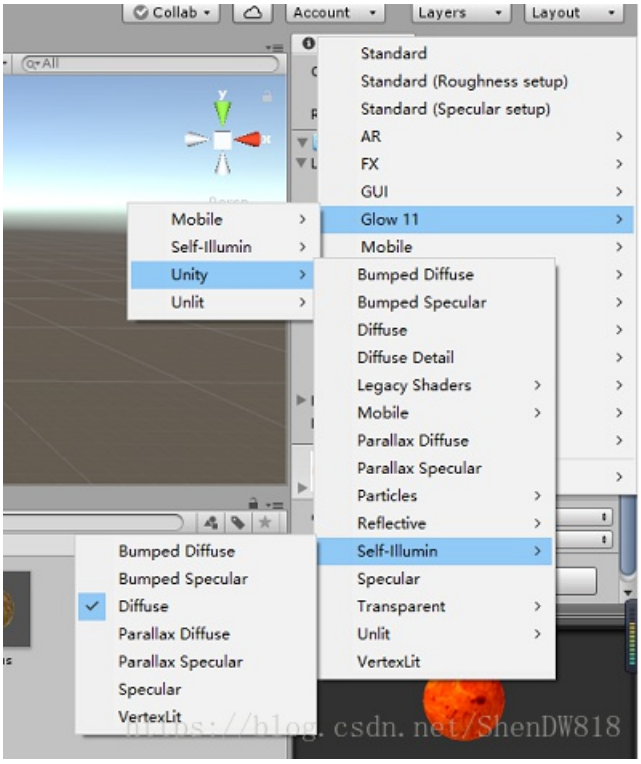
```
        SSDirector director = SSDirector.getInstance();
        director.setFPS(60);
        director.currentScenceController = this;
        director.currentScenceController.LoadResources();

    }
    //生成transform组件的辅助函数
    Transform createTransform(Vector3 position, string name)
    {
        Transform transform = Instantiate<Transform>(Resources.Load<Transform>("prefabs/" + name), posi
        return transform;
    }
    //资源导入
    public void LoadResources()
    {   //导入太阳
        Sun = createTransform(Vector3.zero, "Sun");
        Sun.name = "Sun";
        //导入背景
        background1 = createTransform(new Vector3(0, -100, 500), "background");
        background1.name = "Background";
        background1.localScale += new Vector3(2000, 1000, 100);

        background2 = createTransform(new Vector3(0, -500, 0), "background");
        background2.name = "tBackground";
        background2.localScale += new Vector3(2000, 2, 2000);
        //导入各个行星
        for (int i = 0; i < 8; i++)
        {
            planets[i] = createTransform(position[i], planets_name[i]);
            planets[i].transform.parent = Sun.transform;
            planets[i].name = planets_name[i];
            //用来加尾巴的，不过，有一些属性我试过改了，显现不出来，就干脆，原始这样子吧
            TrailRenderer trail = planets[i].gameObject.AddComponent<TrailRenderer>();
            trail.startWidth = 0.1f;
        }
    }

    // 设置初始化视角为远看
    void Start () {
        ca_0 = GameObject.Find("Camera");
        ca_0.SetActive(true);
        ca_1 = GameObject.Find("Camera1");
        ca_1.SetActive(false);
        ca_2 = GameObject.Find("Camera2");
        ca_2.SetActive(false);
    }

    // Update is called once per frame
    void Update () {
        //每个行星的自转和公转处理
        for(int i = 0; i < 8; i++)
        {
            planets[i].RotateAround(Sun.position, includedAngle[i], gongZhuan[i] * Time.deltaTime);
            planets[i].Rotate(Vector3.up * ziZhuan[i] * Time.deltaTime);
        }
        //获取键盘输入来得到观察的三个视角，C是近处，F是远处，R是俯视
        if(Input.GetKeyDown(KeyCode.C))
        {
            ca_0.SetActive(false);
            ca_1.SetActive(true);
```

```
                ca_2.SetActive(false);
            }
            if(Input.GetKeyDown(KeyCode.F))
            {
                ca_1.SetActive(false);
                ca_0.SetActive(true);
                ca_2.SetActive(false);
            }
            if(Input.GetKeyDown(KeyCode.R))
            {
                ca_2.SetActive(true);
                ca_1.SetActive(false);
                ca_0.SetActive(false);
            }
        }
        //为使用添加一点说明
        void OnGUI()
        {
            GUIStyle fStyle = new GUIStyle();
            fStyle.fontSize = 40;
            fStyle.normal.textColor = Color.red;//红色比较容易看清
            GUI.Label(new Rect(0, 0, 200, 50), "视角：C为近看，F为远看，R为俯视", fStyle);
        }
    }
```

代码实现完之后，也得对场景做一下处理，也很简单，就是把一个空对象和三个摄像头加上去，调整好它们的位置就好了。这里的话，为了观察方便，我三个摄像头都加了一个平行光，将光的强度调到了0.1。然后，在太阳的位置（就是Vector（0，0，0））上，增加一个点光，模拟太阳发出来的光线。

最后，由于点光源不会把太阳照亮，感觉看起来太阳比其他行星还暗，为了解决这个问题，我去下载了一个Glow11的package，然后将这个package导入场景中。然后，在太阳prefab上，改一下Shader属性就可以了。





之后，太阳看起来就是发亮的样子了。至此，这个太阳系就模拟地差不多了。