### 坐标系：

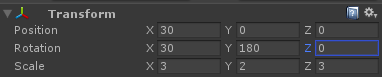
unity的本地坐标系和世界坐标系都是左手坐标系，旋转使用左手法则

在unity中，每个scene都有唯一存在的一个世界坐标系，**世界坐标系也是一个左手坐标系**

变换组件的所有属性都是相对其父物体进行衡量的，如果此物体没有父物体，这些属性则相对于世界坐标进行计算

新创建的物体O的局部坐标系：原点为其质心，轴向与其上一级物体的本地坐标系轴向相同。O平移、旋转时，其局部坐标系跟着平移、旋转。

下图中显示的position、rotation、scale都是物体在其本地坐标系中的坐标：



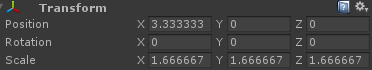
绕x轴转：y->z

绕y轴转：z->x

绕z轴转：x->y

且为静态欧拉角，使用的是Z-X-Y顺规

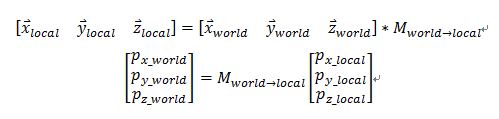
例如，立方体(30，10，0)，尺寸3\*3\*3，无旋转；球体(40，10，0)，尺寸5\*5\*5。将球放置于立方体的下一级，则球的信息变为：



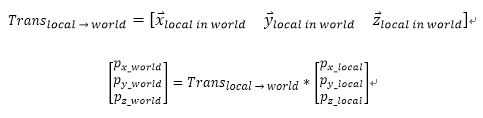
要将本地坐标系的点、向量等转换到世界坐标系中：

使用gameObject.transform.localToWorldMatrix得到矩阵，该矩阵既是点p从本地坐标系变换到世界坐标系的变换矩阵，又是world基到local基的过渡矩阵

world基到local基的过渡矩阵的定义如下：



点p从本地坐标系变换到世界坐标系的变换矩阵：



**点P在坐标系V下的坐标:相当于其在坐标系U下的坐标，左乘矩阵T，其中T为 坐标系U的三个基（列向量）在坐标系V下的坐标构成的矩阵**

**点P在世界坐标系下的坐标=T\*其在局部坐标系下的坐标，其中T为局部坐标系的三个基（列向量）在世界坐标系下的坐标构成的矩阵**

采用以下两种方式进行坐标变换后，点p的坐标相同：

(1)坐标系不动，点p进行 T变换，即 p的新坐标列向量=T\*p的旧坐标列向量

(2)点p不动，坐标系进行T-1变换，即变换后x轴在原坐标系中的位置为T-1\*[1 0 0列向量]、变换后y轴在原坐标系中的位置为T-1\*[0 1 0列向量]、

变换后z轴在原坐标系中的位置为T-1\*[0 0 1列向量]

### 常用快捷键：

旋转视角快捷键：alt-鼠标左键-移动鼠标

### 观察导入的模型属性：

检视器中各属性的含义如下：

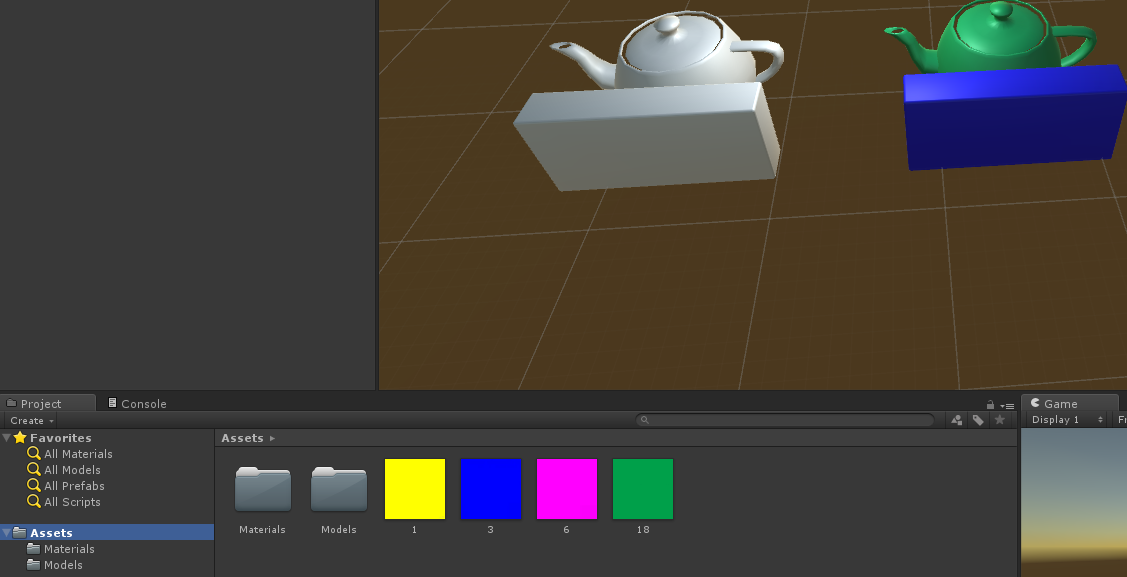
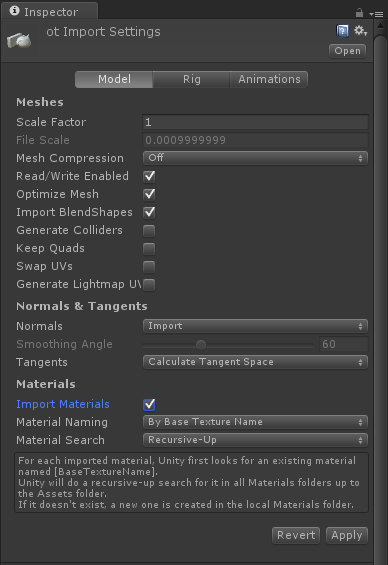
Meshes网格

* Scale Factor：缩放系数。如果导入的模型大小不一样，可通过它调整缩放系数。

Material材质

* Import Naming：导入材质。勾选该项，系统将用默认的漫反射材质取代FBX导入时的材质。

默认勾选Import Naming，这时不能再将图片贴至模型；不勾选，则可以贴图至模型



### 获取模型尺寸的方法：

为模型添加script，在void Start(){}函数体中编写代码：

Vector3 originSize = gameObject.GetComponent<MeshFilter>().mesh.bounds.size; //通过MeshFilter获得模型的mesh，最后返回模型在Unity的原始尺寸

//要获得在Unity场景中的实际尺寸大小还需要乘以模型的localScale.x

float x = originSize.x \* gameObject.transform.localScale.x;

float y = originSize.y \* gameObject.transform.localScale.y;

float z = originSize.z \* gameObject.transform.localScale.z;

Debug.Log("模型在Unity的原始尺寸："+originSize+"\n");

Debug.Log("模型的localScale:" + gameObject.transform.localScale);

Debug.Log("在unity场景中的实际尺寸："+ x +","+y+","+z+"\n");

测试：

1. 在3DMax中建一个80\*40\*20cm长方体模型，保存为FBX格式，导入Unity，导入时设置scale factor为1，则

模型在Unity的原始尺寸：0.4 0.8 0.2

模型的localScale： 1 1 1

在unity场景中的实际尺寸： 0.4 0.8 0.2

1. 在3DMax中建一个80\*40\*20cm长方体模型，保存为FBX格式，导入Unity，导入时设置scale factor为10，则

模型在Unity的原始尺寸：4 8 2

模型的localScale：1 1 1

在unity场景中的实际尺寸： 4 8 2

1. 在3DMax中建一个80\*40\*20cm长方体模型，保存为FBX格式，导入Unity，导入时设置scale factor为10，将Transform-Scale设置为2，2，2则

模型在Unity的原始尺寸： 4 8 2

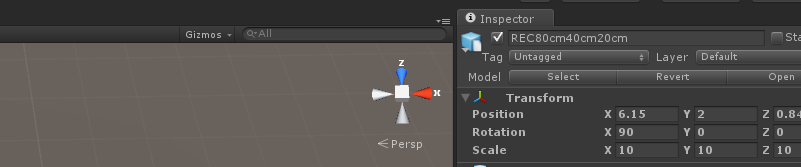
模型的localScale： 2 2 2

在unity场景中的实际尺寸： 8 16 4

导入3DMax FBX模型至Unity时，设置Meshes-Scale Factor，则Unity场景中出现的物体的实际尺寸是模型自身尺寸的Scale Factor倍，“模型在Unity的原始尺寸”。再乘以Transform-Scale，即模型在Unity的实际尺寸

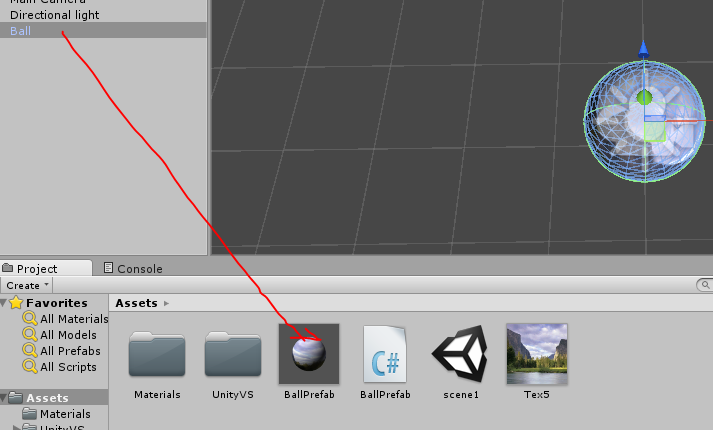
“在模型资源的导入属性中，scale Factor影响着模型的在unity的原始尺寸”

Unity系统单位为米，视图窗口中一个网格大小为1m\*1m，Transform-Position显示的(6.15,2,0.84)即6.15m、2m、0.84m



### Prefab：

Hierarchy View中创建一个球形物体；Asset View中创建一个prefab；将将检视器中的Ball拖动到资源视图中的prefab上。则创建好了一个预设。向场景添加一个预设 (Prefab) 时，就会创建它的一个**实例。**之后在Asset视图修改预设，则场景中预设的所有实例都随之变化



脚本也可以像贴图一样直接拖到gameObject上，成为gameObject的一个component

### Debug.Log

Debug.Log()函数来打印程序中的变量值，默认显示的小数位数只有一位，像0.0xxxxx这种小数就成了0.0!!!!!!

Debug.Log函数把Vector转成字符串的时候进行了截断。

用下面的方法可以控制精度：

Debug.Log(vec.ToString("f4"))

### 代码：

public class InputPlayer : MonoBehaviour {

// Use this for initialization

void Start () {

Debug.Log("start!....go");

}

// Update is called once per frame

void Update () {

// bool isWDown = Input.GetKeyDown(KeyCode.W);

Rigidbody rb = gameObject.GetComponent<Rigidbody>();

bool isWDown = Input.GetKey(KeyCode.W);

if (isWDown == true)

{

rb.AddForce(0, 0, 10);

Debug.Log(isWDown);

}

bool isSDown = Input.GetKey(KeyCode.S);

if (isSDown == true)

{

rb.AddForce(0, 0,-10);

Debug.Log(isSDown);

}

}

}

using UnityEngine;

using System.Collections;

//相机随着player目标运动，即两者之间的相对位置不变

public class FllowoSpherePlayer : MonoBehaviour {

public GameObject player; //添加player作为当前目标的组件

Vector3 posPlayer; //player位置

Vector3 differPos;

void Start () {

Vector3 initPos= gameObject.transform.position; //相机初始位置

posPlayer = player.transform.position; //player位置

differPos = initPos - posPlayer;

}

void Update () {

posPlayer = player.transform.position;

gameObject.transform.position = posPlayer + differPos;

}

}