

Unity 模拟太阳系 民间教学

目录

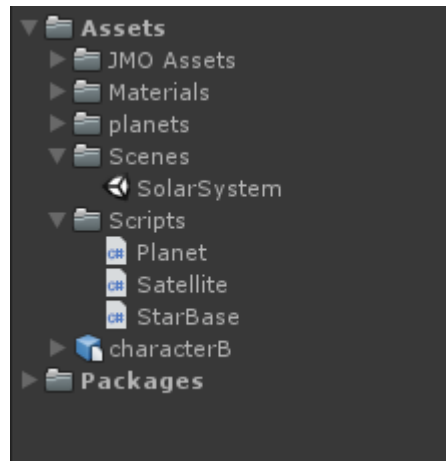
前言.....	2
场景：创建太阳系.....	3
1.1 场景描述	3
1.2 运行场景描述	3
1.3 步骤（含代码解释）	3
1.3.1 创建 10 个球体	3
1.3.2 下载相应的星球的材质	4
1.3.3 导入下载的 Textures.....	6
1.3.4 纹理贴图	6
1.3.5 添加月球和星环	8
1.3.6 调整星球比例	17
1.3.7 关闭平行光源，添加点光源	18
1.3.8 添加粒子效果，让太阳动态发光	21
1.3.9 编写脚本，添加运动信息	22
1.3.10 挂载脚本，设置之前得到的各种参数	39
1.4 场景 最终效果	50

前言

欢迎阅读本教程，所有需要的模型和资源，博主已经制作并打包上传到 GitHub 上。

内容包括：

- 博主自己制作的星环
- 收集的各种行星和星环贴图
- 编写好的脚本



提供资源的地址：

<https://github.com/youhengchan/Unity/tree/master/SolarSystemAssets>

（可直接扫码获取）

博主使用 Unity 版本 2018.3.14.f1

我的博客地址：<https://blog.csdn.net/chenganxuan1999>

我的 github 地址：<https://github.com/youhengchan/>

本文档也可以在 GitHub 上获取：<https://github.com/youhengchan/Unity>

热爱分享，拥抱开源，欢迎留言交流

HNU 陈汉轩

Version: 2019/10/3

Beta 1.0.0

场景：创建太阳系

1.1 场景描述

建立太阳系的模型，实现月球绕地球旋转，行星绕太阳旋转

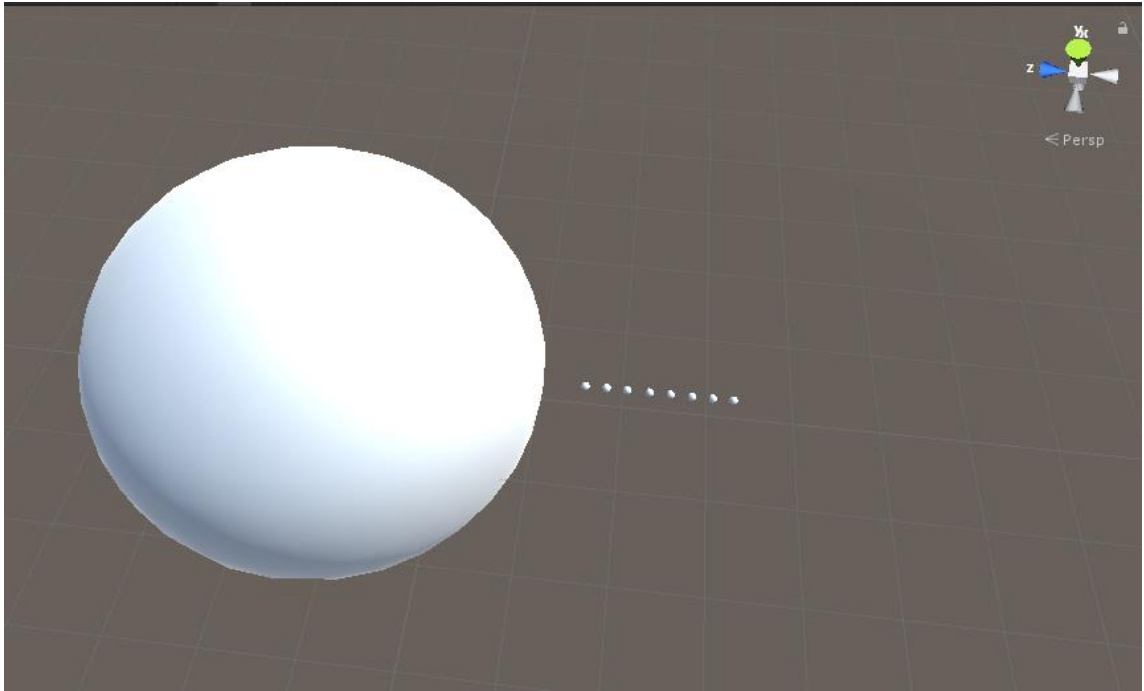
1.2 运行场景描述

太阳位于中央，四周的星球以不同的角速度和不同的运行轨道绕太阳运动。

1.3 步骤（含代码解释）

1.3.1 创建 10 个球体

将左边第一个的 `scale` 设置较大，表示为太阳，剩余的九个球体分别代表太阳系中的 8 大行星和月球。



1.3.2 下载相应的星球的材质

Assets Store 里面免费的资源中没有覆盖所有行星的，全套的都是 10\$起步：



MSGDI
Planets
(not enough ratings)
\$10



NOVA SHADE
Planets
★★★★☆ (32)
\$10



-50%
FORGE3D
Planets
★★★★★ (108)
\$22.50 ~~\$44.99~~



DYLAN AUTY
Simple Planet Generator - Proce.
★★★★★ (4)
\$9.99



MELOWN TECHNOLOGIES SE
VTS Landscape Streaming Plug..
★★★★★ (3)
FREE



EBAL STUDIOS
Star Sparrow Modular Spaceshi..
★★★★★ (138)
FREE



BRETT GREGORY
2D Space Kit
★★★★★ (11)
FREE



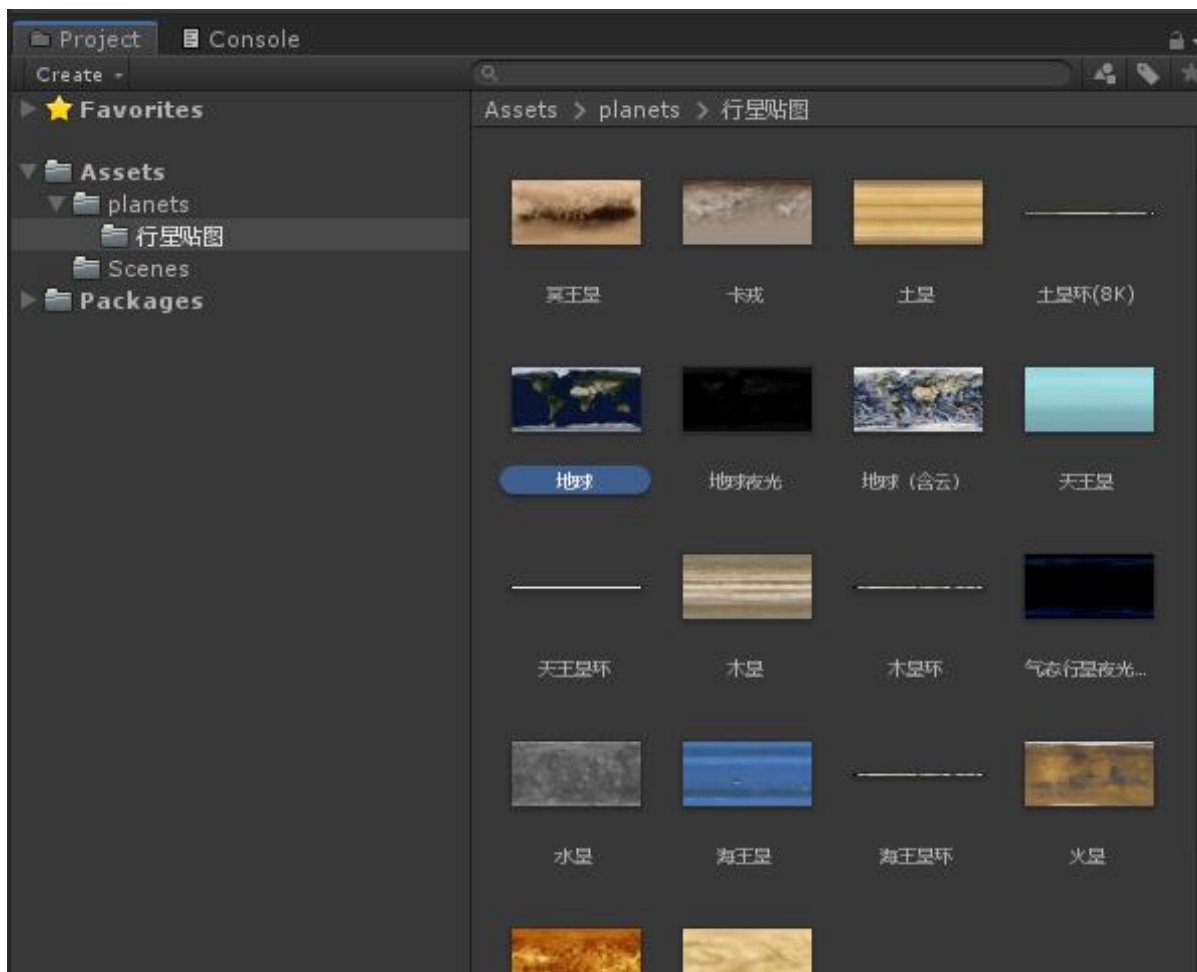
SPACEINVADERSTEAM
Simple Kepler Orbits
★★★★★ (18)
FREE

在百度上查找：

百度贴吧吧友给出了他们自己制作的贴图：

https://tieba.baidu.com/p/4876471245?red_tag=2938874589 (已经打包上传到 Github)

1.3.3 导入下载的 Textures

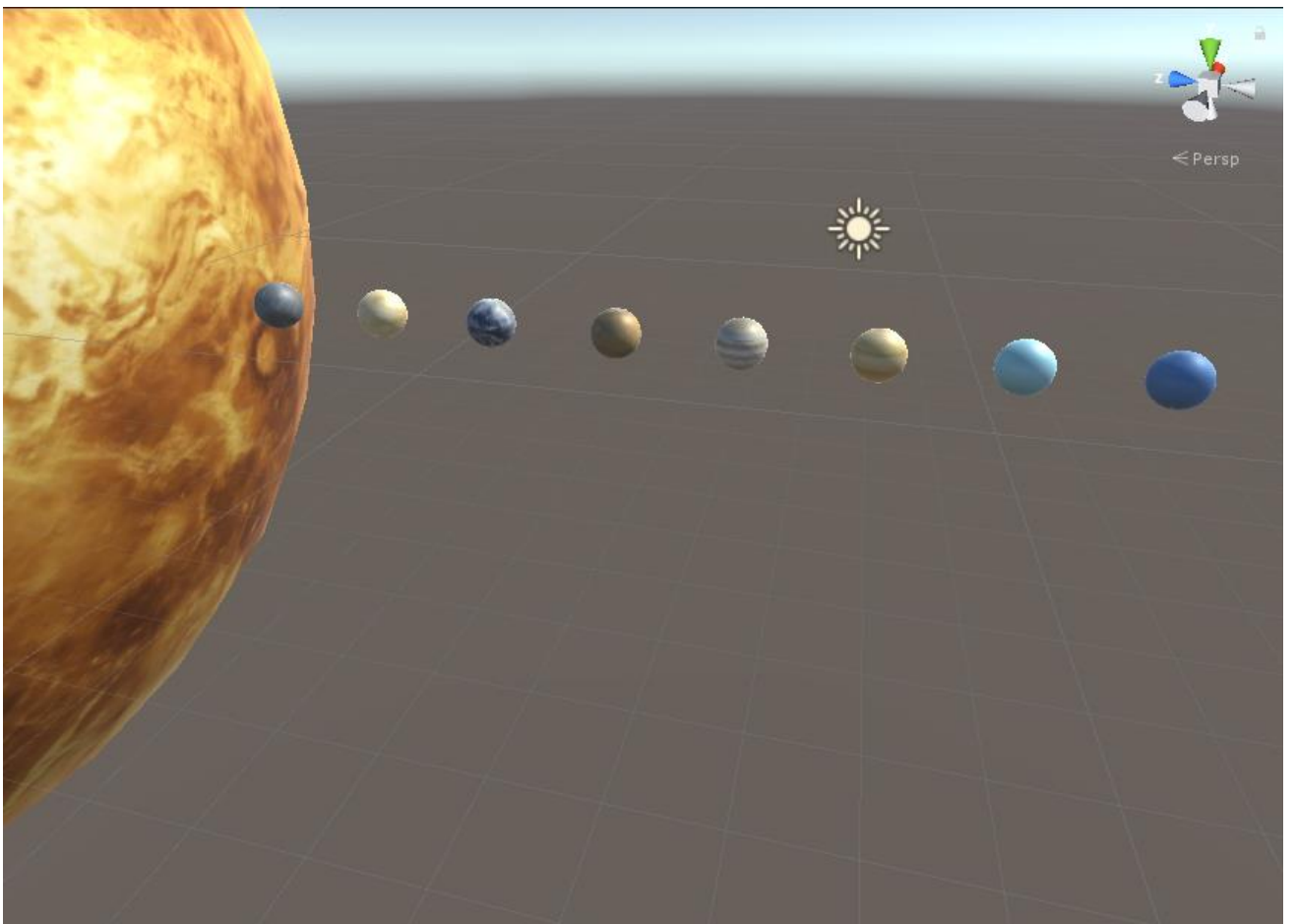
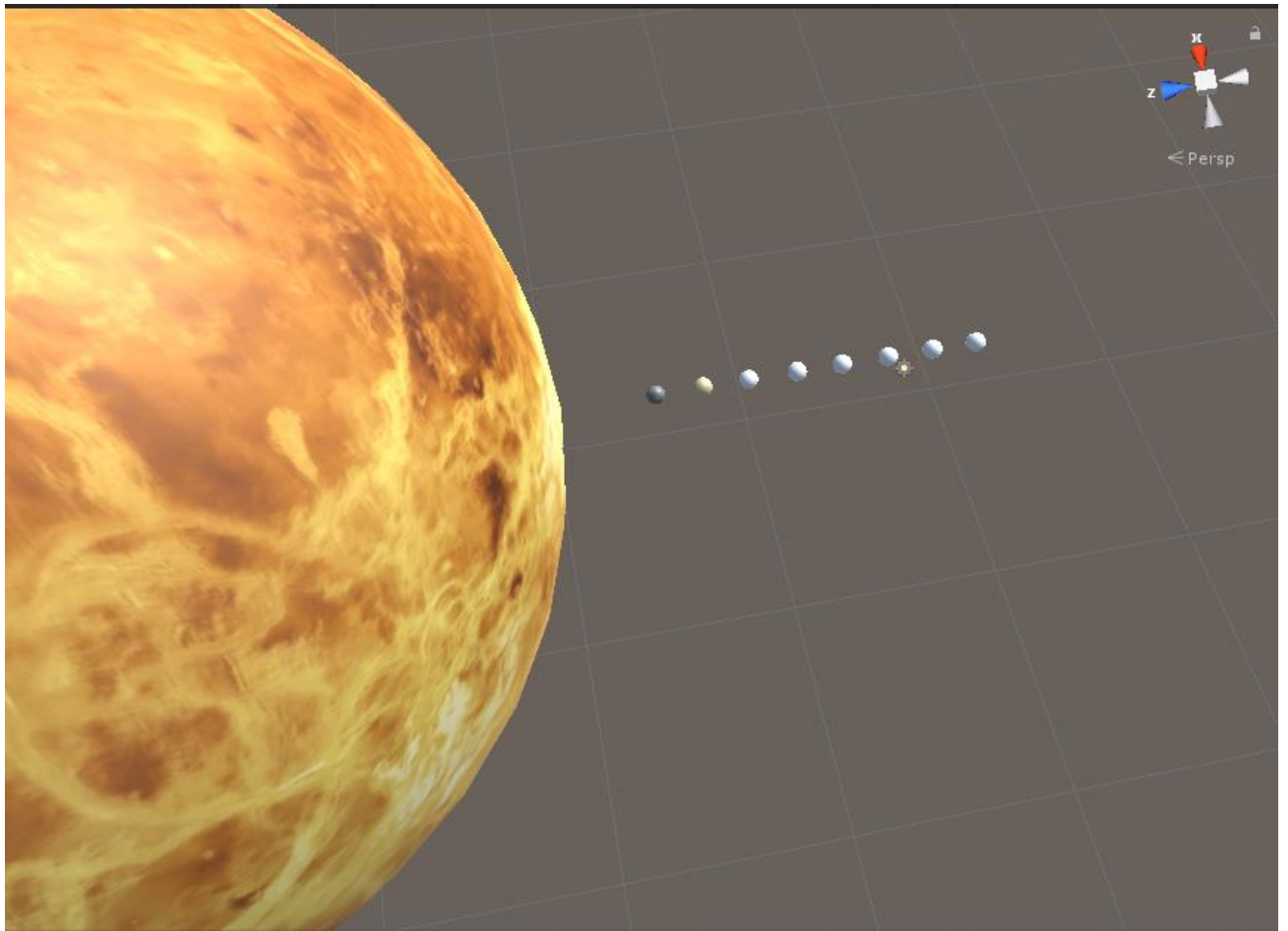


1.3.4 纹理贴图

按照太阳系中的顺序（从靠近太阳到远离太阳）进行贴图：

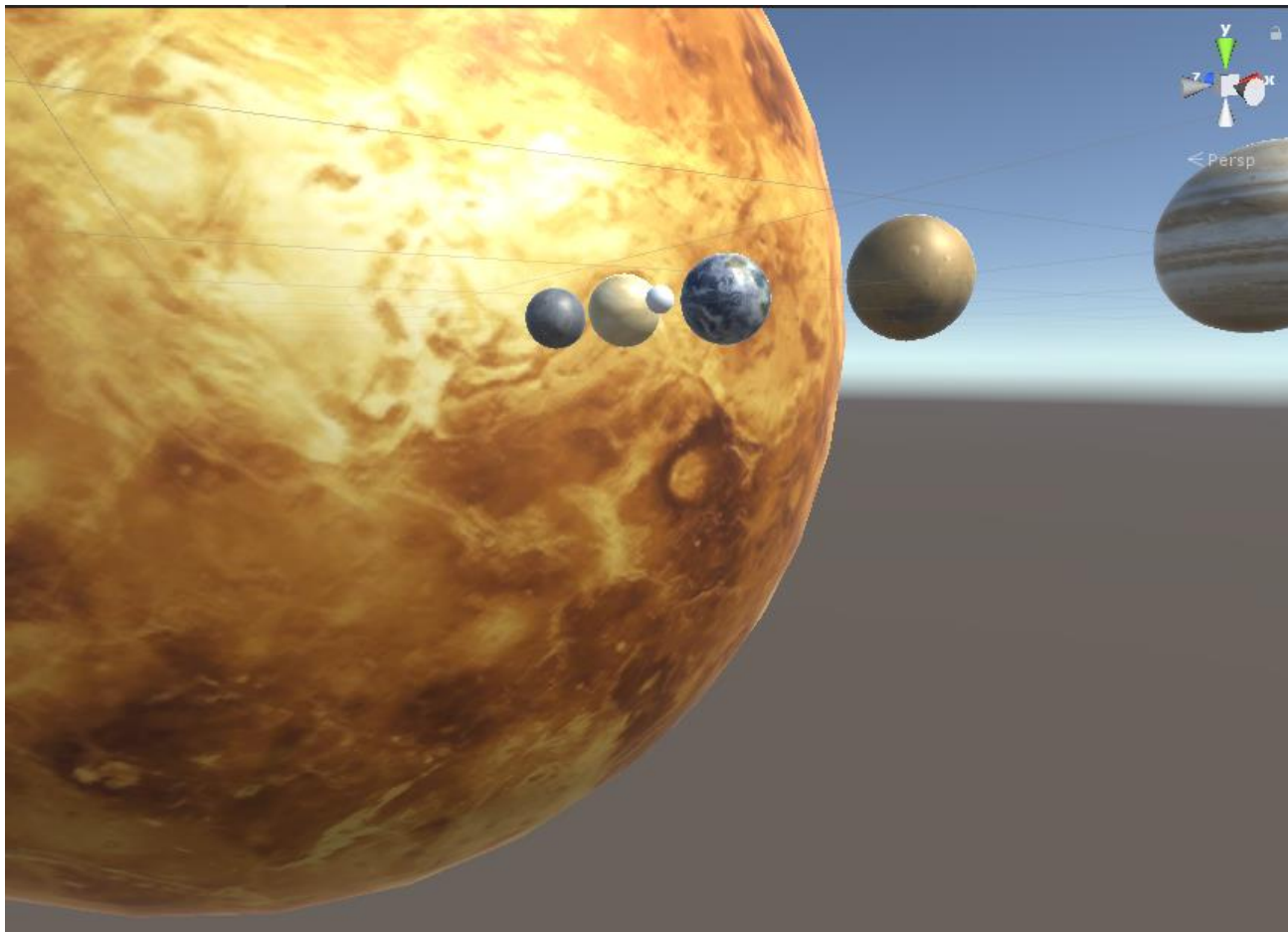
直接将资源拖到对应的物体上就行，没有技巧。

水星、金星、地球、火星、木星、土星、天王星、海王星

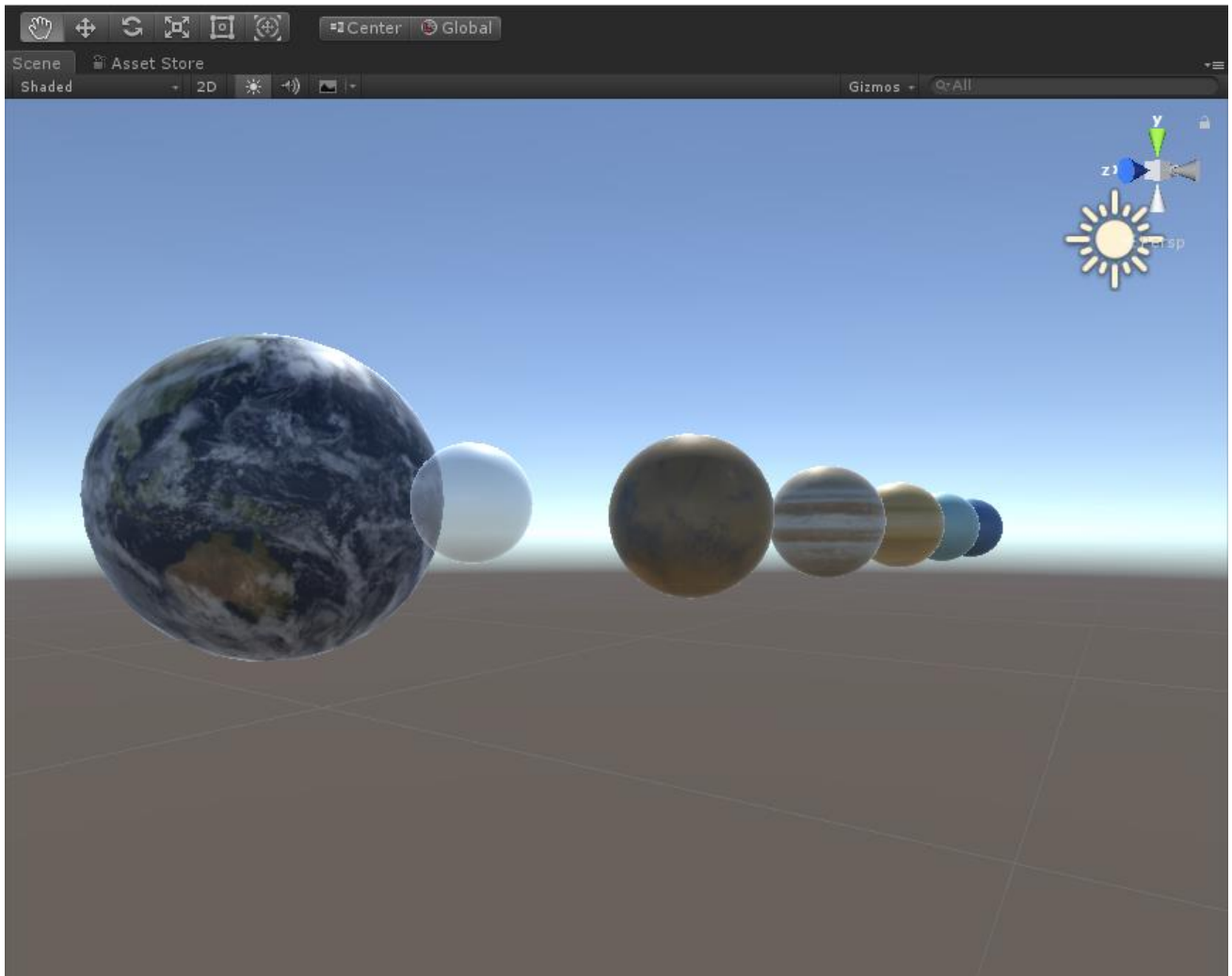


1.3.5 添加月球和星环

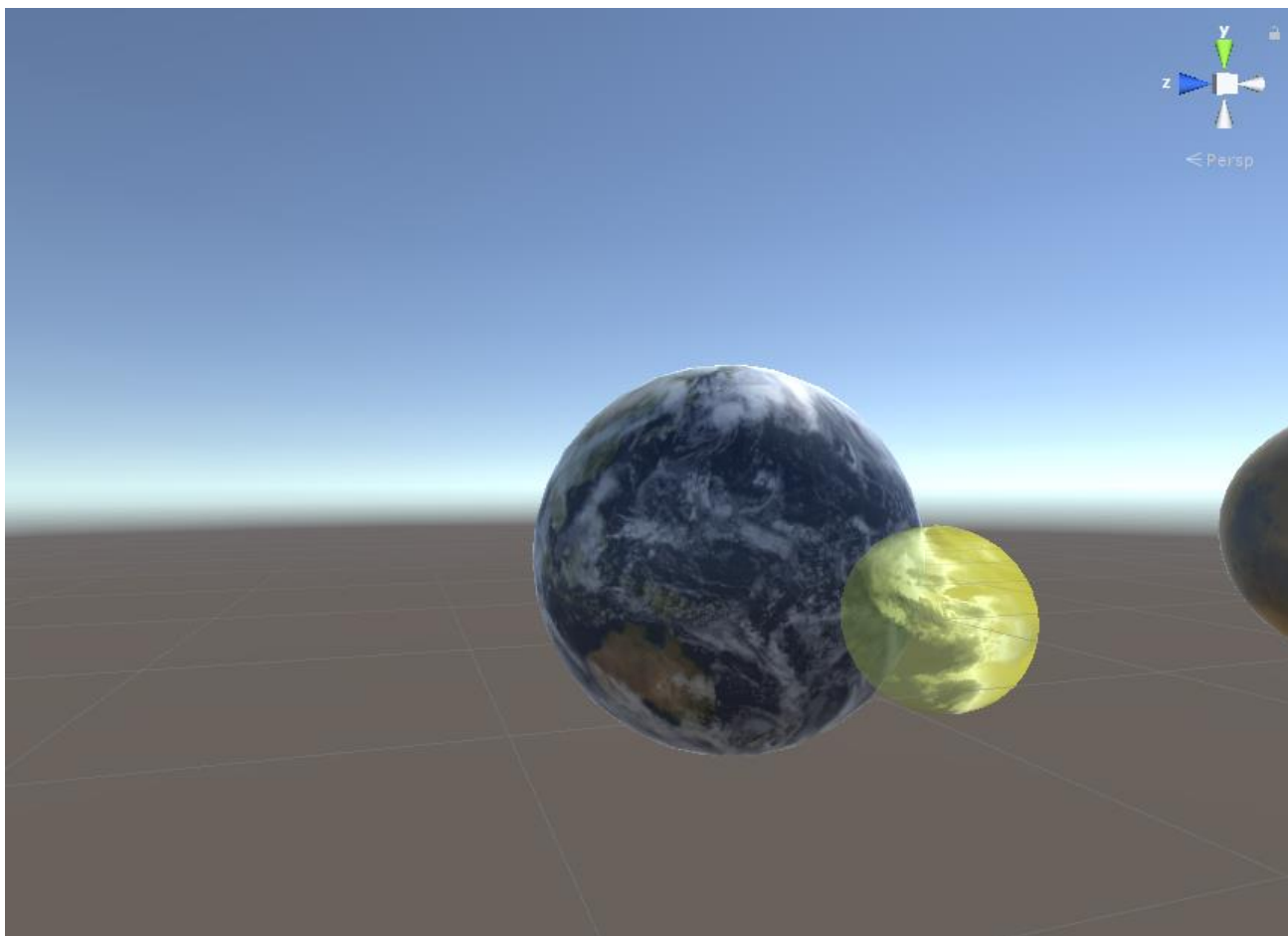
首先添加月球



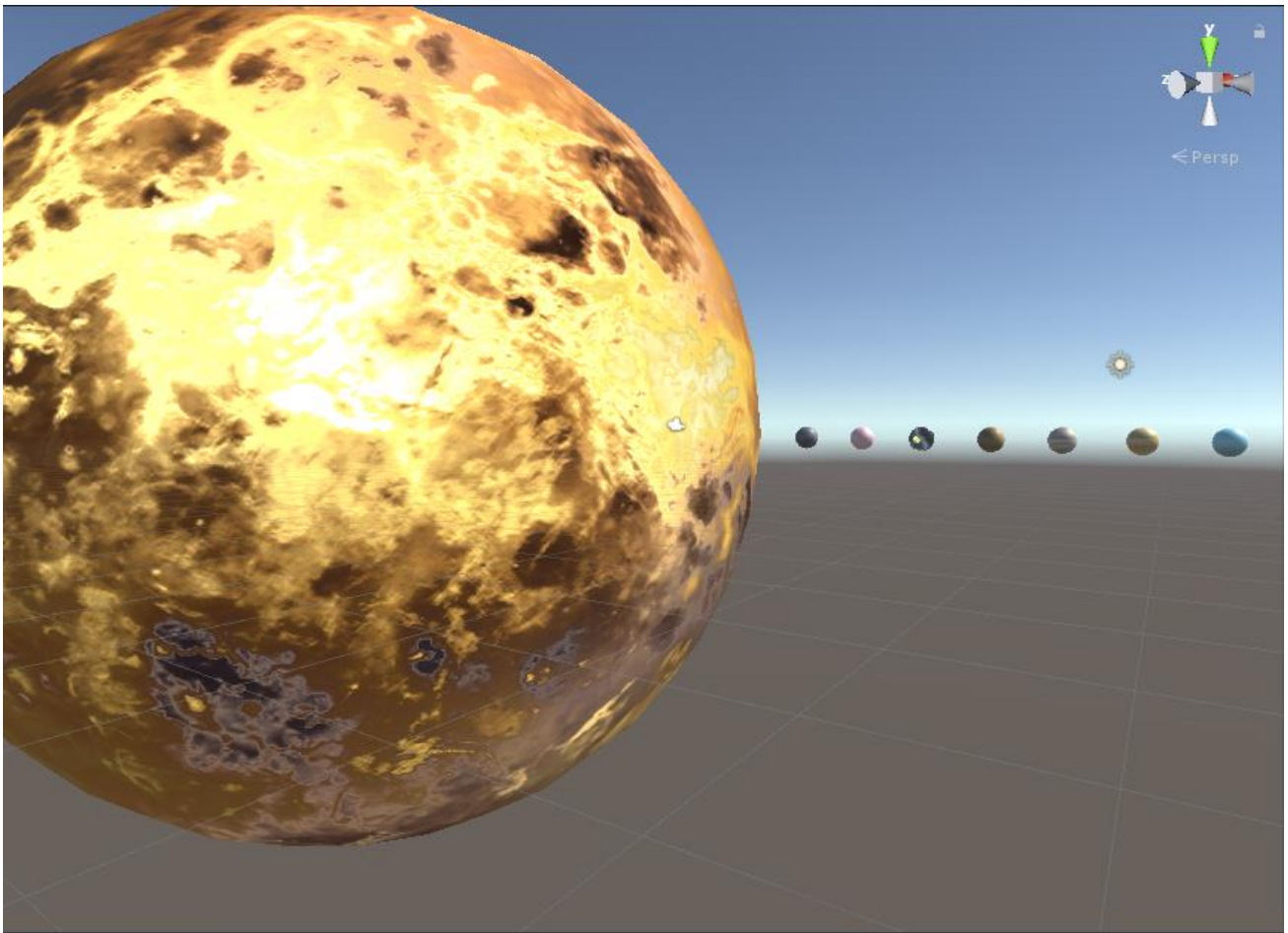
为了添加星球外的气态效果，将月球设置为半透明：



创建一个新的材质，设置模式为 **transparent**，并设置其 **alpha** 通道的强度为 128（50%不透明度），然后将气态星球的纹理放到这个月球上，给其加上黄色的颜色，并添加法线贴图作为气态表面纹理。



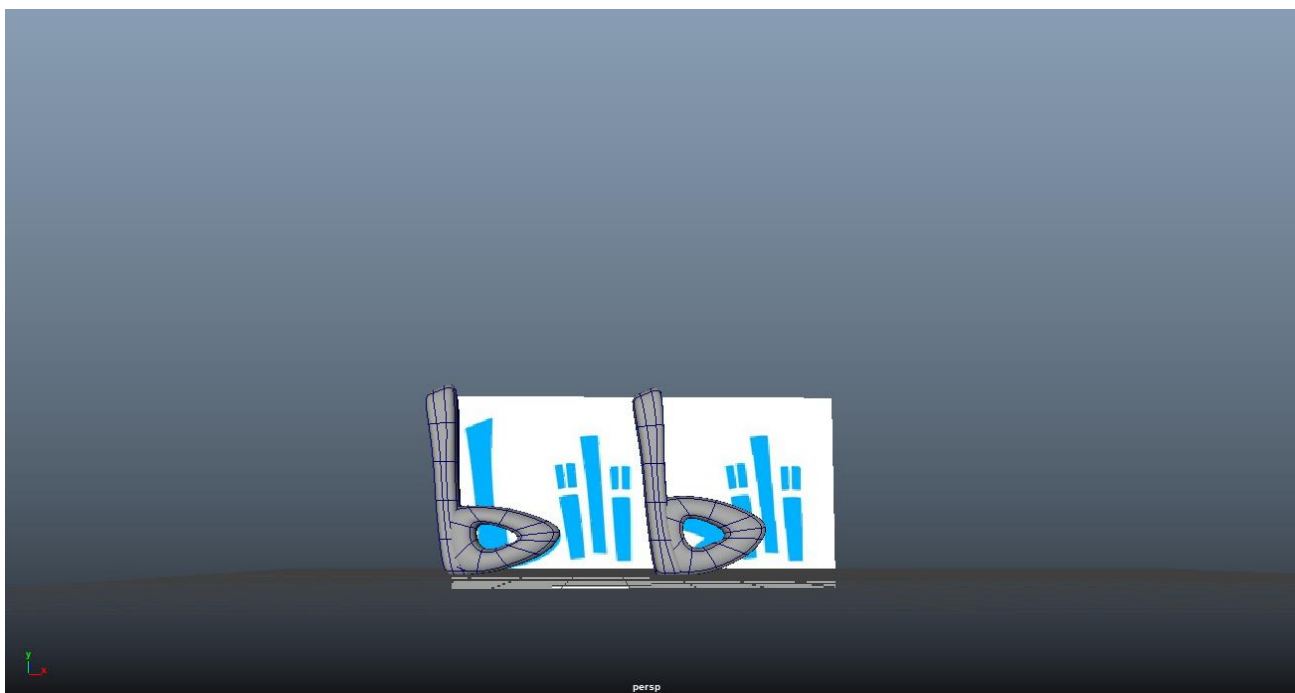
然后给太阳加上法线贴图，增加光照立体感：



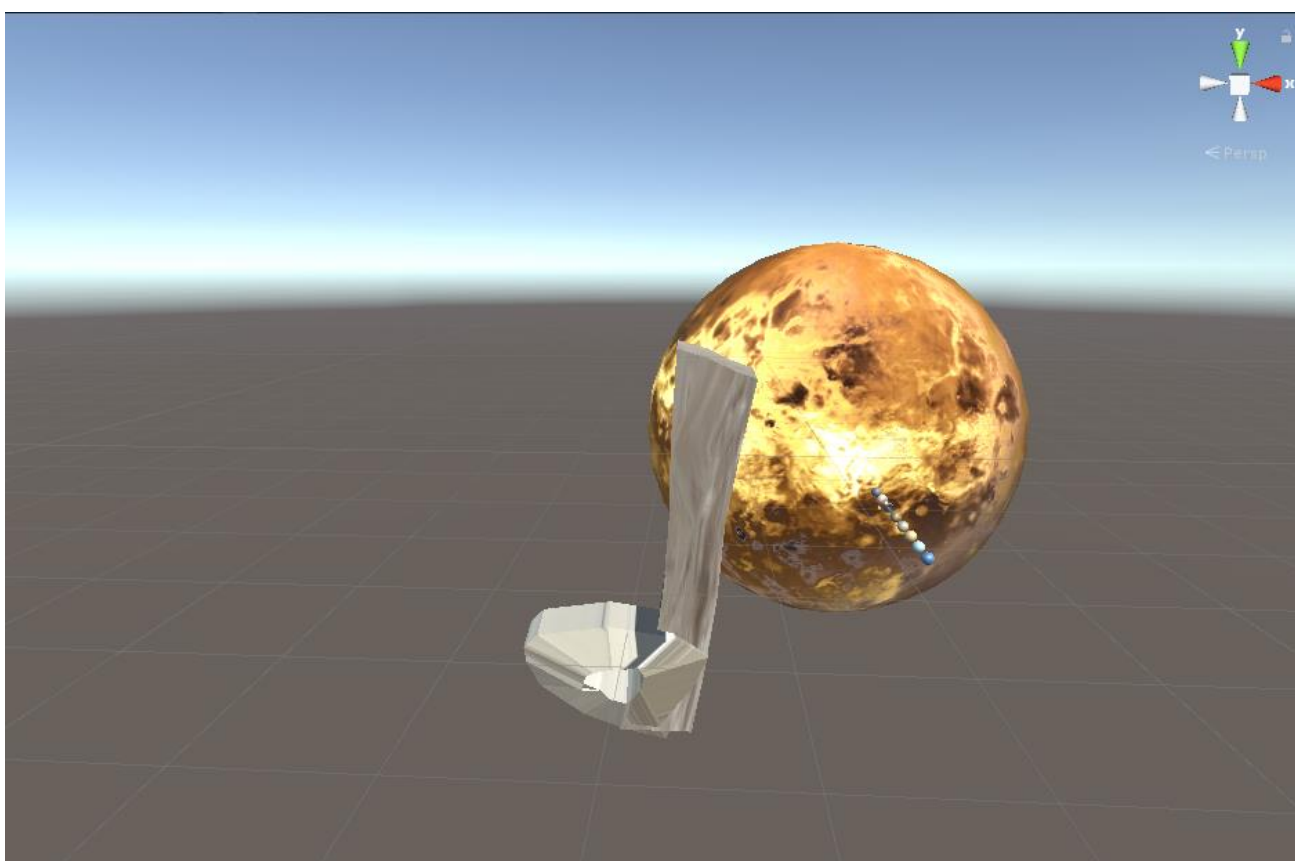
然后添加星环

太阳系中拥有行星环的行星有木星、土星、天王星和海王星。

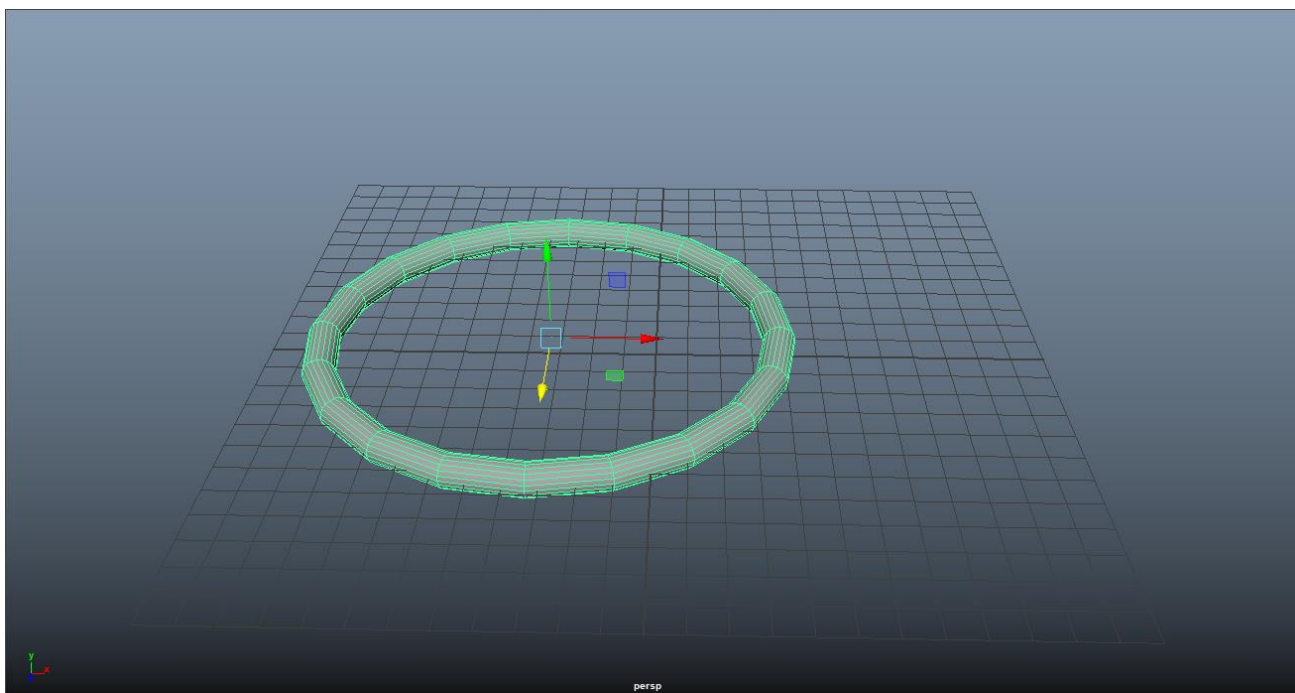
首先尝试导入一个制作好的模型 B 字母(导出 FBX 格式并在 Unity 中 import new asset)



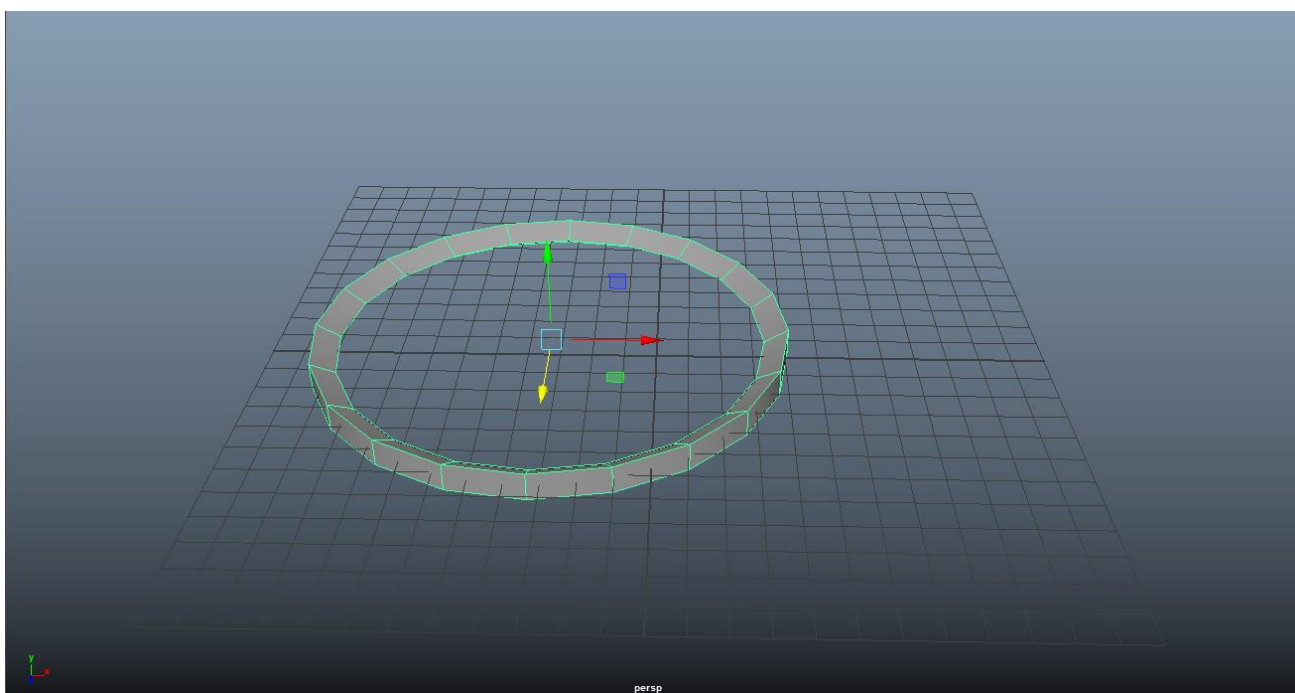
导入 Unity 并添加木星环纹理，一切正常：



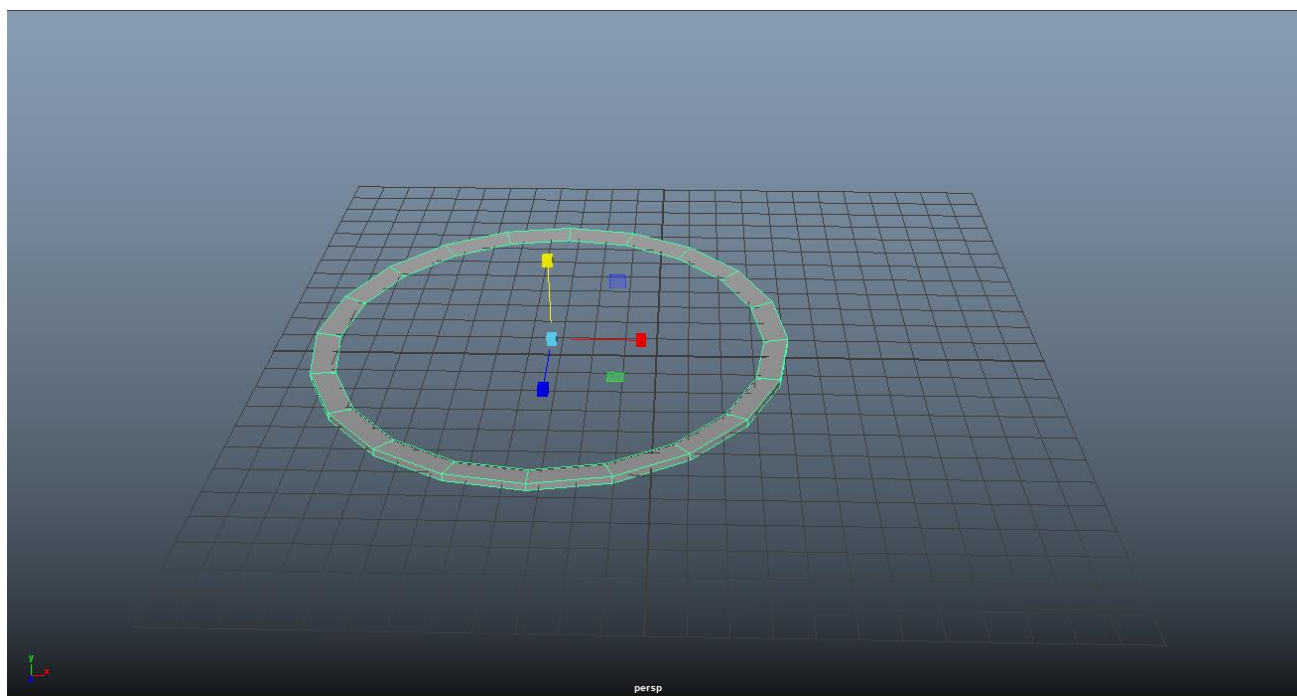
开始制作星环：（已经将 FBX 文件打包上传到 GitHub）



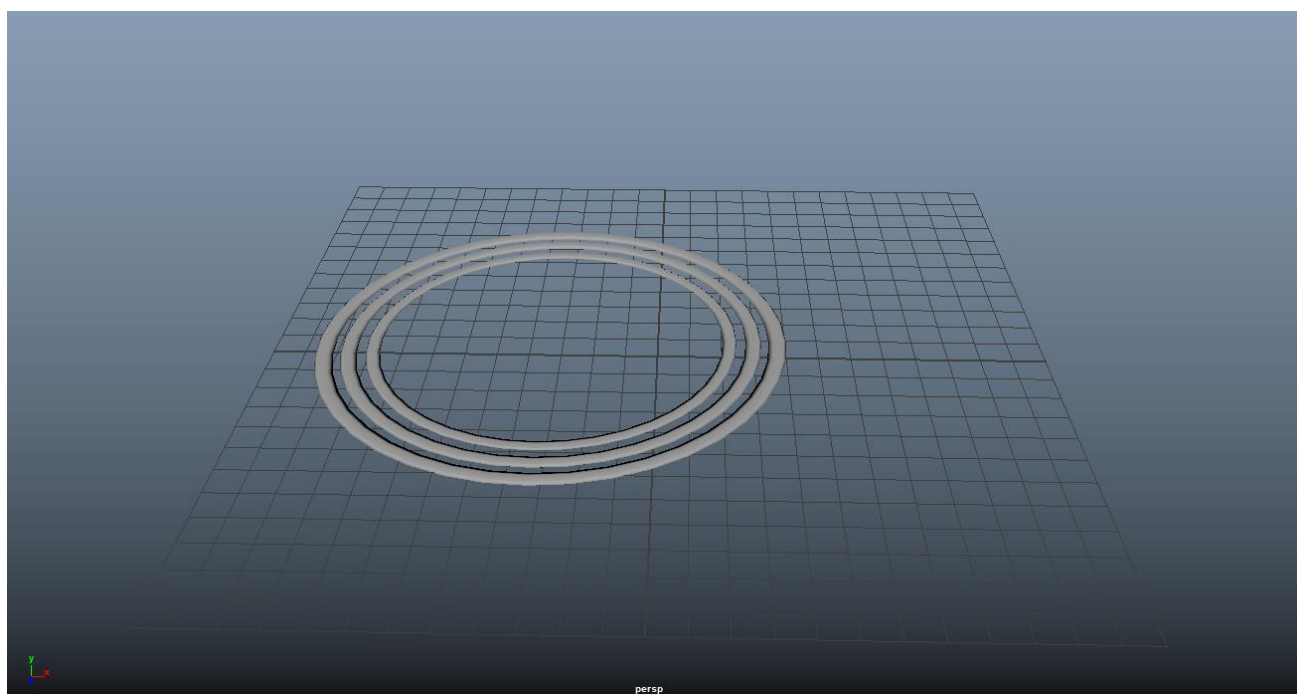
减少高度细分和横向细分 20（高细分） * 3（宽细分）：

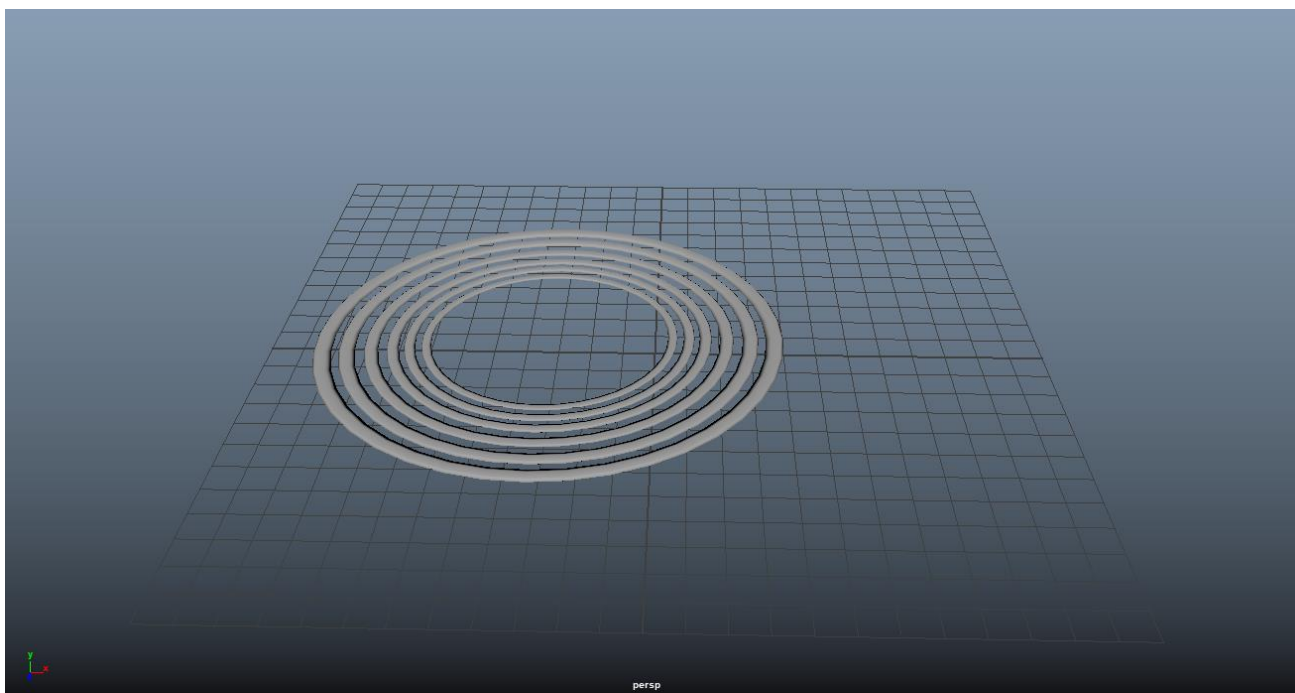


压扁：

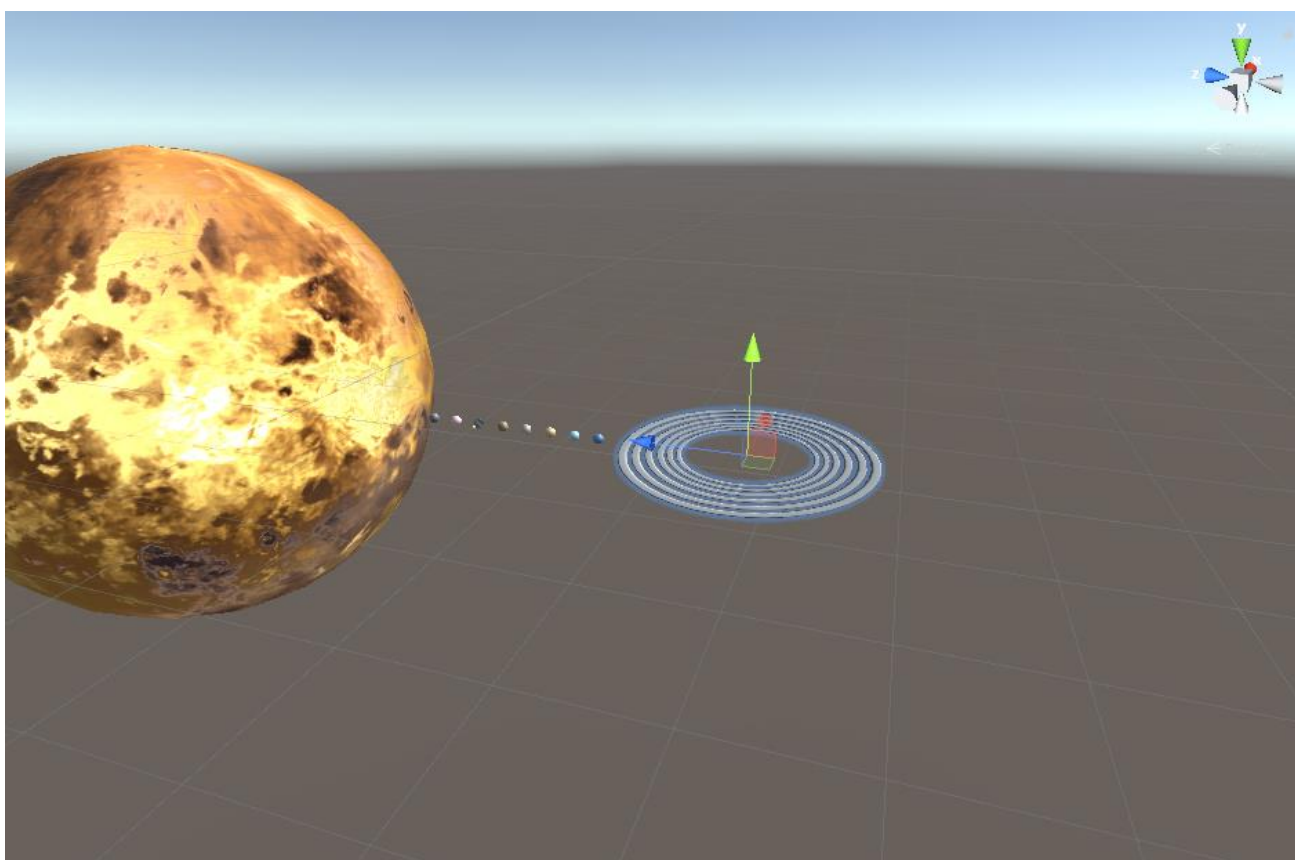


自身扩展，制造多层环效果

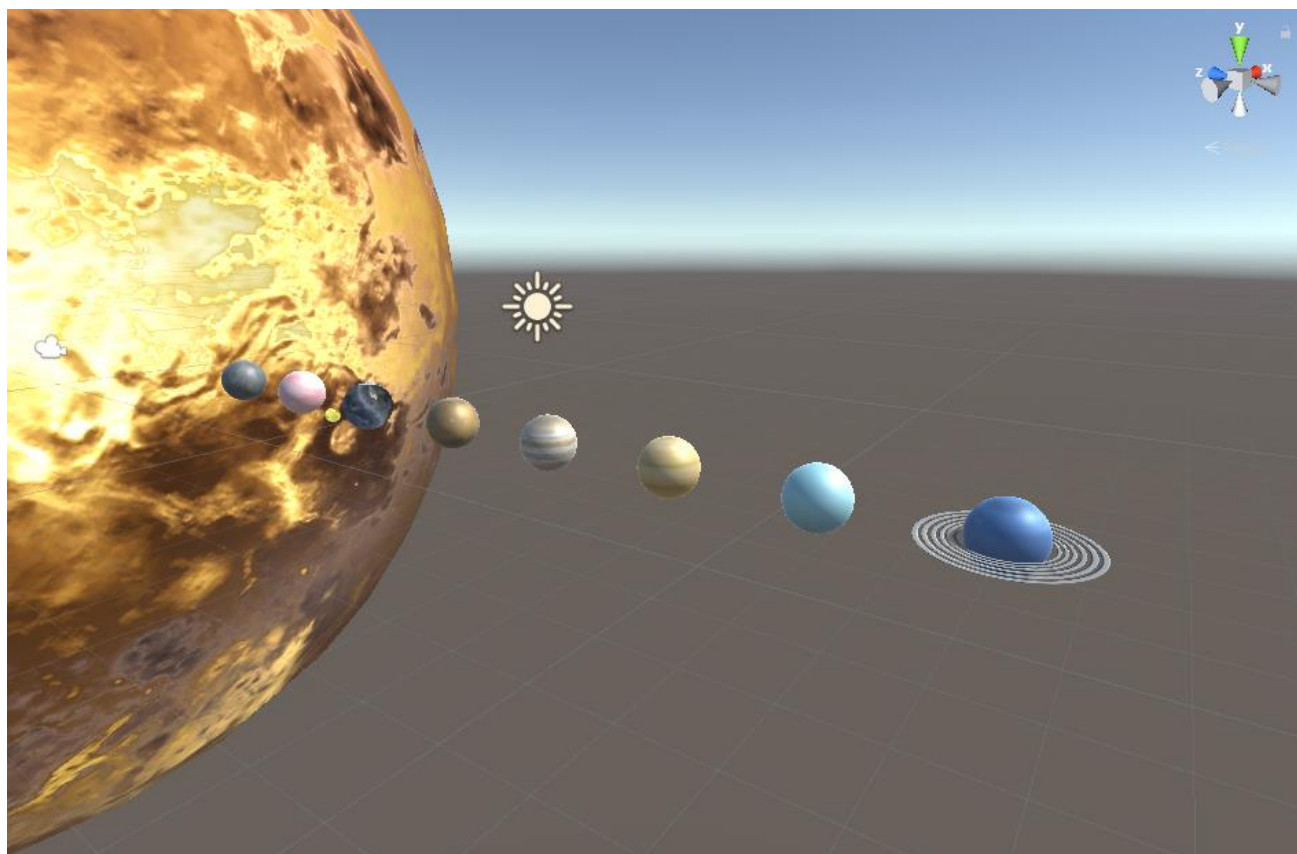




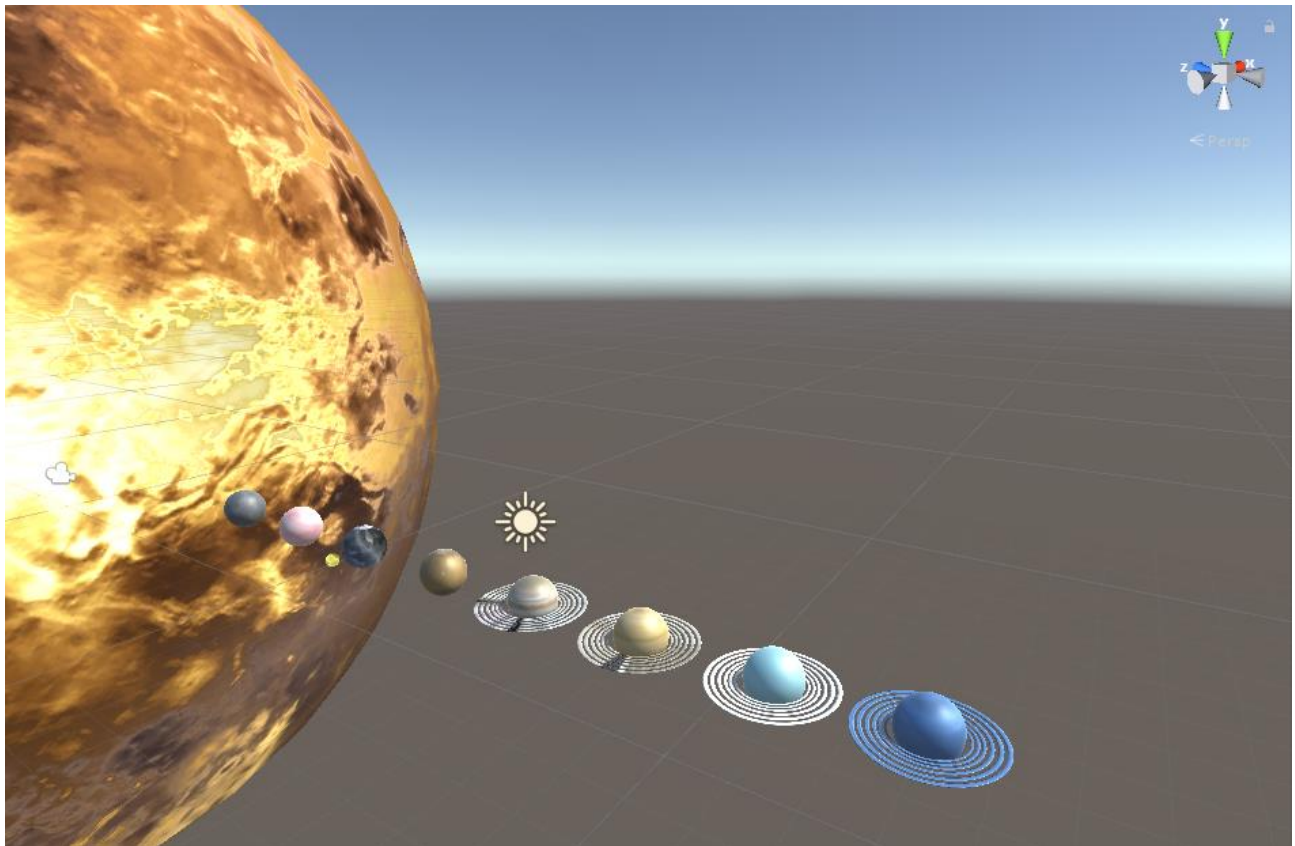
导入 Unity:



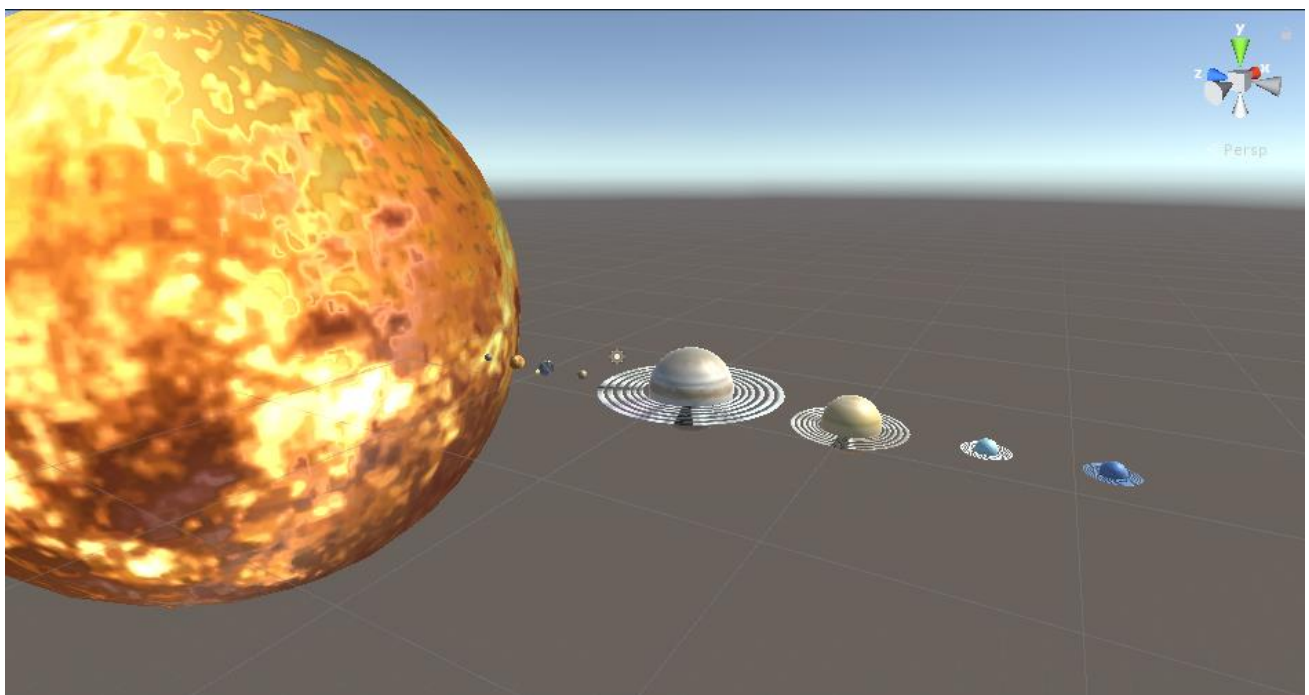
调整位置，依次给木星，土星，天王星，海王星加上星环：



给星环加上纹理：

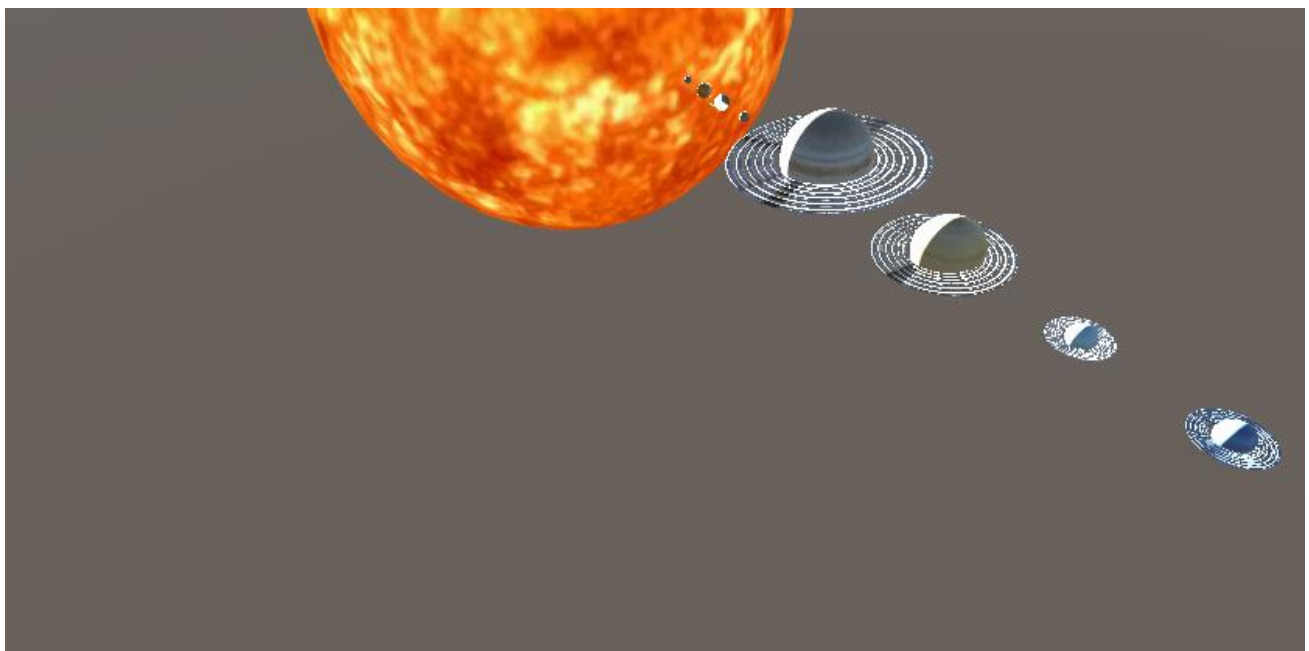
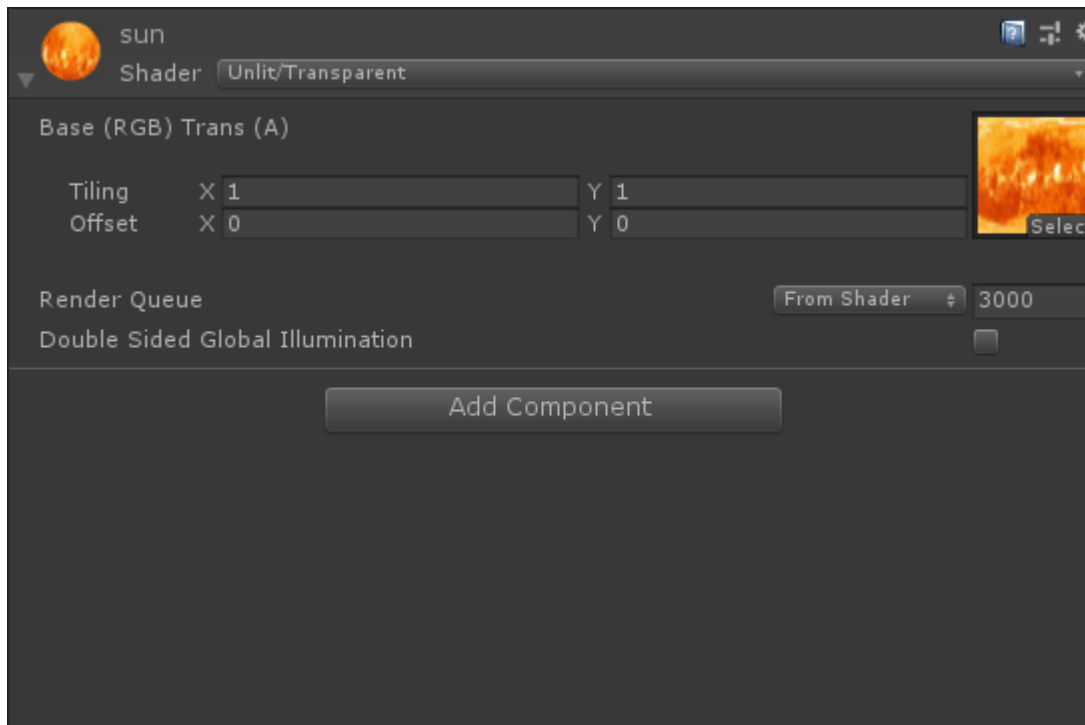


1.3.6 调整星球比例

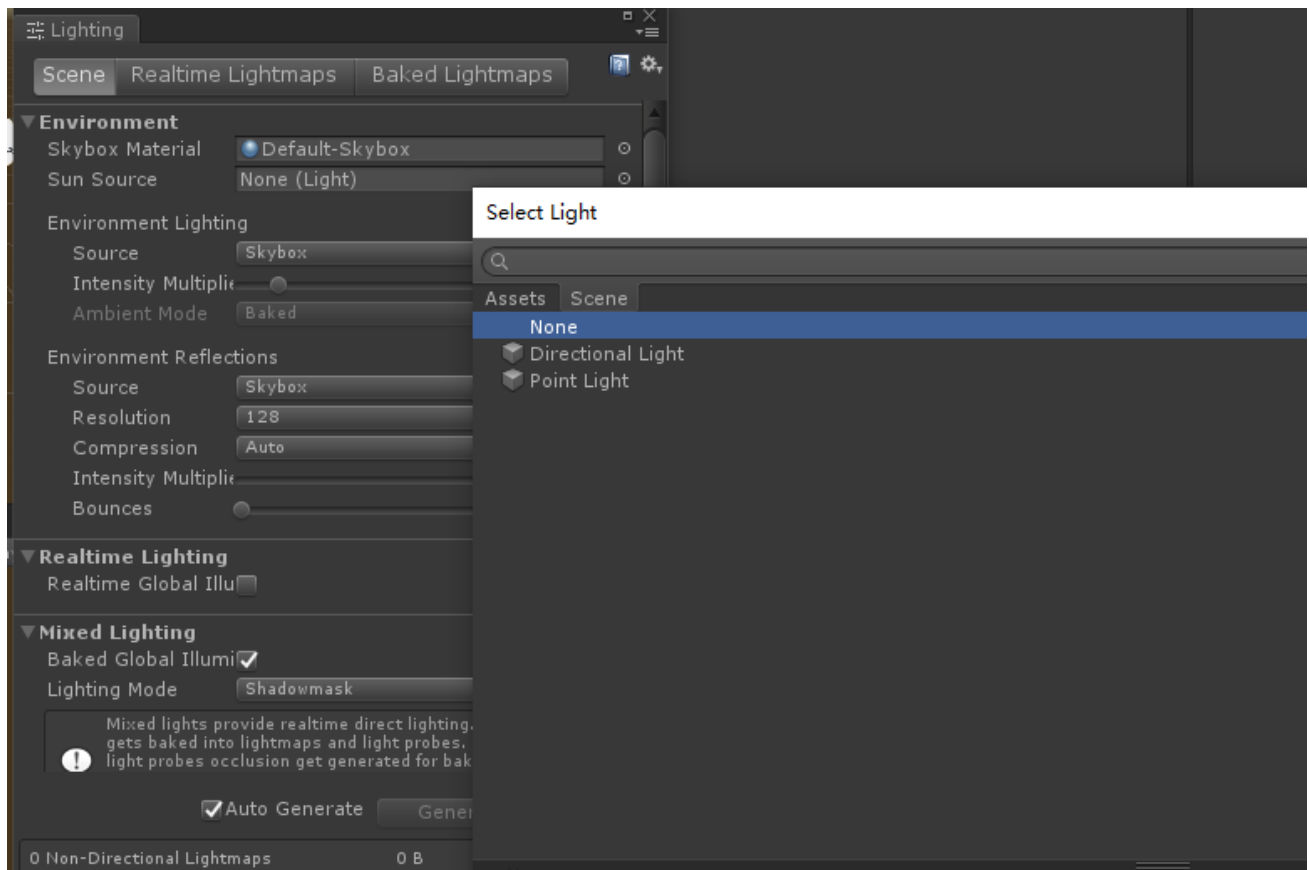


1.3.7 关闭平行光源，添加点光源

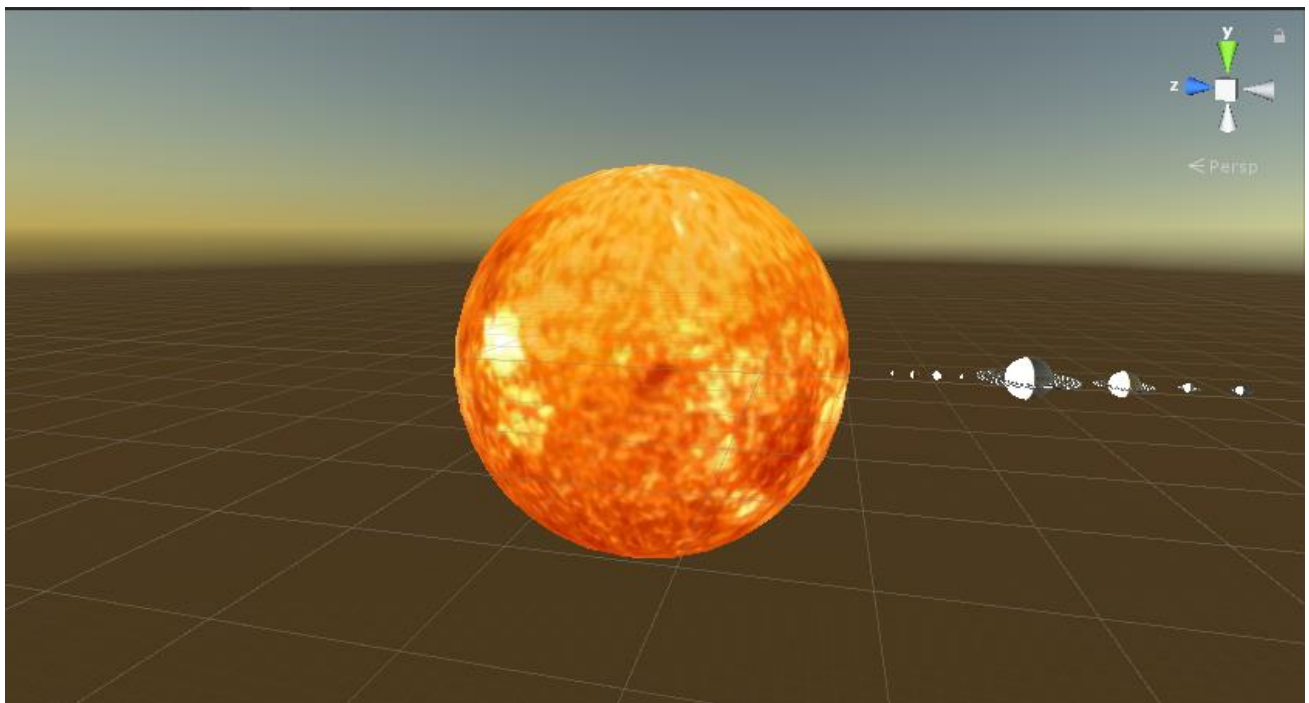
将太阳材质设置为透明，并将点光源置于太阳的中心



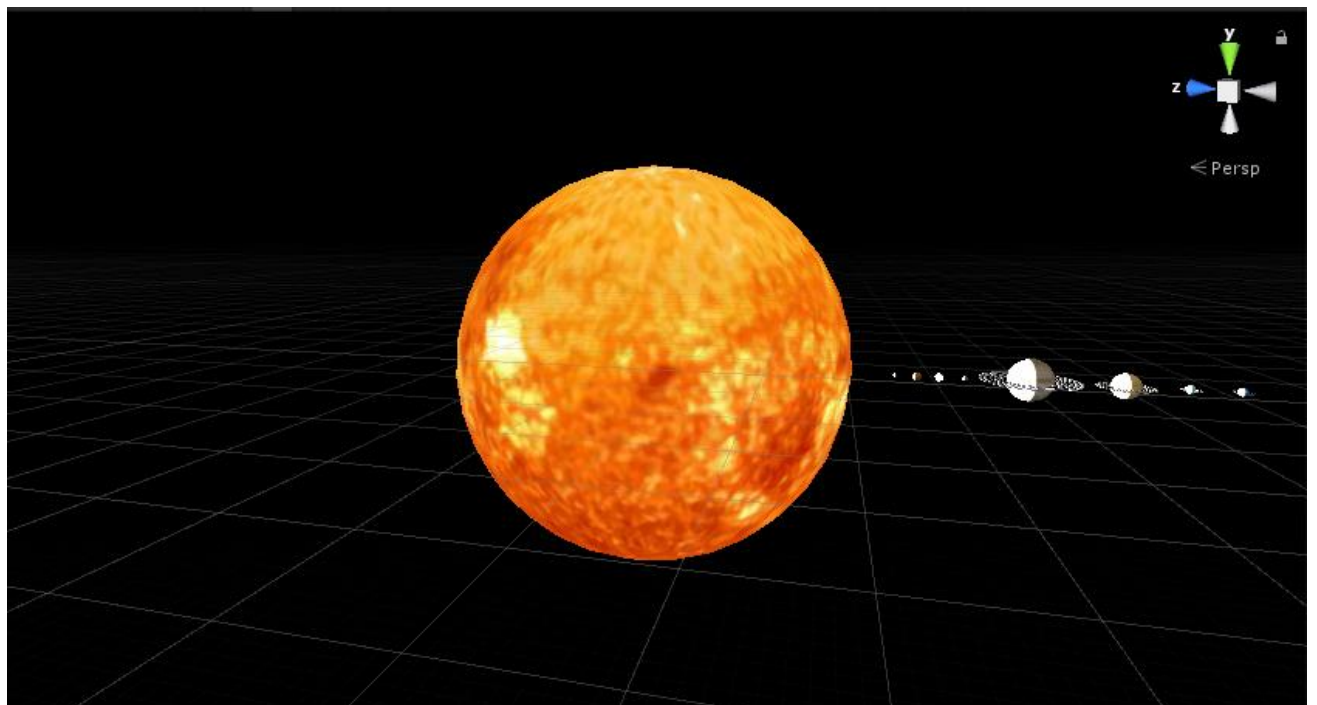
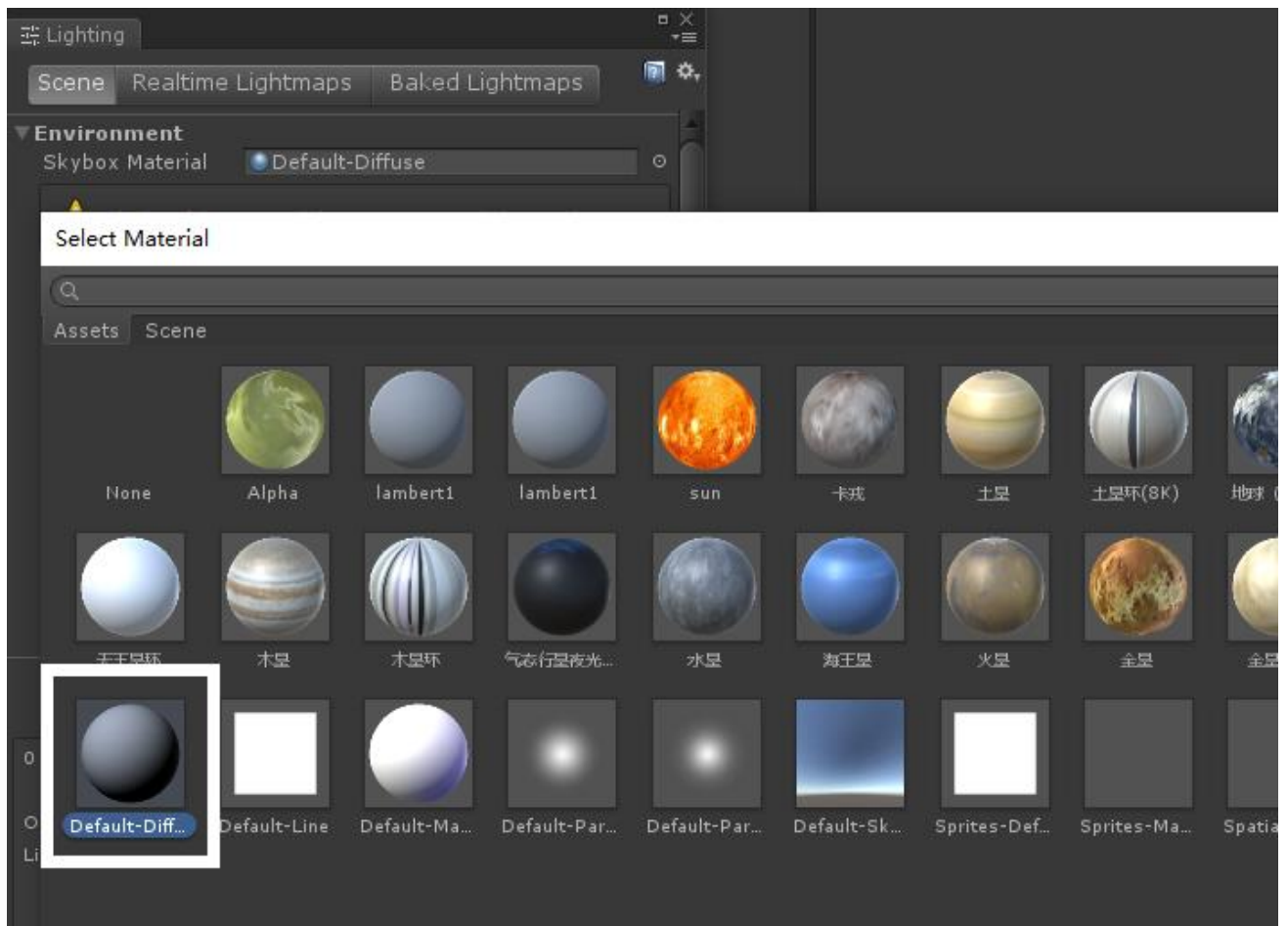
接着关闭默认的阳光：Window->rendering->lighting settings

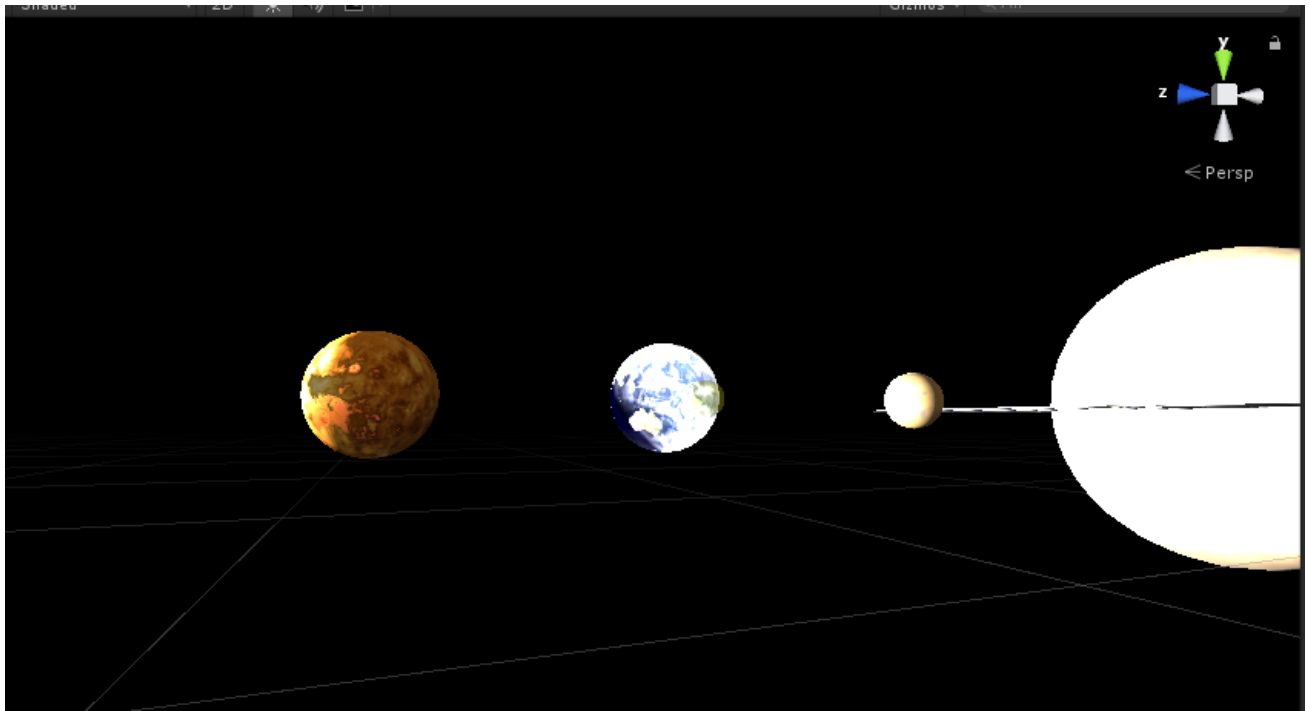


环境变暗了，但是依然能够看到有微光

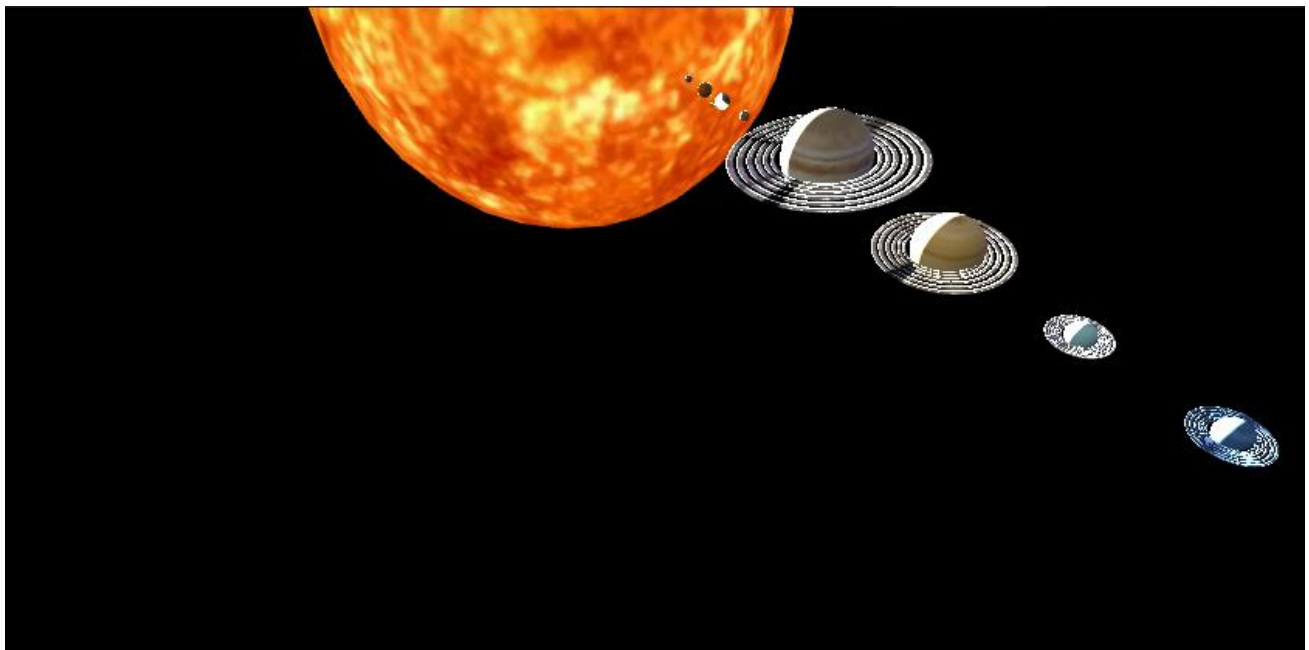


接着将 Sky-box 的材质调整为 Default Diffuse，世界暗下来:





Main Camera:



1.3.8 添加粒子效果，让太阳动态发光

导入爆炸效果包，添加喷火，爆炸，燃烧三种效果，叠加并旋转角度



JEAN MORENO (JMO)

War FX

FREE



48 user reviews

Download

Popular Tags

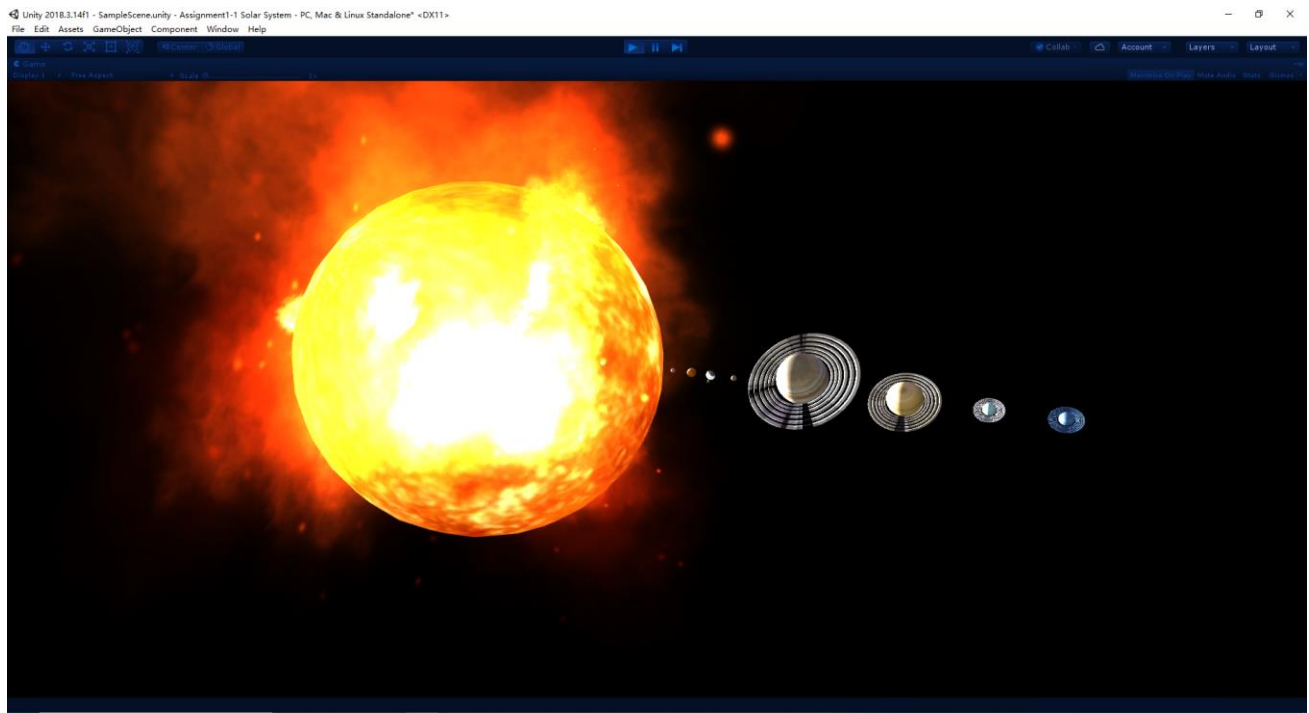
Add a new tag right now?

Add tags

War FX is now **FREE!**

40+ REALISTIC EFFECTS for war games of all genres!

添加后：



1.3.9 编写脚本，添加运动信息

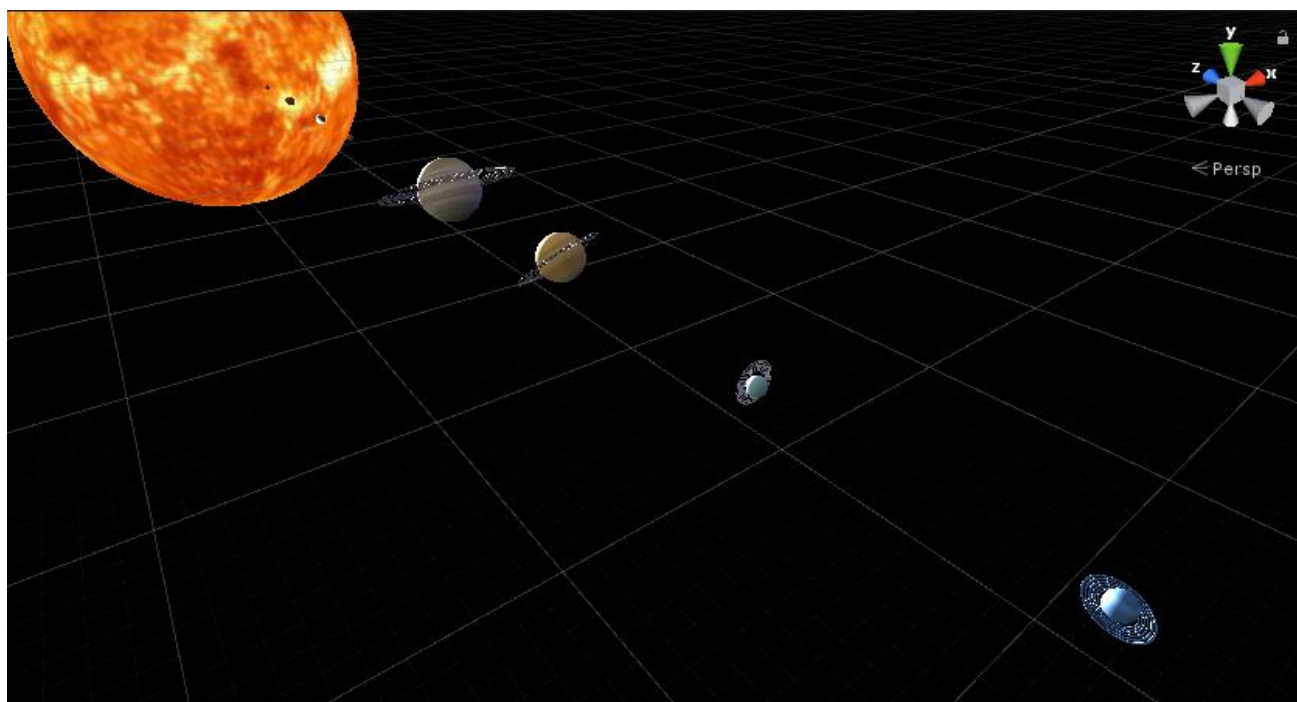
首先分析太阳系中的行星的运动模型：

自转运动模型：

在太阳系八大行星中,有六大行星是自西向东自转的,它们中水星、地球、火星、木星、土星和海王星,如果从地球的北极（也是太阳和大多数行星的北极）方向看,它们都是逆时针方向自转,与它们的公转方向相同.太阳也是这样自转的.

只有金星和天王星与大家的自转方向不同.金星是顺时针自转的,而它的公转方向与其它大行星相同,这样一来,金星上的一天与一年的时间就差不多了.而天王星自转轴与黄道面的夹角很小,看上去差不多是躺在黄道面上转圈.但它也算是由西向东自转的.

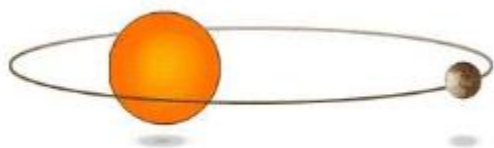
按照自转的运动模型，重新调整了 8 个行星与黄道平面之间的夹角，效果如下。



太阳自转模型：

太阳系的 8 大行星的公转轨道面基本还都是在同一个平面上(黄道面)，但太阳的自转轴却并非与黄道面垂直，而是存在大约 6 度的倾角。

公转运动模型：

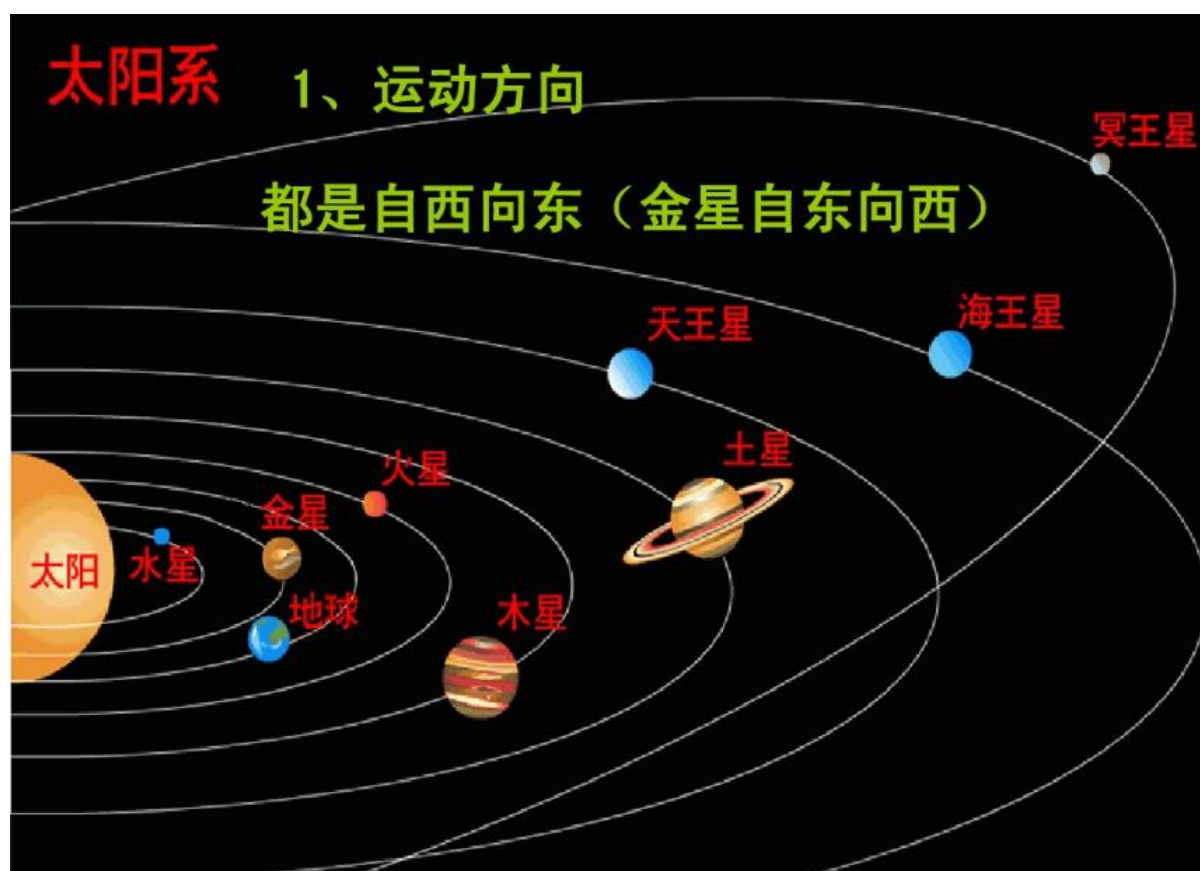


位置：太阳位于椭圆的一个焦点上，行星绕着这个焦点做非匀速率曲线运动。

运动方向：7 大行星除了金星外都是自西向东转。

轨道倾角：都接近在同一个平面上，具体偏转角度见下一页的“太阳系八颗行星轨道倾斜角与偏心率”图片。

偏心率：决定了椭圆轨道的形状，见下一页的“太阳系八颗行星轨道倾斜角与偏心率”图片。



2、太阳系八颗行星轨道倾角与偏心率

	水星	金星	地球	火星	木星	土星	天王星	海王星
轨道倾角	7°	3.4 °	0 °	1.9 °	1.3 °	2.5 °	0.8 °	1.8 °
偏心率	0.206	0.007	0.017	0.093	0.048	0.055	0.051	0.006

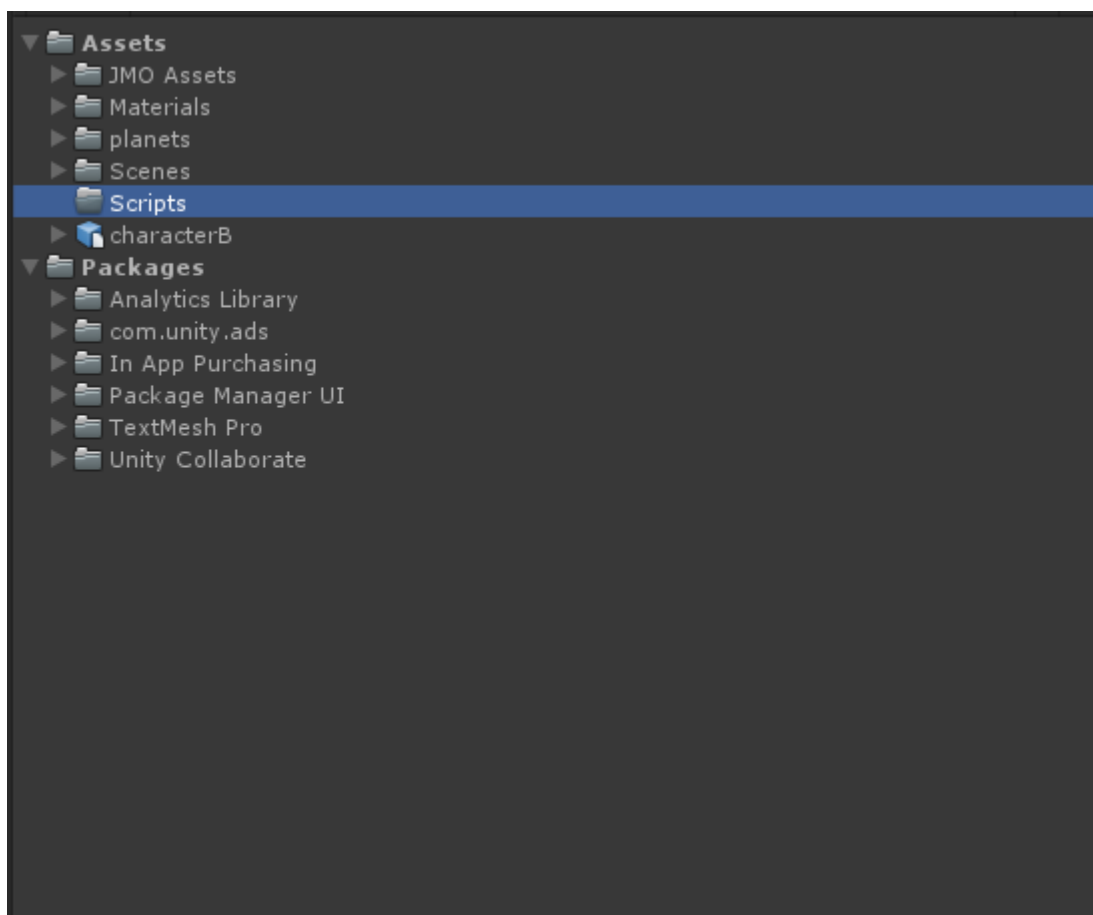
1、其他行星的公转轨道面与地球的公转轨道面之间的夹角大吗？

答：不大，行星的公转轨道几乎在同一平面上。

2、与其他行星相比，地球的轨道形状特殊吗？

答：不特殊，都接近正圆。

创建脚本文件夹 Scripts



1.3.9.1 实现行星和恒星的自转

经过思考，为了体现面向对象的编程思想，设置了基类 **StarBase**，之后派生出行星，卫星，恒星三种星球。

分析三种星球，抽象出一般的公共属性和行为，作为基类 **StarBase** 的属性和行为：

恒星：太阳，在 **Demo** 中仅有自转，不考虑绕银河系中心的公转

方法：

1. 自转 **SelfRotate**

属性：

1. 自转转速 **selfRotateAngularVelocity** 为 **Vector3** 类型（float X, float Y, float Z）
2. 自转倾斜角 **selfRotateTiltAngle** 为 **Vector3** 类型（float X, float Y, float Z）

行星：除了太阳和月球二者外的 8 大行星

方法：

1. 自转 **SelfRotate**
2. 公转 **PubRotate**

属性：

1. 旋转中心 **rotateCenterObject** 为 **GameObject** 类型
2. 公转轨道法线倾斜角 **pubRotateNormalAngle** 为 **Vector3** 类型
3. 自转倾斜角 **selfRotateTiltAngle** 为 **Vector3** 类型
4. 公转转速 **pubRotateAngularVelocity** 为 **float** 类型
5. 自转转速 **selfRotateAngularVelocity** 为 **Vector3** 类型

卫星：月球

方法：

1. 自转 **SelfRotate**
2. 公转 **PubRotate**

属性：

1. 旋转中心 **rotateCenterObject** 为 **GameObject** 类型
2. 公转轨道法线倾斜角 **pubRotateNormalAngle** 为 **Vector3** 类型
3. 自转倾斜角 **selfRotateTiltAngle** 为 **Vector3** 类型
4. 公转转速 **pubRotateAngularVelocity** 为 **float** 类型
5. 自转转速 **selfRotateAngularVelocity** 为 **Vector3** 类型

通过上面分析可以发现，基类 **StarBase** 设计的时候，按照太阳设计，行星和恒星派生后添加相应方法和属性（添加内容相同，类型名称不同）。

下面结合 **Unity** 提供的 **API** 编写代码：

查找 **Unity Scripts API** 文档，**Unity** 提供了两个跟旋转有关的函数：

<https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Transform.html>

Transform.Rotate()

Use Transform.Rotate to rotate GameObjects in a variety of ways. The rotation is often provided as a Euler angle and not a Quaternion.

<https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Transform.Rotate.html>

```
public void Rotate(Vector3 eulers, Space relativeTo = Space.Self);
```

eulers	The rotation to apply.
---------------	------------------------

relativeTo	Determines whether to rotate the GameObject either locally to the GameObject or relative to the Scene in world space.
-------------------	--

查到这个方法有两个参数，第一个参数是一个欧拉角，第二个参数是决定参考系为自身还是全局坐标系。

文档中具体的解释：

Applies a rotation of eulerAngles.z degrees around the z-axis, eulerAngles.x degrees around the x-axis, and eulerAngles.y degrees around the y-axis (in that order).

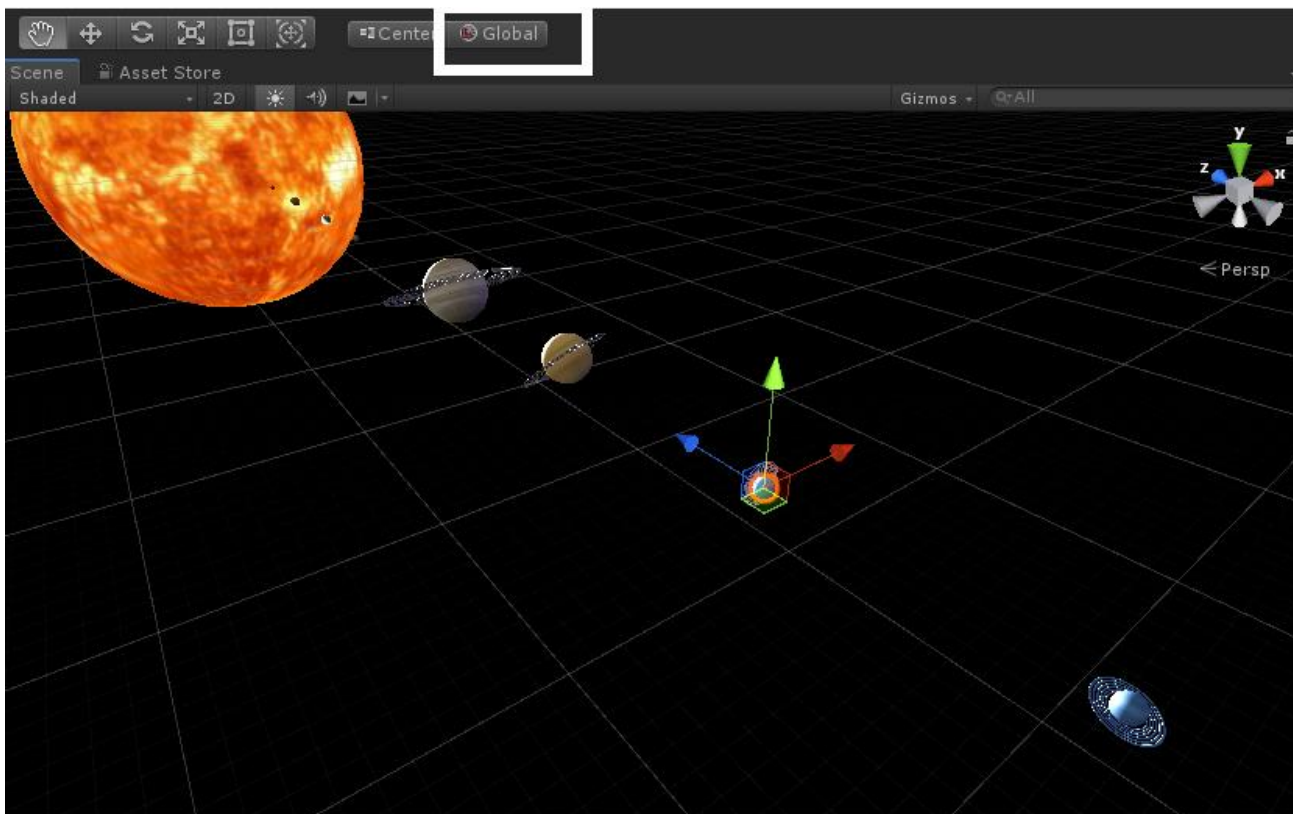
Rotate takes a [Vector3](#) argument as a euler angle. The second argument is the rotation axes, which can be set to local axis ([Space.Self](#)) or global axis ([Space.World](#)). The rotation is by the Euler amount.

照 Z 轴旋转 `eulerAngles.z`，然后按照 X 轴旋转 `eulerAngles.x`，最后按照 Y 轴旋转 `eulerAngles.y`，使用 `Space.Self` 和 `Space.World` 决定应该使用全局参考系还是局部参考系。

自转的时候，使用这个 API 的 `Space.Self` 自身的局部坐标系是合适的。这样行星就有了不依赖于层级关系（其他的父子星球的运动行为）的自转效果分量，后期只需要对父级的运动分量取反并抵消就可以。

在公转的时候，如果要使用这个 API 的 `Space.World` 参数，使用全局的坐标系，应该也是可行的，将太阳放在全局坐标系的中心（0，0，0）处，接下来就可以设置不同轨道半径的行星

的 Rotate 参数(arg1, arg2, arg3, Space.World)



根据图中的坐标系可以知道，在行星旋转的时候其实只需要根据全局的绿色的轴进行旋转即可，图中的 **Space.World** 中心在太阳的正中心，其余的行星在设计自转（假设轨道为圆形）的时候，旋转的角度都是相对于 **Sence** 的 Y 轴（绿轴）有旋转分量，对于 X,Z 轴没有旋转分量，即 $\text{arg1} = 0$, $\text{arg2} > 0$, $\text{arg3} = 0$, arg2 越大，角速度越大，旋转越快。

但是这样做，在公转的时候并不具有普适性，也就是这样写出来的方法月球是无法正常使用的，因为这样做默认了旋转中心点在世界的中心(0,0,0)，而月球的旋转中心不在世界的中心，而是地球，所以 **Transform.Rotate** 方法在这里仅用于编写自转。

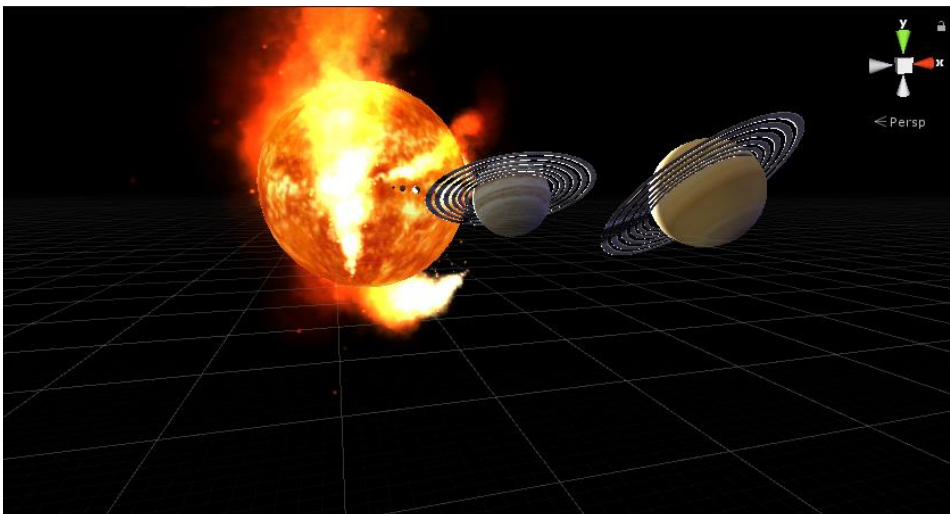
所以可以设计 **StarBase** 基类的方法 **selfRotate** 和初始化自转轨道倾斜角度如下：


```

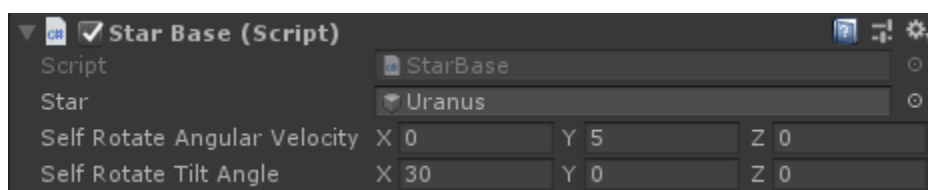
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  1 个引用
6  public class StarBase : MonoBehaviour
7  {
8      public GameObject star;
9      public Vector3 selfRotateAngularVelocity;
10     public Vector3 selfRotateTiltAngle;
11     0 个引用
12     protected void Start()
13     {
14         // Initialize the Rotation of star refering to the world cordinates
15         star.transform.eulerAngles = selfRotateTiltAngle;
16         // For parental transformation :
17         // transform.localEulerAngles = new Vector3(arg1, arg2, arg3);
18     }
19     1 个引用
20     protected void SelfRotate()
21     {
22         star.transform.Rotate(selfRotateAngularVelocity, Space.Self);
23     }
24
25     // Update is called once per frame
26     0 个引用
27     void Update()
28     {
29         SelfRotate();
30         // Do nothing
31     }
32 }

```

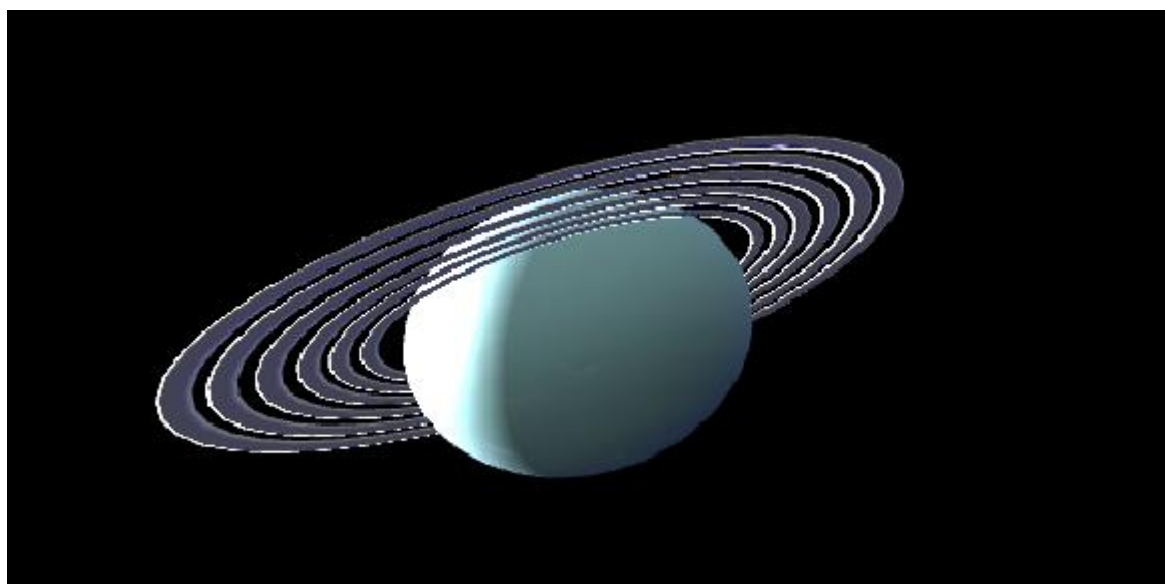
挂载 StarBase 脚本到太阳上做测试，效果：



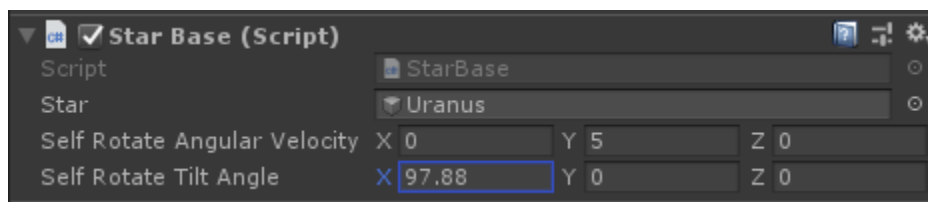
挂载 Uranus，初始化角度 Self Rotate Tilt Angle 和转速 Self Rotate Angular Velocity:



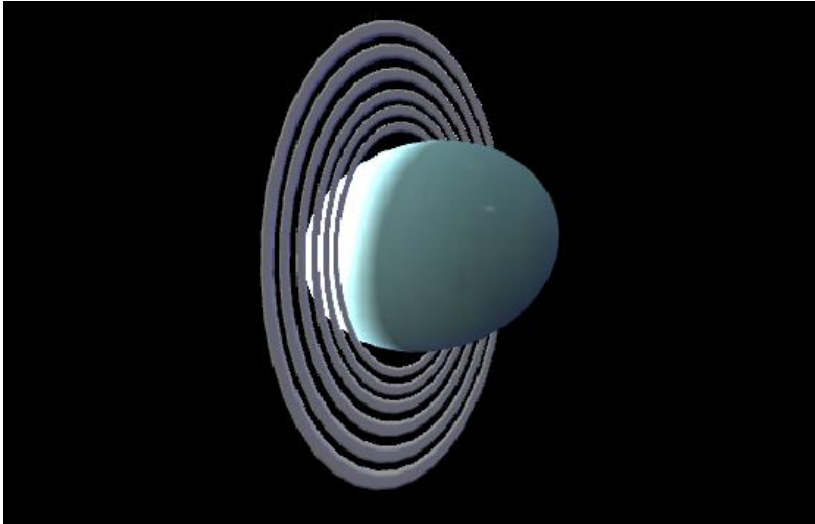
效果为按 (30, 0, 0) 初始化倾斜角，按照 (0, 5, 0) 转速旋转）:



但实际上，Uranus 的自转轨道倾斜角应该是(97.88, 0, 0)，因为其自转的轴线几乎是和其在太阳系中的公转轨道重合的，所以，将 Self Rotate Tilt Angle 设置为 (97.88, 0, 0)



运行效果:



1.3.9.2 实现行星的公转

Transform.RotateAround()

Rotates the transform about axis passing through point in world coordinates by angle degrees.

<https://docs.unity3d.com/2018.3/Documentation/ScriptReference/Transform.RotateAround.html>

public void **RotateAround**(Vector3 point, Vector3 axis, float angle);

官网给了示例代码：

```
using UnityEngine;

public class Example : MonoBehaviour

{

    void Update()

    {

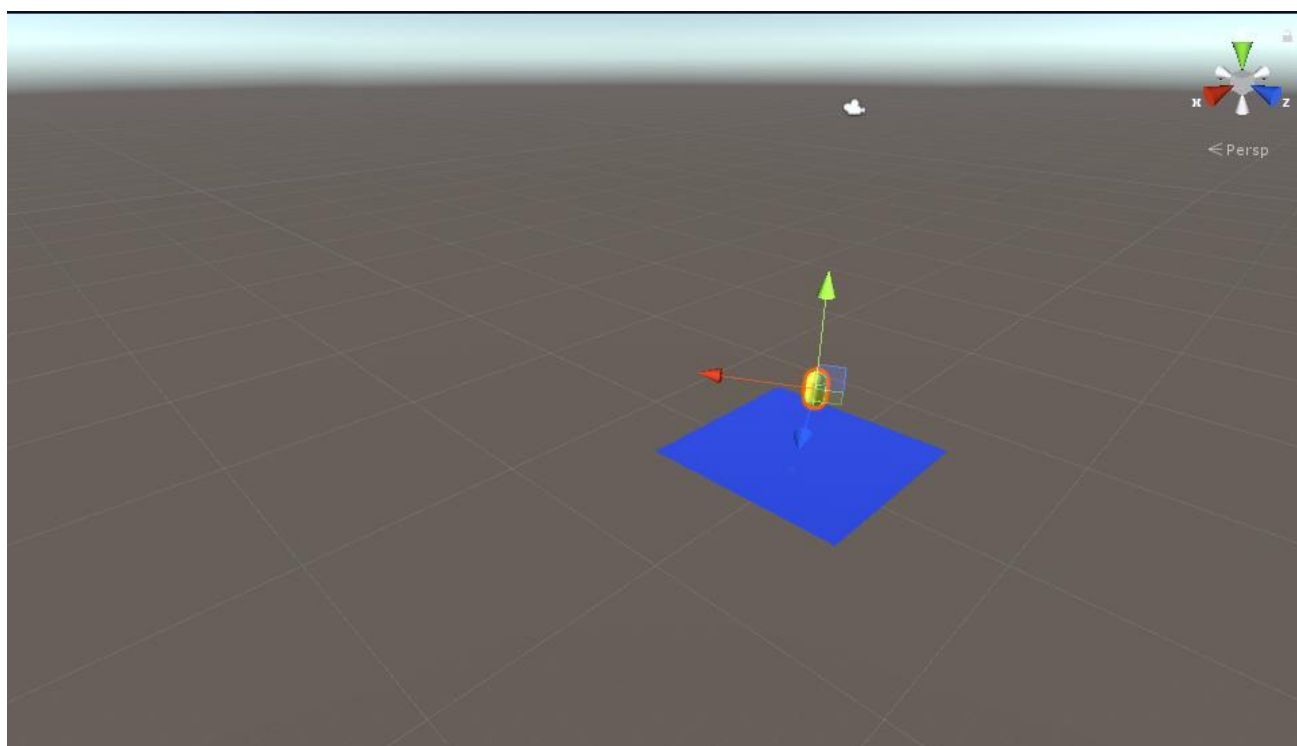
        // Spin the object around the world origin at 20 degrees/second.

        transform.RotateAround(Vector3.zero, Vector3.up, 20 * Time.deltaTime);
```

```
}
```

```
}
```

第一个参数是绕转中心在世界坐标系中的坐标 `Vector3.zero`，第二个参数是绕转的平面的法线（自己实验之后的猜测），第三个参数是角速度，使用 `20*Time.deltaTime` 限制每秒 20 度



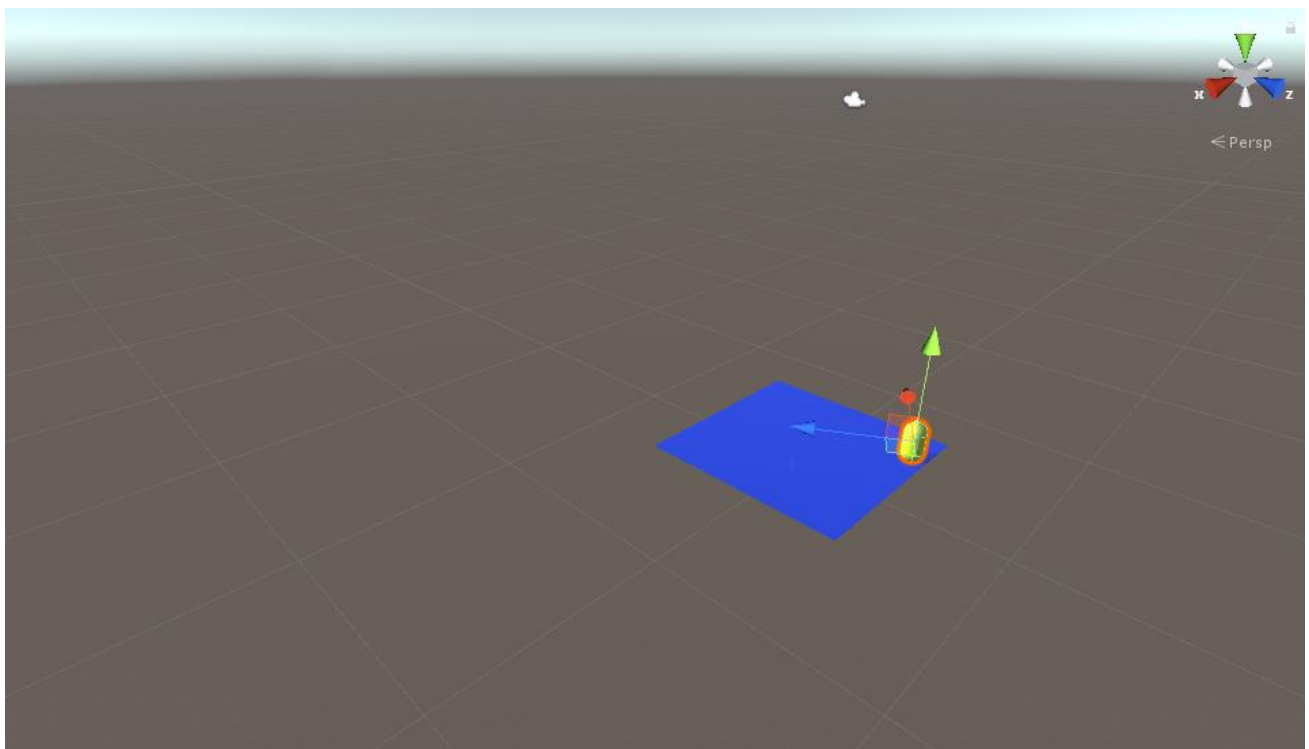
通过实验和查询 `Vector3.up` 就是 `Vector(0,1,0)`，发现 `Vector3.up` 就是上图中绿色箭头所指的方向，也正是物体运动的平面的法线方向，尝试使用非单位化的方向向量：

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  0 个引用
6  public class TransformRotateAround : MonoBehaviour
7  {
8      // Start is called before the first frame update
9      0 个引用
10     void Start()
11     {
12     }
13
14     // Update is called once per frame
15     0 个引用
16     void Update()
17     {
18         // Spin the object around the world origin at 20 degrees/second
19         this.transform.RotateAround(Vector3.zero, new Vector3(0, 2, 0), 20 * Time.deltaTime);
20     }
21 }

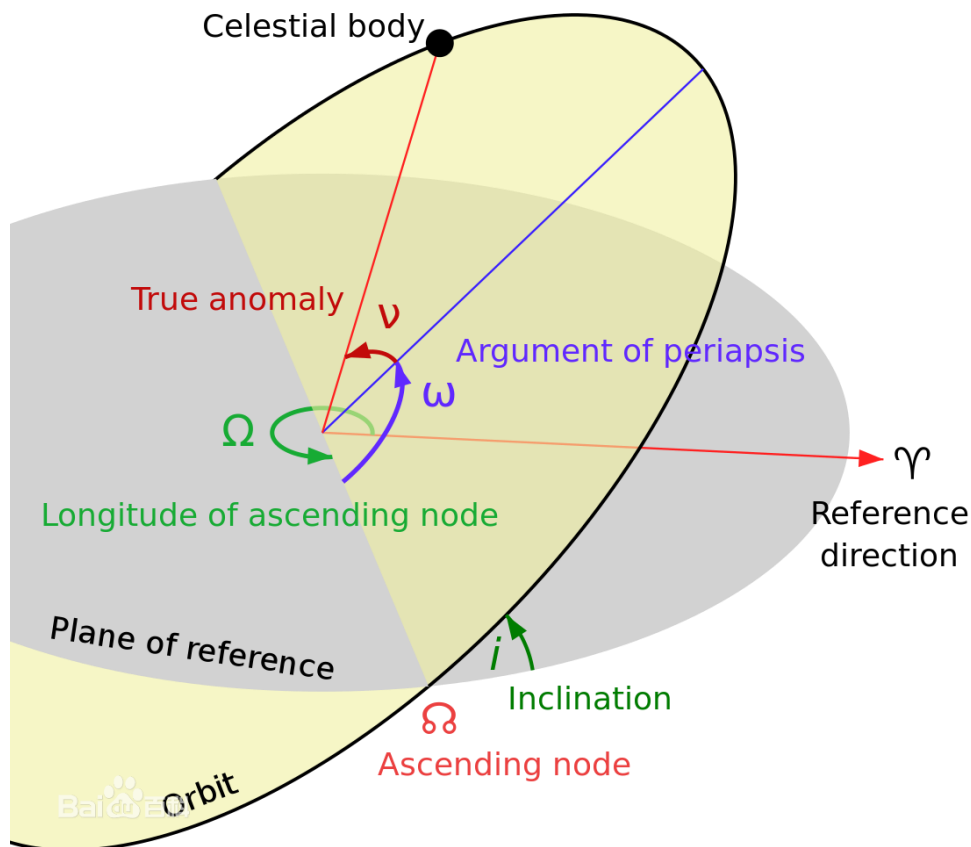
```

运行效果：



发现运行效果依然是绕（0，0，0）做匀速率圆周运动，结论成立。

下面需要编程实现行星公转轨道的倾斜角度，偏心率，转速：（自转相关设置继承基类）



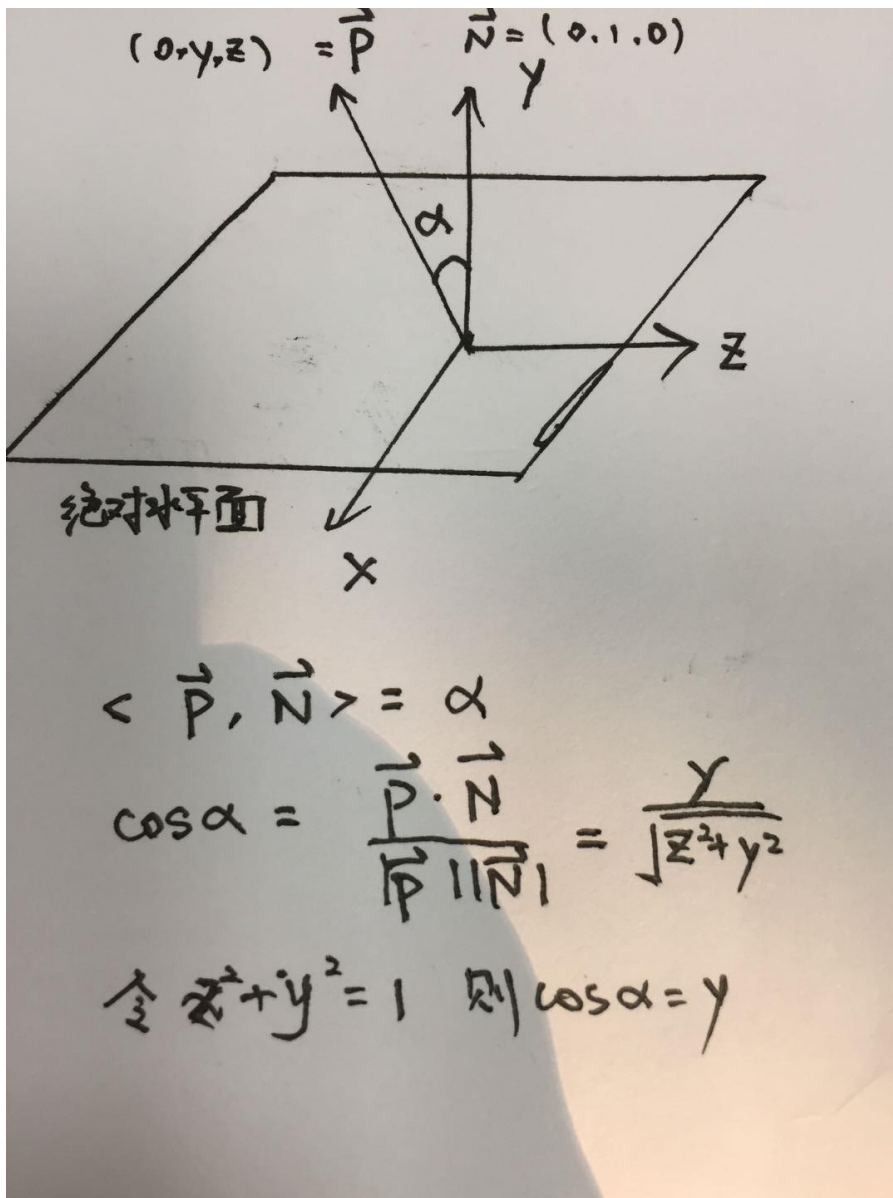
理论设计：

公转轨道倾斜角：是行星运行时的轨道平面与基准水平面之间的夹角，可以通过法线夹角得到，即之前分析过的 `Transform.RotateAround()` 提供了旋转平面的法线作为第二个参数，旋转中心作为第一个参数，在设计 `Planet` 类和 `Satellite` 类的时候就考虑到了参考系无关性，所以在设计公转中心的时候，设计了 `GameObject` 类型的 `pubRotateCenter` 参数，然后通过访问 `pubRotateCenter.position` 来实时获得相对的旋转中心，这样也解决了多层次的父子相互绕转的问题（地球绕太阳转，月球绕地球转，地球的位置实时变化，但是通过 `pubRotateCenter.position` 总是可以实时读到最新的位置，实现相对转动）

借用 Unity API `Transform.RotateAround()`，计算出 8 大行星的运行平面的法线方向：

轨道倾角	名称	黄道倾角	太阳赤道倾斜	不变平面倾角
类地行星	水星	7.01°	3.38°	6.34°
	金星	3.39°	3.86°	2.19°
	地球	0	7.155°	1.57°
	火星	1.85°	5.65°	1.67°
气态行星	木星	1.31°	6.09°	0.32°
	土星	2.49°	5.51°	0.93°
	天王星	0.77°	6.48°	1.02°
	海王星	1.77°	6.43°	0.72°

取不变平面倾角：



经过计算，得到 8 大行星相对于整体空间的旋转轨道的法向量为：

Planet	pubRotateTiltAngle
水星 Mercury	New Vector3(0, 0.992884, -0.01228)
金星 Venus	New Vector3(0, 0.9992696, -0.00146027)
地球 Earth	New Vector3(0, 0.9996246, -0.000751)
火星 Mars	New Vector3(0, 0.9995753, -0.000849)
木星 Jupiter	New Vector3(0, 0.9999884, -0.0000312)
土星 Saturn	New Vector3(0, 0.9998683, -0.000263)
天王星 Uranus	New Vector3(0, 0.99998415, -0.0000317)

海王星 Neptune	New Vector3(0, 0.99992104, -0.0001579)
-------------	--

偏心率：对于设置椭圆轨道，查了一下 Unity 没有提供 API，要实现会非常复杂，使用圆轨道代替。

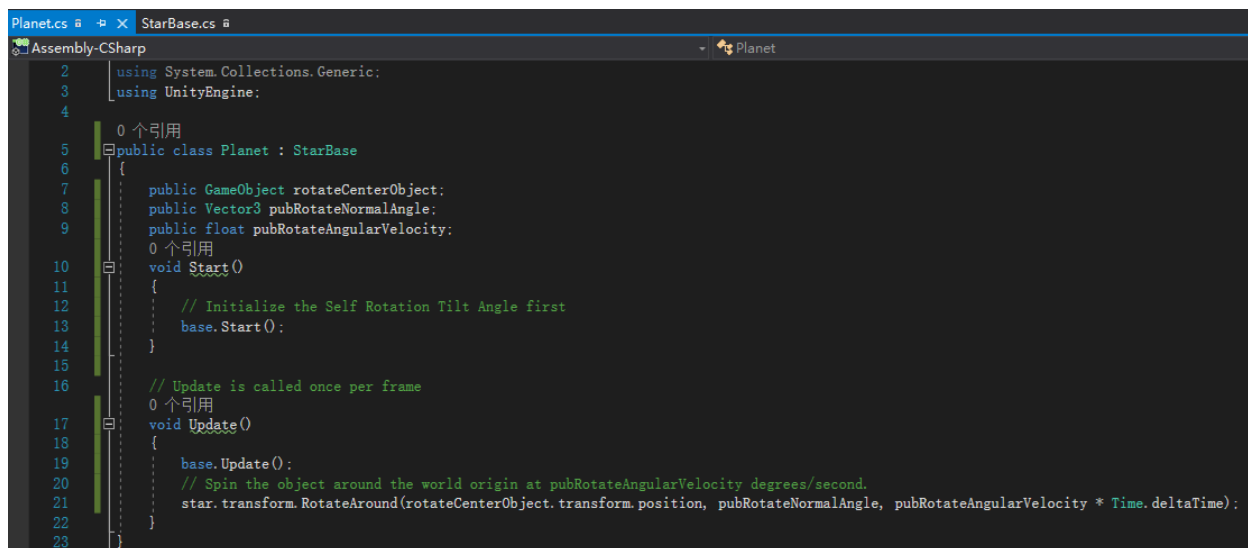
公转转速：借助 Transform.RotateAround()知道最后的一个变量应该为 float 类型，所以设计为一个 float 类型的变量 pubRotateAngularVelocity

编程实现：

于是可以得到公转方法 PubRotate()：

首先要从基类 StarBase 派生出行星类 Planet 和卫星类 Satellite

Planet.cs:



```
Planet.cs  StarBase.cs
Assembly-CSharp Planet
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 0 个引用
6 public class Planet : StarBase
7 {
8     public GameObject rotateCenterObject;
9     public Vector3 pubRotateNormalAngle;
10    public float pubRotateAngularVelocity;
11    0 个引用
12    void Start()
13    {
14        // Initialize the Self Rotation Tilt Angle first
15        base.Start();
16    }
17
18    // Update is called once per frame
19    0 个引用
20    void Update()
21    {
22        base.Update();
23        // Spin the object around the world origin at pubRotateAngularVelocity degrees/second.
24        star.transform.RotateAround(rotateCenterObject.transform.position, pubRotateNormalAngle, pubRotateAngularVelocity * Time.deltaTime);
25    }
26 }
```

Satellite.cs:

```

Satellite.cs Planet.cs StarBase.cs
Assembly-CSharp Satellite
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Satellite : StarBase
6 {
7     public GameObject rotateCenterObject;
8     public Vector3 pubRotateNormalAngle;
9     public float pubRotateAngularVelocity;
10
11 void Start()
12 {
13     // Initialize the Self Rotation Tilt Angle first
14     base.Start();
15 }
16
17 // Update is called once per frame
18 void Update()
19 {
20     base.Update();
21     // Spin the object around the world origin at pubRotateAngularVelocity degrees/second.
22     star.transform.RotateAround(rotateCenterObject.transform.position, pubRotateNormalAngle, pubRotateAngularVelocity * Time.deltaTime);
23 }

```

1.3.9.3 实现卫星（月球）绕地球的旋转

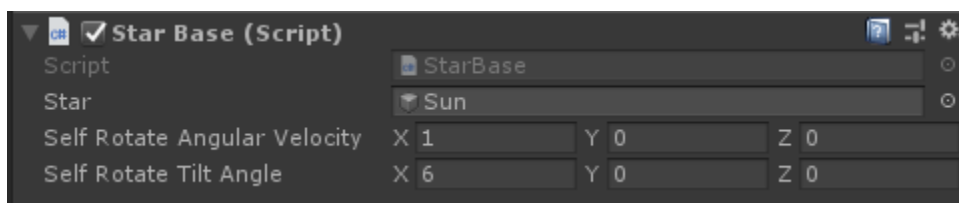
1.3.10 挂载脚本，设置之前得到的各种参数

根据天体物理学知识，知道旋转的角速度和天体于太阳之间的距离的 4 次方成反比

1.3.10.1 太阳 Sun

挂载脚本 StarBase.cs

配置参数：



解释：

恒星（Name）	太阳 Sun
自转角速度（1 unit / sec）	1
自转轴心角（°）	(6,0,0)

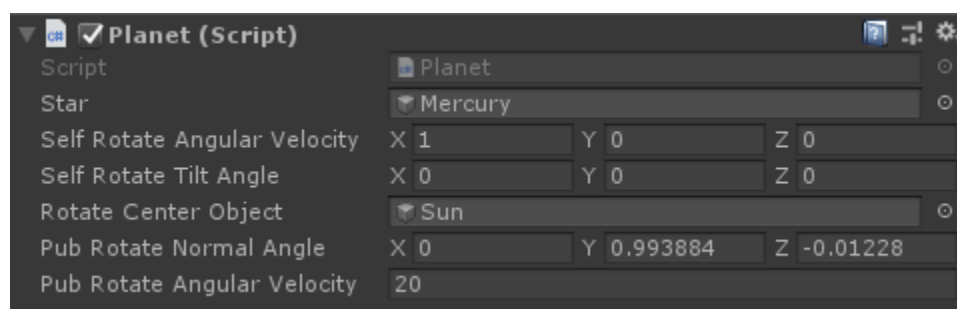
运行效果：



1.3.10.2 水星 Mercury

挂载脚本 Planet.cs

配置参数：

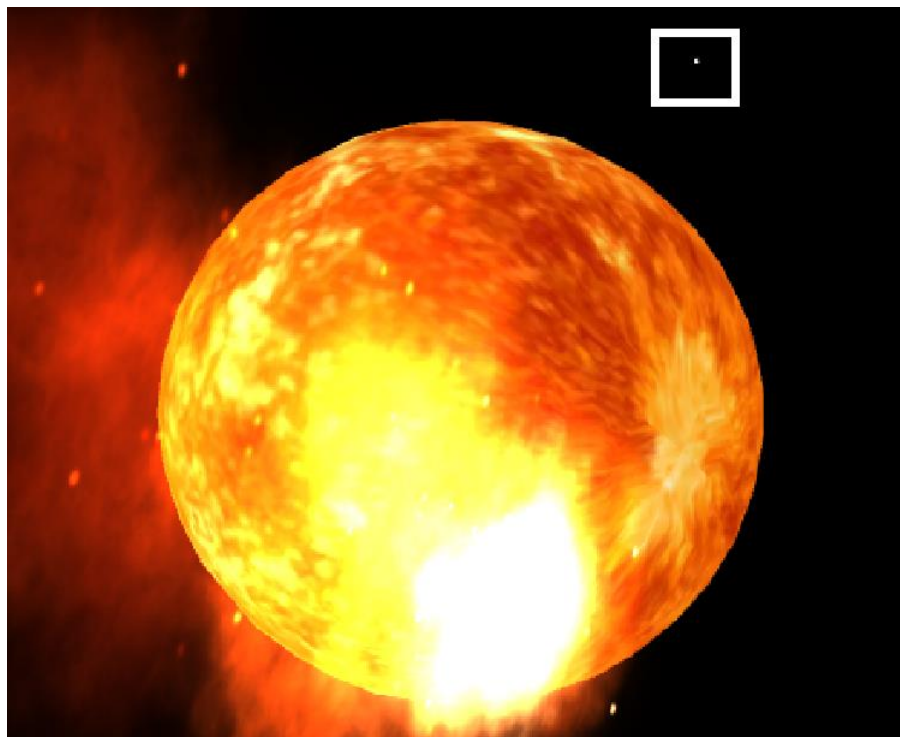


解释：

行星（Name）	水星 Mercury
自转角速度（1 unit / sec）	1

自转轴心角 (°)	(0,0,0)
绕转中心物体 (Name)	Sun
公转平面法向量 (Vector3)	New Vector3(0, 0.9992696, -0.00146027)
公转角速度 (° / sec)	20

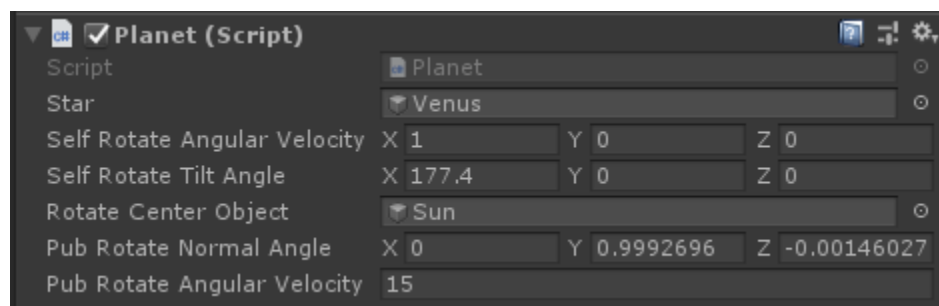
运行效果:



1.3.10.3 金星 Venus

挂载脚本 Planet.cs

配置参数:



解释:

行星（Name）	金星 Venus
自转角速度（1 unit / sec）	1
自转轴心角（°）	177.4
绕转中心物体（Name）	Sun
公转平面法向量（Vector3）	New Vector3(0, 0.9992696, -0.00146027)
公转角速度（° / sec）	15

运行效果：



1.3.10.4 地球 Earth

挂载脚本 Planet.cs

配置参数：

解释：

行星（Name）	地球 Earth
自转角速度（1 unit / sec）	1

自转轴心角 (°)	23.44
绕转中心物体 (Name)	Sun
公转平面法向量 (Vector3)	New Vector3(0, 0.9996246, -0.000751)
公转角速度 (° / sec)	12

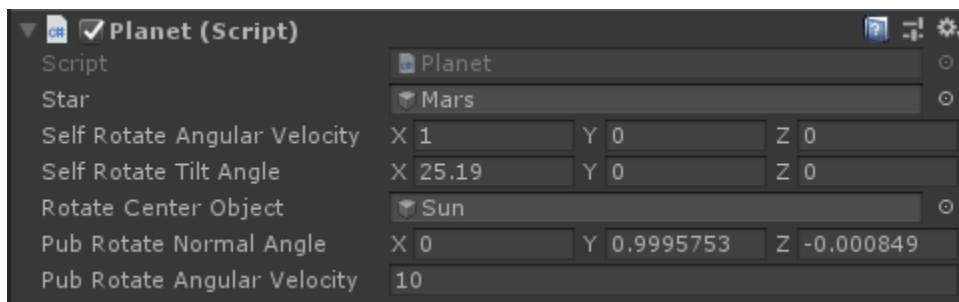
运行效果：



1.3.10.5 火星 Mars

挂载脚本 Planet.cs

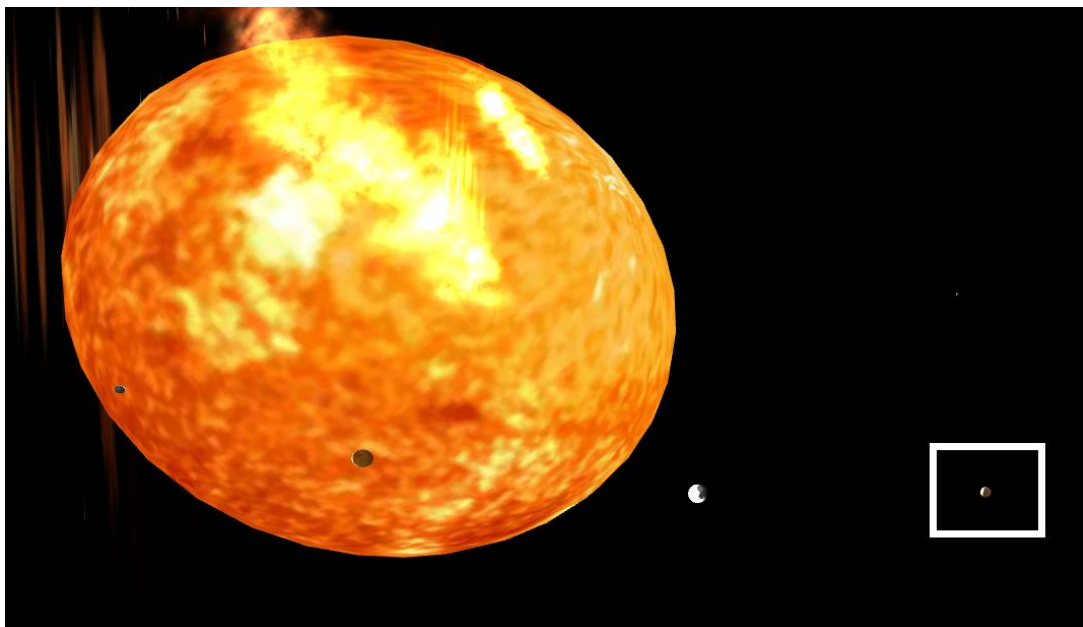
配置参数：



解释：

行星（Name）	火星 Mars
自转角速度（1 unit / sec）	1
自转轴心角（°）	25.19
绕转中心物体（Name）	Sun
公转平面法向量（Vector3）	New Vector3(0, 0.9995753, -0.000849)
公转角速度（° / sec）	10

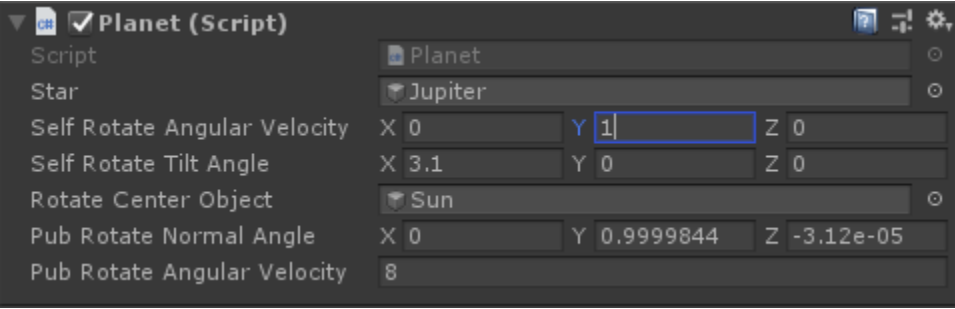
运行效果：



1.3.10.6 木星 Jupiter

挂载脚本 Planet.cs

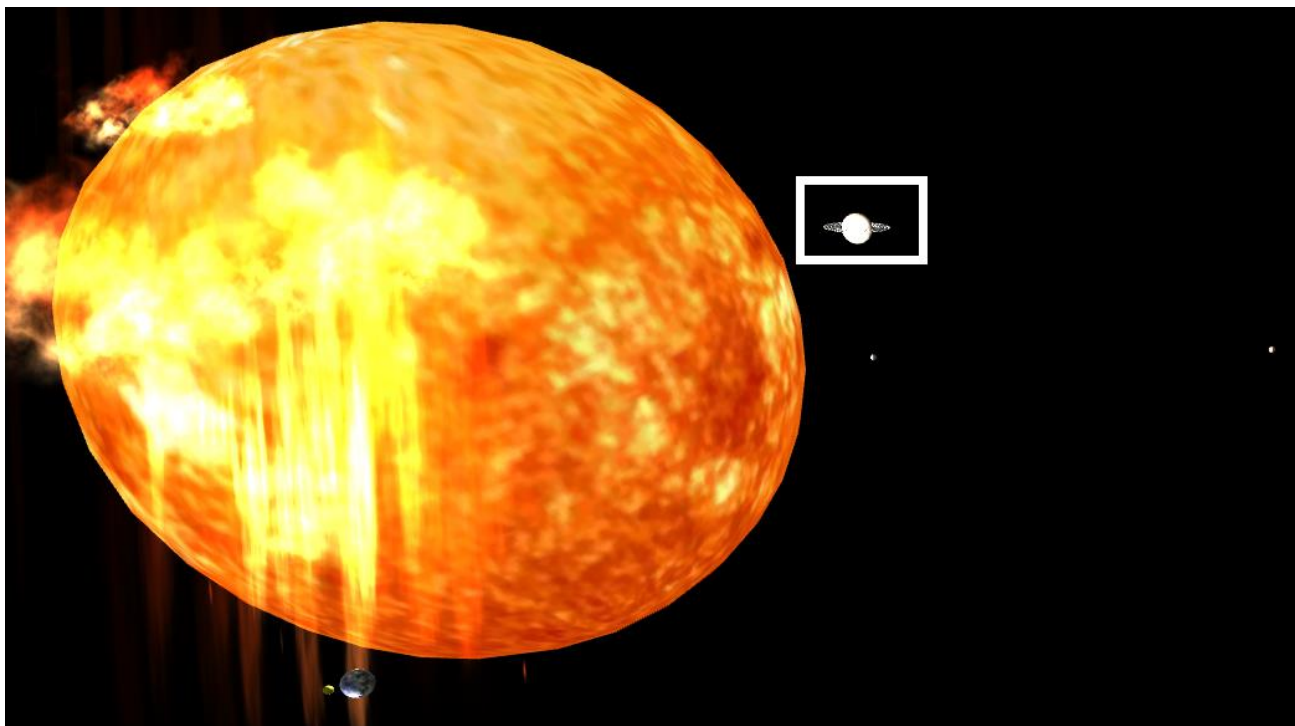
配置参数：



解释：

行星 （Name）	木星 Jupiter
自转角速度 （1 unit / sec）	1
自转轴心角 （° ）	3.1
绕转中心物体 （Name）	Sun
公转平面法向量 （Vector3）	New Vector3(0, 0.9999884, -0.0000312)
公转角速度 （° / sec)	8

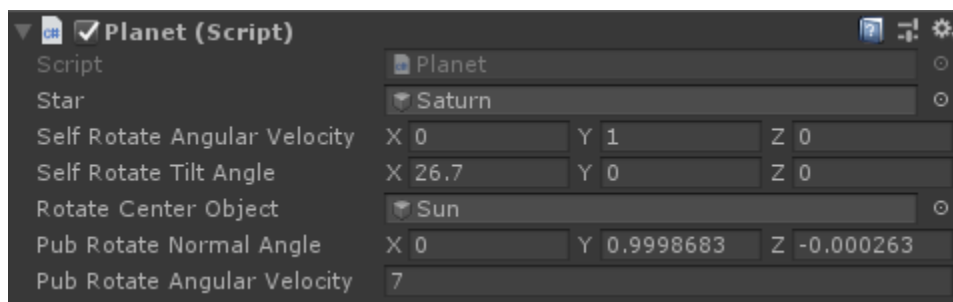
运行效果：



1.3.10.7 土星 Saturn

挂载脚本 Planet.cs

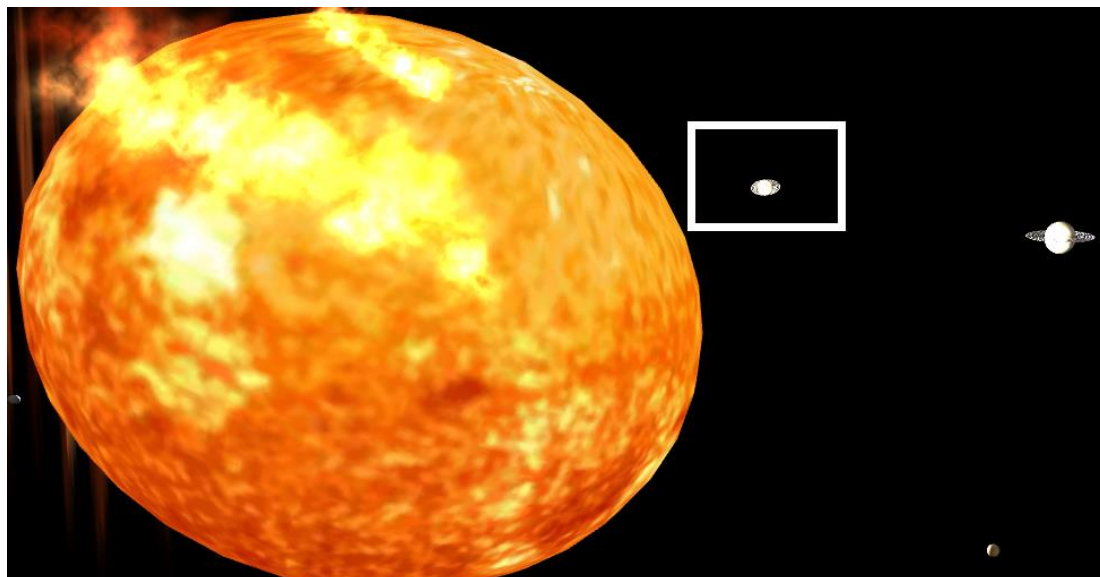
配置参数：



解释：

行星（Name）	土星 Saturn
自转角速度（1 unit / sec）	1
自转轴心角（°）	26.7
绕转中心物体（Name）	Sun
公转平面法向量（Vector3）	New Vector3(0, 0.9998683, -0.000263)
公转角速度（° / sec）	7

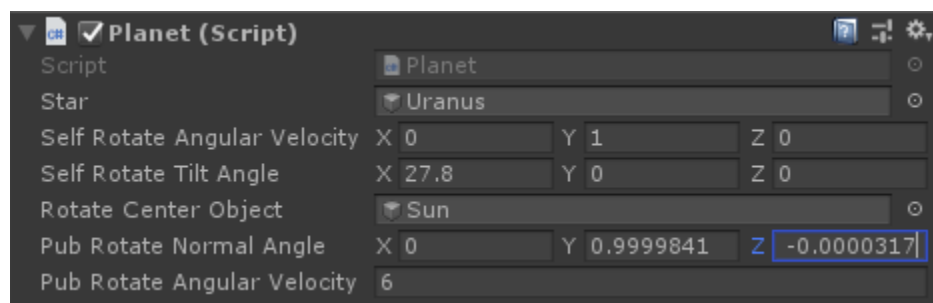
运行效果：



1.3.10.8 天王星 Uranus

挂载脚本 Planet.cs

配置参数：



解释：

行星（Name）	天王星 Uranus
自转角速度（1 unit / sec）	1
自转轴心角（°）	27.8
绕转中心物体（Name）	Sun
公转平面法向量（Vector3）	New Vector3(0, 0.99998415, -0.0000317)
公转角速度（° / sec）	6

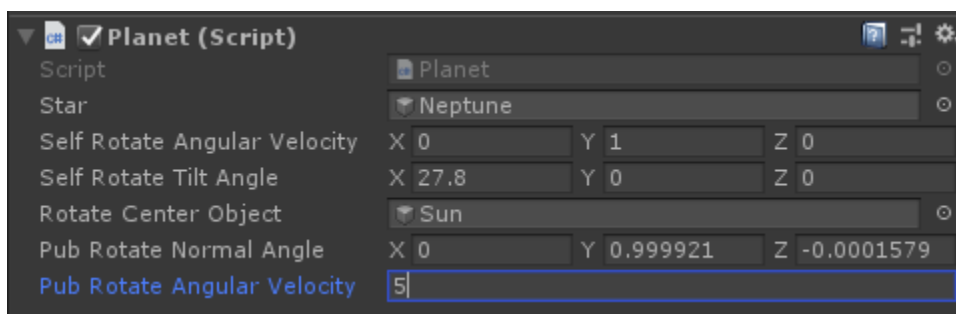
运行效果：



1.3.10.9 海王星 Neptune

挂载脚本 Planet.cs

配置参数：

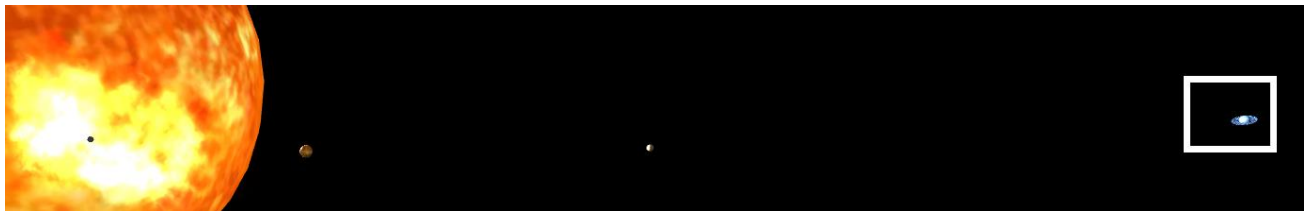


解释：

行星（Name）	海王星 Neptune
自转角速度（1 unit / sec）	1
自转轴心角（°）	27.8
绕转中心物体（Name）	Sun
公转平面法向量（Vector3）	New Vector3(0, 0.99992104, -0.0001579)

公转角速度 ($^{\circ}$ / sec)	5
---------------------------	---

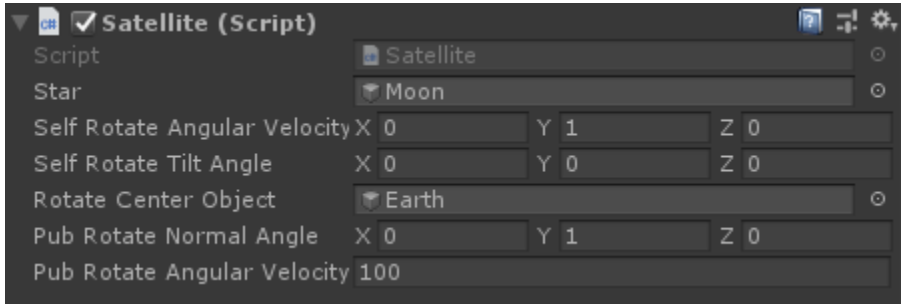
运行效果：



1.3.10.10 月球 Moon

挂载脚本 Satellite.cs

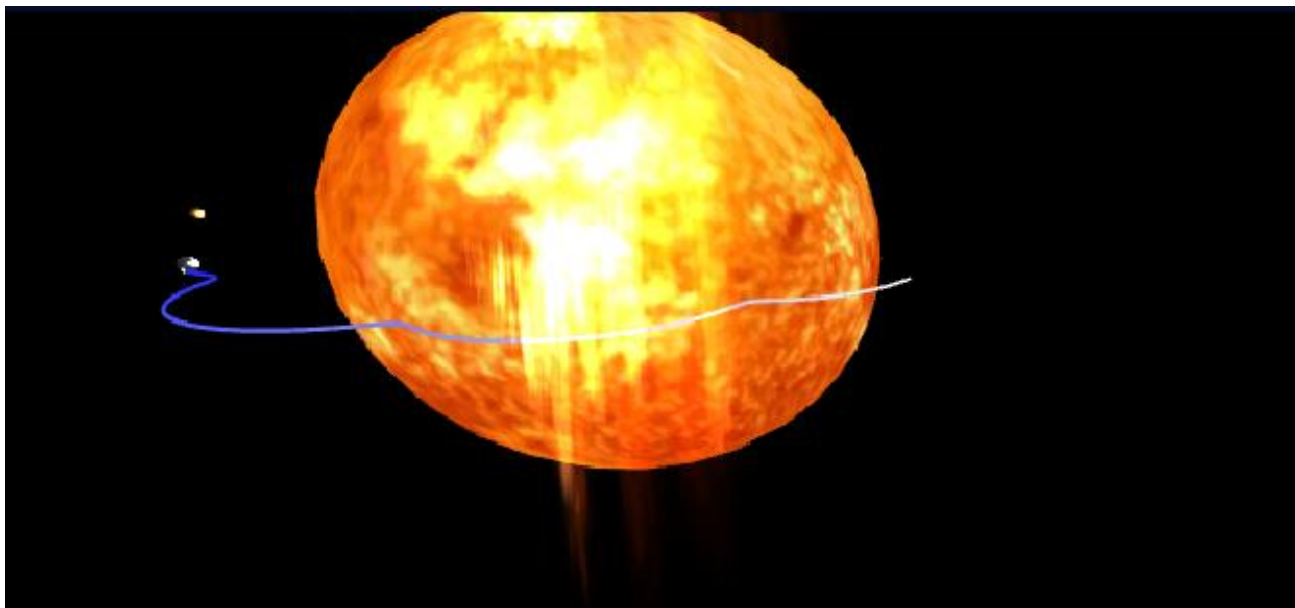
配置参数：



解释：

行星 (Name)	月球 Moon
自转角速度 (1 unit / sec)	1
自转轴心角 ($^{\circ}$)	0
绕转中心物体 (Name)	Earth
公转平面法向量 (Vector3)	New Vector3 (0, 0, 0)
公转角速度 ($^{\circ}$ / sec)	100

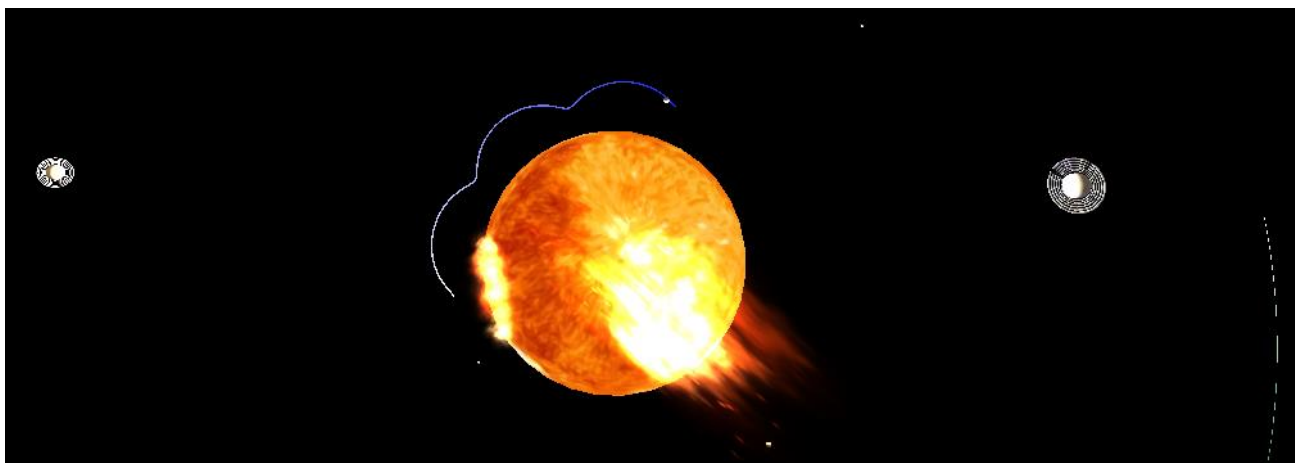
运行效果：



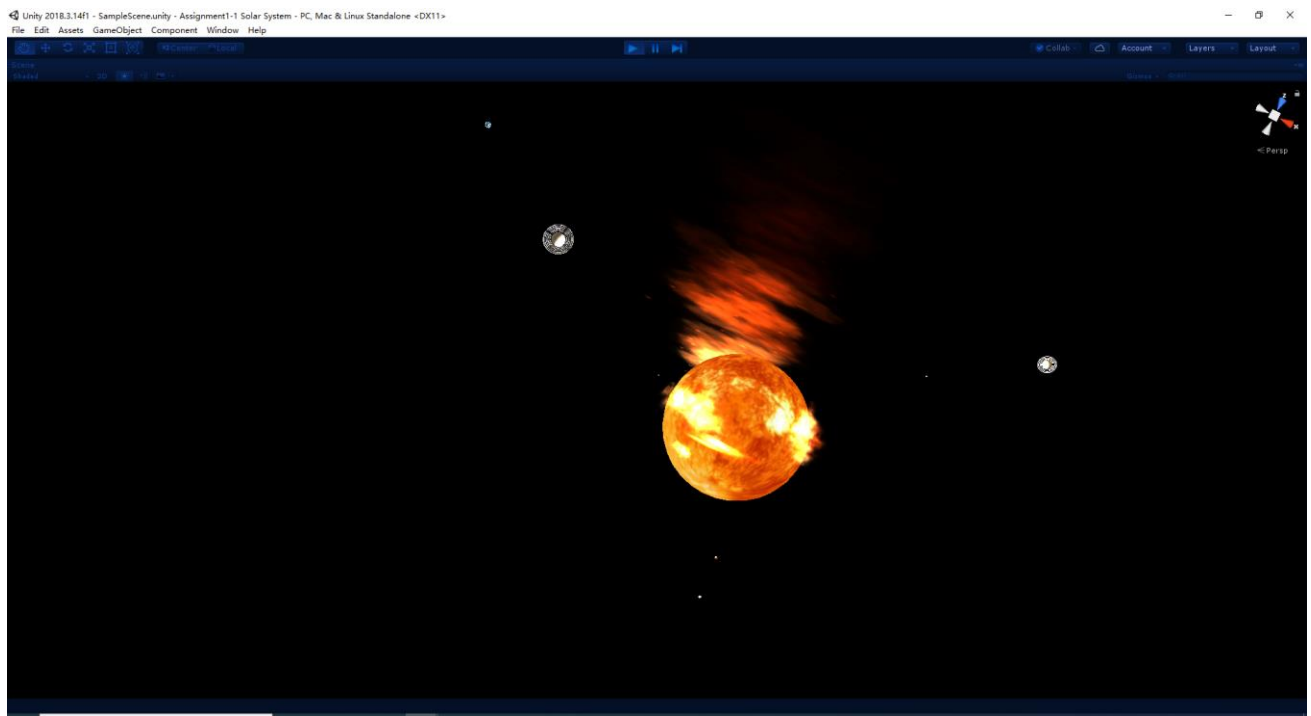
由于月球还在绕地球旋转，所以轨迹不是纯圆形，而是有曲折的螺线

1.4 场景 最终效果

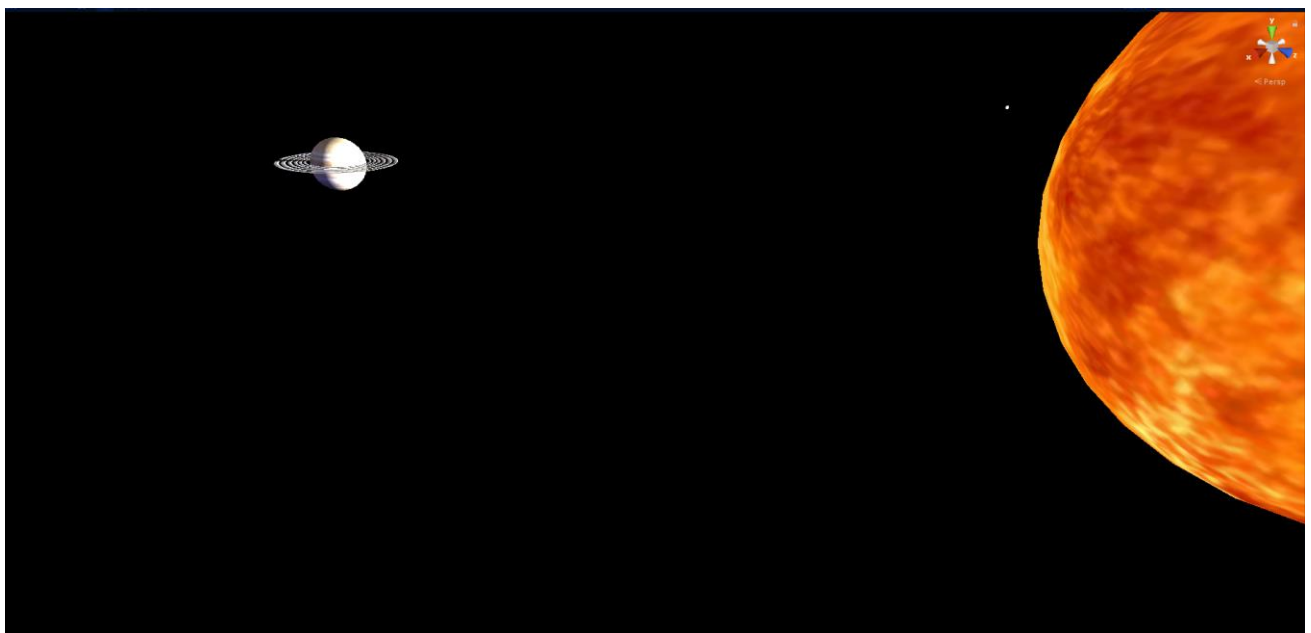
俯视：



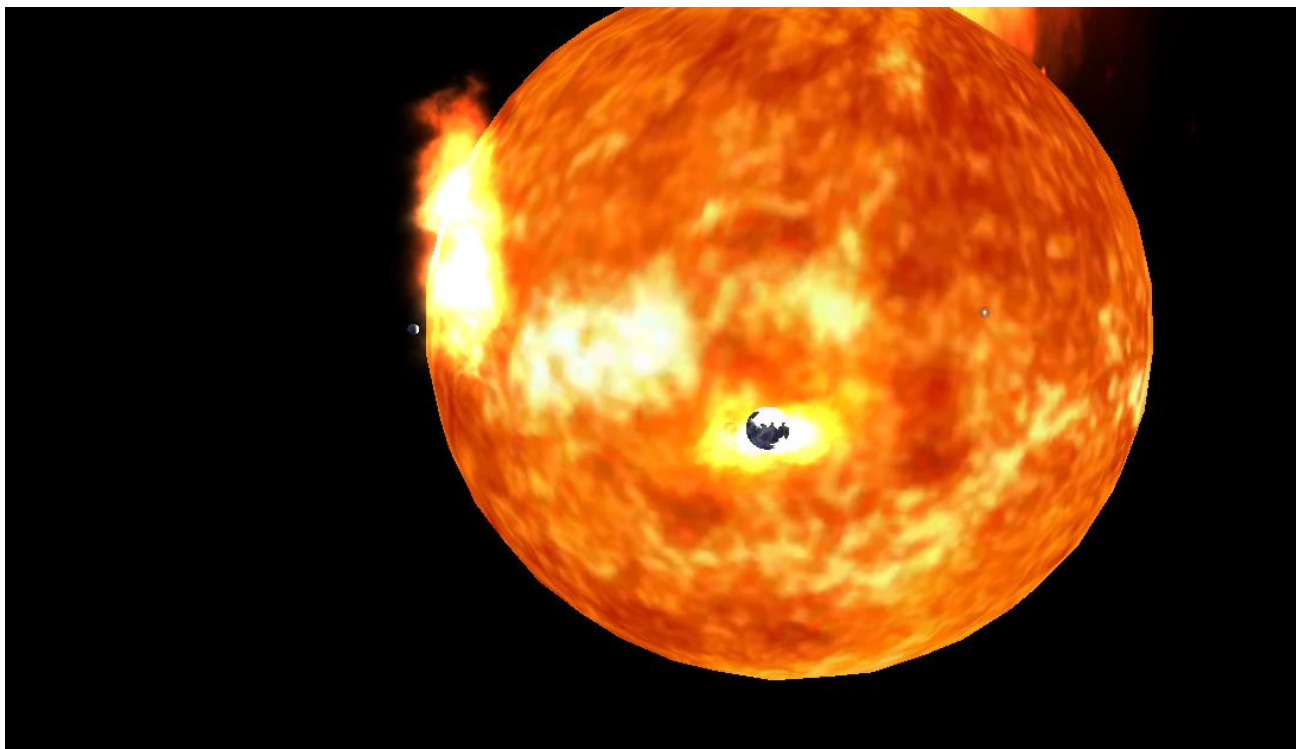
上图中给 Moon 加了 Effect->Trail（轨迹），以及给天王星 Uranus 加了轨迹。



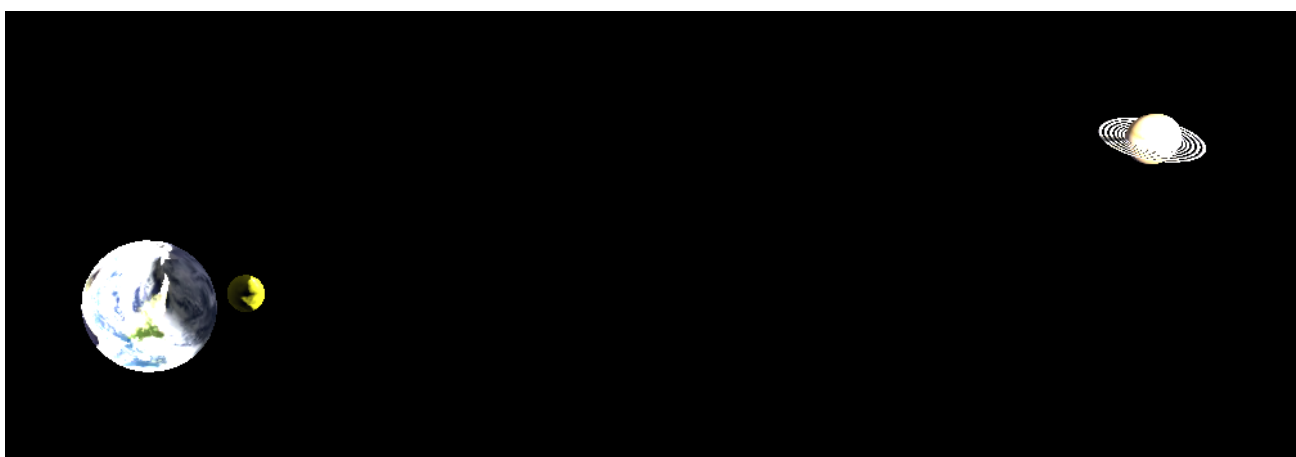
自由视角：



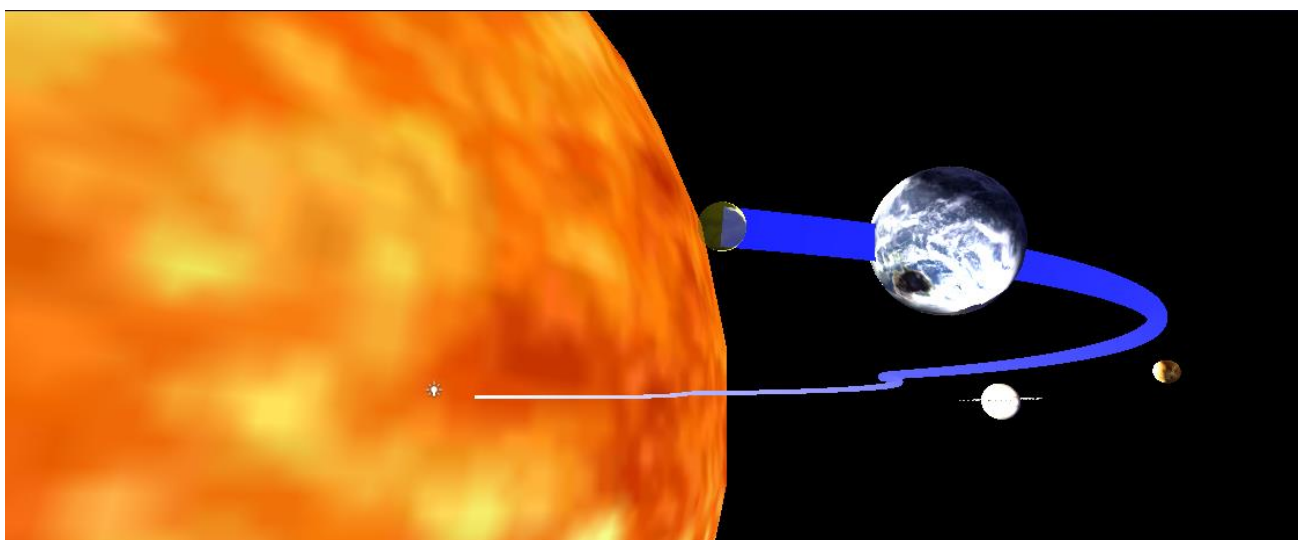
地球掠过：



地球跟踪：



月球围绕：



太阳系全景：

