Computer Graphics

一. 贝塞尔曲线路径

1. 贝塞尔曲线

在 1962 年，贝塞尔(Bezier)曲线由法国雷诺汽车公司的工程师 Pierre Bezier 对伯恩斯坦多项式进行扩展用于汽车工业辅助设计而产生，之后进行推广而得到广泛应用。贝塞尔曲线的优点是能够直观地表示给定条件与曲线形状的关系，只需要定义曲线的起始坐标点、终止坐标点以及两个相互分离的中间点（控制点）即可完成。用贝塞尔曲线来绘制复杂类型的曲线可以先分段定义多段曲线，然后再将这些曲线段连接起来形成所需要的曲线或曲面形。由于贝塞尔曲线的简单性特点，使得很多图形工作者选择使用它来绘制形状较为复杂的图形或模型。

2. 一次 Bezier 曲线

一次贝塞尔曲线函数中的 t 会经过由点 P0 至点 P1 组成的 B(t)所描述的曲线。当 $t = 0.25$ 时，B(t)即是一条由点 P0 至 P1 路径的四分之一处。就像是由数值 0 至数值 1 的连续数值 t，B(t)描述了一条由点 P0 至点 P1 的直线。高阶的贝塞尔曲线全部基于这个线性插值。

一次贝塞尔曲线的数学表达如下：

$$B(t) = P_0 + (P_1 - P_0)t = (1 - t)P_0 + tP_1, t\epsilon[0,1] \quad （1）$$

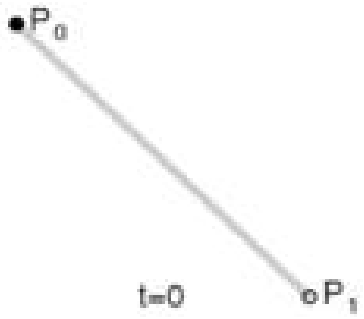


图 3.1 一次贝塞尔曲线 t 在[0,1]之间

3. 二次 Bezier 曲线

二次贝塞尔曲线具有三个控制点。二次的贝塞尔曲线其实是点对点的两个一次贝塞尔曲线的线性插值。由三个点 P_0、P_1 和 P_2 构建的一条二次贝塞尔曲线，其实是两条线性的贝塞尔曲线构成，即是线性贝塞尔曲线 P_0 和 P_1 和线性贝塞尔曲线 P_1 和 P_2。

二次贝塞尔曲线的路径由给定点 P_0、P_1、P_2 控制，这条线由下式给出：

$$B(t) = P_0(1 - t)^2 + 2P_1t(1 - t) + p_2t^2, t \in [0,1] \quad （2）$$

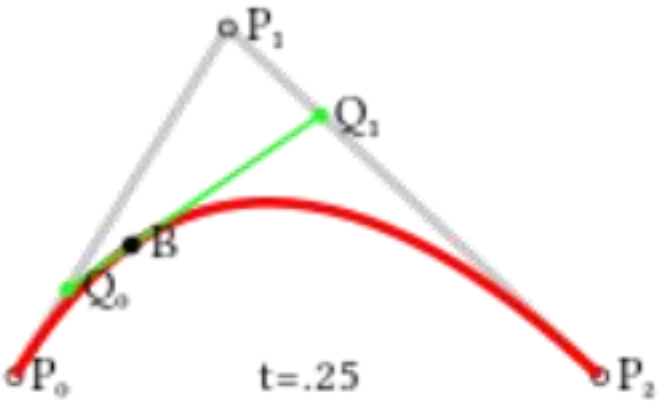


图 3.2 二次贝塞尔曲线的结构 t 在[0,1]之间

4. 三次 Bezier 曲线

三次贝塞尔曲线具有四个控制点。三次的贝塞尔曲线是由点对点的两条二次贝塞尔曲线的线性插值得到。由四个点 P_0、P_1、P_2 和 P_3 组成的三次贝塞尔曲线，其实是由两条二次贝塞尔曲线构成，即是二次贝塞尔曲线 P_0、P_1 和 P_2 和二次贝塞尔曲线 P_1、P_2 和 P_3 得到的线性插值。

三次贝塞尔曲线的路径由给定点 P_0、P_1、P_2、P_3 控制，这条线由下式给出：

$$B(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 3P_2t^2(1-t) + p_3t^3, t \in [0,1] \quad (3)$$

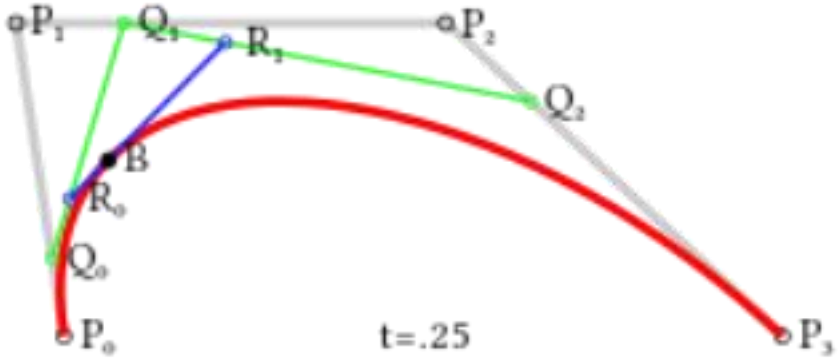


图 3.3 三次贝塞尔曲线的结构 t 在[0,1]之间

5. 高阶 Bezier 曲线

更高阶的贝塞尔曲线总能够通过两个低阶的贝塞尔曲线插值的堆叠获得，通俗的说就是通过对两条低阶的贝塞尔曲线进行线性差值，就可以求得一条高一阶的贝塞尔曲线。

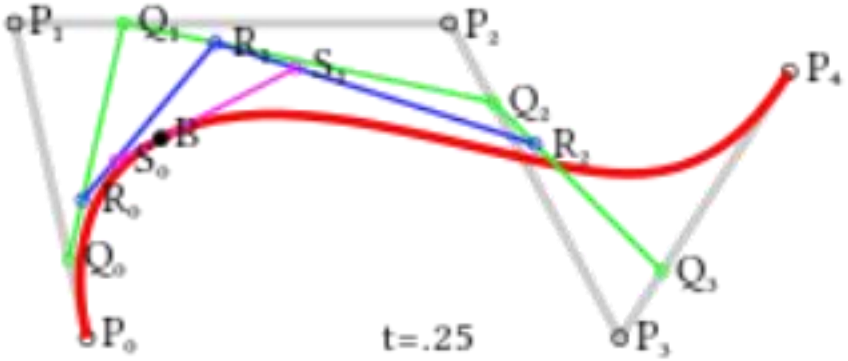


图 3.4 四次贝塞尔曲线的结构 t 在[0,1]之间

二. 程序化网格生成

1. 渲染物体

如果要想在 Unity3D 中渲染出某些内容，可以是渲染从另一个程序导出的 3D 模型，也可以是一个通过程序生成的网格，还可以是 Sprite、UI 或者粒子系统，就应该考虑使用 Unity3D 中的 Mesh。

如果要让 Object 显示 3D 模型，则该 Object 需要包含两个组件。第一个组件是 Mesh Filter（网格过滤器），该组件包含对要显示的网格的引用；第二个组件是 Mesh Renderer（网格渲染器），使用网格渲染器可以配置网格的渲染方式，应该使用何种 Material（材质），是否应该接收阴影等。

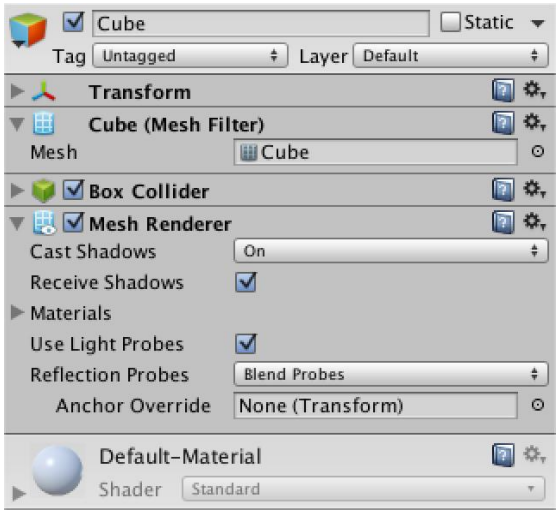


图 3.5 Unity 默认立方体

2. UV

UV 只是 3D 应用程序用于将纹理映射到模型的 2D 坐标，U 坐标表示 2D 纹理的水平轴，V 坐标表示垂直轴。UV 坐标通常位于(0,0)和(1,1)之间，覆盖了整个纹理。根据纹理设置，超出该范围的坐标将被限制或者平铺。

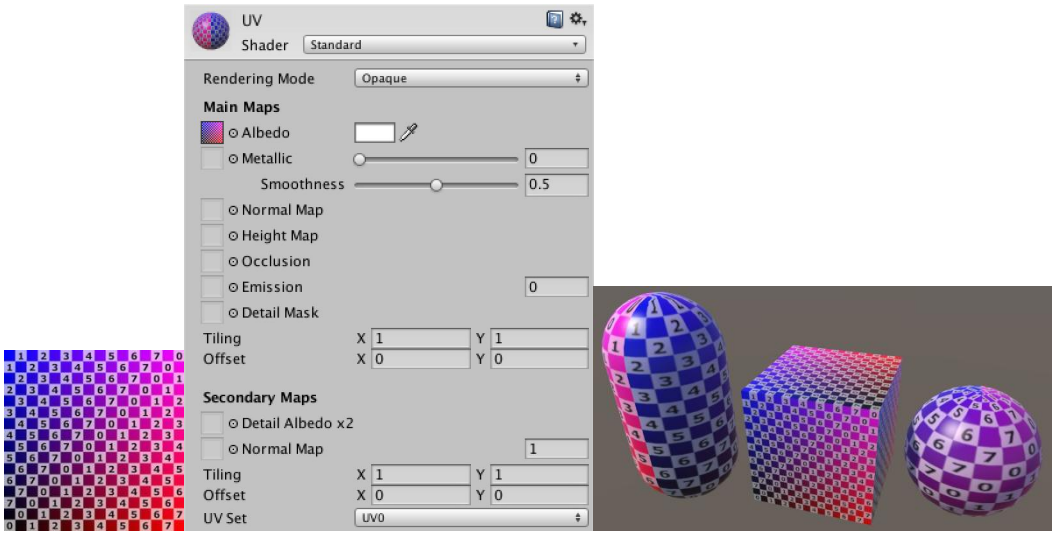


图 3.6 应用于 Unity 的 UV 测试坐标

3. 创建网格

每一个 Mesh 网格都是由若干三角形面片组成的，三角面是通过顶点索引数组定义的。因为每个三角形都具有三个点，因此三个连续的顶点索引就能够描绘出一个三角形。

这个三角形面片从哪一侧可见，是由构成这个三角形面片的顶点索引的方向决定的。在默认情况下，如果顶点索引按照顺时针方向排列，那么该三角形面片就是朝向前方且可见的，而逆时针方向的三角形面片就会被舍弃，因此程序就不需要花费时间去渲染对象的内部。

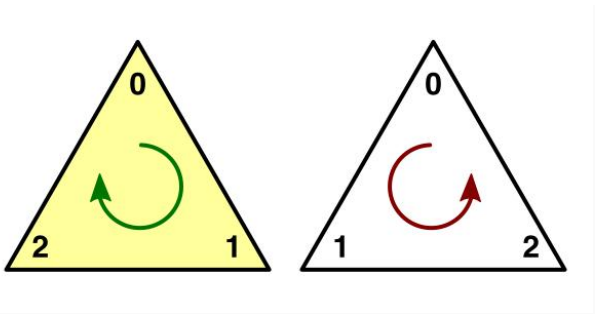


图 3.7 三角形顶点两种遍历情况

因此，要使三角形在 Z 轴向下看时能够显现出来，我们必须更改其顶点的遍历顺序。当我们有了一个三角形，便覆盖了网格第一个图块的一半。

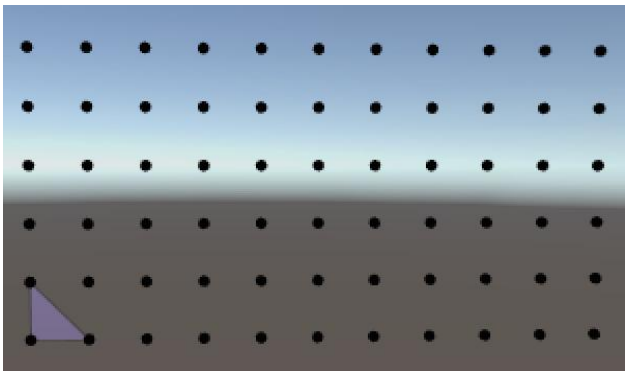
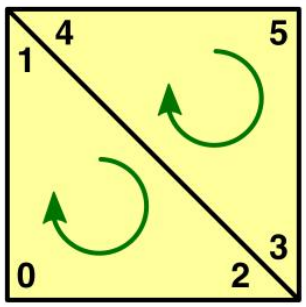


图 3.8 第一个三角形

若想要覆盖整个图块，我们需要两个三角形来组合成矩形。由于这些三角形共享两个顶点，在遍历时只需遍历 4 次即可。然后进行循环生成，我们就可以得到一个完整的矩形网格。



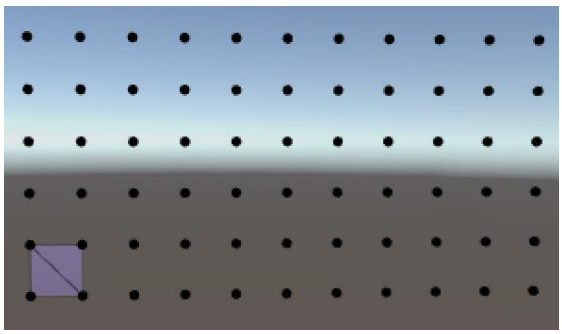


图 1. 图 3.9 由两个三角形组成的四边形 图 3.10 第一个四边形

4. 生成法线

法线是一个垂直于表面的向量，通常情况下使用的都是单位长度的法线，它们指向表面的外部而不是表面的内部。要想确定一束光线照射到表面的角度（如果有的话）可以使用法

线。默认的法线方向是（0，0，1），具体如何使用取决于着色器。由于三角形始终是平坦的，因此不需要提供有关法线的单独信息。但是，提供法线信息可以达到一种欺骗的效果。实际上，顶点没有法线，三角形面片也没有，要想假装拥有的是一个平滑的曲面而不是一堆扁平三角形，可以通过将自定义的法线附加到顶点并在两个三角形之间进行插值。只要不去特别注意网格的鲜明轮廓，这种幻觉就会令人信服。

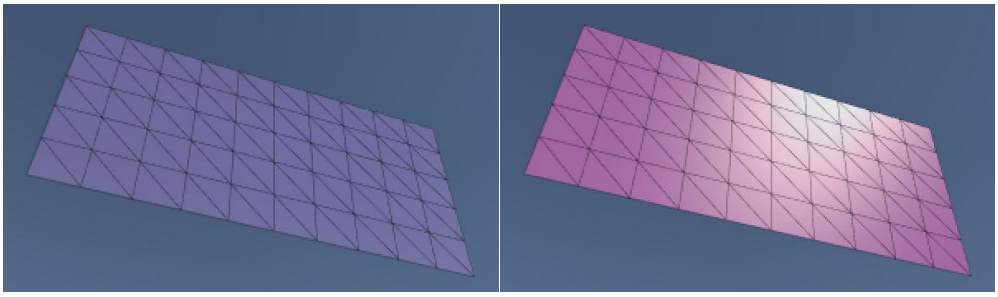


图 2.　　图 3.11 没有法线与有法线

5. 生成 UV 坐标

我们生成网格后，如果不提供 UV 坐标，则它们将全为 0。要使纹理适合我们的整个网络，只需将顶点的位置除以网格尺寸即可。在进行除法运算得到 UV 坐标时可能会出现 UV 坐标不正确的问题，这是因为使用了整数类型（integer）除以整数类型，这就导致最终所得结果是另一个整数。因为 UV 坐标介于 0 和 1 之间，为了确保坐标正确，我们必须使用浮点数（float）来进行计算。

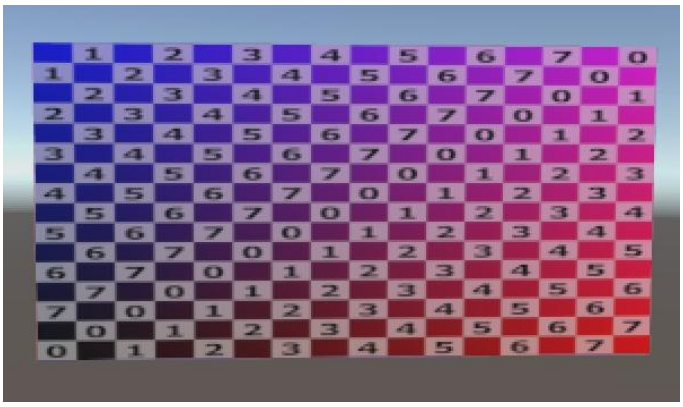


图 3.　　图 3.12 有 UV 坐标的网格

三. 欧拉角

表示飞行姿态的三种方法：欧拉角、四元数、旋转矩阵。飞机的三维坐标只能确定飞机的地理位置，却不能确定飞机的飞行姿态，所以我们需要欧拉角。

欧拉角包括：俯仰角 pitch、偏航角 yaw、翻转角 roll（可以根据字面意思进行理解如何旋转的）。俯仰角是绕 X 轴旋转，偏航角是绕 Z 轴旋转，翻转角是绕 Y 轴旋转。

在介绍 Gimbal Lock （万向节死锁）之前，假设我们有一个物体，我们使用三个互相正交的坐标轴对它建立一个坐标系。

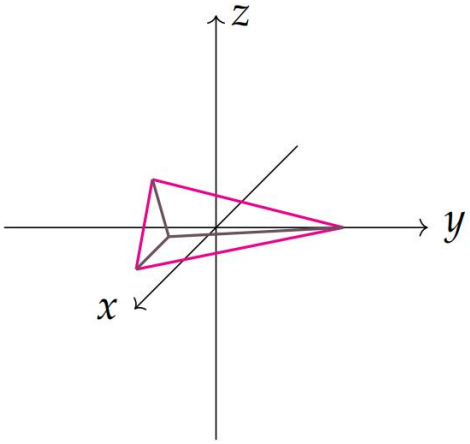


图 4.　　构建一个坐标系

现在，我们的目标就是能改变这个物体的朝向. 有一点需要注意的是，朝向 (Orientation) 和方向 (Direction) 虽然看起来很像，但是它们是完全不同的两个概念。朝向并不是使用一个向量定义的，我们通常会将朝向定义为将某一个「正朝向」旋转至当前朝向所进行的变换，所以当你思考朝向的时候，你需要想到的其实是一个旋转（想象一下，我们需要对物体的每一个顶点都施加一个旋转的变换，将它变换到当前的朝向）。

在很多年以前，欧拉就证明了，3D 空间中的任意一个旋转都可以拆分成为沿着物体自身三个正交坐标轴的旋转（注意，这里指的是沿着物体自己的坐标轴旋转）。而欧拉角就规定了这三次旋转的角度，我们分别称他们为 yaw（偏航角）、$pitch$（俯仰角） 和 $roll$（翻转角）。这也就是说，3D 空间中任意的旋转都可以由三个旋转矩阵相乘的方式得到。

$$\mathrm{E}(\mathrm{yaw},\mathrm{pitch},\mathrm{roll}) = R_z(\mathrm{yaw})R_y(\mathrm{roll})R_x(\mathrm{pitch})$$

其中 Rx、Ry、Rz 是沿着 x、y、z 轴旋转的旋转矩阵，它们就是我们的三维旋转矩阵。

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

因为我们将一个旋转拆分成了三个 3D 旋转矩阵的复合，而且 3D 旋转矩阵的相乘一般是不可交换的（即一般 $Rx(\theta)Rz(\phi) \neq Rz(\phi)Rx(\theta)$），所以进行这三次旋转的顺序非常重要。在上面的例子中，我们是按照 x-y-z 轴的顺序进行旋转的.当然，这并不是唯一的选择，如果想用 y-x-z 的顺序进行旋转也是完全可以的。根据三次旋转所绕轴，欧拉角可以分成多种类型，如：xyz，xzy，yxz，zxy，yzx，zyx，xyx，xzx，yxy，yzy，zxz，zyz 共十二种类型。

然而，要注意的是，一般情况下，我们一般只会选择其中固定的一个顺序来编码旋转或者朝向，而这正是导致 Gimbal Lock 的原因。

四. Gimbal Lock——万向节死锁

前面已经说了，我们可以将 3D 空间中任意一个旋转写成了三个矩阵相乘的形式，在

下面的讨论中，我们将固定使用这一套旋转顺序。

$$\mathrm{E}(\mathrm{yaw},\mathrm{pitch},\mathrm{roll}) = R_z(\mathrm{yaw})R_y(\mathrm{roll})R_x(\mathrm{pitch})$$

其中，每一次旋转变换就代表着有一个 Gimbal（通常译为万向节或者平衡环架）。所以，在这里我们一共有固定的三个 Gimbal。我们可以改变每次旋转的角度，但是不论怎么变化角度这个旋转的顺序都是不能改变的。

大部分情况下，这一套顺序都能够对三个不同的轴分别进行三次旋转。然而，在某一些特殊情况下，它其中的某两个旋转变换可能变换的是同一个轴（这里指的不是物体自身的轴，而是外部或者世界的轴），这样就导致我们丧失了一个轴的自由度，从而导致 Gimbal Lock 的产生。虽然看起来有点不可思议，但是下面我们会用实际的例子来说明。

假设我们使用 x-y-z 的顺序旋转任意一个点。

我们先沿 x 轴旋转任意度数。

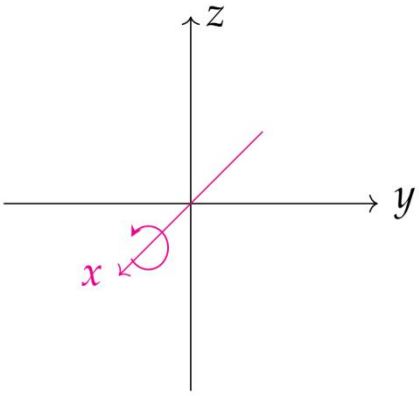


图 5.　　沿 X 轴旋转

再沿 y 轴旋转 $\frac{\pi}{2}$ 弧度。

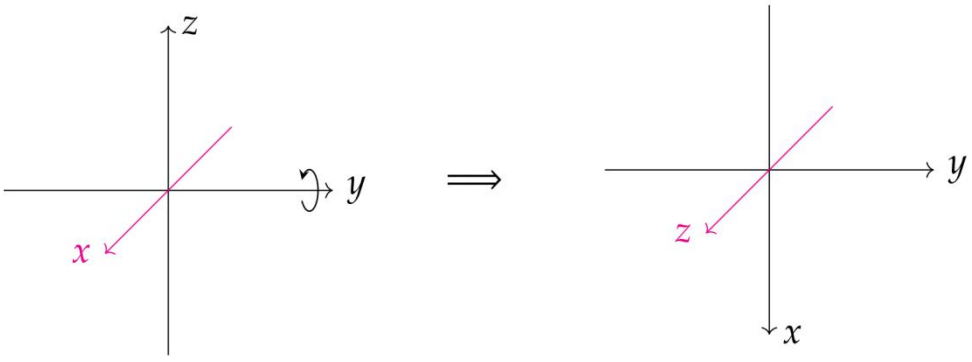


图 6.　　沿 Y 轴旋转$\frac{\pi}{2}$弧度

注意，经过这一次的变换之后，我们将 z 轴变换到原来 x 轴方向，而 x轴变到原来 $-z$ 的方向。

前面的变换执行完毕，最后打算沿着 z 轴旋转任意的度数时。当前的 z 轴与原来的 x 轴重合，也就是说，最后 z 轴的旋转与 x 轴的旋转其实操纵的是同一个轴（同样，这里指的是外部的旋转轴，在图中用红色标出）。三次旋转变换仅仅覆盖了两个外部轴的旋转，一个自由度就这样丢失了，这也就导致了 Gimbal Lock 的现象。

现在，如果我们想要操纵第三个轴的旋转（也就是在 y 轴变换之前 z 轴所表示的竖直方向），就需要对 y 轴变换之后的 x 轴进行旋转. 然而，唯一的 x 轴 Gimbal（或者变换）优先级在最外部，我们当时操纵它的时候，它操纵的外部旋转轴还与现在的 z 方向相同，

并不能补回这个自由度。

这个变换用公式来理解的话，则是这样。

$$E\left(\alpha, \frac{\pi}{2}, \beta\right) = R_z(\beta)R_y\left(\frac{\pi}{2}\right)R_x(\alpha)$$

$$= \begin{bmatrix} 0 & \cos\beta\sin\alpha - \cos\alpha\sin\beta & \sin\alpha\sin\beta + \cos\alpha\cos\beta \\ 0 & \sin\alpha\sin\beta + \cos\alpha\cos\beta & \cos\alpha\sin\beta - \cos\beta\sin\alpha \\ -1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & \sin(\alpha-\beta) & \cos(\alpha-\beta) \\ 0 & \cos(\alpha-\beta) & -\sin(\alpha-\beta) \\ -1 & 0 & 0 \end{bmatrix} = R_y\left(\frac{\pi}{2}\right)R_x(\alpha-\beta)$$

利用一些三角恒等式，将原本由三个旋转矩阵所组成的变换化简成了两个变换矩阵。即便我们分别对 x-y-z 三轴进行了旋转，实际上这个矩阵仅仅旋转了 x-y 两轴，它并没有对（初始的）z 轴进行变换。$R_z(\beta)$ 与$R_x(\alpha)$ 这两个变换被合并为单独的一个 $R_x(\alpha - \beta)$ 变换（因为化简完之后变换顺序不一样了，严格来说并不是合并，只不过是能够使用一步来完成）。

注意！我们在这里化简的并不是单独的一个变换，而是一系列变换。因为它对任意 α 和 β 角都是成立的。也就是说，一旦 y 轴上的变换角将这两个旋转轴对齐，我们就没有任何办法对最初的 z 轴进行旋转了。无论 x轴与 z 轴的旋转角是多少，变换都会丧失一个自由度。

当然这也不是说现在没有办法对最初的 z 轴进行变换。我们只需要在最内部添加一个 x 轴 Gimbal，或者说在变换序列的最后再添加一个 x 轴的旋转变换，就能解决问题了（注意下面经过 Ry 变换之后 Rx 操纵的其实是原本的 z 轴）

$$E(\alpha, \frac{\pi}{2}, \beta, \gamma) = R_x(\gamma)R_z(\beta)R_y(\frac{\pi}{2})R_x(\alpha) = R_x(\gamma)R_y(\frac{\pi}{2})R_x(\alpha - \beta)$$

然而，我们之前说过，我们选择的是固定的三个旋转顺序，实际上并没有这个第四个 Gimbal，所以没办法解决这个问题。虽然在当初选择顺序的时候也可以选择采用四个 Gimbal 的设计，但是这仍不能完全解决问题，因为除了 y 轴之外其它轴的旋转仍然可能将其中的两个轴或者三个轴对齐，只不过现在 Gimbal Lock 产生的几率可能会变小而已。Gimbal Lock 问题的核心还是在于我们采用了固定的旋转顺序。

五. 三维空间中的旋转

在讨论四元数之前，我们先来研究一下三维空间中的旋转。

表示三维空间中旋转的方法有很多种，但我们这里关注的是轴角式 (Axis-angle) 的旋转。虽然使用欧拉角的旋转很常用，但是我们知道欧拉角的表示方法不仅会导致 Gimbal Lock 而且依赖于三个坐标轴的选定，使用四元数正是为了解决这个问题。我们这里所要讨论的轴角式旋转是旋转更加普遍的情况。

假设我们有一个经过原点的（如果旋转轴不经过原点我们可以先将旋转轴平移到原点，进行旋转，再平移回原处）旋转轴 $u = (x, y, z)^T$，我们希望将一个向量 v，沿着这个旋转轴旋转 θ 度，变换到v'。

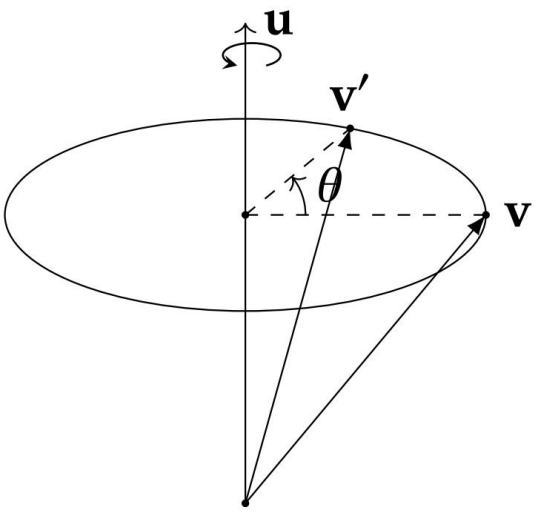


图 7.　沿着旋转轴旋转 θ 度，变换到v'

注意!在这里我们将使用右手坐标系统，并且我们将使用右手定则来定义旋转的正方向。我们可以将右手拇指指向旋转轴 u 的正方向，这时其它四个手指弯曲的方向即为旋转的正方向（在上图中即为逆时针方向）。

在轴角的表示方法中，一个旋转的定义需要使用到四个变量：旋转轴 u 的x，y，z 坐标，以及一个旋转角 θ，也就是我们一共有四个自由度 (Degree of Freedom)。这很明显是多于欧拉角的三个自由度的。实际上，任何三维中的旋转只需要三个自由度就可以定义了，为什么这里我们会多出一个自由度呢？

其实，在我们定义旋转轴 u 的 x，y，z 坐标的同时，我们就定义了 u 的模长（长度）。然而，通常情况下，如果我们说绕着一个向量 u 旋转，我们其实指的是绕着 u 所指的方向进行旋转。回忆一下向量的定义：向量是同时具有大小和方向的量，但是在这里它的大小（长度）并不重要．我们可以说绕着$\mathrm{u}_1 = (0,0,1)^T$ 这个轴进行旋转，也可以说绕着$\mathrm{u}_2 = (0,0,1)^T$旋转。虽然这两个向量完全不同，但是它们指向的都是同一个方向（即 z 轴的方向）。

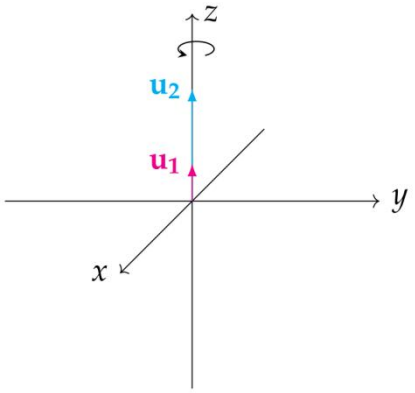


图 8.　只关注方向不关注大小

在三维空间中定义一个方向只需要用到两个量就可以了（与任意两个坐标轴之间的夹角）。最简单的例子就是地球的经纬度，我们仅仅使用经度和纬度两个自由度就可以定义地球上任意一个方位。而如果我们要表示某一个方位上的特定一个点，则还需要添加海拔这个自由度。

为了消除旋转轴 u 的模长这个多余的自由度，我们可以规定旋转轴 u 的模长为$\|u\| =$

$\sqrt{x^2+y^2+z^2}$=1，也就是说 u 是一个单位向量。这样一来，空间中任意一个方向上的单位向量就唯一代表了这个方向。我们其实可以将模长规定为任意的常量，但是规定 $\|u\|=1$ 能为我们之后的推导带来很多的便利，这也是数学和物理中对方向定义的惯例。

在实际代码的实现中，我们也可以让用户输入一个非单位长度的旋转轴向量 u，但是在进行任何的计算之前我们必须要记得先将它转化为一个单位向量，即$\hat{u}=\frac{u}{\|u\|}$。

1. 旋转的分解

有了这个约束，我们现在就可以开始思考怎么样进行这个旋转了。首先，我们可以将 v 分解为平行于旋转轴 u 以及正交（垂直）于 u 的两个分量，$v_{\parallel}$和 $v_{\perp}$，即 $v=v_{\parallel}+v_{\perp}$。

我们可以分别旋转这两个分向量，再将它们旋转的结果相加获得旋转后的向量，$v'=v'_{\parallel}+v'_{\perp}$。

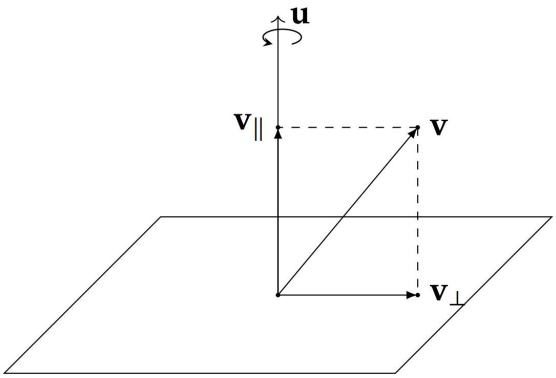


图 9.　　分解的示意图

可以看到，$v_{\parallel}$ 其实就是 v 在 u 上的正交投影 (Orthogonal Projection)，根据正交投影的公式，我们可以得出$v_{\parallel}=proj_u(v)=\frac{u\cdot v}{u\cdot u}u=\frac{u\cdot v}{\|u^2\|}u=(u\cdot v)u$。（$\|u^2\|=u\cdot u=1$）

由于 $v=v_{\parallel}+v_{\perp}$，我们可以得到 $v=v_{\parallel}+v_{\perp}=v-(u\cdot v)u$。既然我们已经知道怎么分解 v，接下来我们只需要分别讨论对 $v_{\parallel}$ 和 $v_{\perp}$ 的旋转就可以了。

2. $v_{\parallel}$的旋转

首先，我们来看一下平行于 u 的 $v_{\parallel}$。这种情况其实非常简单，从之前的图示中就可以看到，$v_{\parallel}$ 其实根本就没有被旋转，仍然与旋转轴 u 重合，所以当 $v_{\parallel}$ 平行于旋转轴 u 时，旋转 θ 角度之后的$v'_{\parallel}$为$v'_{\parallel}=v_{\parallel}$。（3D 旋转公式——向量型——平行情况）

3. $v_{\perp}$的旋转

接下来，我们需要处理正交于 u 的$v_{\perp}$。因为这两个向量是正交的，这个旋转可以看做是平面内的一个旋转。因为旋转不改变$v_{\perp}$的长度，所以路径是一个圆。

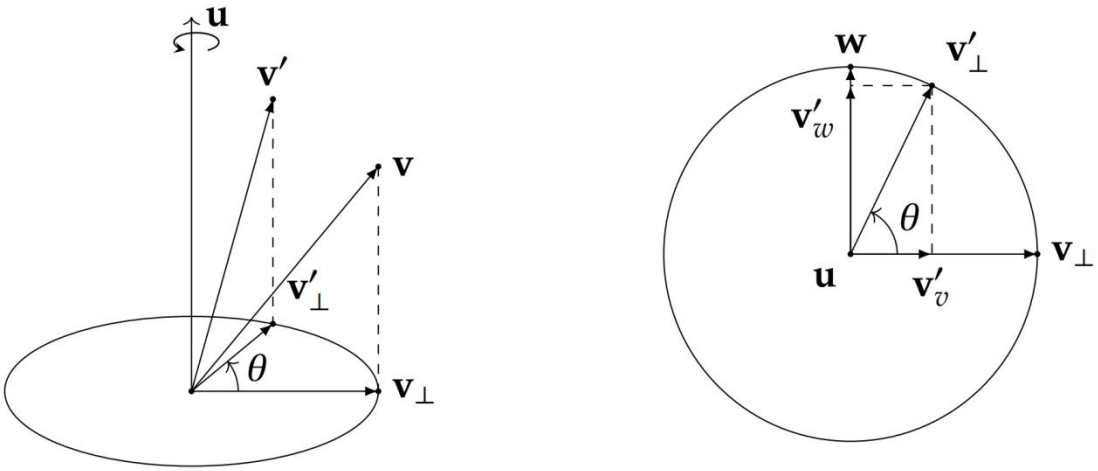


图 10.　左侧为旋转的示意图，右侧的为俯视图

现在，3D 的旋转就被我们转化为了 2D 平面上的旋转。由于在这个平面上我们只有一个向量 $v_\perp$，用它来表示一个旋转是不够的，我们还需要构造一个同时正交于 u 和 $v_\perp$ 的向量 w，这个向量可以通过叉乘来获得 $w = u \times v_\perp$。

注意叉乘的顺序，因为我们使用的是右手坐标系统，按照右手定则你可以发现这个新的向量 w 指向 $v_\perp$ 逆时针旋转 $\frac{\pi}{2}$ 后的方向，并且和 $v_\perp$ 一样也处于正交于 u 的平面内。因为$\|u\| = 1$，我们可以发现$\|w\| = \|u \times v_\perp\| = \|u\| \cdot \|v_\perp\| \cdot \sin\frac{\pi}{2} = \|v_\perp\|$。(前面的$\frac{\pi}{2}$是 u 与 $v_\perp$ 的夹角)

也就是说，w 和 $v_\perp$ 的模长是相同的，所以 w 也位于圆上。有了这个新的向量 w，就相当于我们在平面内有了两个坐标轴。我们现在可以把 $v'_\perp$投影到到 w 和$v_\perp$上，将其分解为v'_v和v'_w。使用一点三角学的知识我们就能得到$v'_\perp = v'_v + v'_w = \cos\theta\ v_\perp + \sin\theta\, w = \cos\theta * v_\perp + \sin\theta\,(u \times v_\perp)$ 。

这也就完成了旋转的第二步，我们可以得到这样一个定理。当 $v_\perp$ 正交于旋转轴 u 时，旋转 θ 角度之后的$v'_\perp$为$v'_\perp = \cos\theta * v_\perp + \sin\theta\,(u \times v_\perp)$。(3D 旋转公式——向量型——正交情况)

4. v 的旋转

将上面的两个结果组合就可以获得，$v' = v'_\parallel + v'_\perp = v_\parallel + \cos\theta\ v_\perp + \sin\theta\,(u \times v_\perp)$ 。

因为叉乘遵守分配律，$u \times v_\perp = u \times (v - v_\parallel) = u \times v - u \times v_\parallel = u \times v$（u 平行于$v_\parallel$，所以 $u \times v_\parallel = 0$）。

最后，将$v_\parallel = (u \cdot v)u$与$v_\perp = v - (u \cdot v)u$代入，得到$v' = (u \cdot v)u + \cos\theta\,(v - (u \cdot v)u) + \sin\theta\,(u \times v) = \cos\theta\, v + (1 - \cos\theta)(u \cdot v)u + \sin\theta\,(u \times v)$ 。

这样我们就得到了一般形式的旋转公式。

1) 3D 旋转公式—向量型—一般情况

3D 空间中任意一个$\boldsymbol{v}$沿着单位向量$\mathbf{u}$旋转θ角度之后的$\boldsymbol{v}'$为

$$v' = \cos\theta\, v + (1 - \cos\theta)(u \cdot v)u + \sin\theta\,(u \times v)$$

虽然最后结果看起来很复杂，但我们很快就能看到四元数与上面这些公式之间的联系。

六. 四元数

终于，我们可以开始讨论四元数与旋转之间的关系了。因为 3D 空间还是在我们理解范围之内的，所以四元数与三维旋转的一些关系可以直接使用一些基础的几何学和线性代数的

知识来推导和理解，并不会那么复杂。我们在大部分的时间中也会采用这一方式来理解四元数。

四元数的定义和复数非常类似，唯一的区别就是四元数一共有三个虚部，而复数只有一个。所有的四元数 $q \in H$（H 代表四元数的发现者 William Rowan Hamilton）都可以写成这种形式，即 $q = a + bi + cj + dk$，$(a, b, c, d \in R)$。其中，$i^2 = j^2 = k^2 = ijk = -1$，这个看似简单的公式就决定了四元数的一切性质。

与复数类似，因为四元数其实就是对于基{1，i，j，k}的线性组合，四元数也可以写成向量的形式 $q = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$。除此之外，我们经常将四元数的实部与虚部分开，并用一个三维的向量来表示虚部，将它表示为标量和向量的有序对形式 $q = [s，\ v]\ (v = \begin{bmatrix} x \\ y \\ z \end{bmatrix},\ s, x, y, z \subset R)$。

1. 为什么要用四元数

四元数最早于 1843 年由 Sir William Rowan Hamilton 发明，作为复数(complex numbers)的扩展。直到 1985 年才由 Shoemake 把四元数引入到计算机图形学中。四元数在一些方面优于 Euler angles(欧拉角)和 matrices。任意一个三维空间中的定向(orientation，即调置朝向)都可以被表示为一个绕某个特定轴的旋转。给定旋转轴及旋转角度，很容易把其它形式的旋转表示转化为四元数或者从四元数转化为其它形式。因此，四元数可以用于稳定的、经常性的(constant)的 orientations(即旋转)插值，而这些在欧拉角中是很难实现的。

1) 优点

① 解决万向节死锁（Gimbal Lock）问题。

② 仅需存储 4 个浮点数，相比矩阵更加轻量。

③ 四元数无论是求逆、串联等操作，相比矩阵更加高效。

2. 与四元数有关的一些概念

1) 空间中的子空间

一般而言，空间（维度>2）都存在更低维的子空间，比如二维空间中一维子空间，也就是直线；三维空间中的一维子空间和二维子空间，也就是直线和面。当超过三维的概念我们就很难去想象是什么样子，但四维空间一定会存在三维子空间或二维子空间。我们这里会加一个前缀超（hyper）来形容这些空间，比如高维空间中的平面我们称为超平面。

2) 空间和子空间的映射

我们将二维空间表示为(x,y)，当 y=0 时，其实可以看成是一维的，只不过它表示成(x,0)这种形式。推到四维，(w,x,y,z)，当 w=0 时，(0,x,y,z)就是一个三维子空间，这也是为什么我们可以用单位四元数对三维向量进行操作，其实我们是将三维向量映射到四维的三维子空间（w=0，这种形式也称纯四元数），然后对其进行旋转，最终得到的向量结果依然是这个三维子空间中的，因而可以映射回三维空间。

3) 广义球

这里的球是广义上的。我们在二维平面上，广义球其实指代 circle，三维空间上就是我们认知上的球，称为 two-sphere，而四维空间中广义球其实是一个超球（hyper-sphere），又称为 three-sphere。单位向量其实就是广义球上面的点，而单位四元数也就是 three-sphere 上面的点。

4) 约束与特征向量

空间中的一点由 x, y, z 等参数来表示，一般来说参数的数量与维数相等，二维空间的点

用{x, y}参数，四维空间的点用{x, y, z, w}参数。但是对于空间的点加以约束，则会减少参数的数量，比如三维空间的点在某一单位球面上，原本三个参数{x, y, z}才能表达的点现在只需要两个参数{u, v}就可以表达。如果{u, v}是单位向量，也可以称{u, v}是{x, y, z}的特征向量。

上述概念给了大家一个思路，四元数这样一个东西并不是一蹴而就的，从空间来说，它与我们熟知的低维空间本质上是没有区别的，或者说是有很大共性的。四元数的很多特性都是从低维拓展而来的，更具体的说是从复数这一概念拓展的。

复数概念总结起来很简单，它其实就是为了满足数学家的强迫症，-1 的平方根要有意义。看似很没有意义的复数，在实际中却用处非凡。我们首先看一下它的几何表达形式，令 x 为实部，y 为虚部，则 x + iy 可以表示为下图。

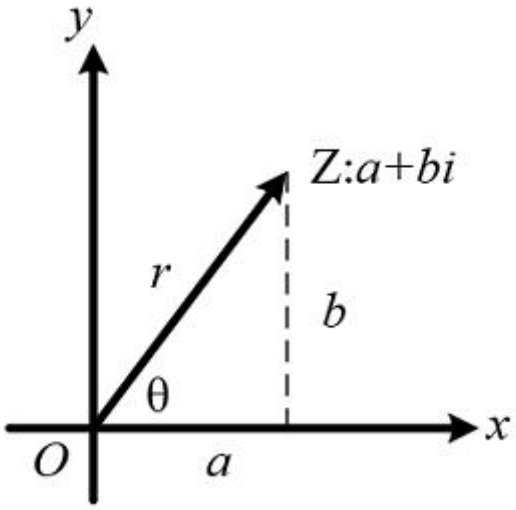


图 11.　复数的几何表示

3. 定义与性质

1) 模长（范数）

仿照复数的定义，我们可以暂时将一个四元数 $q = a + bi + cj + dk$ 的模长（或者说范数（Norm））定义为$\|q\| = \sqrt{a^2 + b^2 + c^2 + d^2}$。

如果用标量向量有序对的形式进行表示的话，$q = [s, v]$的模长为

$\|q\| = \sqrt{s^2 + \|v\|^2} = \sqrt{s^2 + v \cdot v}(v \cdot v = \|v\|^2)$。显然，四元数的模长很难用几何的方法来进行理解，因为它代表的是一个四维的长度。但是，和高维向量的模长一样，这只是类比复数模长进行衍生定义的结果，我们只需要将它理解为一个定义就可以了。

2) 四元数加法和减法

与复数类似，四元数的加法只需要将分量相加就可以了。如果我们有两个四元数$q_1 = a + bi + cj + dk$，$q_2 = e + fi + gj + hk$，那么它们的和$q_1 + q_2 = a + bi + cj + dk + e + fi + gj + hk = (a + e) + (b + f)i + (c + g)j + (d + h)k$。

减法也是同理，只要将加号改为减号就可以了，$q_1 - q_2 = (a - e) + (b - f)i + (c - g)j + (d - h)k$。

如果四元数是以标量向量有序对形式定义的，比如说$q_1 = [s, v]$，$q_2 = [t, u]$，那么$q_1 \pm q_2 = [s \pm t, v \pm u]$。

3) 乘法

a. 和标量相乘

如果我们有一个四元数 $q = a + bi + cj + dk$ 和一个标量 s，那么它们的乘积为 $sq = s(s + bi + cj + dk) = sa + sbi + scj + sdk$。通过这个计算公式可以知道，四元数与标量相乘，就是标量乘以四元数的实部和每一个虚部。四元数与标量的乘法是遵守交换律的，也就是说 $sq = qs$。

b. 四元数相乘

四元数之间的乘法比较特殊，它们是不遵守交换律的，也就是说一般情况下 $q_1q_2 \neq q_2q_1$，这也就有了左乘和右乘的区别。如果是 q_1q_2，那么我们就说「q_2 左乘以 q_1」，如果是 q_2q_1，那我们就说「q_2 右乘以 q_1」。除了交换律之外，我们经常使用的结合律和分配律在四元数内都是成立的。

那么，如果有两个四元数 $q_1 = a + bi + cj + dk$ 和 $q_2 = e + fi + gj + hk$，那么它们的乘积为 $q_1q_2 = (a + bi + cj + dk)(e + fi + gj + hk)$

$= ae + afi + agj + ahk + bei + bfi^2 + bgij + bhik + cej + cfji + cgj^2 + chjk + dek + dfki + dgkj + dhk^2$。

这样乘法最终的结果显然非常凌乱，但是我们可以根据 $i^2 = j^2 = k^2 = ijk = -1$ 这个定义来化简。通过这个公式我们可以知道，$ijk = -1$，$iijk = -i$（等式两边同时乘以 i），$-jk = -i(ii = i^2 = -1)$，$jk = i$。

同理，$ijk = -1$，$ijkk = -k$（等式两边同时乘以 k），$-ij = -k$，$ij = k$。

利用 $ij = k$ 这个公式，我们可以继续推导，$ij = k$，$ijj = kj$（等式两边同时乘以 j），$-i = kj$，$kj = -i$。

从这里我们就已经可以发现交换律不成立了，因为 $jk \neq kj$。按照类似的步骤进行推导，我们可以得出下面的一个表格。

	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

表格最左列中一个元素右乘以顶行中一个元素的结果就位于这两个元素行列的交叉处。比如说 $ji = -k$。用颜色标记的格子代表着乘法交换律不成立。

利用这个表格，我们就能进一步化简四元数乘积的结果。

$q_1q_2 = ae + afi + agj + ahk + bei - bf + bgk - bhj + cej - cfk - cg + chi + dek + dfj - dgi - dh$

$= (ae - bf - cg - dh) + (be + af - dg + ch)i + (ce + df + ag - bh)j + (de - cf + bg + ah)k$。

4) 矩阵形式

可以看到，四元数的相乘其实也是一个线性组合，我们同样可以将它写成矩阵的形式

$$q_1q_2 = \begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}。$$

注意，上述矩阵所做出的变换等价于左乘 q_1。因为四元数不符合交换律，所以右乘 q_1 的变换是一个不同的矩阵，它可以使用完全相同的方法推导而得，这里我直接给出结果。

$$q_2q_1 = \begin{bmatrix} a & -b & -c & -d \\ b & a & d & -c \\ c & -d & a & b \\ d & c & -b & a \end{bmatrix} \begin{bmatrix} e \\ f \\ g \\ h \end{bmatrix}。$$

5) Grassmann 积

回到之前的结果，我们重新整理一下乘积中的每一项，观察一下，看看能不能发现什么规律。

$q_1q_2 = (ae - (bf + cg + dh)) + (be + af + ch - dg)i + (ce + ag + df - bh)j + (de + ah + bg - cf)k$。

如果令 $v = \begin{bmatrix} b \\ c \\ d \end{bmatrix}$，$u = \begin{bmatrix} j \\ g \\ h \end{bmatrix}$，那么 $v \cdot u = bf + cg + dh$。

$$v \times u = \begin{vmatrix} i & j & k \\ b & c & d \\ f & g & h \end{vmatrix} = (ch - dg)i - (bh - df)j + (bg - cf)k。$$

注意 $v \times u$ 的结果是一个向量，这里的 i、j、k 是向量的基，写成这种形式结果应该就非常清楚了。如果使用标量向量有序对形式来表示，q_1q_2的结果可以用向量点乘和叉乘的形式表示出来（其实按照真实历史的顺序来说叉乘是在这里被定义的）。

$q_1q_2 = [ae - v \cdot u, au + ev + v \times u]$，这个结果也被叫做 Grassmann 积 (Grassmann Product) 。一般来说，对任意四元数$q_1 = [s, v]$，$q_2 = [t, u]$，q_1q_2的结果是$q_1q_2 = [st - v \cdot u, su + tv + v \times u]$。

如果还记得之前推导 3D 旋转公式时的结果，应该就能注意到上面这个定理会成为将四元数与旋转联系起来的关键。

6) 纯四元数

在我们正式进入四元数的讨论之前，我们还需要更多关于四元数的定义。如果一个四元数能写成这样的形式，即 $v = [0, \mathbf{v}]$。

那我们则称 v 为一个纯四元数 (Pure Quaternion)，即仅有虚部的四元数。因为纯四元数仅由虚部的 3D 向量决定，我们可以将任意的 3D 向量转换为纯四元数。在后续内容中，如果一个 3D 向量为 $\mathbf{v}$，那么不加粗、没有上标的 v 则为它对应的纯四元数 $v = [0, \mathbf{v}]$。

纯四元数有一个很重要的特性：如果有两个纯四元数 $v = [0, \mathbf{v}]$，$u = [0, \mathbf{u}]$。

$$vu = [0 - \mathbf{v} \cdot \mathbf{u}, 0 + \mathbf{v} \times \mathbf{u}] = [-\mathbf{v} \cdot \mathbf{u}, \mathbf{v} \times \mathbf{u}]。$$

我们在之后还需要证明更多有关纯四元数的定理，但这一条暂时就够用了。

7) 逆和共轭

因为四元数是不遵守交换律的，我们通常不会将两个四元数相除写为$\frac{p}{q}$的形式。取而代之的是将乘法的逆运算定义为 pq^{-1}或者 $q^{-1}p$，注意它们的结果一般是不同的。

其中，q^{-1}是 q 的逆(Inverse)，我们规定 $qq^{-1} = q^{-1}q = 1 (q \neq 0)$。

这也就是说$(pq)q^{-1} = p(qq^{-1}) = p \cdot 1 = p$，$q^{-1}(qp) = (q^{-1}q)p = 1 \cdot p = p$。所以，右乘 q 的逆运算为右乘q^{-1}，左乘 q 的逆运算为左乘q^{-1}，这个与矩阵的性质非常相似。

显然，要在无数的四元数中寻找一个满足 $qq^{-1} = q^{-1}q = 1$ 的 q^{-1}是非常困难的，但是实际上我们可以使用四元数共轭的一些性质来获得q^{-1}。

我们定义，一个四元数 $q = a + bi + cj + dk$ 的共轭为 $q^* = a - bi - cj - dk$（q^*读作「q star」）。如果用标量向量有序对的形式来定义的话，$q = [s, v]$ 的共轭为 $q^* = [s, -v]$。

共轭四元数的一个非常有用的性质就是，$qq^* = [s, v] \cdot [s, -v] = [s^2 - v \cdot (-v), s(-v) + sv + v \times (-v)] = [s^2 + v \cdot v, 0]$。（v 平行于-v，由叉乘的定义得$v \times (-v) = 0$）

可以看到，这最终的结果是一个实数，而它正是四元数模长的平方，即$q^*q = (q^*)(q^*) = \|q^*\|^2 = s^2 + x^2 + y^2 + z^2 = \|q\|^2 = qq^*$。

所以我们得到，$q^*q = qq^*$，这个特殊的乘法是遵守交换律的。

如果还记得之前四元数逆的定义的话，$qq^{-1} = 1$，

$q^*qq^{-1}=q^*$(左右两边同乘四元数的共轭)，$(q^*)qq^{-1} = q^*$，$\|q^*\|^2 \cdot q^{-1} = q^*(q^*q = \|q^*\|^2)$，

$q^{-1} = \frac{q^*}{\|q^*\|^2}$。

用这种办法寻找一个四元数的逆会非常高效，我们只需要将一个四元数的虚部改变符号，除以它模长的平方就能获得这个四元数的逆了．如果$\|q\| = 1$，也就是说 q 是一个单位四元数 (Unit Quaternion)，那么$q^{-1} = \frac{q^*}{1} = q^*$。

4. 四元数与 3D 旋转

了解了复数和四元数的基本知识，我们就能开始讨论四元数与 3D 旋转之间的关联了。

回忆一下我们之前讨论过的内容：如果我们需要将一个向量 v 沿着一个用单位向量所定义的旋转轴 u 旋转 θ 度，那么我们可以将这个向量 v 拆分为正交于旋转轴的 $v_\perp$ 以及平行于旋转轴的$v_\parallel$。我们可以对这两个分向量分别进行旋转，获得$v' = v'_\parallel + v'_\perp$。

我们可以将这些向量定义为纯四元数，$v = [0, \mathbf{v}]$，$v' = [0, \mathbf{v}']$，$v_\perp = [0, \mathbf{v}_\perp]$，$v'_\perp = [0, \mathbf{v}'_\perp]$，$v_\parallel = [0, \mathbf{v}_\parallel]$，$v'_\parallel = [0, \mathbf{v}'_\parallel]$，$u = [0, \mathbf{u}]$。那么我们就能得到，$v = v_\parallel + v_\perp$，$v' = v'_\parallel + v'_\perp$，和之前一样，我们在这里也分开讨论 $v_\perp$ 和 $v_\parallel$ 的情况。

1) $v_\perp$的旋转

我们首先讨论正交于旋转轴的$v_\perp$．我们之前推导过，如果一个向量$\mathbf{v}_\perp$ 正交于旋转轴 $\mathbf{u}$，那么$\mathbf{v}'_\perp = \cos\theta * \mathbf{v}_\perp + \sin\theta\,(\mathbf{u} \times \mathbf{v}_\perp)$。

我们可以很容易地将前面的$\mathbf{v}'_\perp$和$\mathbf{v}_\perp$**替换为**$v'_\perp$和$v_\perp$，但是我们仍遗留下来了一个 $\mathbf{u} \times \mathbf{v}_\perp$。幸运的是，利用四元数的性质，我们可以将它写成四元数积的形式。我们之前推导过，如果有两个纯四元数 $v = [0, \mathbf{v}]$，$u = [0, \mathbf{u}]$，那么 $vu = [-\mathbf{v} \cdot \mathbf{u}, \mathbf{v} \times \mathbf{u}]$。

类似地，$uv_\perp = [-\mathbf{u} \cdot \mathbf{v}_\perp, \mathbf{u} \times \mathbf{v}_\perp]$。因为$\mathbf{v}_\perp$正交于 $\mathbf{u}$，所以$\mathbf{u} \cdot \mathbf{v}_\perp = 0$，也就是说 $uv_\perp = [0, \mathbf{u} \times \mathbf{v}_\perp] = \mathbf{u} \times \mathbf{v}_\perp$。

注意，$uv_\perp$ 同样是一个纯四元数。我们离最终的结论已经很近了！将这个等式以及之前定义的纯四元数代入，我们就能获得$v'_\perp = \cos\theta * v_\perp + \sin\theta\,(uv_\perp)$。因为四元数的乘法遵守分配律，我们可以继续变换这个等式$v'_\perp = \cos\theta * v_\perp + \sin\theta\,(uv_\perp) = (\cos\theta + \sin\theta * u)v_\perp$。

应该可以注意到，如果我们将 $(\cos\theta + \sin\theta * u)$看做是一个四元数，我们就能将旋转写成四元数的乘积了。到此为止，我们已经将旋转与四元数的积联系起来了。如果令 $q = \cos\theta + \sin\theta * u$我们能得到$v'_\perp = qv_\perp$。

如果能构造一个 q，那么我们就能完成这个旋转了。我们可以对 q 继续进行变形，$q = \cos\theta + \sin\theta * u = [\cos\theta, 0] + [0, \sin\theta * \mathbf{u}] = [\cos\theta, \sin\theta * \mathbf{u}]$。也就是说，如果旋转轴 $\mathbf{u}$ 的坐标为$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$，旋转角为 θ，那么完成这一旋转所需要的四元数 q 可以构造为 $q = \cos\theta +$

$\sin\theta\, u_x i + \sin\theta\, u_y j + \sin\theta\, u_z k$。

这样我们就完成了对$v_\perp$的旋转，我们可以得到一个定理。当$\mathbf{v}_\perp$正交于旋转轴 **u** 时，旋转 θ 角度之后的$\mathbf{v}'_\perp$可以使用四元数乘法来获得。令$v_\perp = [o, \mathbf{v}_\perp]$，$q = [\cos\theta, \sin\theta * \boldsymbol{u}]$，那么$v'_\perp = qv_\perp$。（3D 旋转公式—四元数型—正交情况）

这个四元数 q 其实还有一个性质，我们可以注意到$\|q\| = \sqrt{\cos^2\theta + (\sin\theta * \boldsymbol{u} \cdot \sin\theta * \boldsymbol{u})} = \sqrt{\cos^2\theta + \sin^2\theta\,(\boldsymbol{u}\cdot\boldsymbol{u})} = \sqrt{\cos^2\theta + \sin^2\theta\,(\|\boldsymbol{u}\|^2)}(\boldsymbol{u}\cdot\boldsymbol{u} = \|\boldsymbol{u}\|^2) = \sqrt{\cos^2\theta + \sin^2\theta}(\|\boldsymbol{u}\| = 1) = 1$（三角恒等式）。

所以，我们构造出来的这个 q 其实是一个单位四元数。我们之前讨论过，复数的乘法的几何意义可以理解为缩放与旋转的复合。这个性质可以类比到四元数，而因为$\|q\| = 1$，它所代表的变换并不会对原向量进行缩放，是一个纯旋转。

2) $v_\parallel$的旋转

接下来是平行于旋转轴的 $v_\parallel$。我们之前讨论过，如果一个向量 $v_\parallel$平行于 **u**，那么旋转不会对它作出任何的变换，也就是说当$v_\parallel$平行于旋转轴 **u** 时，旋转 θ 角度之后的$v'_\parallel$用四元数可以写为$v'_\parallel = v_\parallel$。（3D 旋转公式—四元数型—平行情况）

3) V 的旋转

有了这些知识，我们能够获得一般情况下v'的结果了，即$v' = v'_\parallel + v'_\perp = v_\parallel + qv_\perp$（其中 $q = [\cos\theta, \sin\theta * \boldsymbol{u}]$）。我们当然可以像以前那样将$v_\parallel$和$v_\perp$拆开，继续进行化简。但是这里我们不会这么做，因为我们有更好的办法来进一步化简它。

在进一步化简之前，我们需要证明几个引理。

a. 引理一

如果 $q = [\cos\theta, \sin\theta * \boldsymbol{u}]$，而且 **u** 为单位向量，那么$q^2 = qq = [\cos 2\theta, \sin(2\theta)\,\mathbf{u}]$。

这个引理的证明很简单，只需要使用 Grassmann 积和一些三角恒等式就可以了。

PS：

$q^2 = [\cos\theta, \sin\theta * \boldsymbol{u}] \cdot [\cos\theta, \sin\theta * \boldsymbol{u}]$

$= [\cos^2\theta - (\sin\theta\,\boldsymbol{u} \cdot \sin\theta\,\boldsymbol{u}), (\cos\theta\sin\theta + \sin\theta\cos\theta)\boldsymbol{u} + (\sin\theta\,\boldsymbol{u} \times \sin\theta\,\boldsymbol{u})]$（这里使用了两角和差公式和二倍角公式）

$= [\cos^2\theta - \sin^2\theta\,\|\boldsymbol{u}\|^2, 2\sin\theta\cos\theta\,\boldsymbol{u} + \mathbf{0}]$

$= [\cos^2\theta - \sin^2\theta, 2\sin\theta\cos\theta\,\boldsymbol{u}]$

$= [\cos 2\theta, \sin 2\theta * \boldsymbol{u}]$

其实这个引理的几何意义就是是，如果绕着同一个轴 $\boldsymbol{u}$ 连续旋转θ度两次，那么所做出的变换等同于直接绕着 $\boldsymbol{u}$ 旋转 2θ 度。

有了这个引理，我们再来回忆一下四元数逆的定义，$qq^{-1} = 1$。

现在，我们就能够对原本的旋转公式进行变形了。

PS：

$v' = v_\parallel + qv_\perp (q = [\cos\theta, \sin\theta * \boldsymbol{u}])$

$= 1 \cdot v_\parallel + qv_\perp = pp^{-1}v_\parallel + ppv_\perp$。（令$q = p^2$，则$p = [\cos(\frac{1}{2}\theta), \sin(\frac{1}{2}\theta)\,\mathbf{u}]$）。在这里，我们引入了一个新的四元数$p = [\cos(\frac{1}{2}\theta), \sin(\frac{1}{2}\theta)\,\mathbf{u}]$。

根据刚刚证明的引理，我们可以验证$pp = p^2 = \left[\cos\left(2 * \frac{1}{2}\theta\right), \sin\left(2 * \frac{1}{2}\theta\right)\mathbf{u}\right] =$

$[\cos(\theta), \sin(\theta)\mathbf{u}] = q$。我们应该能够注意到，和 q 一样， $\|p\| = 1$，p 也是一个单位四元数，也就是说$\mathrm{p}^{-1} = p^*$。

将这个结果代入之前的等式中，得到$v' = pp^{-1}\mathrm{v}_{\parallel} + pp\mathrm{v}_{\perp} = pp^*\mathrm{v}_{\parallel} + pp\mathrm{v}_{\perp}$。我们还能进一步化简这个公式，但在继续之前还需要再证明两个引理。

b. 引理二

假设$\mathrm{v}_{\parallel} = [0, \mathbf{v}_{\parallel}]$是一个纯四元数，而 $\mathrm{q} = [\alpha, \beta\mathbf{u}]$，其中$\mathbf{u}$是一个单位向量，$\alpha, \beta \in \mathbb{R}$。在这种条件下，如果$\mathrm{v}_{\parallel}$平行于$\mathbf{u}$，那么 $\mathrm{q}\mathrm{v}_{\parallel} = \mathrm{v}_{\parallel}\mathrm{q}$。

这个引理的证明同样需要使用 Grassmann 积，我们先计算等式左边。

PS：

$\mathrm{LHS} = \mathrm{q}\mathrm{v}_{\parallel}$

$= [\alpha, \beta\mathbf{u}] \cdot [0, \mathbf{v}_{\parallel}]$

$= [0 - \beta\mathbf{u} \cdot \mathbf{v}_{\parallel}, \alpha\mathbf{v}_{\parallel} + 0 + \beta\mathbf{u} \times \mathbf{v}_{\parallel}]$

$= [-\beta\mathbf{u} \cdot \mathbf{v}_{\parallel}, \alpha\mathbf{v}_{\parallel}]$。（$\mathbf{v}_{\parallel}$平行于$\mathbf{u}$，所以$\beta\mathbf{u} \times \mathbf{v}_{\parallel} = 0$）

接下来再计算等式的右边。

PS：

$\mathrm{RHS} = \mathrm{v}_{\parallel}\mathrm{q}$

$= [0, \mathbf{v}_{\parallel}] \cdot [\alpha, \beta\mathbf{u}]$

$= [0 - \mathbf{v}_{\parallel} \cdot \beta\mathbf{u}, 0 + \alpha\mathbf{v}_{\parallel} + \mathbf{v}_{\parallel} \times \beta\mathbf{u}]$

$= [-\mathbf{v}_{\parallel} \cdot \beta\mathbf{u}, \alpha\mathbf{v}_{\parallel}]$。（$\mathbf{v}_{\parallel}$平行于$\mathbf{u}$，所以$\beta\mathbf{u} \times \mathbf{v}_{\parallel} = 0$）

$= [-\beta\mathbf{u} \cdot \mathbf{v}_{\parallel}, \alpha\mathbf{v}_{\parallel}] = \mathrm{LHS}$（点乘遵守交换律）

c. 引理三

假设$\mathrm{v}_{\perp} = [0, \mathbf{v}_{\perp}]$是一个纯四元数，而 $\mathrm{q} = [\alpha, \beta\mathbf{u}]$，其中$\mathbf{u}$是一个单位向量，$\alpha, \beta \in \mathbb{R}$。在这种条件下，如果$\mathbf{v}_{\perp}$正交于$\mathbf{u}$，那么 $\mathrm{q}\mathrm{v}_{\perp} = \mathrm{v}_{\perp}\mathrm{q}^*$。

引理三的证明与之前的引理完全类似。

PS：

$\mathrm{LHS} = \mathrm{q}\mathrm{v}_{\perp}$

$= [\alpha, \beta\mathbf{u}] \cdot [0, \mathbf{v}_{\perp}]$

$= [0 - \beta\mathbf{u} \cdot \mathbf{v}_{\perp}, \alpha\mathbf{v}_{\perp} + 0 + \beta\mathbf{u} \times \mathbf{v}_{\perp}]$

$= [0, \alpha\mathbf{v}_{\perp} + \beta\mathbf{u} \times \mathbf{v}_{\perp}]$。（$\mathbf{v}_{\perp}$正交于$\mathbf{u}$，所以$\beta\mathbf{u} \cdot \mathbf{v}_{\perp} = 0$）

接下来再计算等式的右边。

PS：

$\mathrm{RHS} = \mathrm{v}_{\perp}\mathrm{q}^*$

$= [0, \mathbf{v}_{\perp}] \cdot [\alpha, -\beta\mathbf{u}]$

$= [0 + \mathbf{v}_{\perp} \cdot \beta\mathbf{u}, 0 + \alpha\mathbf{v}_{\perp} + \mathbf{v}_{\perp} \times (-\beta\mathbf{u})]$

$= [0, \alpha\mathbf{v}_{\perp} + \mathbf{v}_{\perp} \times (-\beta\mathbf{u})]$。（$\mathbf{v}_{\perp}$正交于$\mathbf{u}$，所以$\mathbf{v}_{\perp} \cdot \beta\mathbf{u} = 0$）

$= [0, \alpha\mathbf{v}_{\perp} - (-\beta\mathbf{u}) \times \mathbf{v}_{\perp}]$（$a \times b = -b \times a$ 叉乘的运算规则）

$= [0, \alpha\mathbf{v}_{\perp} + \beta\mathbf{u} \times \mathbf{v}_{\perp}] = \mathrm{LHS}$

现在，我们就能对引理一的公式做出最后的变形了。

即$v' = pp^*\mathrm{v}_{\parallel} + pp\mathrm{v}_{\perp} = p\mathrm{v}_{\parallel}p^* + p\mathrm{v}_{\perp}p^* = \mathrm{p}(\mathrm{v}_{\parallel} + \mathrm{v}_{\perp})p^*$。

显然，$(\mathrm{v}_{\parallel} + \mathrm{v}_{\perp})$其实就是 v，所以$v' = pvp^*$。到此为止，我们就解开了四元数与 3D 旋

转之间的谜题。虽然我们之前将v 划分成了两个分量，但是推导的结果其实并没有包含$v_{\parallel}$和$v_{\perp}$。3D 空间中任意一个旋转都能够用三个四元数相乘的形式表达出来。虽然推导的结果可能比较冗长，但是最终的结果非常简短。

4) 3D 旋转公式（四元数型，一般情况）

任意向量 **v** 沿着以单位向量定义的旋转轴**u**旋转 θ 度之后的$\boldsymbol{v}'$可以使用四元数乘法来获得。令 $v = [0, \mathbf{v}]$q=$\left[\cos\left(\frac{1}{2}\theta\right), \sin\left(\frac{1}{2}\theta\right)\mathbf{u}\right]$，那么$v' = qvq^* = qvq^{-1}$。

换句话说，如果我们有 q=[cos θ , sin θ **u**]，那么$v' = qvq^*$可以将 **v** 沿着**u** 旋转 2θ度。虽然这个公式非常简洁，但是它并不是那么直观。如果想了解它真正的含义的话，还需要将它还原到变形之前的式子（即引理一的公式）。

$$v' = qvq^* = qq^*v_{\parallel} + qqv_{\perp} = v_{\parallel} + q^2v_{\perp}。$$

这也就是说，qvq^*这个变换，对 v 平行于旋转轴的分量$v_{\parallel}$实施的变换是qq^*，这两个变换完全抵消了，也就是没有旋转。而对于正交于旋转轴的分量$v_{\perp}$，则实施的是两次变换$q^2 = qq$，将它旋转$\frac{\theta}{2} + \frac{\theta}{2} = \theta$度。

实际上，这个公式也是与前面推导的 3D 旋转公式—向量型—一般情况是完全等价的。如果感兴趣的话，可以试试证明下面这个等式，来确认我们的推导是没有错误的。（提示：证明可能会用到$\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \cdot \mathbf{c})\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{c}$ 这个公式）

$$qvq^* = [0, \cos\theta\,(\mathbf{u} \cdot \mathbf{v})\mathbf{u} + \sin\theta\,(\mathbf{u} \times \mathbf{v})]$$

虽然这个等式的证明可能看起来会非常麻烦，但我仍推荐去尝试证明一下。除了一些三角公式，证明所需要的全部知识我们都已经涉及到了。它会让我们更清楚地了解变换的整个过程以及为什么变换的结果是一个纯四元数。

因为所有的旋转四元数的实部都只是一个角度的余弦值，假设有一个单位四元数 q = [a, **b**]，如果我们想要提取它所对应旋转的角度，那么我们可以直接得到$\frac{\theta}{2} = \cos^{-1} a$，如果想要再获得旋转轴，那么只需要将**b**的每一项都除以$\sin\frac{\theta}{2}$就可以了$\mathbf{u} = \frac{\boldsymbol{b}}{\sin(\cos^{-1} a)}$。

5. 3D 旋转的矩阵形式

在实际的应用中，我们可能会需要将旋转与平移和缩放进行复合，所以需要用到四元数旋转的矩阵形式。它可以很容易地从上面这个公式推导出来。我们之前讨论过(四元数矩阵形式)，左乘一个四元数 q = a + bi + cj + dk 等同于下面这个矩阵。

$$L(q) = \begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix}$$

而右乘 q 等同于下面的矩阵。

$$R(q) = \begin{bmatrix} a & -b & -c & -d \\ b & a & d & -c \\ c & -d & a & b \\ d & c & -b & a \end{bmatrix}$$

所以，我们可以利用这两个公式将$v' = qvq^*$（3D 旋转公式四元数型一般情况）写成矩阵形式。假设 a = $\cos\left(\frac{1}{2}\theta\right)$，b = $\sin\left(\frac{1}{2}\theta\right)u_x$，b = $\sin\left(\frac{1}{2}\theta\right)u_y$，b = $\sin\left(\frac{1}{2}\theta\right)u_z$，q = a + bi + cj + dk，我们就能得到$qvq^* = L(q)R(q^*)v$（或者$R(q^*)L(\mathrm{q})v$，它们是等价的）

$$= \begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix} \begin{bmatrix} a & b & c & d \\ -b & a & -d & c \\ -c & d & a & -b \\ -d & -c & b & a \end{bmatrix} v$$（注意$R(q^*) = R(q)^T$）

$$= \begin{bmatrix} a^2+b^2+c^2+d^2 & ab-ab-cd+cd & ac+bd-ac-bd & ad-ac+cb-ad \\ ab-ab+cd-cd & b^2+a^2-d^2-c^2 & bc-ad-ad+bc & bd+ac+bd+ac \\ ac-bd-ac+bd & bc+ad+ad+bc & c^2-d^2+a^2-b^2 & cd+cd-ab-ab \\ ad+bc-bc-ad & bd-ac+bd-ac & cd+cd+ab+ab & d^2-c^2-b^2+a^2 \end{bmatrix} v$$

因为$a^2+b^2+c^2+d^2=1$，上面的公式能够化简为。

$$qvq^* = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-2c^2-2d^2 & 2bc-2ad & 2ac+2bd \\ 0 & 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab \\ 0 & 2bd-2ac & 2ab+2cd & 1-2b^2-2c^2 \end{bmatrix} v$$

这样我们就得到了 3D 旋转的矩阵形式。因为矩阵的最外圈不会对 v 进行任何变换，我们可以将它压缩成 3 × 3 矩阵（用作 3D 向量的变换）。

1) 3D 旋转公式—矩阵型

任意向量**v**沿着以单位向量定义的旋转轴 **u** 旋转 θ 角度之后的$\boldsymbol{v}'$可以使用矩阵乘法来获得。令 $a = \cos\left(\frac{1}{2}\theta\right)$，$b = \sin\left(\frac{1}{2}\theta\right)u_x$，$b = \sin\left(\frac{1}{2}\theta\right)u_y$，$b = \sin\left(\frac{1}{2}\theta\right)u_z$。

$$\boldsymbol{v}' = \begin{bmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2ac+2bd \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab \\ 2bd-2ac & 2ab+2cd & 1-2b^2-2c^2 \end{bmatrix} \mathbf{v}$$

虽然 3D 旋转的矩阵形式可能不如四元数形式简单，而且占用更多的空间，但是对于大批量的变换，使用预计算好的矩阵是比四元数乘法更有效率的。

6. 旋转的复合

在这一小节里，我们来证明一下使用四元数的旋转复合。旋转的复合其实在我们之前证明$q^2 = qq = [\cos 2\theta, \sin(2\theta)\,\mathbf{u}]$（引理一）的时候就已经涉及到一点了，但是这里我们考虑的是更一般的情况。

假设有两个表示沿着不同轴，不同角度旋转的四元数q_1, q_2，我们先对 v进行 q_1的变换，再进行q_2的变换，变换的最终结果是什么呢？

我们不妨将这两次变换分步进行。首先，我们实施q_1 的变换，变换之后的v'为$v' = q_1 v q_1^*$。

接下来，对v'进行q_2的变换，得到v''。

$$v'' = q_2 v' q_2^* = q_2 q_1 v q_1^* q_2^*。$$

我们需要对这两个变换进行复合，写为一个等价变换的形式，即$v'' = q_{net} v q_{net}^*$。

为了写成上面这种形式，我们还需要一个引理。

1) 引理四

对任意四元数$q_1 = [s, \boldsymbol{v}]$、$q_2 = [t, \boldsymbol{u}]$，有$q_1^* q_2^* = (q_2 q_1)^*$。

仍然使用与之前类似的方法来加以证明。

PS：

LHS $= q_1^* q_2^*$

$= [s, -\boldsymbol{v}] \cdot [t, -\boldsymbol{u}]$

$= [st - (-\boldsymbol{v}) \cdot (-\boldsymbol{u}), s(-\boldsymbol{u}) + t(-\boldsymbol{v}) + (-\boldsymbol{v}) \times (-\boldsymbol{u})]$

$= [st - \boldsymbol{v} \cdot \boldsymbol{u}, -s\boldsymbol{u} - t\boldsymbol{v} + \boldsymbol{v} \times \boldsymbol{u}]$。

接下来再计算等式的右边。

PS：

RHS $= (q_2q_1)^*$

$= ([t, \boldsymbol{u}] \cdot [s, \boldsymbol{v}])^*$

$= [ts - \boldsymbol{u} \cdot \boldsymbol{v}, t\boldsymbol{v} + s\boldsymbol{u} + \boldsymbol{u} \times \boldsymbol{v}]^*$

$= [st - \boldsymbol{v} \cdot \boldsymbol{u}, -s\boldsymbol{u} - t\boldsymbol{v} - \boldsymbol{u} \times \boldsymbol{v}]$

$= [st - \boldsymbol{v} \cdot \boldsymbol{u}, -s\boldsymbol{u} - t\boldsymbol{v} + \boldsymbol{v} \times \boldsymbol{u}] =$ LHS

通过上面的证明，我们可以确定$v'' = q_2q_1vq_1^*q_2^* = (q_2q_1)v(q_2q_1)^*$。

这也就是说，$q_{net} = q_2q_1$。注意四元数乘法的顺序，我们先进行的是q_1的变换，再进行q_2的变换。这和矩阵与函数的复合非常相似，都是从右往左叠加。

要注意的是，q_1与q_2的等价旋转q_{net}并不是分别沿着q_1和q_2的两个旋转轴进行的两次旋转。它是沿着一个全新的旋转轴进行的一次等价旋转，仅仅只有旋转的结果相同。

虽然我们讨论的是两个旋转的复合，但是它可以很容易推广到更多个旋转的复合。比如说我们还需要进行第三个旋转q_3.

$$v''' = q_3(q_2q_1)v(q_2q_1)^*q_3^* = (q_3q_2q_1)v(q_3q_2q_1)^*$$

它的等价旋转就是$q_{net} = q_3q_2q_1$。

7. 双倍覆盖

四元数与 3D 旋转的关系并不是一对一的，同一个 3D 旋转可以使用两个不同的四元数来表示。对任意的单位四元数$q = \left[\cos\left(\frac{1}{2}\theta\right), \sin\left(\frac{1}{2}\theta\right)\mathbf{u}\right]$，$q$与$-q$代表的是同一个旋转。如果$q$表示的是沿着旋转轴 **u** 旋转 θ 度，那么$-q$代表的是沿着相反的旋转轴−**u** 旋转 $(2\pi - \theta)$ 度。

$$-q = \left[-\cos\left(\frac{1}{2}\theta\right), -\sin\left(\frac{1}{2}\theta\right)\mathbf{u}\right] = \left[-\cos\left(\pi - \frac{1}{2}\theta\right), -\sin\left(\pi - \frac{1}{2}\theta\right)(-\mathbf{u})\right]$$

($\cos(\pi - \theta) = -\cos\theta, \sin(\pi - \theta) = \sin\theta$)，使用的是三角函数的诱导公式。

所以，这个四元数旋转的角度为 $2\left(\pi - \frac{1}{2}\theta\right) = 2\pi - \theta$。从下面的图中我们可以看到，这两个旋转是完全等价的。

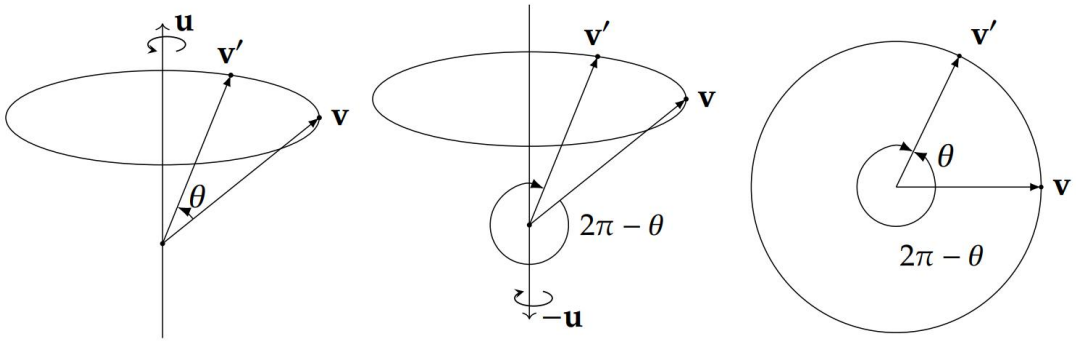


图 12.　完全等价的两个旋转

其实从四元数的旋转公式中也能推导出相同的结果，$(-q)\mathrm{v}(-q)^* = (-1)^2qvq^* = qvq^*$。

所以，我们经常说单位四元数与 3D 旋转有一个「2 对 1 满射同态」(2-1 Surjective Homomorphism) 关系，或者说单位四元数双倍覆盖 (Double Cover)了 3D 旋转。它的严格证明会用到一些李群的知识，但我相信这里给出的解释已经足够直观了。我不想让这个教程变得太复杂，所以在这里就省略了。

因为这个映射是满射，我们可以说所有的单位四元数都对应着一个 3D 旋转。或者说，

一个四维单位超球面（也叫做$\mathbb{s}^3$）上任意一点所对应的四元数（$\|q\| = 1$）都对应着一个 3D 旋转，这一点会在之后讨论旋转插值的时候会用到。

有一点需要注意的是，虽然q与$-q$是两个不同的四元数，但是由于旋转矩阵中的每一项都包含了四元数两个分量的乘积，它们的旋转矩阵是完全相同的。旋转矩阵并不会出现双倍覆盖的问题。

8. 指数形式

在讨论复数的时候，我们利用欧拉公式将 2D 的旋转写成了$v' = e^{i\theta}v$这样的指数形式。实际上，我们也可以利用四元数将 3D 旋转写成类似的形式。

类似于复数的欧拉公式，四元数也有一个类似的公式。如果 **u** 是一个单位向量，那么对于单位四元数 $u = [0, \mathbf{u}]$，有$e^{u\theta} = \cos\theta + u\sin\theta = \cos\theta + \boldsymbol{u}\sin\theta$。

这也就是说，$q = [\cos\theta + \sin(\theta)\,\boldsymbol{u}]$，可以使用指数表示为$e^{u\theta}$。这个公式的证明与欧拉公式的证明非常类似，直接使用级数展开就可以了。

注意，因为**u**是一个单位向量，$u^2 = [\mathbf{u}\cdot\mathbf{u}, 0] = -\|q\|^2 = -1$。这与欧拉公式中的 i 是非常类似的。

有了指数型的表示方式，我们就能够将之前四元数的旋转公式改写为指数形式了。

1) 3D 旋转公式—指数型

任意向量 **v** 沿着以单位向量定义的旋转轴 **u**旋转 θ 角度之后的v'可以使用四元数的指数表示。令 $v = [0, \boldsymbol{v}]$、$u = [0, \boldsymbol{u}]$，那么$v' = e^{u\frac{\theta}{2}}ve^{-u\frac{\theta}{2}}$。

有了四元数的指数定义，我们就能够定义四元数的更多运算了。首先是自然对数 log，对任意单位四元数$q = [\cos\theta, \sin\theta\,\mathbf{u}]$，$\log(q) = \log(e^{u\theta}) = [0, \boldsymbol{u}\theta]$。

接下来是单位四元数的幂运算，$q^t = (e^{u\theta})^t = [\cos(t\theta), \sin(t\theta)\,\mathbf{u}]$。

可以看到，一个单位四元数的 t 次幂等同于将它的旋转角度缩放至 t 倍，并且不会改变它的旋转轴（u 必须是单位向量，所以一般不能与 t 结合）。这些运算会在之后讨论四元数插值时非常有用。

七. 四元数插值（基础）

有了这么多四元数的前置知识，我们就能开始讨论四元数的插值 (Interpolation)了。与其它的旋转表示形式不同，四元数的一些性质让它的插值变得非常简单。

假设有两个旋转变换$q_0 = [\cos(\theta_0), \sin(\theta_0)\,\mathbf{u}_0]$和$q_1 = [\cos(\theta_1), \sin(\theta_1)\,\mathbf{u}_1]$，我们希望找出一些中间变换$q_t$，让初始变换$q_0$能够平滑地过渡到最终变换$q_1$。$t$ 的取值可以是 $t \in [0,1]$。当 $t = 0$ 时 q_t 等同于初始变换q_0，而 $t = 1$ 时q_t等同于最终变换q_1。

由于插值的对象是两个变换，想象起来可能非常困难，我们不妨假设 3D 空间中有任意一个向量 v。那么q_0会将v变换到$v_0 = q_0 v q_0^*$，而q_1会将v变换到$v_1 = q_1 v q_1^*$。我们需要找出中间向量$v_t = q_t v q_t^*$所对应的变换q_t，使v旋转到v_0与v_1中间的某个位置v_t'。

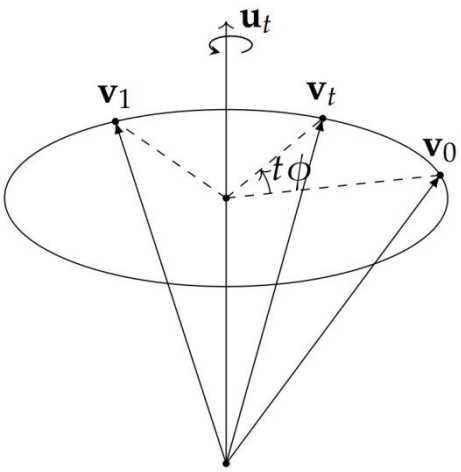


图 13.　中间向量

我们可以看到，这个旋转的变化量其实对应的仍是一个旋转。它将由q_0变换到v_0的向量进一步旋转到v_t。这个旋转拥有某一个固定的旋转轴$\mathbf{u}_t$，我们只需要缩放这个变换所对应的角度 ϕ 就能够达到插值的目的了。

那么，现在的问题是，我们该怎么获得这个旋转的变化量呢？我们不妨思考一下有什么变换Δq，能将已经旋转到v_0的向量v直接变换到v_1。这其实就是一个旋转的复合，我们先进行q_0变换，再进行Δq变换，它们复合的结果需要等于q_1变换。

也就是说，$\Delta q q_0 = q_1$。

则有$\Delta q q_0 q_0^{-1} = q_1 q_0^{-1}$（右乘$q_0^{-1}$）

$\Delta q = q_1 q_0^{-1}$（运用了四元数的逆）

因为所有的旋转q都是单位四元数，$q^{-1} = q^*$（运用了单位四元数逆和共轭），它又可以写成$\Delta q = q_1 q_0^*$。

如果我们对Δq取 t 次方，$(\Delta q)^t$就能缩放这个旋转所对应的角度了。所以，我们就能得出插值的公式$q_t = Slerp(q_0, q_1; t) = (q_1 q_0^*)^t q_0$。

可以发现，当 $t = 0$ 时，$q_t = (q_1 q_0^*)^0 q_0 = q_0$；而当 $t = 1$ 时，$q_t = (q_1 q_0^*)^1 q_0 = q_1 (q_0^* q_0) = q_1$。如果 t 为中间值，比如说 $t = 0.4$ 时，$q_t = (q_1 q_0^*)^{0.4} q_0$，它会先进行$q_0$变换将$v$变换到$v_0$，并在此基础上向$v_1$旋转 40%。

这个公式虽然对四元数插值的分析很有帮助，但是它的计算不仅涉及到多个四元数的乘法，而且包含幂运算，在实际应用中的效率很低，我们希望找出一个更高效的插值方法。

我们将这个插值方法叫做「Slerp」，但是我们暂时不会解释它具体是什么意思，为了理解它我们还需要研究一下 3D 空间的旋转与四元数的 4D 向量空间之间的关系。

八. 3D 空间旋转变化量 vs. 四元数 4D 向量空间夹角

为了探讨这个关系，我们来实际计算一下Δq。由于这个关系和角度有关，我们只需要关心 Δq的实部就可以了。

$$\begin{aligned}\Delta q = q_1 q_0^* &= [\cos(\theta_1), \sin(\theta_1)\mathbf{u}_1][\cos(\theta_0), \sin(\theta_0)\mathbf{u}_0] \\ &= [\cos(\theta_0)\cos(\theta_1) - (\sin(\theta_1)\mathbf{u}_1)\cdot(-\sin(\theta_0)\mathbf{u}_0), \ldots] \\ &= [\cos(\theta_0)\cos(\theta_1) + (\sin(\theta_1)\mathbf{u}_1)\cdot(\sin(\theta_0)\mathbf{u}_0), \ldots]\end{aligned}$$

如果将q_0和q_1看作是两个四维向量，我们能发现非常巧合的一个情况。Δq的实部正好是q_0与 q_1点乘的结果$q_0 \cdot q_1$！

因为q_0与q_1都是单位四元数，$q_0 \cdot q_1$正好是这两个四元数在 4D 空间中夹角的余弦值，我们将这个夹角称为 θ。那么$q_0 \cdot q_1 = \cos\theta$。

我们又知道，Δq表示的也是一个旋转，而如果它代表的旋转的角度是 2ϕ，那么Δq的实

数部为$\Delta q = [\cos\phi, ...]$。

$$\Delta q = [\cos\phi, ...] = [\cos\theta, ...] \qquad (\cos\phi = \cos\theta)$$

因为ϕ和θ都是夹角，ϕ、$\theta \in [0, \pi]$，所以这个方程有唯一的解$\phi = \theta$。

这也就是说，q_0与q_1作为向量在 4D 四元数空间中的夹角θ，正好是它们旋转变化量 Δq 的所代表旋转的角度的一半，即$\theta = \frac{1}{2}(2\phi)$。所以，我们可以直接用插值向量的方法对旋转进行插值。

为了更直观地理解这层关系，请看下面这两幅图。虽然四元数是处于四维空间之内的，但是因为只有两个四元数，我们可以将它们投影到一个二维的圆上来，也就是左图。右图则是 3D 空间中发生的旋转改变。

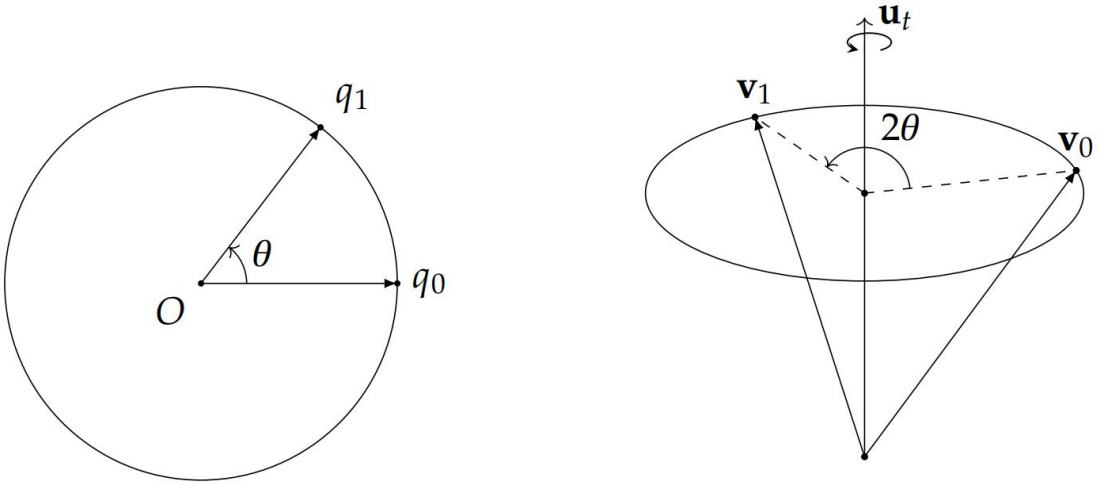


图 14.　左图是二维视角，右图是三维视角

可以看到，当 q_1与q_0之间的夹角为θ时，旋转的变化量正好是 2θ。如果我们在圆上有一个单位四元数q_t，使得它与q_0 的夹角为 tθ，与q_1的夹角为$(1 - t)\theta$，那么我们就能保证在 3D 空间中，它相对于q_0的旋转变化量为 2tθ，相对于q_1的旋转变化量为 $2(1 - t)\theta$。

现在，两个单位四元数的插值就被我们简化到了一个圆上（其实是超球面的一部分）两个向量的插值，我们能直接套用向量的插值公式对两个四元数进行插值。接下来，我们就来讨论如何对两个向量进行插值。

九. Lerp、Nlerp、Slerp

不管是哪种插值方法，我们都希望将中间向量$\boldsymbol{v_t}$写为初始向量$\boldsymbol{v_0}$和最终向量$\boldsymbol{v_1}$的线性组合，也就是说$\boldsymbol{v_t} = \alpha\boldsymbol{v_0} + \beta\boldsymbol{v_1}$。其中，系数$\alpha$与$\beta$都是$t$的函数。不同的插值方法只是拥有不同的系数而已。

1. Lerp

我们首先来看一下两个向量插值最简单的一种形式：线性插值（Linear Interpolation），也叫做「Lerp」。Lerp 会沿着一条直线进行插值，如果将$\boldsymbol{v_0}$和$\boldsymbol{v_1}$看做是三角形的两个边，那么$\boldsymbol{v_t}$会指向三角形的第三条边。

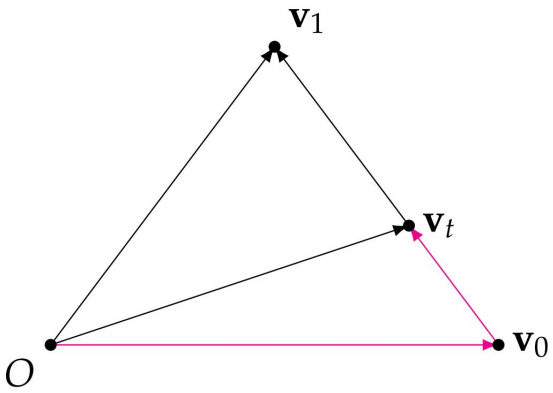


图 15.　线性插值示意图

从图中可以看到，我们能将$\boldsymbol{v_t}$写为两个向量的和（用红色标出，即向量加法）。其中一个向量正是$\boldsymbol{v_0}$，而另一个向量则是 $(\boldsymbol{v_1}-\boldsymbol{v_0})$乘上一个系数，我们直接将 t 作为这个系数，所以就有。

$$\boldsymbol{v_t}=\mathrm{Lerp}(\boldsymbol{v_0},\boldsymbol{v_1},\mathrm{t})=\boldsymbol{v_0}+t(\boldsymbol{v_1}-\boldsymbol{v_0})=(1-t)\boldsymbol{v_0}+t\boldsymbol{v_1}$$

当 t=0 时，$\boldsymbol{v_t}=(1-0)\boldsymbol{v_0}+0\boldsymbol{v_1}=\boldsymbol{v_0}$；当 t=1 时，$\boldsymbol{v_t}=(1-1)\boldsymbol{v_0}+1\boldsymbol{v_1}=\boldsymbol{v_1}$。

如果将 Lerp 的结果应用到单位四元数上，我们就能得到$q_t=\mathrm{Lerp}(q_0,q_1,\mathrm{t})=(1-\mathrm{t})q_0+tq_1$。

当然，因为我们是沿着一条直线（也就是圆上的一个弦）进行插值的，这样插值出来的四元数并不是单位四元数。

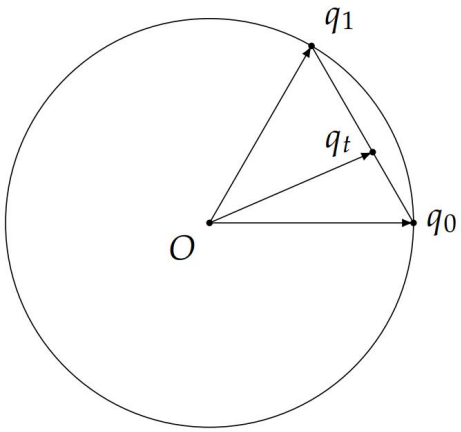


图 16.　插值结果

2. Nlerp

虽然这样插值出来的q_t并不是单位四元数，但只要将q_t除以它的模长$\|q_t\|$就能够将其转化为一个单位四元数了。

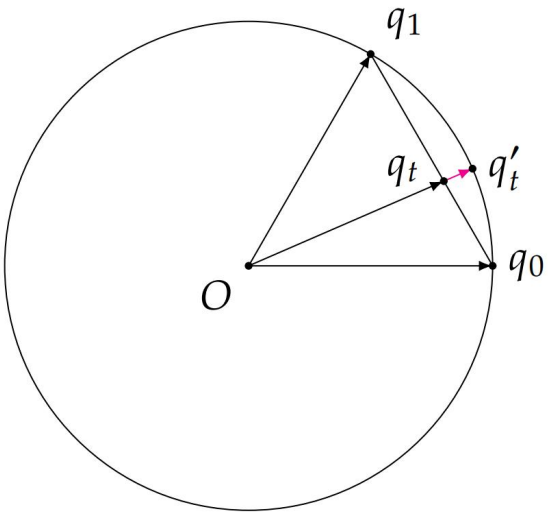


图 17.　插值结果转化为单位四元数

我们将这种先对向量进行插值，再进行规一化 (Normalization) 的插值方法称为规一化线性插值（Normalized Linear Interpolation），或者「Nlerp」。与 Lerp 不同，Nlerp 的两个输入向量必须是单位向量，否则插值出来的结果不会经过初始和最终向量。下面分别是向量和四元数的 Nlerp 公式。

$$\boldsymbol{v_t} = \mathrm{Nlerp}(\boldsymbol{v_0}, \boldsymbol{v_1}, \mathrm{t}) = \frac{(1-t)\boldsymbol{v_0} + t\boldsymbol{v_1}}{\|(1-t)\boldsymbol{v_0} + t\boldsymbol{v_1}\|}$$

$$q_t = \mathrm{Nlerp}(q_0, q_1, \mathrm{t}) = \frac{(1-t)q_0 + tq_1}{\|(1-t)q_0 + tq_1\|}$$

Nlerp 插值仍然存在有一定的问题，当需要插值的弧比较大时，$\boldsymbol{v_t}$的角速度会有显著的变化。

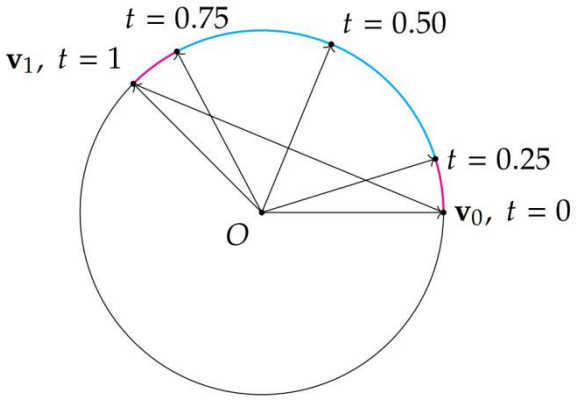


图 18.　插值的弧比较大时

这五个 t 值将整个弧和弦分割成了四个部分。虽然弦上的四段是等长的，但是四个弧是完全不相等的。t = 0 到 t = 0.25 之间的弧（红色）明显比 t = 0.25 到 t = 0.50 的弧（蓝色）要短了不少。

这也就是说，在同等时间内，$\boldsymbol{v_t}$扫过的角度是不同的。$\boldsymbol{v_t}$扫过的速度（或者说角速度）首先会不断地增加，到t = 0.50 之后会开始减速，所以 Nlerp 插值不能保证均匀的角速度。

3. Slerp-球面线性插值

为了解决 Nlerp 的问题，我们可以转而对角度进行线性插值。也就是说，如果$\boldsymbol{v_1}$和$\boldsymbol{v_2}$ 之间的夹角为θ，那么$\theta_t = (1-t)\cdot 0 + t\theta = t\theta$。

因为对角度线性插值直接是让向量在球面上的一个弧上旋转，所以又称球面线性插值

(Spherical Linear Interpolation)，或者「Slerp」。类比于 Lerp 是平面上的线性插值，Slerp 是球面上的线性插值. 我们前面讨论的四元数插值公式正是一个对四元数在四维超球面上的旋转，所以它是 Slerp 的一个等价公式。

上面公式并没有涉及到任何的向量，我们希望将 $\boldsymbol{v_t}$写为$\boldsymbol{v_0}$和$\boldsymbol{v_0}$的线性组合，$\boldsymbol{v_t} = \alpha\boldsymbol{v_0} + \beta\boldsymbol{v_1}$。 (1)

注意这里的$\boldsymbol{v_0}$和$\boldsymbol{v_1}$仍是单位向量。为了求出这其中的 α 和 β，我们需要借助图像来找出一些关系。

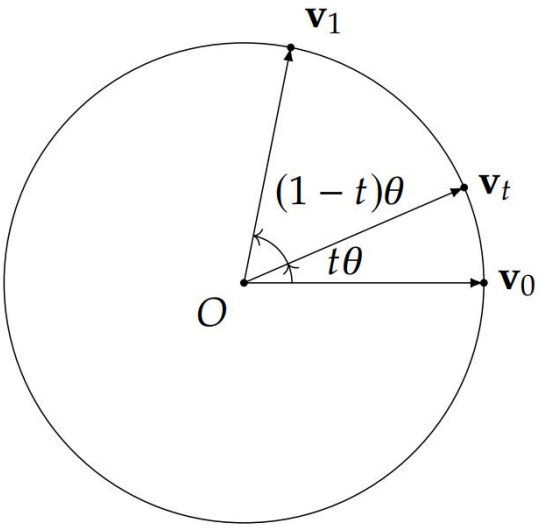


图 19.　角度之间的关系图像

因为图中涉及到很多的角度关系，我们可以先对(1)式的两边同时点乘$\boldsymbol{v_0}$。

$$\boldsymbol{v_0} \cdot \boldsymbol{v_t} = \boldsymbol{v_0} \cdot (\alpha\boldsymbol{v_0} + \beta\boldsymbol{v_1})$$

$$\boldsymbol{v_0} \cdot \boldsymbol{v_t} = \alpha(\boldsymbol{v_0} \cdot \boldsymbol{v_0}) + \beta(\boldsymbol{v_0} \cdot \boldsymbol{v_1})$$（分配率）

我们知道，$\boldsymbol{v_0}$和$\boldsymbol{v_t}$之间的夹角是$t\theta$，$\boldsymbol{v_0}$与它自身之间的夹角为 0，$\boldsymbol{v_0}$和$\boldsymbol{v_1}$之间的夹角是θ，而且所有的向量都是单位向量。

$$\cos(t\theta) = \alpha + \beta\cos\theta \quad (2)$$

同理，我们将(1)的两边同时点乘$\boldsymbol{v_1}$，构造出第二个方程。

$$\boldsymbol{v_1} \cdot \boldsymbol{v_t} = \boldsymbol{v_1} \cdot (\alpha\boldsymbol{v_0} + \beta\boldsymbol{v_1})$$

$$\boldsymbol{v_1} \cdot \boldsymbol{v_t} = \alpha(\boldsymbol{v_1} \cdot \boldsymbol{v_0}) + \beta(\boldsymbol{v_1} \cdot \boldsymbol{v_1})$$

$$\cos((1-t)\theta) = \alpha\cos\theta + \beta \quad (3)$$

现在，我们就有了两个方程以及两个未知数，我们只需要解(2)、(3)这两个方程，求出 α 和 β 就能获得 Slerp 的公式了。

由(2)我们可以得到$\alpha = \cos(t\theta) - \beta\cos(\theta)$ (4)

将(4)代入(3)，利用一些三角恒等式，我们就能解出β。

$$\cos((1-t)\theta) = (\cos(t\theta) - \beta\cos\theta)\cos\theta + \beta$$

$$\cos((1-t)\theta) = \cos(t\theta)\cos\theta - \beta\cos^2\theta + \beta$$

$$\beta(1-\cos^2\theta) = \cos((1-t)\theta) - \cos(t\theta)\cos\theta$$

$$\beta = \frac{\cos(\theta - t\theta) - \cos(t\theta)\cos\theta}{\sin^2\theta}$$（使用了三角函数的平方关系）

$$\beta = \frac{\cos\theta\cos(t\theta) + \sin\theta\sin(t\theta) - \cos(t\theta)\cos\theta}{\sin^2\theta}$$（使用了三角函数的两角和差）

$$\beta = \frac{\sin\theta\sin(t\theta)}{\sin^2\theta}$$

$$\beta = \frac{\sin(t\theta)}{\sin\theta}$$

现在我们就得到了β，将β代入(4)就能解出α。

$$\alpha = \cos(t\theta) - \left(\frac{\sin(t\theta)}{\sin\theta}\right)\cos\theta$$

$$= \frac{\cos(t\theta)\sin\theta - \sin(t\theta)\cos\theta}{\sin\theta}$$

$$= \frac{\sin((1-t)\theta)}{\sin\theta}$$

将α和β代回(1)，我们可以得到向量的 Slerp 的公式。

$$\boldsymbol{v_t} = \mathrm{Slerp}(\boldsymbol{v_0}, \boldsymbol{v_1}, t) = \frac{\sin\left((1-t)\theta\right)}{\sin\theta}\boldsymbol{v_0} + \frac{\sin(t\theta)}{\sin\theta}\boldsymbol{v_1}$$

类似地，我们有四元数的 Slerp 公式。

$$q_t = \mathrm{Slerp}(q_0, q_1, t) = \frac{\sin\left((1-t)\theta\right)}{\sin\theta}q_0 + \frac{\sin(t\theta)}{\sin\theta}q_1$$

其中q_0与q_1之间的夹角θ可以直接使用它们点乘的结果来得出，即$\theta = cos^{-1}(q_0 \cdot q_1)$。

这里导出的公式会比之前利用幂运算的公式要高效很多，但是它仍然涉及到三个三角函数以及一个反三角函数的运算，所以还是会比 Nlerp 要慢一点。

如果要插值的角度比较小的话，Nlerp 其实相对于 Slerp 的误差并没有那么大。为了提高效率，我们经常会使用 Nlerp 来代替 Slerp。我们也能用一些数值分析的方法来近似并优化四元数的 Slerp。

除了效率问题之外，我们在实现 Slerp 时要注意，如果单位四元数之间的夹角θ非常小，那么 $\sin(\theta)$ 可能会由于浮点数的误差被近似为 0.0，从而导致除以 0 的错误。所以，我们在实施 Slerp 之前，需要检查两个四元数的夹角是否过小（或者完全相同）。一旦发现这种问题，我们就必须改用 Nlerp 对两个四元数进行插值，这时候 Nlerp 的误差非常小所以基本不会与真正的 Slerp 有什么区别。

4. 双倍覆盖带来的问题

如果还记得，两个不同的单位四元数 q 与 $-q$ 对应的其实是同一个旋转（双倍覆盖），这个特性显然会对我们的插值造成一些影响。虽然 q 与 $-q$ 对于向量变换的最终效果是完全相同的，但是它们作为向量相差了 π 弧度。

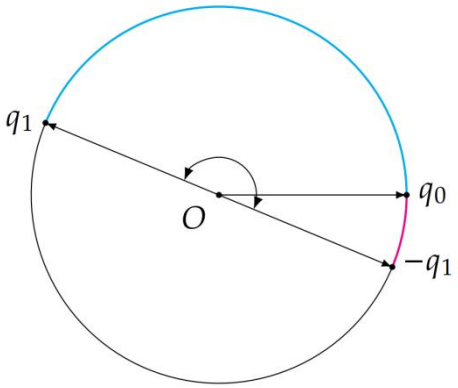


图 20.　q 与 $-q$相差π弧度

可以看到，虽然我们能够将q_0向左插值至q_1（蓝色的弧），但这会将 3D 空间中的向量旋转接近360°，而实际上这两个旋转相差并没有那么多，它并不是 3D 空间中的弧面最短路径（Geodesic）。而如果我们将q_0向右插值至等价的 $-q_1$（红色的弧），它的旋转变化量

就会比插值到q_1要小很多，所以q_0插值到$-q_1$才是插值的最短路径。

这也就告诉我们，在对两个单位四元数进行插值之前，我们需要先检测q_0与q_1之间是否是钝角，即检测它们点积的结果$q_0 \cdot q_1$ 是否为负数（点积用于判断角度）。如果$q_0 \cdot q_1 < 0$，那么我们就反转其中的一个四元数，比如说将q_1改为$-q_1$，并使用q_0与$-q_1$之间新的夹角来进行插值，这样才能保证插值的路径是最短的。

十. Squad

Slerp 其实仍然存在一定的问题。因为我们是对角度进行线性插值，旋转的角速度虽然是固定的，但它的值是与夹角θ成正比的，（$\frac{d\theta_t}{dt} = \frac{d}{dt}(t\theta) = \theta$，准确说是角速率）。如果我们需要对多个四元数进行插值，对每一对四元数使用 Slerp 插值虽然能够保证每两个四元数之间的角速度是固定的，但是角速度会在切换插值的四元数时出现断点，或者说在切换点不可导。比如说我们有 q_0，q_1，q_2三个四元数所组成的序列，如果分别对q_0q_1、q_1q_2使用 Slerp 插值，那么当$q_t = q_1$时，角速度会突然改变，这并不是我们所希望的结果。我们希望能以牺牲固定角速度为条件，让插值的曲线不仅是连续的，而且让它的一阶甚至是高阶导数也是连续的（曲线连续我们称为C^0连续，达到一阶导数连续就叫做C^1连续，在此基础上达到二阶导数连续叫做C^2连续，以此类推）。

解决这个问题的方法有很多，在网上能找到很多论文，但是每一种方法想要完全理解都不是那么容易的，而且它们一般都比普通的 Slerp 或者 Nlerp 要慢得多。我们在这里只会简单介绍最常见到而且也是最直观的球面四边形插值 (Spherical and quadrangle)，也叫做「Squad」。它由 Ken Shoemake 在 1987 年提出（这篇 Paper 已经找不到了）。

Squad 仍然是平面上的向量插值衍生到超球面的结果，所以我们关注的重点首先仍然是向量的插值。下面的内容可能会需要一些 Bézier 曲线以及样条的前置知识，但即便不知道也没有关系，我会在下面简单地概括一下这些知识。

假设我们有一个向量的序列$\boldsymbol{v_0}$，$\boldsymbol{v_1}$，...，$\boldsymbol{v_n}$，如果我们想对这个序列进行插值，那么我们可以分别对每一对向量$\boldsymbol{v_i}$和$\boldsymbol{v_{i+1}}$进行插值，然后将插值的曲线连接起来，也就是我们所说的样条（Spline）。如果直接使用 Lerp 的话，我们会得到这样的结果（假设我们只有五个向量需要插值$\boldsymbol{v_0}$，$\boldsymbol{v_1}$，$\boldsymbol{v_2}$，$\boldsymbol{v_3}$，$\boldsymbol{v_4}$）。

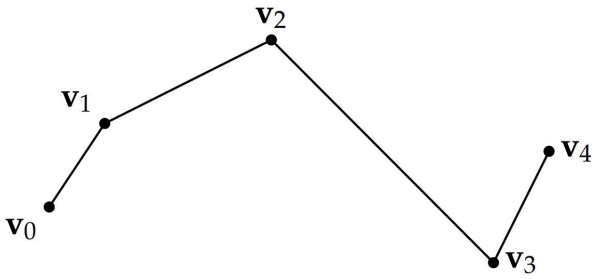


图 21.　线性插值结果

很明显，这个曲线虽然是连续的，但是它的一阶导数（切线）在切换插值向量时都不是连续的。为了解决这个问题，我们最常使用的就是 Bézier 曲线。我们一开始的想法可能会是将中间的$\boldsymbol{v_1}$，$\boldsymbol{v_2}$，$\boldsymbol{v_3}$作为控制点，直接使用一个四次 Bézier 曲线（因为有五个点）来生成这个近似曲线。但是 Bézier 曲线只会经过初始点与最终点（插值），一般不会经过中间的控制点（近似），所以这样求出来的曲线虽然是可导的，但是插值曲线不会经过中间的三个向量。

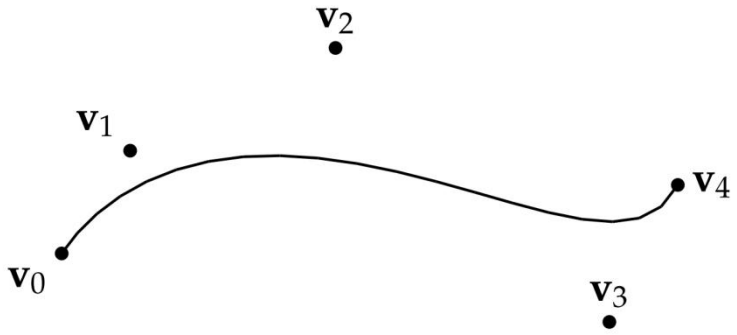


图 22.　四阶贝塞尔曲线

1. 三阶 Bézier 曲线

为了解决这个问题，我们可以分段对每两个向量$\boldsymbol{v_i}$和$\boldsymbol{v_{i+1}}$之间使用 Bézier 曲线进行插值，之后将所有的曲线（样条）连接起来。因为我们需要让曲线的一阶导数（或者说曲线的趋势）连续，我们还需要知道它们的前一个向量$\boldsymbol{v_{i-1}}$和后一个向量$\boldsymbol{v_{i+2}}$，并且用它们生成两个控制点 $\boldsymbol{S_i}$和 $\boldsymbol{S_{i+1}}$ 来控制曲线的趋势。我们会使用$\boldsymbol{v_i}$和$\boldsymbol{v_{i+1}}$作为端点（曲线会经过这两个点），$\boldsymbol{S_i}$和 $\boldsymbol{S_{i+1}}$作为中间的控制点，使用一个三阶 Bézier 曲线（Cubic Bézier Curve，四个点）来近似这个两个向量之间的插值。

在我们的例子中，因为我们一共有四对向量（$\boldsymbol{v_0v_1}$、$\boldsymbol{v_1v_2}$、$\boldsymbol{v_2v_3}$、$\boldsymbol{v_3v_4}$），我们会使用四个三阶 Bézier 曲线对这五个点进行插值。我们知道，对于三阶 Bézier 曲线所产生的样条，如果想让最终的插值曲线达到C^1连续，则需要让前一个样条在$\boldsymbol{v_i}$的控制点与当前样条在$\boldsymbol{v_i}$的控制点分别处于最终曲线在$\boldsymbol{v_i}$处切线对等的两侧。

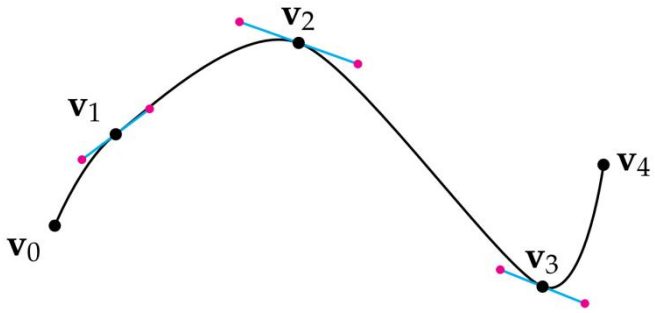


图 23.　构建控制杆

在上面的曲线中，蓝色的线就是曲线在点$\boldsymbol{v_i}$处的切线，红色的点就是三阶 Bézier 曲线的控制点，它们分别处于切线对等的两侧。对于两个端点$\boldsymbol{v_0}$和$\boldsymbol{v_4}$，我们直接将这两个向量的控制点取为它们本身（这不是唯一的做法，但这样是可行的），最终得到一个平滑的曲线。

我们希望将类似的逻辑带到四元数的超球面上，得到四元数序列的插值的方法，但在此之前我们需要了解如何构造一个三阶 Bézier 曲线。

2. de Casteljau 算法

Bézier 曲线的构造有个著名的递归算法叫做 de Casteljau 算法（de Casteljau’s Algorithm），它对任意次方的 Bézier 曲线都是成立的，但是这里我们只关注三阶 Bézier 曲线的情况。

这个算法最基本的思想就是线性插值的嵌套。假设我们有四个向量$\boldsymbol{v_0}$，$\boldsymbol{v_1}$，$\boldsymbol{v_2}$，$\boldsymbol{v_3}$，那么我们可以同下面的方法获得最终的三阶 Bézier 曲线。

首先，我们对每一对向量 $\boldsymbol{v_0v_1}$、$\boldsymbol{v_1v_2}$、$\boldsymbol{v_2v_3}$进行线性插值，获得 $\boldsymbol{v_{01}}$、$\boldsymbol{v_{12}}$、$\boldsymbol{v_{23}}$。

$$\boldsymbol{v_{01}} = \mathrm{Lerp}(\boldsymbol{v_0}, \boldsymbol{v_1}; \mathrm{t})$$

$$\boldsymbol{v_{12}} = \mathrm{Lerp}(\boldsymbol{v_1}, \boldsymbol{v_2}; \mathrm{t})$$

$$\boldsymbol{v_{23}} = \mathrm{Lerp}(\boldsymbol{v_2}, \boldsymbol{v_3}; \mathrm{t})$$

之后，我们对$\boldsymbol{v_{01}v_{12}}$和$\boldsymbol{v_{12}v_{23}}$这两对向量进行线性插值，获得$\boldsymbol{v_{012}}$和$\boldsymbol{v_{123}}$。

$$\boldsymbol{v_{012}} = \mathrm{Lerp}(\boldsymbol{v_{01}}, \boldsymbol{v_{12}}; \mathrm{t})$$

$$\boldsymbol{v_{123}} = \mathrm{Lerp}(\boldsymbol{v_{12}}, \boldsymbol{v_{23}}; \mathrm{t})$$

最后，对$\boldsymbol{v_{012}}$和$\boldsymbol{v_{123}}$进行线性插值获得$\boldsymbol{v_{0123}}$，这个向量就是我们想要的最终结果，它就是三阶 Bézier 曲线上的点，$\boldsymbol{v_{0123}} = \mathrm{Lerp}(\boldsymbol{v_{012}}, \boldsymbol{v_{123}}; \mathrm{t})$。

虽然这个算法看起来很繁琐，但是我们可以通过一张图来理解它（取t ＝0.4）。

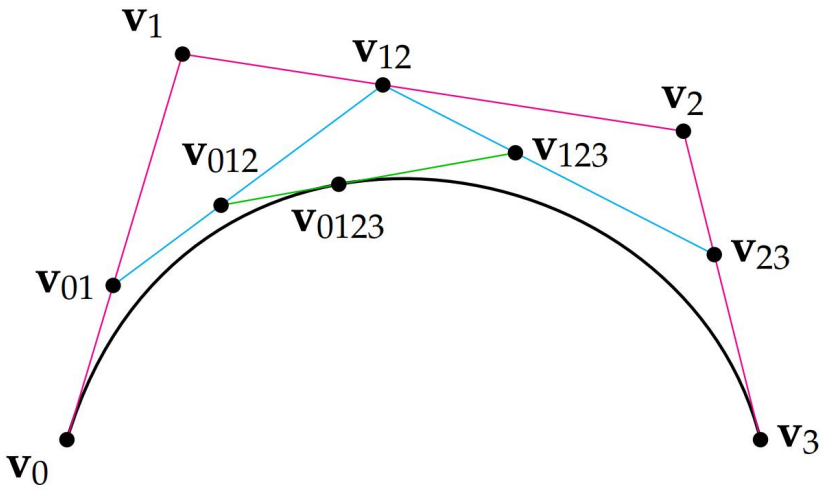


图 24.　三阶贝塞尔

可以看到，虽然我们一直在使用线性插值，最终获得的却是一条三阶 Bézier 曲线（黑色的线）。

如果将这些式子合并起来，我们就能得到三阶 Bézier 曲线的递归公式。因为这个式子太长了，我将 $\mathrm{Lerp}(\boldsymbol{v_i}, \boldsymbol{v_{i+1}}; \mathrm{t})$简写成 $\mathrm{L}(\boldsymbol{v_i}, \boldsymbol{v_{i+1}}; \mathrm{t})$。

$$\mathit{Bézier}(\boldsymbol{v_0}, \boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}; \mathrm{t}) = \mathrm{L}(\mathrm{L}(\mathrm{L}(\boldsymbol{v_0}, \boldsymbol{v_1}; \mathrm{t}), (\boldsymbol{v_1}, \boldsymbol{v_2}; \mathrm{t}); t), L(L(\boldsymbol{v_1}, \boldsymbol{v_2}; \mathrm{t}), L(\boldsymbol{v_2}, \boldsymbol{v_3}; \mathrm{t}); t); t)$$

如果将 Lerp 的定义 $\mathrm{Lerp}(\boldsymbol{v_i}, \boldsymbol{v_{i+1}}; \mathrm{t}) = (1-t)\boldsymbol{v_i} + t\boldsymbol{v_{i+1}}$不断代入并展开的话，我们能获得下面的一个式子。

$$\mathit{Bézier}(\boldsymbol{v_0}, \boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}; \mathrm{t}) = (1-t)^3\boldsymbol{v_0} + 3(1-t)^2t\boldsymbol{v_1} + \boldsymbol{3}(1-t)t^2\boldsymbol{v_2} + t^3\boldsymbol{v_3}。$$

因为每项的次数都是 3，所以我们说它是一个三阶 Bézier 曲线。

我们可以直接将递归的公式运用到四元数上，得到四元数的球面 Bézier 曲线公式，但因为球面的线性插值不是 Lerp 而是 Slerp，我们需要将公式中所有的 Lerp 全部换成 Slerp（可以想象一下，将四个向量形成的四边形看作是一个网格（Mesh），之后将这个网格贴在球面上）。

同样因为公式太长，我会将 $\mathrm{Slerp}(q_i, q_{i+1}; \mathrm{t})$简写为 $\mathrm{S}(q_i, q_{i+1}; \mathrm{t})$。

$$\mathit{SBézier}(q_0, q_1, q_2, q_3; \mathrm{t}) = \mathrm{S}(\mathrm{S}(\mathrm{S}(q_0, q_1; \mathrm{t}), (q_1, q_2; \mathrm{t}); t), S(S(q_1, q_2; \mathrm{t}), S(q_2, q_3; \mathrm{t}); t); t)。$$

这个其实就是 Ken Shoemake 在 1985 年的那篇 Paper 里提出的插值方法，他也提供了控制点的公式。然而，很明显这个方法实在是太复杂了。仅仅是一个 Slerp 就要使用四个三角函数，而我们这里一共有 7 个 Slerp，如果真的要使用它进行插值会对性能产生非常大的影响。

3. Squad

于是，Shoemake 在 1987 年提出了一个更高效的近似算法，也就是我们熟悉的 Squad。

我们首先仍然来看平面中向量的情况．我把向量的 Squad 算法叫做 Quad，代表「Quadrangle」，Quad 有很多歧义，而且我好像没看到别人这么叫过，但是为了简便这里就

暂时这样称呼它吧。

与三阶 Bézier 曲线嵌套了三层一阶插值不同，Quad 使用的是一层二阶插值嵌套了一层一阶插值。

我们首先是分别对$\boldsymbol{v_0v_3}$和$\boldsymbol{v_1v_2}$进行插值，获得$\boldsymbol{v_{03}}$和$\boldsymbol{v_{12}}$。

$$\boldsymbol{v_{03}} = \mathrm{Lerp}(\boldsymbol{v_0}, \boldsymbol{v_3}; \mathrm{t})$$

$$\boldsymbol{v_{12}} = \mathrm{Lerp}(\boldsymbol{v_1}, \boldsymbol{v_2}; \mathrm{t})$$

之后，我们使用 $2\mathrm{t}(1-\mathrm{t})$为参数，对$\boldsymbol{v_{03}}$和$\boldsymbol{v_{12}}$进行二次插值，获得最终的$\boldsymbol{v_{0312}}$。

$$\boldsymbol{v_{0312}} = \mathrm{Lerp}\left(\boldsymbol{v_{03}}, \boldsymbol{v_{12}}; 2\mathrm{t}(1-\mathrm{t})\right)$$

注意最终的 Lerp 使用的参数是二次的 $2\mathrm{t}(1-\mathrm{t})$，不是通常情况下使用的 t，而且它插值的两个向量也变为了$\boldsymbol{v_{03}}$和 $\boldsymbol{v_{12}}$。

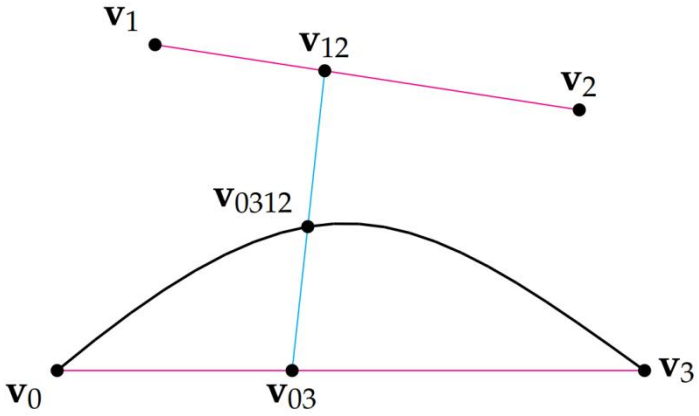


图 25.　$t = 0.4$ 的图像

同样，我们可以将 Quad 写为递归形式。

$$Quad(\boldsymbol{v_0}, \boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}; \mathrm{t}) = \mathrm{L}(\mathrm{L}(\boldsymbol{v_0}, \boldsymbol{v_3}; \mathrm{t}), L(\boldsymbol{v_1}, \boldsymbol{v_2}; \mathrm{t}); 2t(1-t))$$

可以看到，这样的插值要比三阶 Bézier 曲线简单很多，将七次 Lerp 减少到了三次。虽然最终的曲线与三次 Bézier 曲线不完全相同，但是已经很近似了。

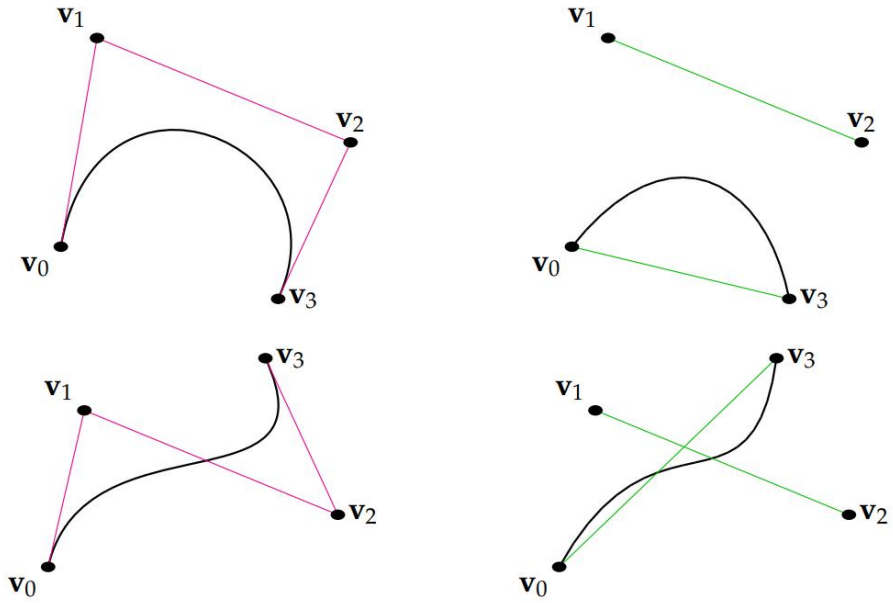


图 26.　左边是三阶 Bézier 曲线，右边是 Quad 曲线

如果利用 Lerp 的定义 $\mathrm{Lerp}(\boldsymbol{v_i}, \boldsymbol{v_{i+1}}; \mathrm{t}) = (1-t)\boldsymbol{v_i} + t\boldsymbol{v_{i+1}}$将递归式展开的话，我们能得到下面这样的式子。

$Quad(\boldsymbol{v_0}, \boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}; \mathrm{t}) = (2t^2 - 2\mathrm{t} + 1)(1 - \mathrm{t})\boldsymbol{v_0} + 2(1 - \mathrm{t})^2 t\boldsymbol{v_1} + 2(1 - t)t^2\boldsymbol{v_2} + t(2t^2 - 2t + 1)\boldsymbol{v_3}$。

它仍是一个三次的曲线，只不过系数有所不同。

如果我们将 Quad 的递归公式用于球面，就能得到四元数的 Squad。

1) Squad 一般形式

$$Squad(q_0, q_1, q_2, q_3; \mathrm{t}) = Slerp(Slerp(q_0, q_3; \mathrm{t}), Slerp(q_1, q_2; \mathrm{t}); 2t(1 - t))$$

我们知道$Slerp(q_i, q_{i+1}; \mathrm{t}) = (q_{i+1}q_i^*)^t q_i$所以我们可以将 Squad 写成指数形式。

2) Squad 指数形式

$$Squad(q_0, q_1, q_2, q_3; \mathrm{t}) = (Slerp(q_1, q_2; \mathrm{t})(Slerp(q_0, q_3; \mathrm{t}))^*)^{2t(1-t)} Slerp(q_0, q_3; \mathrm{t})$$

这个式子会对 Squad 的分析非常有用。

4. Squad 应用

接下来，我们回到最初的主题，对多个单位四元数进行插值。如果我们有一个四元数序列$q_0, q_1, \ldots, q_n$，我们希望对每一对四元数q_i和q_{i+1}都使用 Squad 进行插值，所以我们有。

$$Squad(q_i, S_i, S_{i+1}, q_{i+1}; \mathrm{t}) = Slerp(Slerp(q_i, q_{i+1}; \mathrm{t}), Slerp(S_i, S_{i+1}; \mathrm{t}); 2t(1 - t))$$

现在，留下来的问题就是找出中间的控制点S_i和S_{i+1}了。类似于 Bézier 曲线的样条，我们同样需要前一个四元数q_{i-1}以及q_{i+2}的信息。

S_i的推导还是比较复杂的，但是它最基本的理念非常简单。让 Squad 在切换点可导，从而达到C^1连续。也就是说，我们希望$q_{i-1}q_i$插值时在 t=1 处的导数，与q_iq_{i+1}插值时在 t=0 处的导数相等。

$$Squad'(q_{i-1}, S_{i-1}, S_i, q_i; 1) = Squad'(q_i, S_i, S_{i+1}, q_{i+1}; 0)$$

如果我们想要从这里继续推导下去的话，就需要用到单位四元数导数的定义（它和$q = [\cos\theta, \sin\theta\,\mathbf{u}] = e^{u\theta}$有关），但是因为这一块涉及到很多的证明，我并没有进行介绍。所以在这里我会省略控制点S_i的推导。

如果按照这个思路推导下去的话，最终能得到（或者它的等价形式），$S_i = q_i \exp\left(-\frac{\log(q_i^* q_{i-1}) + \log(q_i^* q_{i+1})}{4}\right)$。

注意，和 Bézier 曲线的样条不同的是，这里的S_i在对$q_{i-1}q_i$插值时和对q_iq_{i+1}插值时都是相同的，不像之前是处于切线的两端不同的两个向量。

与两个四元数之间的插值一样，Squad 同样会受到双倍覆盖的影响。我们在计算中间控制点和插值之前，需要先选中一个四元数，比如说q_i，检测它与其它三个四元数之间的夹角，如果是钝角就翻转，将插值的路线减到最小。

十一. 二重四元数

除了普通的四元数之外，几何代数中还衍生出来了一个二重四元数（Dual Quaternion）。它不仅能够表示 3D 旋转，还能够表示 3D 空间中的任何的刚体运动（Rigid Motion），即旋转、平移、反射和均匀缩放，所以有时候也被用于图形学中。

二重四元数同样可以表示为$q = a + bi + cj + dk$，但与普通的四元数不同的是，这里的系数 a, b, c, d 都不再是实数，它们在这里是二重数（Dual Number）。一个二重数 z 可以被定义为 $z = a + b\epsilon$（$\epsilon^2 = 0, \epsilon \neq 0$）。

所以，我们还可以将一个二重四元数写为 $q = r + d\epsilon\ (r, d \in H)$。其中，$r$ 是这个二重四元数的实部，d 是这个二重四元数的虚部 (Dual Part)。这样的定义会给我们带来更多的性质，让我们能够表示其它的线性变换。

十二. 左手坐标系统下的旋转

如果有注意到的话，前面的文章里大部分的图都没有画坐标系统，但这不代表坐标系统

对于旋转是没有关系的。我们在讲解三维空间中的旋转时定义了我们在使用右手坐标系统，并且我们使用的是右手定则来定义的旋转正方向。虽然这两个概念都是用右手来定义的，但这都只是数学内的一些惯例定义（Convention）。它们不仅没有直接的关系，而且我们也可以随意改变这个惯例。

然而，在这之后一切的结论都需要按照惯例的不同进行细微的调整。下面，我们就来尝试改动这个惯例，来看一看结论会有什么不同。

1. 右手坐标系 - 左手定则

我们首先来看一个比较简单的情况。如果我们仍然在使用右手坐标系统，但是使用左手定则来定义旋转的正方向，四元数的旋转公式会发生什么样的改变呢?

当我们用左手定则定义旋转的时候，旋转的方向会与右手定则定义的旋转完全相反。在之前，我们说「旋转 θ 角度」，我们指的是沿着旋转轴逆时针旋转 θ 度。而现在，我们则指的是沿着旋转轴顺时针旋转 θ 度。

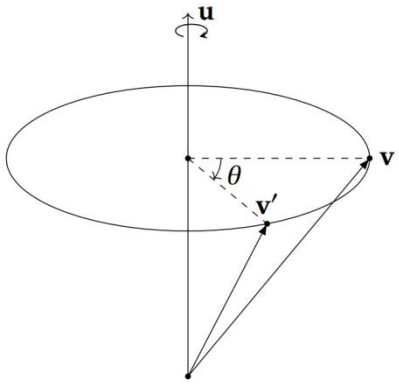


图 27. 使用左手定则定义旋转

这种情况相对还是比较容易的。我们知道，顺时针旋转 θ 度等于逆时针旋转 −θ 度。这也就是说，我们可以直接将「右手坐标系——右手定则」时的结论直接拿过来用。

我们知道，在我们用右手定则定义旋转的正方向时，四元数旋转的公式是这样进行定义的。如果我们有一个旋转四元数$q = [\cos(\frac{1}{2}\theta), \sin(\frac{1}{2}\theta)\,\mathbf{u}]$，那么将任意向量$v$沿着以单位向量定义的旋转轴 u 逆时针旋转 θ 度之后的v'可以通过公式$v' = qvq^* = qvq^{-1}$来获得。

如果我们现在想要将 v 逆时针旋转−θ度，我们需要的新的旋转四元数p则是，$p = \left[\cos\left(-\frac{1}{2}\theta\right), \sin\left(-\frac{1}{2}\theta\right)\mathbf{u}\right] = \left[\cos\left(\frac{1}{2}\theta\right), -\sin\left(\frac{1}{2}\theta\right)\mathbf{u}\right] = q^*$。（这里运用了正弦和余弦的诱导公式）

如果我们使用这个新的四元数 p 来对 v 进行旋转，$v' = pvp^*$我们就能得到将 v 顺时针旋转 θ 度的 v'了。然而，我们可以注意到 p 的共轭其实就是q ，$p^* = (q^*)^* = q$。这也就是说，顺时针旋转 θ 度的公式其实可以写成$v' = pvp^* = q^*vq$。其中，$q = \left[\cos\left(\frac{1}{2}\theta\right), \sin\left(\frac{1}{2}\theta\right)\mathbf{u}\right]$与之前保持不变，唯一改动只是 q 施加的顺序。现在我们是对 v 左乘 q^*，右乘 q，与之前正好相反。我们其实也可以通过四元数旋转的复合来得出同样的结论。

现在我们应该就清楚在「右手坐标系 —— 左手定则」的情况下该怎样对 v 进行旋转了。

1) 四元数旋转公式（右手坐标系，左手定则定义正方向）

任意向量 **v** 沿着以单位向量定义的旋转轴 **u** 顺时针旋转 θ 度之后的v'可以使用四元

数乘法来获得。令 $\mathrm{v} = [0, \mathbf{v}]$ $\mathrm{q} = \left[\cos\left(\frac{1}{2}\theta\right), \sin\left(\frac{1}{2}\theta\right)\mathbf{u}\right]$，那么$v' = q^{*}vq = q^{-1}vq$。

2. 左手坐标系 - 右手定则

接下来，我们来讨论一个相对比较复杂的情况。现在，我们将使用左手坐标系统，但是仍然使用右手定则来定义旋转的正方向。这就是说，当我们说「旋转 θ 角度」的时候，我们仍然需要的是沿着旋转轴逆时针旋转 θ 度。

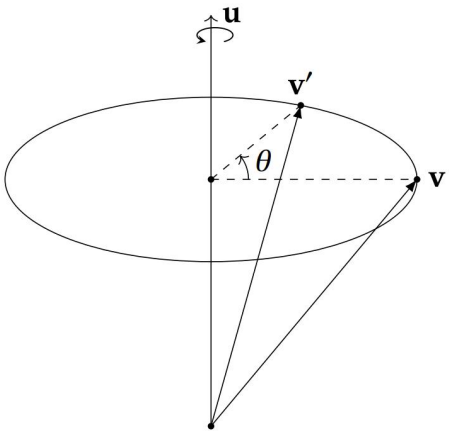


图 28. 沿着旋转轴逆时针旋转

虽然之前我们很早就说明了使用的是右手坐标系统，但是在前面所有的推导中，我们仅仅只在一个地方用到了坐标系的手性（Handedness），那就是在三维空间中的旋转-v1的旋转中构造 **w** 的时候使用的叉乘。

改变坐标系的手性会怎样改变 **w** 呢？我们首先要明白的一点是，坐标系手性的改变并不会对任何的数值运算造成影响，它只会影响我们如何想象（Visualize）向量之间的关系。

当我们在进行叉乘的时候，坐标系手性的改变并不会改变结果的数值。举一个最简单的例子，假设我们有两个向量$\mathbf{u} = (1,0,0)^T$和 $\mathbf{v} = (0,1,0)^T$，现在我们想要计算它们叉乘的结果 $\mathbf{w} = \mathbf{u} \times \mathbf{v}$。不论在哪个坐标系中，它的计算方式和结果都是一样的。

$$\mathbf{u} \times \mathbf{v} = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix} = 0\mathbf{i} + 0\mathbf{j} + 1\mathbf{k} = (0,1,0)^T$$

在左手坐标系和右手坐标系中，$\mathbf{u} \times \mathbf{v}$的结果都是 $\mathbf{w} = (0,1,0)^T$。然而，当我们把结果绘制到图中的时候，我们就会发现这个向量 $\mathbf{w} = \mathbf{u} \times \mathbf{v}$有非常重大的区别。

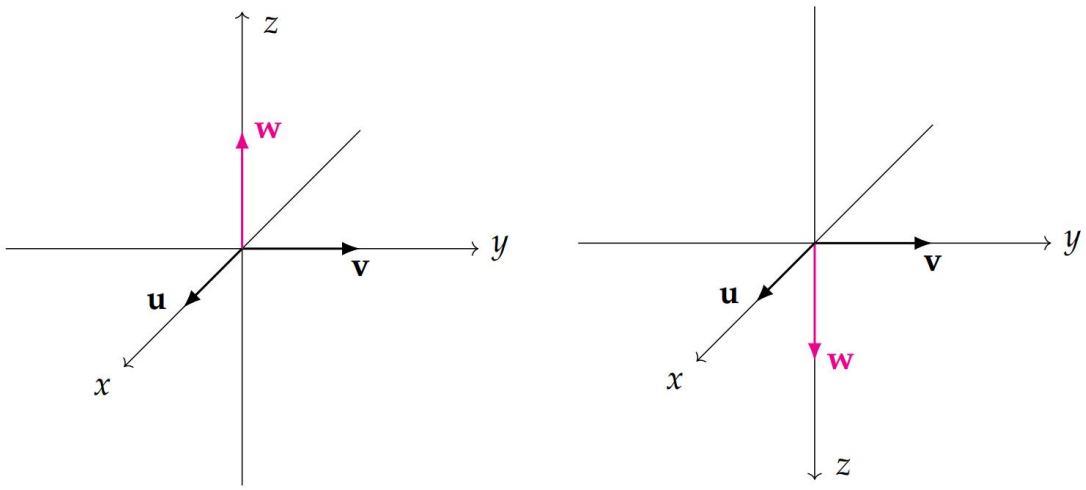


图 29.　左图是右手坐标系中的叉乘，右图是左手坐标系中的叉乘

我们应该能够注意到，虽然 **w** 的坐标都是(0,0,1)，但是它与向量 **u**和 **v** 视觉上的关系在右手坐标系和左手坐标系中完全不同。在右手坐标系中，**w** 的方向可以通过右手定则来决定，而在左手坐标系中，**w** 的方向则需要通过左手定则来决定。

知道了这个结果之后，我们应该就差不多能够预见改用左手坐标系的后果是什么了。

现在，我们需要检查一下改用左手坐标系之后三维空间中的旋转-$v_\perp$的旋转推导的结果。因为我们仍然使用右手定则定义旋转，所以 3D 的旋转示意图是完全没有改变的。

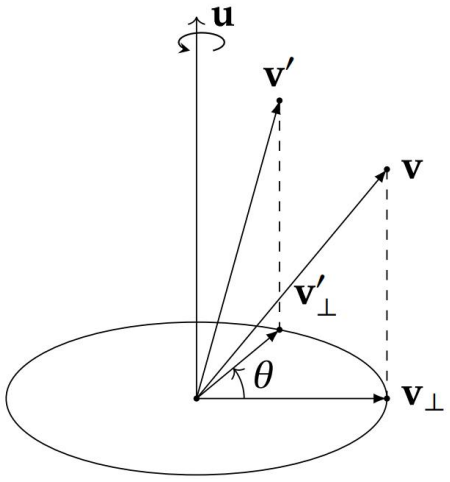


图 30.　3D 旋转示意图未发生改变

之后，我们将所有的向量投影到了发生旋转的 2D 平面上，并且利用叉乘构造了一个向量$\mathbf{w} = \mathbf{u} \times \mathbf{v}_\perp$。坐标系的手性就是在这里造成了差异。我们知道，在右手坐标系中，**w** 指向 $\mathbf{v}_\perp$ 逆时针旋转$\frac{\pi}{2}$后的方向。然而在左手坐标系中，**w** 则指向$\mathbf{v}_\perp$ 顺时针旋转$\frac{\pi}{2}$后的方向。

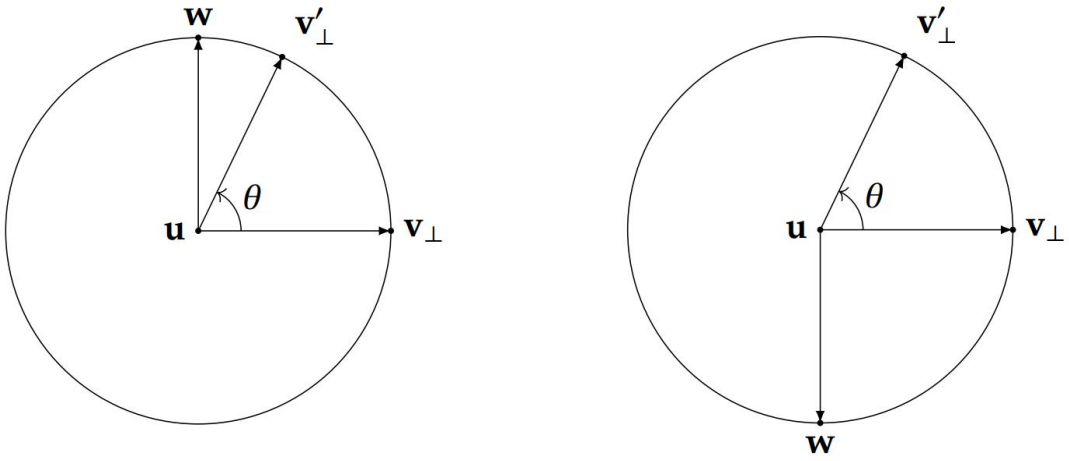


图 31.　左图为右手坐标系中的情况，右图为左手坐标系中的情况

如果我们仍然按照这个公式$v'_{\perp} = \cos\theta * v_{\perp} + \sin\theta\, w = \cos\theta * v_{\perp} + \sin\theta\,(u \times v_{\perp})$。来计算旋转的结果$v'_{\perp}$的话，我们得到的会是$v_{\perp}$顺时针旋转 θ 度，或者说逆时针旋转 −θ 度之后的结果，我们将这个实际计算的结果叫做$\tilde{v}'_{\perp}$。

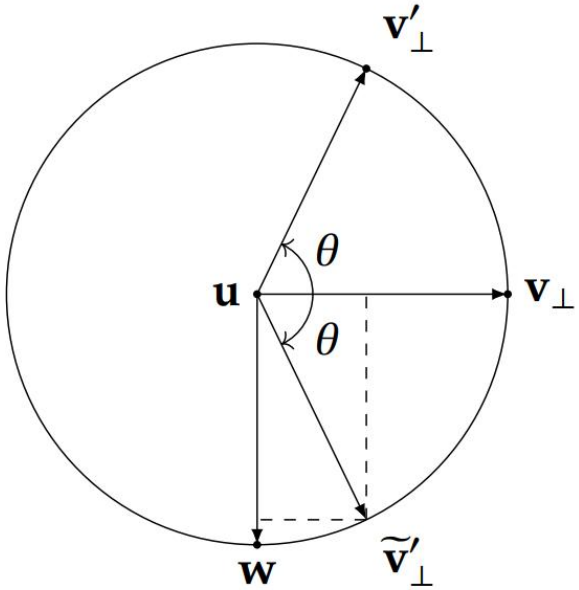


图 32.　实际情况的图示

因为我们一直是按照上面的公式推导出的四元数形式的旋转，所以说，在左手坐标系中，如果我们有一个旋转四元数 $q = \left[\cos\left(\frac{1}{2}\theta\right), \sin\left(\frac{1}{2}\theta\right) u\right]$，一个向量 v 和一个单位向量定义的旋转轴 u，那么这个公式会将 v 沿着 u 逆时针旋转 −θ 度，即$v' = qvq^{*} = qvq^{-1}$。

如果我们需要将 v 沿着 u 逆时针旋转 θ 度，参照上一节的结果，我们需要交换 q 与 q^{*} 相乘的位置。这也就是说，如果使用这个公式$v' = q^{*}vq = q^{-1}vq$，我们就能获得真正想要的结果，将 v 沿着 u 逆时针旋转 θ 度的v'。这也就解答了左手坐标系中的四元数的旋转。

1) 四元数旋转公式（左手坐标系，右手定则定义正方向）

任意向量 v 沿着以单位向量定义的旋转轴 u 逆时针旋转 θ 度之后的v'可以使用四元数乘法来获得。令 $v = [0, v]$ $q = \left[\cos\left(\frac{1}{2}\theta\right), \sin\left(\frac{1}{2}\theta\right) u\right]$，那么$v' = q^{*}vq = q^{-1}vq$。

2) 四元数旋转公式（左手坐标系，左手定则定义正方向）

任意向量 $\mathbf{v}$ 沿着以单位向量定义的旋转轴 $\mathbf{u}$ 顺时针旋转 θ 度之后的$\boldsymbol{v}'$可以使用四元数乘法来获得。令 $\mathrm{v} = [0, \mathbf{v}]$ $\mathrm{q} = \left[\cos\left(\frac{1}{2}\theta\right), \sin\left(\frac{1}{2}\theta\right)\mathbf{u}\right]$，那么$v' = qvq^* = qvq^{-1}$。

可以看到，如果在左手坐标系用左手定则定义旋转正方向，和在右手坐标系中用右手定则定义旋转正方向的公式是完全一样的。

十三. 坐标系转换

有一点需要注意的是，实际应用中的坐标转换其实涉及到的不止是左右手坐标系的问题。我们有可能需要将同一个变换在不同的坐标系中表示出来，比如说下面这种情况。

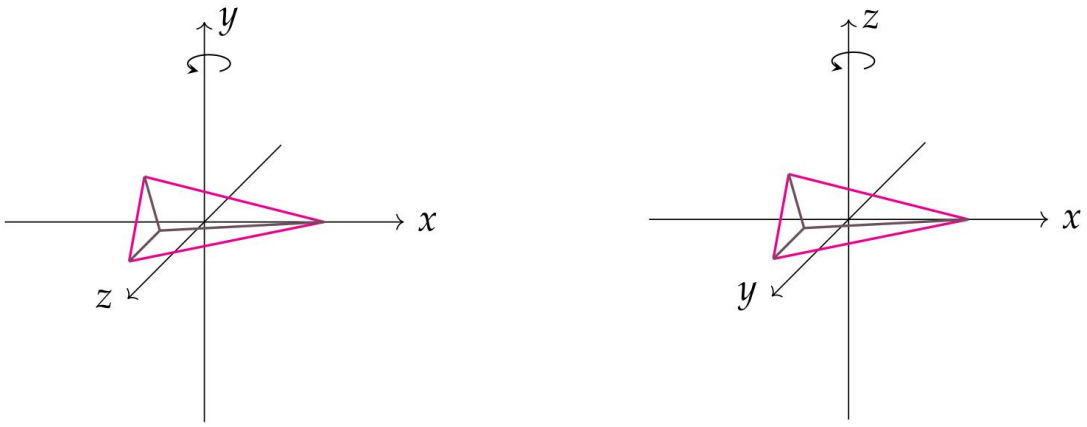


图 33. 左图是坐标系 A 下的旋转，右图是坐标系 B 下的同一旋转

我们应该能注意到，虽然两个图中的旋转看起来一模一样，但是这两个坐标系不仅手性改变了，坐标轴的位置也被调换了。坐标系 A 下的 (x, y, z) 轴分别对应着坐标系 B 下的 (x, z, y) 轴。

如果我们在坐标系 A（右手坐标系）中使用右手定则定义旋转正方向，在坐标系 B（左手坐标系）中使用左手定则定义正方向（也就是都用$v' = qvq^* = qvq^{-1}$这个个公式来进行旋转），那么在坐标系 A 中就是沿着 y 轴$(0,1,0)^T$ 向正方向旋转 θ 度，而在坐标系 B 中则是沿着 z 轴$(0,1,0)^T$向正方向旋转 $-\theta$ 度。

旋转角度需要取反这一点其实可以直接从坐标系手性的改变看出来，除此之外我们还需要将旋转轴从坐标系 A 转换到坐标系 B 中。我们假设这个旋转轴在坐标系 A 中是一个单位向量$\boldsymbol{u}_A$，我们希望找出一个变换 T 将 $\boldsymbol{u}_A$ 转化到坐标系 B 中，我们将坐标系 B 中的旋转轴称之为 $\boldsymbol{u}_B$. 这也就是说我们需要 $\mathrm{T}\boldsymbol{u}_A = \boldsymbol{u}_B$。

接下来我们就需要找出这个变换 T。因为坐标系的转换只有两种情况，一种是旋转（Rotation，不改变手性），另一种是反射（Reflection，改变手性），所以 T 一定是一个线性变换。我们只需要知道三个基向量在 T 下做出的变换就能够得到关于这个变换 T 的所有信息。

那么，这个 T 对于坐标系 A 下的基向量做出了什么变换呢？我们可以选择使用坐标系 A 下的标准基：x 轴 $(1,0,0)^T$,、y 轴 $(0,1,0)^T$ 和 z 轴 $(0,0,1)^T$来解答这个问题。

首先，我们来考虑一下 x 轴$(1,0,0)^T$所做出的变换。假设我们在坐标系 A中有一个以 x 轴为旋转轴的旋转变换，如果我们想让这个旋转在坐标系 B 中看起来完全一样，就需要在坐标系 B 中同样沿着 x 轴 $(1,0,0)^T$ 进行旋转，这也就是说 $\mathrm{T}\begin{bmatrix}1\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\end{bmatrix}$。

接下来是 y 轴 $(0,1,0)^T$。因为坐标系 A 中的 y 轴看起来与坐标系 B 中的 z轴 $(0,0,1)^T$指向的是同一个方向，所以我们能得到 $\mathrm{T}\begin{bmatrix}0\\1\\0\end{bmatrix}=\begin{bmatrix}0\\0\\1\end{bmatrix}$。

同理，我们知道坐标系 A 中的 z 轴被变换到了坐标系 B 中的 y 轴，所以 $\mathrm{T}\begin{bmatrix}0\\0\\1\end{bmatrix}=\begin{bmatrix}0\\1\\0\end{bmatrix}$。

我们将这三个等式用矩阵结合起来可以得到 $\mathrm{T}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}=\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}$。

将等式两边同时右乘以$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$的逆矩阵，$\mathrm{T}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}^{-1}=$ $\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}^{-1}$，$\mathrm{T}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}=\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$。

即 $\mathrm{T}=\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}$。

现在，如果我们在坐标系中有一个旋转轴 $\boldsymbol{u}_A$，想要获得其在坐标系 B 中对应的旋转轴$\boldsymbol{u}_B$，就可以通过以下的计算来获得。

$$\boldsymbol{u}_B=T\boldsymbol{u}_A=\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}\boldsymbol{u}_A。$$

这个变换 T 的矩阵里其实还隐藏了一个很重要的信息，我们可以通过计算它的行列式（Determinant）来确定坐标系的手性有没有被翻转。如果坐标系的手性被翻转了，那么 T 对应的是一个反射，这时候它的行列式的值等于−1。我们可以确认一下上面求出来的这个 T 改变了手性，$\det T=\begin{vmatrix}1&0&0\\0&0&1\\0&1&0\end{vmatrix}=-1$ 。

如果坐标系的手性没有被翻转，那么 T 对应的是一个旋转，这时候它行列式的值等于1。比如我们想将坐标系 A 中的旋转轴转化到一个新的坐标系 C 中来。

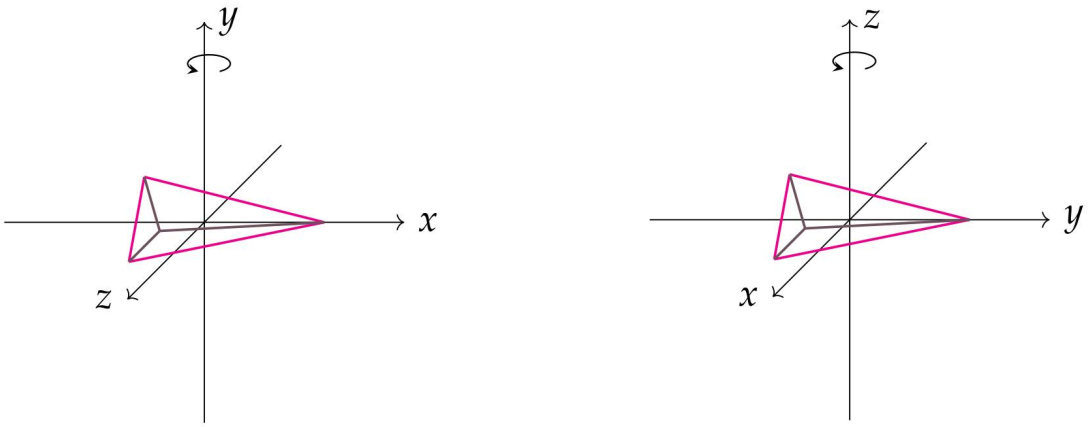


图 34.　左图是坐标系 A 下的旋转，右图是坐标系 C 下的旋转

坐标系 A 和坐标系 C 都是右手坐标系，我们可以按照上面的方法计算出所需的变换矩阵T'，即$T' = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$。

我们可以计算它的行列式验证这个坐标系变换没有改变坐标系的手性，$\det T = \begin{vmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{vmatrix} = 1$ 。

十四. 常用空间数据结构——Spatial[1] Data Structures

1. 层次包围盒——BVH(Bounding Volume Hierarchies)

2. BSP 树——BSP Trees

3. 八叉树——Octrees

4. 场景图——Scene Graphs

BVH、BSP 树和八叉树都是使用某种形式的树来作为基本的数据结构，它们的具体区别在于各自是如何进行空间分割和几何体存储，且他们均是以层次的形式来保存几何物体。

然而，渲染三维场景不仅仅只是渲染出几何图形，对动画，可见性，以及其他元素的控制，往往需要通过场景图（Scene Graphs）来完成。

场景图被誉为“当今最优秀且最为可重用的数据结构之一。”Wiki 中的对场景图的定义是“场景图（Scene Graph）是组织和管理三维虚拟场景的一种数据结构，是一个有向无环图（Directed Acyclic Graph， DAG）。”

场景图是一个面向用户的树结构，可以通过纹理、变换、细节层次、渲染状态（例如材质属性）、光源以及其他任何合适的内容进行扩充。它由一棵以深度优先遍历来渲染整个场景的树来表示。

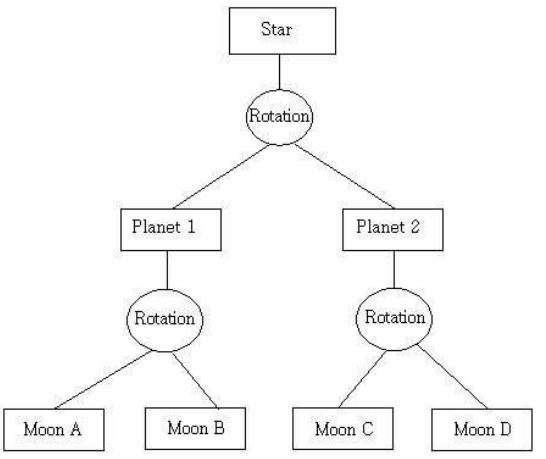


图 35. 通过创建场景图来表示对象

另外提一句，开源的场景图有 Open Scene Graph 和 OpenSG 等，有兴趣的朋友们可以进行进一步了解。

5. 游戏场景管理中 BVH 相比八叉树有什么优劣

BVH（层次包围盒）的构建比较耗时间，而且对于动态物体的支持比较麻烦，但是运行效率会更高，因此常用于离线渲染和光线追踪等对性能要求较高的场合。

[1] adj.空间的，与空间有关的；空间理解能力的

八叉树的构建简洁明了，比较直观。八叉树的结构是固定的，不会受物体位置的影响，这样对动态物体非常友好。游戏中用的八叉树常常是松散八叉树，就是子节点的范围会比正常的范围略微大一圈，这样如果物体和分割轴相交，也可以放到子节点下。游戏中物体的大小通常是不变的，物体的移动的处理就很简单，只需要改变物体挂载的子节点位置就行了，而不要调整挂载子节点的深度。游戏中常常有非常多的动态对象，使用松散八叉树是非常合适的。

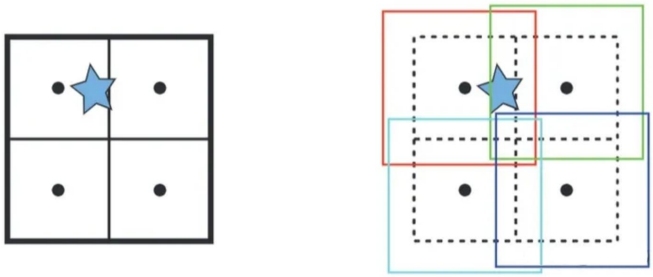


图 36.　四叉树和松散四叉树

BVH 当然也能实现动态的更新，但是会比较麻烦。

游戏中通常只会使用场景管理来做视锥剔除，不会用来做光线追踪等功能，不需要特别高的运行速度，因此八叉树就足够了。

十五.　抗锯齿与常见抗锯齿类型总结

抗锯齿（英语：anti-aliasing，简称 AA），也译为边缘柔化、消除混叠、抗图像折叠有损，反走样等。它是一种消除显示器输出的画面中图物边缘出现凹凸锯齿的技术，那些凹凸的锯齿通常因为高分辨率的信号以低分辨率表示或无法准确运算出 3D 图形坐标定位时所导致的图形混叠（aliasing）而产生的，抗锯齿技术能有效地解决这些问题。

1. SSAA——超级采样抗锯齿

超级采样抗锯齿（Super-Sampling Anti-Aliasing，简称 SSAA）是比较早期的抗锯齿方法，比较消耗资源，但简单直接。这种抗锯齿方法先把图像映射到缓存并把它放大，再用超级采样把放大后的图像像素进行采样，一般选取 2 个或 4 个邻近像素，把这些采样混合起来后，生成的最终像素，令每个像素拥有邻近像素的特征，像素与像素之间的过渡色彩，就变得近似，令图形的边缘色彩过渡趋于平滑。再把最终像素还原回原来大小的图像，并保存到帧缓存也就是显存中，替代原图像存储起来，最后输出到显示器，显示出一帧画面。这样就等于把一幅模糊的大图，通过细腻化后再缩小成清晰的小图。如果每帧都进行抗锯齿处理，游戏或视频中的所有画面就都带有抗锯齿效果。

超级采样抗锯齿中使用的采样法一般有两种：

① OGSS，顺序栅格超级采样（Ordered Grid Super-Sampling，简称 OGSS），采样时选取 2 个邻近像素。

② RGSS，旋转栅格超级采样（Rotated Grid Super-Sampling，简称 RGSS），采样时选取 4 个邻近像素。

另外，作为概念上最简单的一种超采样方法，全场景抗锯齿（Full-Scene Antialiasing，FSAA）以较高的分辨率对场景进行绘制，然后对相邻的采样样本进行平均，从而生成一幅新的图像。

2. MSAA——多重采样抗锯齿

多重采样抗锯齿（Multi Sampling Anti-Aliasing，简称 MSAA），是一种特殊的超级采样抗锯齿（SSAA）。MSAA 首先来自于 OpenGL。具体是 MSAA 只对 Z 缓存（Z-Buffer）和模板缓存(Stencil Buffer)中的数据进行超级采样抗锯齿的处理。可以简单理解为只对多边形的

边缘进行抗锯齿处理。这样的话，相比 SSAA 对画面中所有数据进行处理，MSAA 对资源的消耗需求大大减弱，不过在画质上可能稍有不如 SSAA。

3. CSAA——覆盖采样抗锯齿

覆盖采样抗锯齿（Coverage Sampling Anti-Aliasing，简称 CSAA）是 NVIDIA 在 G80 及其衍生产品首次推向实用化的 AA 技术。CSAA 就是在 MSAA 基础上更进一步的节省显存使用量及带宽，简单说 CSAA 就是将边缘多边形里需要采样的子像素坐标覆盖掉，把原像素坐标强制安置在硬件和驱动程序预先算好的坐标中。这就好比采样标准统一的 MSAA，能够最高效率的执行边缘取样，效能提升非常的显著。比方说 16xCSAA 取样性能下降幅度仅比 4xMSAA 略高一点，处理效果却几乎和 8xMSAA 一样。8xCSAA 有着 4xMSAA 的处理效果，性能消耗却和 2xMSAA 相同。

4. HRAA——高分辨率抗锯齿

高分辨率抗锯齿方法(High Resolution Anti-Aliasing，简称 HRAA)，也称 Quincunx 方法，也出自 NVIDIA 公司。“Quincunx”意思是 5 个物体的排列方式，其中 4 个在正方形角上，第五个在正方形中心，也就是梅花形，很像六边模型上的五点图案模式。此方法中，采样模式是五点梅花状，其中四个样本在像素单元的角上，最后一个在中心。

5. CFAA——可编程过滤抗锯齿

可编程过滤抗锯齿（Custom Filter Anti-Aliasing，简称 CFAA）技术起源于 AMD-ATI 的 R600 家庭。简单地说 CFAA 就是扩大采样面积的 MSAA，比方说之前的 MSAA 是严格选取物体边缘像素进行缩放的，而 CFAA 则可以通过驱动灵活地选择对影响锯齿效果较大的像素进行缩放，以较少的性能牺牲换取平滑效果。显卡资源占用也比较小。

6. MLAA——形态抗锯齿

形态抗锯齿（Morphological Anti-Aliasing，简称 MLAA），是 AMD 推出的完全基于 CPU 处理的抗锯齿解决方案。与 MSAA 不同， MLAA 将跨越边缘像素的前景和背景色进行混合，用第 2 种颜色来填充该像素，从而更有效地改进图像边缘的变现效果。

7. FXAA——快速近似抗锯齿

快速近似抗锯齿(Fast Approximate Anti-Aliasing，简称 FXAA) ，是传统 MSAA(多重采样抗锯齿)效果的一种高性能近似。它是一种单程像素着色器，和 MLAA 一样运行于目标游戏渲染管线的后期处理阶段，但不像后者那样使用 DirectCompute，而只是单纯的后期处理着色器，不依赖于任何 GPU 计算 API。正因为如此，FXAA 技术对显卡没有特殊要求，完全兼容 NVIDIA、AMD 的不同显卡(MLAA 仅支持 A 卡)和 DirectX 9.0、DirectX 10、DirectX 11。

8. TXAA——时间性抗锯齿

时间性抗锯齿（Temporal Anti-Aliasing，简称 TXAA），将 MSAA、时间滤波以及后期处理相结合，用于呈现更高的视觉保真度。与 CG 电影中所采用的技术类似，TXAA 集 MSAA 的强大功能与复杂的解析滤镜于一身，可呈现出更加平滑的图像效果。此外，TXAA 还能够对帧之间的整个场景进行抖动采样，以减少闪烁情形，闪烁情形在技术上又称作时间性锯齿。目前，TXAA 有两种模式：TXAA 2X 和 TXAA 4X。TXAA 2X 可提供堪比 8X MSAA 的视觉保真度，然而所需性能却与 2X MSAA 相类似；TXAA 4X 的图像保真度胜过 8XMSAA，所需性能仅仅与 4X MSAA 相当。

9. MFAA——多帧采样抗锯齿

多帧采样抗锯齿（Multi-Frame Sampled Anti-Aliasing，MFAA）是 NVIDIA 公司根据 MSAA 改进出的一种抗锯齿技术。目前仅搭载 Maxwell 架构 GPU 的显卡才能使用。可以

将 MFAA 理解为 MSAA 的优化版，能够在得到几乎相同效果的同时提升性能上的表现。MFAA 与 MSAA 最大的差别就在于在同样开启 4 倍效果的时候 MSAA 是真正的针对每个边缘像素周围的 4 个像素进行采样，MFAA 则是仅仅只是采用交错的方式采样边缘某个像素周围的两个像素。