暑培-Unity3D

SAST

2023 年 8 月 5 日

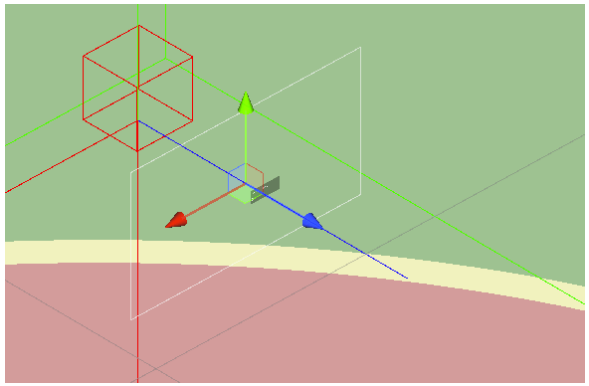
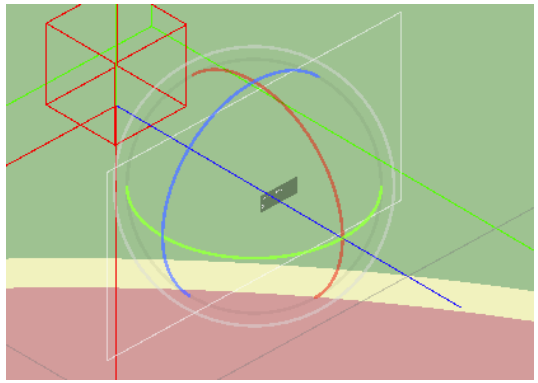
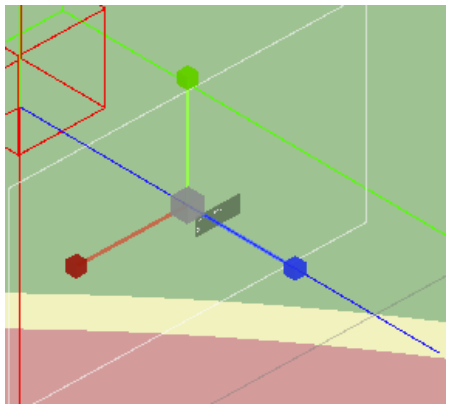
目录

当前进度

基本概念

Unity 以物件的中心坐标、旋转和缩放唯一确定其各个顶点的位置，存放在 Transform 类里。

- 中心坐标：三维矢量 (.position)。
- 旋转：四元数 (.rotation)。
- 缩放：三维数组 (.lossyscale)。* 建议用相对缩放处理

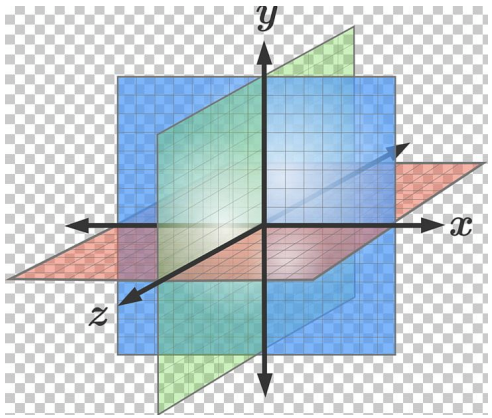




绝对坐标

对于每个场景，Unity 定义了一个世界坐标系。
类比：经度、纬度的概念。
操作方式：

```
// 读
Vector3 glob_pos = gameobj.transform.position;
// 写
gameobj.transform.position = (Vector3) ...;
```

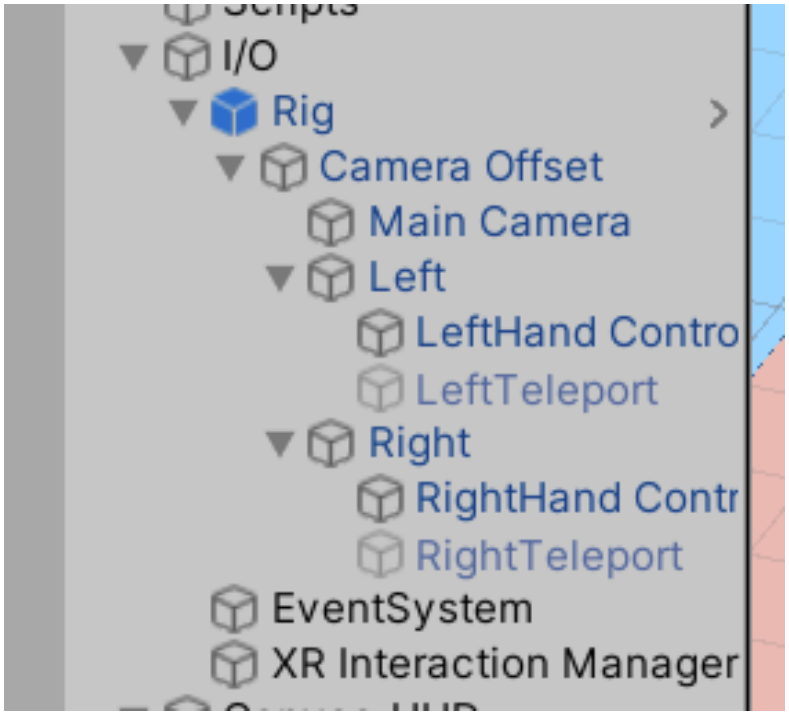
需要注意的是，纵向坐标被表示为 y 轴，而与之垂直的平面是 xOz。



相对坐标

GameObject 嵌套

Unity 可以在一个 GameObject 内嵌套多个其他 GameObject



相对坐标

GameObject 嵌套

Unity 可以在一个 GameObject 内嵌套多个其他 GameObject

相对坐标，指的是该字段记录的坐标，是相对父物件的坐标，那就是把坐标系设置在父物件的坐标。使用方法：

```
// 读
Vector3 local_pos = gameobj.transform.localPosition;
// 写
gameobj.transform.localPosition = (Vector3) ...;
```

本质上就是在Transform 类有关字段前面加上 local。
类似的有相对旋转、相对缩放，概念也差不多，本质上就是一些矩阵运算的技巧。

平移

设平移量为 $\vec{v}$，则

$$\vec{x}' = \vec{x} + \vec{v}$$

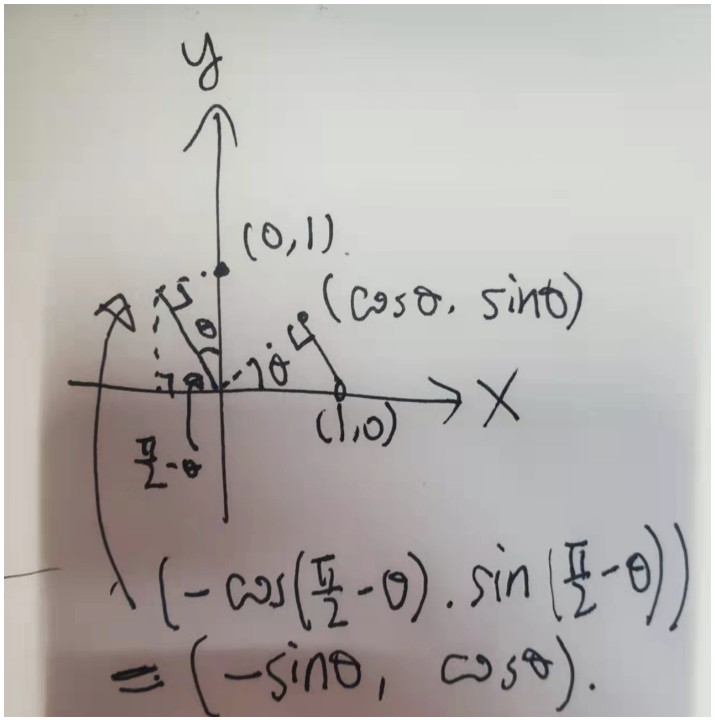
代码实现：

```
transform.Translate(v);
```

请注意：Translate 函数有个可选参数，选择平移相对的坐标系。

旋转的表示

二维平面下的旋转：
对于平面的标准坐标系，其基向量为 $\vec{e}_1 = \langle 1, 0 \rangle, \vec{e}_2 = \langle 0, 1 \rangle$.



$$\mathrm{Rot}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

三维空间的旋转

- 绕 z 轴:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- 绕 y 轴:

$$\begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

- 绕 x 轴:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

- 如何便捷的表示绕任意轴旋转呢？

四元数

游戏引擎普遍用四元数做旋转运算，原因有二：

1. 轴、角表示很容易被转化为四元数。
2. 可以有效避免在运算过程出现万向节死锁 (Gimbal Lock) 的问题。

设有单位旋转轴为 $\vec{u}$，以右手定则决定的旋转角度为 θ，则

$$\vec{q} = \cos(\theta/2) + \vec{u}\sin(\theta/2)$$

如果要旋转某个点 $\vec{x}$，可以用这种方法：

$$\vec{x}' = \vec{q}\vec{x}\vec{q}^*$$

理解起来很抽象，其实可以类比复平面上的旋转：

$$\exp(i\theta) = \cos\theta + i\sin\theta$$

旋转的做法

对一般人而言，由于理解四元数过于复杂，可以直接利用 Unity 提供的接口进行操作。

```
transform.RotateAround(point, axis,angle);
```

- point: 旋转轴穿过的点。
- axis: 旋转轴。
- angle: 旋转角。

当前进度

1. Unity 坐标系统
   - 绝对坐标与相对坐标
   - 旋转
2. 材质
   - 网格
   - 纹理
   - 渲染器与碰撞体
   - 具体实现
3. 相机
   - 透视相机
   - 正交相机

网格

计算机对立体物件的表示：

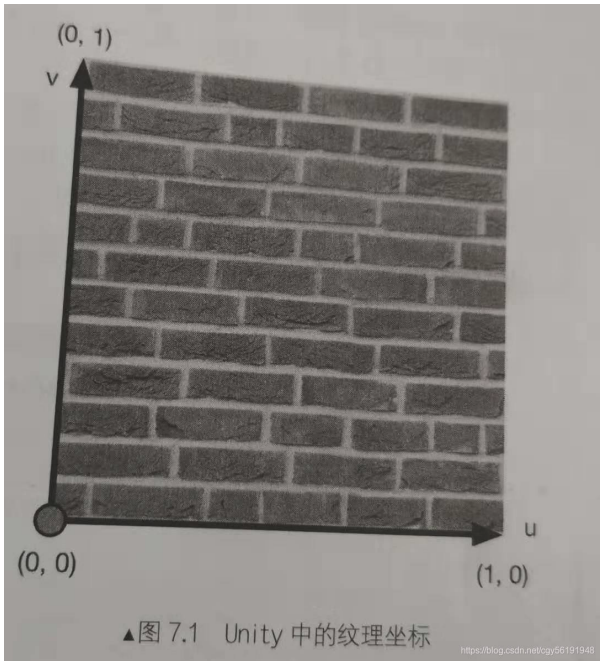
1. 列出该物件的所有顶点。
2. 按实际需要把这些顶点连成多个三角形。
3. 对各个三角形贴上纹理。
4. 计算各个三角形的光照属性（可选）。

例子：(obj 文件) 表示单个方块

```
v -1 -1 -1
v 1 -1 -1
v -1 1 -1
v 1 1 -1
...
f 1 3 4
...
```

纹理映射

好比你拿着高达贴纸贴在模型上面。
我们假设那张图能连续表示，放在二维平面上四个角落为 $(0,0),(0,1),(1,1),(1,0)$。



▲图 7.1 Unity 中的纹理坐标

对 Mesh 类修改 `mesh.uvs` 数组即可。

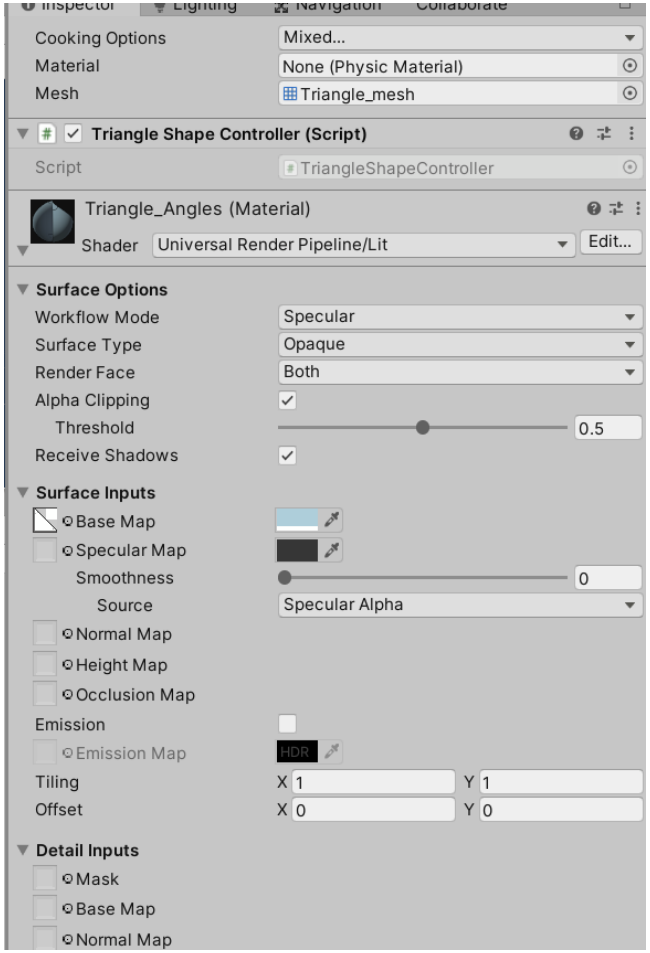
网格渲染器

Mesh Renderer，配合 Mesh Filter 使用，可以在场景把网格绘制出来。
对于没有贴材质的网格，Unity 显示物体为紫色。

网格碰撞体

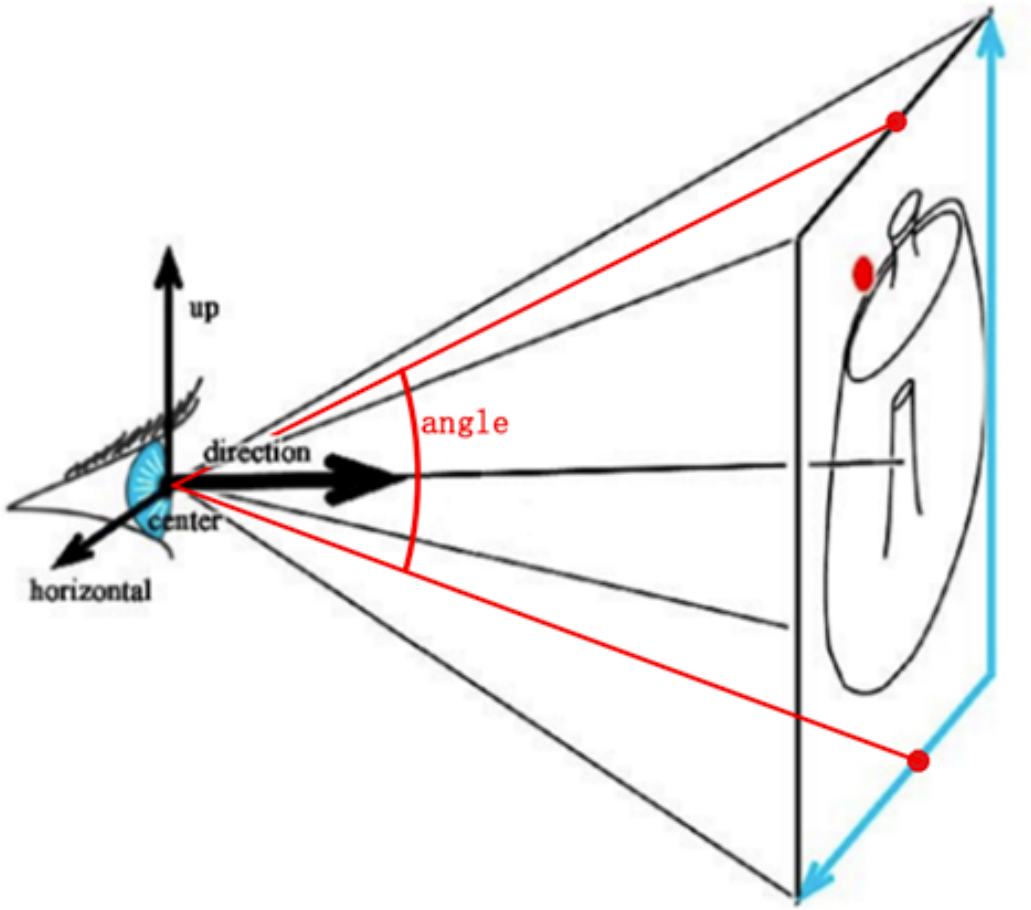
既然网格可以定义物体表面，那就说明其可以用这些表面是否出现相交的情况，判断两个物体间是否出现碰撞。
对物体挂接 MeshCollider，可以在脚本复写 OnCollisionEnter() 函数，实现对物体碰撞的探测。

Unity 材质控制

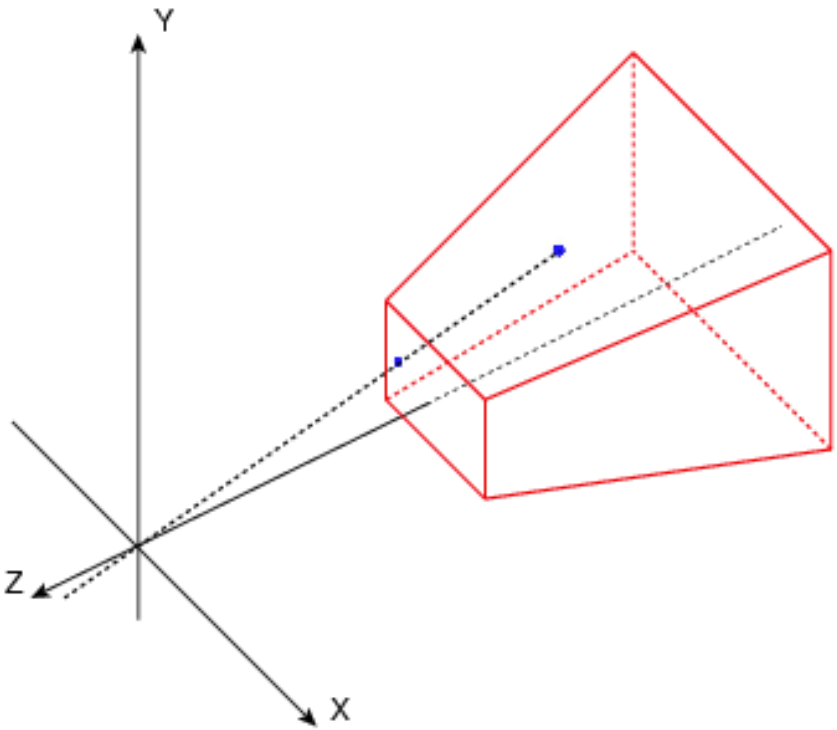

Cooking Options
Mixed...
Material
None (Physic Material)
Mesh
Triangle_mesh
Triangle Shape Controller (Script)
Script
TriangleShapeController
Triangle_Angles (Material)
Shader
Universal Render Pipeline/Lit
Edit...
Surface Options
Workflow Mode
Specular
Surface Type
Opaque
Render Face
Both
Alpha Clipping
Threshold
0.5
Receive Shadows
Surface Inputs
Base Map
Specular Map
Smoothness
0
Source
Specular Alpha
Normal Map
Height Map
Occlusion Map
Emission
Emission Map
HDR
Tiling
X 1
Y 1
Offset
X 0
Y 0
Detail Inputs
Mask
Base Map
Normal Map

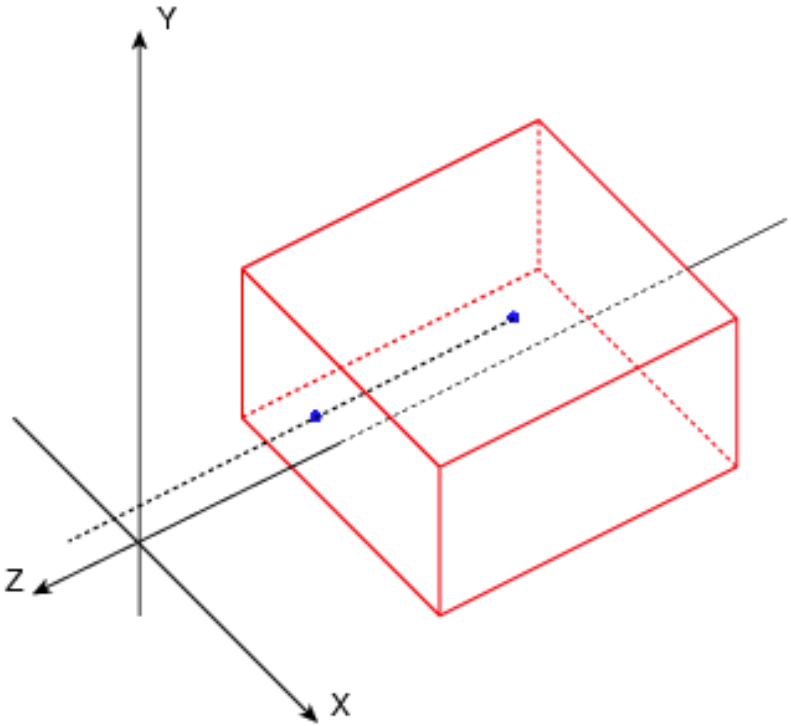
当前进度

透视相机



正交相机





目录

结束