<http://blog.csdn.net/honey199396/article/details/51423630>

<https://www.cnblogs.com/diamond/p/4168083.html>

<https://www.baidu.com/link?url=M2xiF0ozsSwrujp_S3QoPYjZp76YbxRdPU7T050ZfnritIOu-GYufiqUOh1P1O8xU6Y3o-6BqYJB5zC51aVF1Ir0LTb_RW1r7-rteHs4On_HoWT9jqwjtM_rpbK3-wxj&wd=&eqid=f657be880005caab000000035b4336a6>

**ScreenToWorldPoint**

使用Unity3D ScreenToWorldPoint(Vector3 position)时，如果将z坐标设置为零的话，那么转换的结果可能是出错，实际上此时得到的是摄像机坐标，会变成摄像机的位置，所以想让结果正确要将position的z坐标设置为非零的值，比如100，或者是-100。

Screenspace is defined in pixels. The bottom-left of the screen is (0,0); the right-top is ([pixelWidth](https://docs.unity3d.com/ScriptReference/Camera-pixelWidth.html),[pixelHeight](https://docs.unity3d.com/ScriptReference/Camera-pixelHeight.html)). **The z position is in world units from the camera.**

**注意：上面的参数z值是距离相机的z值，ScreenToWorldPoint得到的结果是相机的世界坐标+距离相机的坐标，如果距离相机的坐标等于0，则得到的结果是相机的世界坐标**

public void OnDrawGizmos()

{

Vector3 v = new Vector3(0, 0, 550f);

Vector3 v1 = new Vector3(290, 80.4f, 125);

camera = Camera.main;

Vector3 worldPosition = camera.ScreenToWorldPoint(Input.mousePosition + v);

Vector3 screenPosition = camera.WorldToScreenPoint(v1);

Gizmos.color = Color.red;

Gizmos.DrawLine(camera.transform.position, worldPosition);

Debug.Log("--------------------------------------------------");

Debug.Log("camera.aspect = " + camera.aspect);

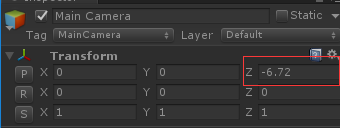
Debug.Log("camera.projectionMatrix = " + camera.projectionMatrix);

Debug.Log("Input.mousePosition = " + (Input.mousePosition + v));

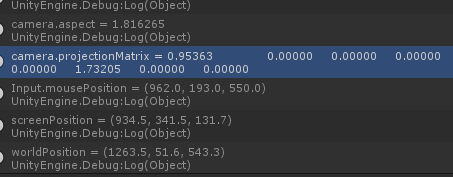
Debug.Log("screenPosition = " + screenPosition);

Debug.Log("worldPosition = " + worldPosition);

}







摄像机的fov角度为60度，因此投影矩阵

第一行：1：cot(60/2)/aspect=1.732/1.816265=0.9536

第二行：2：cot(60/2)=1.732

第三行：3：(0.3+1000)/(0.3-1000)=-1.0006 4：2\*0.3\*1000/(0.3-1000)=-0.60018

第四行：3：-1

最终得到的结果和camera.projectionMatrix一模一样

**ScreenPointToRay**

其中得到的Ray中的

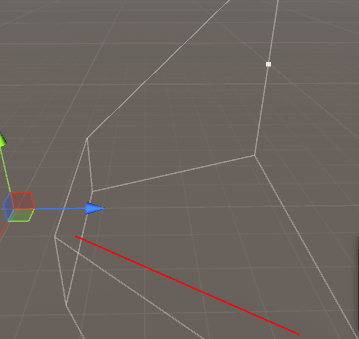
Ray.origin是摄像机与近平面的交点

Ray.direction是方向



注意：这里得到的Ray是世界坐标系中的





实际上我们发现Ray.origin确实是摄像机与近平面的交点（并且是世界坐标系中的，通过上面的z值就可以发现）

camera = Camera.main;

Ray ray = camera.ScreenPointToRay(Input.mousePosition);

Debug.Log("ray.origin = " + ray.origin);

Debug.Log("ray.direction = " + Vector3.Normalize(ray.direction));

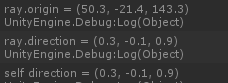
Gizmos.color = Color.red;

Gizmos.DrawLine(ray.origin, ray.origin + 500 \* ray.direction);

Vector3 worldPosition = camera.ScreenToWorldPoint(Input.mousePosition + new Vector3(0, 0, 400));

Vector3 direction = worldPosition - camera.transform.position;

Debug.Log("self direction = " + Vector3.Normalize(direction));



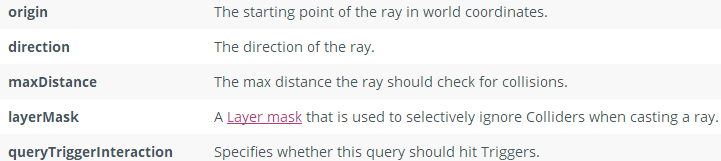
实际上我们可以将屏幕上的点通过ScreenToWorldPoint（注意不要让z值等于0）转换为世界坐标系上的点

然后通过摄像机的起点与转换后的世界坐标系上的点得到方向，最终得到的方向与通过

Camera.main.ScreenPointToRay(Input.mousePoint)的结果一样

**Physics.Raycast**





RaycastHit hit;

camera = Camera.main;

Ray ray = camera.ScreenPointToRay(Input.mousePosition);

if (Physics.Raycast(ray.origin, ray.direction, out hit, Mathf.Infinity, 1 << 0))

{

Gizmos.color = Color.red;

Gizmos.DrawLine(camera.transform.position, hit.point);

Debug.Log("hit.transform.position = " + hit.transform.position);

Debug.Log("hit.distance = " + hit.distance);

Debug.Log("hit.length = " + Vector3.Distance(ray.origin, hit.point));

}

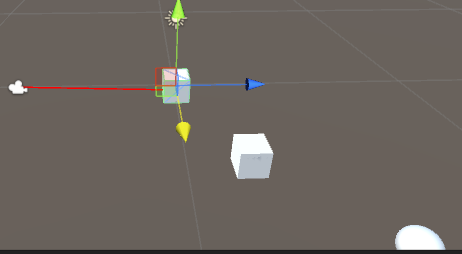
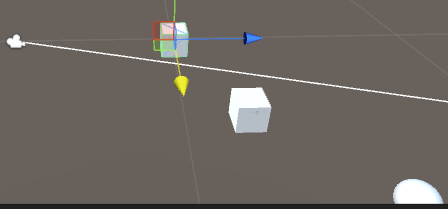
else

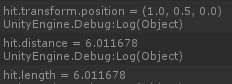
{

Gizmos.color = Color.white;

Gizmos.DrawRay(camera.transform.position, 400 \* ray.direction);

}



通过上面的结果发现hit.distance实际上就是起始点到碰撞点的距离

还有一个Physics.RaycastAll表示射线碰撞到的所有物体

**注意：上面的所有都需要有Collider**