**重要的类**

以下是在使用 Unity 编写脚本时将要使用的一些最重要的类。这些类涵盖 Unity 脚本化系统的一些核心领域，为查找可用的函数和事件提供了良好的起点。

| ***类：*** | ***描述：*** |
| --- | --- |
| [MonoBehaviour](file:///E:\UnityDocumentation\ScriptReference\MonoBehaviour.html) | 所有新 Unity 脚本的基类，MonoBehaviour 引用提供了附加到游戏对象的标准脚本可以使用的所有函数和事件的列表。如果需要对游戏中的各个对象进行任何类型的交互或控制，请使用此类。 |
| [Transform](file:///E:\UnityDocumentation\ScriptReference\Transform.html) | 每个游戏对象都在空间（无论是 3D 还是 2D 空间）中具有位置、旋转和缩放，而这些信息由变换组件表示。除了提供这些信息之外，变换组件还有许多有用的函数可用于移动、缩放、旋转、重新父子化和操纵对象，并可将坐标从一个空间转换为另一个空间。 |
| [Rigidbody](file:///E:\UnityDocumentation\ScriptReference\Rigidbody.html) / [Rigidbody2D](file:///E:\UnityDocumentation\ScriptReference\Rigidbody2D.html) | 对于大多数游戏元素，物理引擎提供了最简单的工具集来移动对象、检测触发器和碰撞以及施加作用力。Rigidbody 类（或其 2D 等效类 Rigidbody2D）提供了设定速度、质量、阻力、力、扭矩、碰撞等所需的所有属性和函数。 |

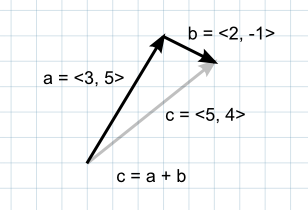
**矢量说明书**

虽然矢量运算很容易描述，但它们非常精巧和强大，在游戏编程中有很多用途。以下页面将提供有关在代码中有效使用矢量的一些建议。  
了解矢量算术

矢量算术是 3D 图形、物理和动画的基础，深入了解这一主题对于充分发挥 Unity 的功能很有帮助。以下是主要运算的说明以及有关它们的用途的一些建议。

## 加法

当两个矢量相加时，结果相当于将原始矢量依次作为“步骤”。请注意，两个参数的顺序无关紧要，因为两种方式的结果都相同。



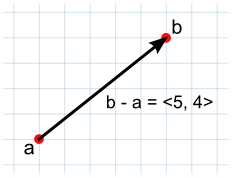
如果将第一个矢量视为空间中的一个点，那么第二个矢量可以解释为从该位置的偏移或“跳跃”。例如，为了找到地面上某个位置上方 5 个单位的点，可使用以下计算：

var pointInAir = pointOnGround + new Vector3(0, 5, 0);

如果矢量代表力，那么从力的方向和大小来考虑它们将会更直观。两个力矢量相加会产生一个等于力的组合的新矢量。施加的力同时有若干单独分量起作用时（例如，向前推进的火箭还可能受到侧风影响），此概念通常很有用。

## 减法

矢量减法通常用于获取从一个对象到另一个对象的方向和距离。请注意，两个参数的顺序对于减法很\_\_重要\_\_：



// 矢量 d 的大小与 c 相同，但指向相反的方向。

var c = b - a;

var d = a - b;

与数字一样，与负向矢量相加相当于减去正向矢量。

// 这两者得出相同的结果。

var c = a - b;

var c = a + -b;

负向矢量的大小与原始矢量相同并沿着同一条线指向，但在完全相反的方向上。

## 标量乘法和除法

在讨论矢量时，通常将普通数（例如，浮点值）称为标量。这意味着标量只有“标度”或大小，而矢量兼具大小和方向。

将矢量乘以标量会产生与原始矢量方向相同的矢量。但是，新矢量的大小等于原始大小乘以标量值。

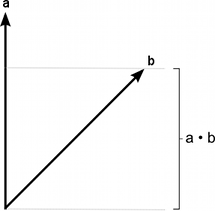
同样，标量除法将原始矢量的大小除以标量。

当矢量表示移动偏移或力时，这些运算很有用。通过这些运算可以更改矢量的大小而不影响其方向。

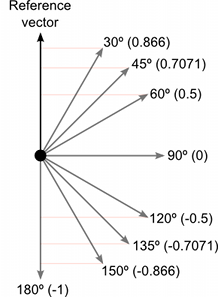
当任何矢量除以其自身的大小时，得到的结果是大小为 1 的矢量，即所谓的归一化矢量。如果归一化矢量乘以标量，则结果的大小将等于该标量值。当力的方向恒定但强度可控时（例如，来自车轮的力总是向前推动，但是动力由驾驶员控制），这会很有用。

## 点积

点积取两个矢量并返回标量。该标量等于两个矢量相乘的大小，得到的结果再乘以矢量之间角度的余弦。当两个矢量都被归一化时，余弦本质上表示第一个矢量在第二个矢量的方向上延伸的距离（反之亦然 - 参数的顺序无关紧要）。



想象成角度再使用计算器找到相应的余弦，这种处理方式比较容易。但是，还有一种有用的方法可以直观了解一些主余弦值，如下图所示：

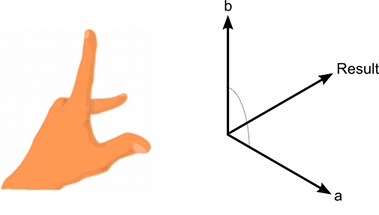


点积是一种非常简单的运算，在某些情况下可用于代替 Mathf.Cos 函数或矢量大小运算（功能不完全相同但有时效果相同）。但是，计算点积函数所需的 CPU 时间要少得多，因此可作为一种有价值的优化。

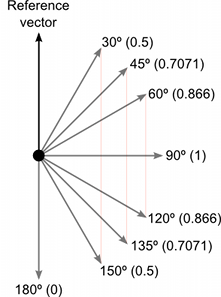
## 差积

其他运算是针对 2D 和 3D 矢量甚至任意维度的矢量定义的。相比之下，差积仅对 3D 矢量有意义。它需要两个矢量作为输入，并返回另一个矢量作为结果。

结果矢量垂直于两个输入向量。可使用“左手规则”根据输入矢量的排序确定输出矢量的方向。如果第一个参数对应于手的拇指，而第二个参数对应于食指，则结果将指向中指的方向。如果参数的顺序颠倒，得到的矢量将指向完全相反的方向，但大小相同。



结果的大小等于输入矢量的大小相乘，然后该值再乘以二者之间角度的正弦。正弦函数的一些有用值如下所示：



差积看起来很复杂，因为它在返回值中结合了多方面的有用信息。然而，就像点积一样，它在数学上的效率非常高，可用于优化代码，否则这些代码将不得不依赖于缓慢的超越函数。

# 从一个对象到另一个对象的方向和距离

如果从空间中的一个点减去另一个点，则得到的结果是从一个对象“指向”另一个对象的矢量：

// 获取从玩家位置指向目标位置的矢量。

var heading = target.position - player.position;

除了指向目标对象的方向之外，该矢量的大小等于两个位置之间的距离。通常需要归一化矢量来提供目标的方向以及到目标的距离（例如用于指挥飞弹）。对象之间的距离等于方向矢量的大小，该矢量可以通过除以其大小来归一化：

var distance = heading.magnitude;

var direction = heading / distance; // This is now the normalized direction.

此方法优于单独使用大小和归一化属性，因为大小和归一化属性都非常耗费 CPU（都涉及计算平方根）。

如果只需要使用距离进行比较（例如，进行接近检查），则可以完全避免大小计算。sqrMagnitude 属性给出大小值的平方，计算方式与大小相似，但不需要进行耗时的平方根运算。不要将大小与已知距离进行比较，可将大小的平方与距离的平方进行比较：

if (heading.sqrMagnitude < maxRange \* maxRange) {

// 目标在范围内。

}

这种算法比在比较中使用真实大小要高效得多。

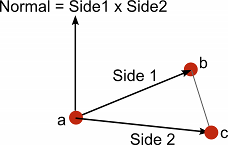
有时，需要平行于地面的目标方向。例如，想象一个站在地面上的玩家需要接近漂浮在空中的目标。如果从目标位置减去玩家位置，那么产生的矢量将向上指向目标。这种情况下不适合对玩家变换进行定向，因为变换也会指向上方；真正需要的是从玩家位置到目标正下方地面位置的矢量。通过使用减法结果并将 Y 坐标设置为零，很容易获得该矢量：

var heading = target.position - player.position;

heading.y = 0; // 这是平行于地面的方向。

**计算法向/垂直矢量**

在网格生成期间经常需要法向矢量（即，垂直于平面的矢量），此外，法向矢量还可能用于路径跟踪和其他情况。在平面中给定三个点（比如网格三角形的角点），就很容易找到法线。选择三个点中的任意一个点，然后分别从其他两个点减去这个点，即可得到两个矢量：



var a: Vector3;

var b: Vector3;

var c: Vector3;

var side1: Vector3 = b - a;

var side2: Vector3 = c - a;

这两个矢量的差积将得出垂直于表面的第三矢量。使用“左手规则”可确定将这两个矢量传递到叉积函数的顺序。在表面上方从上往下看（法线将指向外部）时，第一个矢量应顺时针扫过第二个矢量：

var perp: Vector3 = Vector3.Cross(side1, side2);

如果输入矢量的顺序颠倒，结果将指向完全相反的方向。

对于网格，法向矢量也必须归一化。可通过归一化属性来实现此目的，但是还有另一个偶尔有用的技巧。还可以将垂直矢量除以其大小，从而将其归一化：

var perpLength = perp.magnitude;

perp /= perpLength;

事实证明，三角形的面积等于 perpLength / 2。如果需要得出整个网格的表面积，或者想要根据相对面积随机选择三角形，这会很有用。

# 一个矢量的大小位于另一个矢量方向上的量

汽车的速度计一般用于测量车轮的转速。汽车可能不会直接向前移动（例如，可能侧向打滑），在此情况下，部分运动不在速度计可以测量的方向上。对象的 rigidbody.velocity 矢量的大小将给出整体运动方向上的速度，但是为了单独考虑向前的速度，应使用点积：

var fwdSpeed = Vector3.Dot(rigidbody.velocity, transform.forward);

当然，方向可以是您喜欢的任何方向，但为了进行此计算，必须对方向矢量进行归一化。这样，不仅结果比速度大小更准确，而且无需执行在查找大小时需要的缓慢的平方根运算。