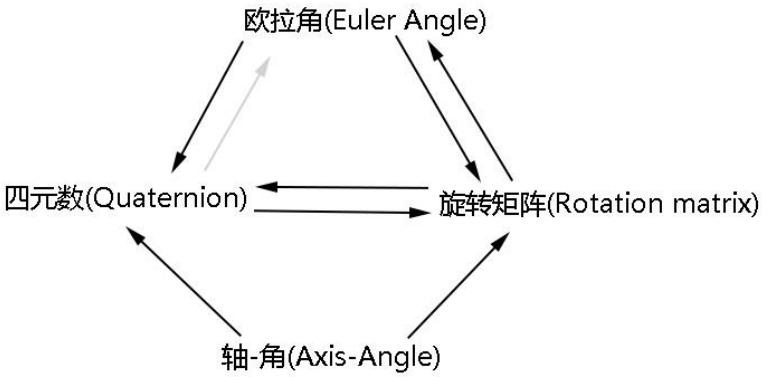
# Conversions Between 3D Rotation Representations

Author：练孙鸿(sunhonglian)

## 0 引言

因为最近要给 Noise3D 集成一个 RigidTransform 之类的 class，动机是做 SH Lighting Preview Utility 的时候想要用鼠标旋转球体，结果发现如果只能控制 Euler Angle 的话实现 delta rotation 很麻烦，干脆就把四元数也加进来然后实现一个 RigidTransform 的类，把几种旋转表示的转换都封装一下。这里考虑了几种旋转表示方法：Euler Angle，Quaternion，Rotation Matrix，(还有个擦边的 Lookat 转 Euler Angle)。网上很多 blog 的旋转表示转换方法都是基于 XYZ 的欧拉角顺规的，这就很不清真了，因为 D3D 是 y 轴向上的左手系而且顺规也不一样，那么很多公式就不能直接 copy 了。一篇 NASA 1977 的 Technical Report[1] 给出了 12 种欧拉角顺规的 euler-to-matrix conversion，以及与 quaternion 的转换等，可以参考一下（但是没给推导）。所以我觉得可以在这基础上看一手，并且记下一点与 D3D 相关的旋转转换公式。




图：本文介绍的转换

# 1 欧拉角(Euler Angle)与旋转矩阵(Rotation Matrix)

## 1.1 Euler Angles To Rotation Matrix

D3D 和 OpenGL 不同，用的坐标系是 Y 轴竖直向上的左手系，所以欧拉角 Yaw-Pitch-Roll 的顺规是跟广大 blog、OpenGL 不一样的，那么博客上、甚至维基百科[2]上的各种基于右手系 xyz 顺规(分别对应 roll, pitch,yaw)的看起来就不太能随随便便直接用了。虽然有一部分转换是 DirectXMath 支持的，例如 XMQuaternionRotation()，XMMatrixRotationQuaternion()等。但是为了不要用得稀里糊涂，以及实现一些接口不提供的东西，我还是决定自己实现几种 Rotation Representation 的 conversion 吧...

首先欧拉角旋转序列(Euler Angle Rotational Sequence)一共有 12 种顺规，6 种绕三条轴的旋转(XYZ,XZY,YXZ,YZX,ZXY,ZYX)，另外 6 种绕两条轴的旋转(XYX,YXY,XZX,ZXZ,YZY,ZYZ)。如果相邻两次旋转是绕同一条轴，例如 XXY，那么其实可以坍缩成 XY。那么只绕一条轴旋转就根本不够自由度就不需要说了。所以一共是 12 种旋转顺规（可以表示所有旋转的集合），其中基于 D3D 的 Noise3D 采用的是 Z(roll)X(pitch)Y(Yaw)顺规，写成列向量矩阵时应该是：(与维基百科 Euler Angles[3]给出的结果一致)

$$
\begin{aligned}
\boldsymbol{R}(\alpha,\beta,\gamma) &= \boldsymbol{R}_y(\alpha)\boldsymbol{R}_x(\beta)\boldsymbol{R}_z(\gamma) \\
&= \begin{bmatrix} cos\alpha & 0 & -sin\alpha \\ 0 & 1 & 0 \\ sin\alpha & 0 & cos\alpha \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\beta & sin\beta \\ 0 & -sin\beta & cos\beta \end{bmatrix}\begin{bmatrix} cos\gamma & -sin\gamma & 0 \\ sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_1 & 0 & -s_1 \\ 0 & 1 & 0 \\ s_1 & 0 & c_1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & c_2 & -s_2 \\ 0 & s_2 & c_2 \end{bmatrix}\begin{bmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_1 & s_1s_2 & s_1c_2 \\ 0 & c_2 & -s_2 \\ -s_1 & c_1s_2 & c_1c_2 \end{bmatrix}\begin{bmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 & c_2s_1 \\ c_2s_3 & c_2c_3 & -s_2 \\ c_1s_2s_3 - s_1c_3 & s_1s_3 + c_1c_3s_2 & c_1c_2 \end{bmatrix}
\end{aligned}
$$

其中α是绕 y 轴旋转的 yaw，β是绕 x 轴旋转的 pitch，γ是绕 z 旋转的 roll，即：

$$
\begin{aligned}
c_1 &= cos\,\alpha = cos(Y_{yaw}), s_1 = sin\,\alpha = sin(Y_{yaw}) \\
c_2 &= cos\beta = cos(X_{pitch}), s_2 = sin\beta = sin(X_{pitch}) \\
c_3 &= cos\gamma = cos(Z_{roll}), s_3 = sin\gamma = sin(Z_{roll})
\end{aligned}
$$

注意两种不同的旋转顺规想要把同一物体旋转到同一姿态，那么他们的欧拉角 Yaw-Pitch-Roll 角度值

是不一样的。

从欧拉角构建旋转矩阵就很简单啦，按照顺规把三个 Elemental Rotation Matrix 连乘起来就好了。但是由于(2018.9.18)现在要实现用鼠标拖拽一个 sphere 进行 delta Rotation 的功能，就是在当前姿态上再加上一个旋转，然后还要 update 其他的旋转表示。直接给 Euler Angle 加上一个 delta 值是不能得到正确的结果的，一个反例就是，如果物体的正面面对着用户时，鼠标向上拖，物体的 pitch 增加就好了。但是如果物体的背面面对着用户，鼠标向上拖，pitch 应该是要减少的。总之得用其他的旋转表示来做 delta Rotation 比较合适，欧拉角不适合做 delta rotation。那么四元数或者旋转矩阵其实也是可以的，这两种形式可以很方便地加上一个 delta rotation，而且 delta rotation 也都可以很方便地由 Axis-Angle 的方法构造出来。给旋转矩阵乘多一个 delta rotation matrix 然后再提取出欧拉角，就可以实现各个 rotation representation 的同步。所以这一节接下来再讨论一下从旋转矩阵提取欧拉角的方法。

## 1.2 Euler Angles From Rotation Matrix

参考 NASA technical Report[1]的 Appedix-A6 和[5]，我们可以用旋转矩阵元素的相乘、相除、反三角函数等操作去提取出欧拉角。[5]给出了从 XYZ 顺规提取欧拉角的方法、步骤、思路，[1]则给出了全部 12 种顺规的欧拉角提取公式，但是没有给一些细节注意事项。所以总结一下，根据[1]、[5]、[7]《Real Time Rendering 3rd Edition》4.2.2 和自己的推导，Z(roll)X(pitch)Y(Yaw)顺规的欧拉角提取公式（[1]原文下标似乎有点小问题）：

- Y axis yaw angle:

$$\alpha = atan2(sin\alpha cos\beta, cos\alpha cos\beta) = atan2(m_{13}, m_{33})$$

- X axis pitch angle:

$$\beta = arcsin(sin\beta) = arcsin(-m_{23})$$

- Z axis roll angle:

$$\gamma = atan2(cos\beta sin\gamma, cos\beta cos\gamma) = atan2(m_{21}, m_{22})$$

要注意一个很关键的问题，注意到矩阵的每一个元素都是 pitch angle β的函数...所以当$\mathrm{m}_{23} = -sin\beta =$

1，$\cos\beta = 0$，这时候其他的欧拉角 Extraction formula 的表达式就无意义了（因为分子分母都是 0，atan2 都没用）....在 ZXY 顺规下，pitch angle $\beta = \pm\frac{\text{pi}}{2}$恰好就是 Gimbal Lock 的位置。在 Gimbal Lock 的时候，旋转矩阵会退化为：

$$\boldsymbol{R}(\alpha,\beta,\gamma) = \begin{bmatrix} c_1c_3 \pm s_1s_3 & \pm c_3s_1 - c_1s_3 & 0 \\ 0 & 0 & \pm 1 \\ \pm c_1s_3 - s_1c_3 & s_1s_3 \pm c_1c_3 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\alpha \pm \gamma) & \sin(\alpha \pm \gamma) & 0 \\ 0 & 0 & \pm 1 \\ -sin(\alpha \pm \gamma) & -\cos(\alpha \pm \gamma) & 0 \end{bmatrix}$$

所以在$cos\beta = 0$的 gimbal lock corner case 时，甚至连$atan2(m_{13}, m_{33})$函数都无济于事了，因为$atan2(0,0)$也是无意义的。这个时候，我们不按《Real Time Rendering》[7]的比较粗暴的解决方法来，而按[5]的思路和自己的推算，gimbal lock corner case 应该分两种情况处理：

- $\beta = -\frac{\pi}{2}$时，$sin\beta = -1$，$cos\beta = 0$

$$\boldsymbol{R}_{gl}(\alpha,\beta,\gamma) = \begin{bmatrix} c_1c_3 - s_1s_3 & -c_3s_1 - c_1s_3 & 0 \\ 0 & 0 & 1 \\ -c_1s_3 - s_1c_3 & s_1s_3 - c_1c_3 & 0 \end{bmatrix}$$

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

$$\Rightarrow \alpha + \gamma = atan2(-m_{12}, m_{11})$$

$$\Rightarrow \alpha = -\gamma + atan2(-m_{12}, m_{11})$$

其中α和γ可以先随便固定一个，计算出另外一个。

- $\beta = \frac{\pi}{2}$时，$sin\beta = 1$，$cos\beta = 0$

$$\boldsymbol{R}_{gl}(\alpha,\beta,\gamma) = \begin{bmatrix} c_1c_3 + s_1s_3 & c_3s_1 - c_1s_3 & 0 \\ 0 & 0 & -1 \\ c_1s_3 - s_1c_3 & s_1s_3 + c_1c_3 & 0 \end{bmatrix}$$

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

$$\Rightarrow \alpha - \gamma = atan2(m_{12}, m_{11})$$

$$\Rightarrow \alpha = \gamma + atan2(m_{12}, m_{11})$$

其中α和γ可以先随便固定一个，计算出另外一个。

最后要注意的是，从旋转矩阵提取欧拉角的公式跟欧拉角顺规的选取有关，因为旋转矩阵的元素会略有不同，但是思路都是一样的，就是根据旋转矩阵的解析表达式+反三角函数凑出来23333。

# 2 四元数(Quaternion)与旋转矩阵

## 2.1 Quaternion to Rotation Matrix

众所周知的是，欧拉角和旋转矩阵表示法是有万向锁(Gimbal Lock)的问题的。幸好我们有四元数这种东西。

我们可以用把一个旋转写成四元数$\boldsymbol{q}(x,y,z,w) = w + x\boldsymbol{i} + y\boldsymbol{j} + z\boldsymbol{k}$的形式。如果我们设ω是旋转角，单位旋转轴为$\boldsymbol{u} = (u_x, u_y, u_z)$，由轴角Axis-Angle方式构造四元数方式如下：

$$x = u_x \, sin\left(\frac{\omega}{2}\right)$$

$$y = u_y \, sin\left(\frac{\omega}{2}\right)$$

$$z = u_z \sin\left(\frac{\omega}{2}\right)$$

$$w = cos\left(\frac{\omega}{2}\right)$$

注意：

$$\|\boldsymbol{q}\| = \mathrm{x}^2 + y^2 + z^2 + w^2 = 1$$

这里按右手定则来定义旋转方向( 笔者疑问 :左右手系和左右手法则有没有什么影响呢？感觉不太有，毕竟这里的旋转都是代数定义，与手性无关），那么给定一个四元数$\boldsymbol{q}(x,y,z,w)$，可以构造旋转矩阵[1][4][8][14]（参考Wikipedia -Rotation Matrix: Conversion- Quaternion）:

$$\boldsymbol{R}(\boldsymbol{q}) = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2zw & 2xz+2yw \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 \end{bmatrix}$$

用四元数的分量直接构造旋转矩阵是非常高效的，因为只有乘和加减法，而不需要三角函数。

## 2.2 Quaternion From Rotation Matrix

然后从旋转矩阵提取四元数，也是可以像提取欧拉角那样，用正向构造的矩阵表达式凑出来。参考[8]《RTR》Chap4的思路，我们观察一下$\boldsymbol{R}(\boldsymbol{q})$的元素，可以发现：

$$m_{32} - m_{23} = (2yz + 2xw) - (2yz - 2xw) = 4xw$$
$$\mathrm{m}_{13} - m_{31} = (2xz + 2yw) - (2xz - 2yw) = 4yw$$
$$\mathrm{m}_{21} - m_{12} = (2xy + 2zw) - (2xy - 2zw) = 4zw$$

意思是我们只需要再凑出个$w$，那么四元数的四个分量都可以求出来了。于是我们又发现了一个恒等式：

$$\begin{aligned} tr\big(\boldsymbol{R}(\boldsymbol{q})\big) &= m_{11} + m_{22} + m_{33} = 3 - 4(x^2 + y^2 + z^2) \\ &= 4(1 - (x^2 + y^2 + z^2)) - 1 \\ &= 4w^2 - 1 \end{aligned}$$

其中$tr()$是矩阵的迹(trace)。于是四元数$\boldsymbol{q}(x, y, z, w)$就可以从旋转矩阵里面提取出来了：

$$w = \frac{\sqrt{(tr(\boldsymbol{R}) + 1}}{2}$$
$$\mathrm{x} = \frac{\mathrm{m}_{32} - m_{23}}{4w}$$
$$\mathrm{y} = \frac{m_{13} - m_{31}}{4w}$$
$$\mathrm{z} = \frac{m_{21} - m_{12}}{4w}$$

考虑到w在很小的时候可能会有数值不稳定的情况，《RTR》Chap4给出了一个数值相对稳定的解法，具体还是看文献吧2333。

# 3 欧拉角与四元数

## 3.1 Euler Angles to Quaternion

思路可以参考[2]。首先两个四元数相乘长这个样子（注意中文维基[12]的四元数乘积表达式是有点符号问题的，[13]的blog看起来比较对，但反正我都自己推了一下，[13]对一点）：

$$\boldsymbol{p} = w_1 + \boldsymbol{v_1} = w_1 + x_1\boldsymbol{i} + x_2\boldsymbol{j} + x_3\boldsymbol{k}$$
$$\boldsymbol{q} = w_2 + \boldsymbol{v_2} = w_2 + x_2\boldsymbol{i} + y_2\boldsymbol{j} + z_2\boldsymbol{k}$$
$$\Rightarrow \boldsymbol{pq} = w_1w_2 - \boldsymbol{v_1} \cdot \boldsymbol{v_2} + w_2\boldsymbol{v_1} + w_1\boldsymbol{v_2} + \boldsymbol{v_1} \times \boldsymbol{v_2}$$
$$= (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2) + w_2\overrightarrow{(x_1, y_1, z_1)} + w_1\overrightarrow{(x_2, y_2, z_2)} + \overrightarrow{(y_1z_2 - z_1y_2, z_1x_2 - x_1z_2, x_1y_2 - y_1x_2)}$$
$$= (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2) + (x_1w_2 + w_1x_2 + y_1z_2 - z_1y_2)\boldsymbol{i} + (y_1w_2 + w_1y_2 + z_1x_2 - x_1z_2)\boldsymbol{j}$$
$$+ (z_1w_2 + w_1z_2 + x_1y_2 - y_1x_2)\boldsymbol{k}$$

首先是欧拉角转四元数，由于欧拉角表示旋转一般是可以由3个Elemental Rotation组合而成，这一个

跟EulerAngleToRotationMatrix是一样道理的，所以用于旋转的四元数$\boldsymbol{q}(x,y,z,w)$表达式是：

$$
\begin{aligned}
&\boldsymbol{q}(\alpha,\beta,\gamma)=\boldsymbol{q}_y(\alpha)\boldsymbol{q}_x(\beta)\boldsymbol{q}_z(\gamma)\\
&=\begin{bmatrix}0\\ \sin\left(\frac{\alpha}{2}\right)\\ 0\\ \cos\left(\frac{\alpha}{2}\right)\end{bmatrix}\begin{bmatrix}\sin\left(\frac{\beta}{2}\right)\\ 0\\ 0\\ \cos\left(\frac{\beta}{2}\right)\end{bmatrix}\begin{bmatrix}0\\ 0\\ \sin\left(\frac{\gamma}{2}\right)\\ \cos\left(\frac{\gamma}{2}\right)\end{bmatrix}\\
&=\begin{bmatrix}\cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\\ \sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\\ -\sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\\ \cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\end{bmatrix}\begin{bmatrix}0\\ 0\\ \sin\left(\frac{\gamma}{2}\right)\\ \cos\left(\frac{\gamma}{2}\right)\end{bmatrix}\\
&=\begin{bmatrix}\cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right)+\sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right)\\ \sin\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right)-\cos\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right)\\ -\sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right)+\cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right)\\ \cos\left(\frac{\alpha}{2}\right)\cos\left(\frac{\beta}{2}\right)\cos\left(\frac{\gamma}{2}\right)+\sin\left(\frac{\alpha}{2}\right)\sin\left(\frac{\beta}{2}\right)\sin\left(\frac{\gamma}{2}\right)\end{bmatrix}
\end{aligned}
$$

这个自己推导的结果跟[1]NASA Technical Report Appendix A6对比过了（[1]中四元数$\boldsymbol{q}=q_1+q_2\boldsymbol{i}+q_3\boldsymbol{j}+q_4\boldsymbol{k}$），看起来没什么问题。

3.2 Quaternion to Euler Angles

本来我以为，从四元数提取欧拉角的思路跟旋转矩阵提取欧拉角类似，也是用四元数的元素运算和反三角函数凑出公式来。后来我发现这简直就是一个极其硬核的任务，展开之后每一项都是六次多项式，画面非常暴力且少儿不宜，直接强行凑的话画风大概是这样：

$$
\begin{aligned}
xw&=(c_1s_2c_3+s_1c_2s_3)(c_1c_2c_3+s_1s_2s_3)\\
&=c_1^2c_2s_2c_3^2+c_1s_1s_2^2c_3s_3+c_1s_1c_2^2c_3s_3+s_1^2c_2s_2s_3^2\\
yz&=(s_1c_2c_3-c_1s_2s_3)(-s_1s_2c_3+c_1c_2s_3)\\
&=-s_1^2c_2s_2c_3^2+c_1s_1c_2^2c_3s_3+c_1s_1s_2^2c_3s_3-c_1^2c_2s_2s_3^2
\end{aligned}
$$

$$
\begin{aligned}
\Longrightarrow 2(xw-yz)&=2(c_1^2c_2s_2+s_1^2c_2s_2)\\
&=2c_2s_2=sin\beta\\
\Longrightarrow \beta&=arcsin(2(xw-yz))
\end{aligned}
$$

（译者注：这个结果的符号和旋转矩阵元素刚好相反，感觉有一点点问题orz）

这还只是最好凑的那一个，惹不起惹不起。所以还是要先转矩阵再转欧拉角，用旋转矩阵的元素展开会好很多。这里就不把公式展开了，因为四元数直接转欧拉角依旧是要处理gimbal lock的corner case，还是那么麻烦，所以这里先鸽了23333

# 4 轴-角(Axis-Angle)

## 4.1 AxisAngle To Quaternion

轴-角顾名思义就是绕某条顶轴旋转一定角度，从这个意义上看，它构造四元数是非常和谐的，毕竟几何意义有一点点类似，轴角构造四元数具体构造在Chap2有提到，绕单位轴$\boldsymbol{u}$旋转$\theta$的四元数是：

$$\boldsymbol{q}(w,\boldsymbol{v}) = \left(cos\left(\frac{\theta}{2}\right), \boldsymbol{u}sin\left(\frac{\theta}{2}\right)\right)$$

## 4.2 AxisAngle To Rotation Matrix

因为已经有了欧拉角/旋转矩阵/四元数来表示当前姿态了，所以Axis Angle我就打算用来构造其他表示的delta Rotation。

首先是Axis Angle转Rotation Matrix。可以从[9]罗德里格斯旋转公式Rodrigues Rotation Formula开始推导。设$\boldsymbol{v}$是我们要旋转的单位向量，$\boldsymbol{k}$是旋转轴，$\boldsymbol{v}$绕$\boldsymbol{k}$旋转角度θ，那么：

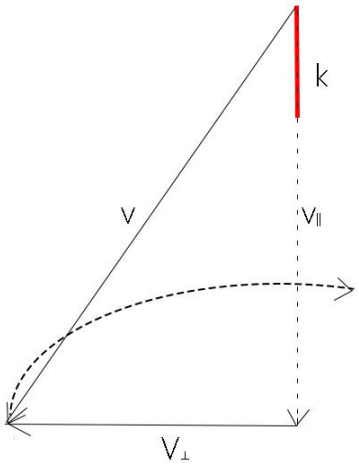
$$\boldsymbol{v}_{\mathrm{rot}} = \boldsymbol{v}cos\theta + (\boldsymbol{k}\times\boldsymbol{v})sin\theta + \boldsymbol{k}(\boldsymbol{k}\cdot\boldsymbol{v})(1-cos\theta)$$

推导的思路是这样子的，我们先对被旋转的向量$\boldsymbol{v}$进行正交分解，分解成投影到旋转轴$\boldsymbol{k}$的分量和垂直于$\boldsymbol{k}$的分量：

$$\boldsymbol{v} = \boldsymbol{v}_{\parallel} + \boldsymbol{v}_{\perp}$$

其中：

$$\boldsymbol{v}_{\parallel} = (\boldsymbol{v}\cdot\boldsymbol{k})\boldsymbol{k}$$
$$\boldsymbol{v}_{\perp} = -\boldsymbol{k}\times(\boldsymbol{k}\times\boldsymbol{v})$$




图：就假设旋转轴k和向量v都在屏幕这个平面上吧

于是绕$\boldsymbol{k}$旋转向量$\boldsymbol{v}$其实就是把上面正交投影的向量分别旋转之后加起来。那么很明显，$\boldsymbol{v}_{\parallel}$与旋转轴同向，旋转后没什么变化。所以我们只需要旋转一下$\boldsymbol{v}_{\perp}$就好了= =。要注意$\boldsymbol{v}_{\perp}$的旋转平面与旋转轴垂直，所以$\boldsymbol{v}_{\perp}$的旋转平面的二维笛卡尔坐标轴分别是$\boldsymbol{v}_{\perp}$与$\boldsymbol{k}\times\boldsymbol{v}$，那么旋转后的$\boldsymbol{v}_{\perp}$表达式为：

$$\boldsymbol{v}_{\perp-\boldsymbol{rotated}} = cos\theta\boldsymbol{v}_{\perp} + sin\theta\boldsymbol{k}\times\boldsymbol{v}$$

Wikipedia里面，旋转后的$\boldsymbol{v}$表达式的推导变形是这样的：

$$\begin{aligned}\boldsymbol{v} &= \boldsymbol{v}_{\perp-\boldsymbol{rotated}} + \boldsymbol{v}_{\parallel-\boldsymbol{rotated}}\\ &= \boldsymbol{v}_{\parallel} + cos\theta(\boldsymbol{v}-\boldsymbol{v}_{\parallel}) + sin\theta\boldsymbol{k}\times\boldsymbol{v}\\ &= cos\theta\boldsymbol{v} + (1-cos\theta)(\boldsymbol{k}\cdot\boldsymbol{v})\boldsymbol{k} + sin\theta\boldsymbol{k}\times\boldsymbol{v}\end{aligned}$$

但是上面的变形比较坑，没有为下文转成矩阵表达做铺垫。所以如果把表达式变得没有点乘的话，旋转后的$\boldsymbol{v}$表达式是：

$$\begin{aligned}\boldsymbol{v} &= \boldsymbol{v}_{\perp-\boldsymbol{rotated}} + \boldsymbol{v}_{\parallel-\boldsymbol{rotated}}\\ &= cos\theta\boldsymbol{v}_{\perp} + sin\theta\boldsymbol{k}\times\boldsymbol{v} + \boldsymbol{v}_{\parallel}\\ &= -cos\theta\boldsymbol{k}\times(\boldsymbol{k}\times\boldsymbol{v}) + sin\theta\boldsymbol{k}\times\boldsymbol{v} + (\boldsymbol{v}-\boldsymbol{v}_{\perp})\\ &= -cos\theta\boldsymbol{k}\times(\boldsymbol{k}\times\boldsymbol{v}) + sin\theta\boldsymbol{k}\times\boldsymbol{v} + (\boldsymbol{v}-(-\boldsymbol{k}\times(\boldsymbol{k}\times\boldsymbol{v}))\\ &= \boldsymbol{v} + (1-cos\theta)\boldsymbol{k}\times(\boldsymbol{k}\times\boldsymbol{v}) + sin\theta\boldsymbol{k}\times\boldsymbol{v}\end{aligned}$$

然后我们可以把绕轴旋转的变换写成矩阵形式：

$$\boldsymbol{R}(\boldsymbol{u},\theta) = \boldsymbol{I} + sin\theta\boldsymbol{M} + (1-cos\theta)\boldsymbol{M}^2$$

其中M是叉积矩阵[10]：

$$\boldsymbol{M} = \begin{bmatrix}0 & -u_z & u_y\\ u_z & 0 & -u_x\\ -u_y & u_x & 0\end{bmatrix}$$

$$\boldsymbol{Mv} = \boldsymbol{k} \times \boldsymbol{v}$$

所以再继续展开，直接用Axis-Angle构造旋转矩阵就是（最终结果可参考[11]）：

$$\boldsymbol{R}(\boldsymbol{u},\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + sin\theta \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix} + (1 - cos\theta) \begin{bmatrix} -(u_y^2 + u_z^2) & u_x u_y & u_x u_z \\ u_x u_y & -(u_x^2 + u_z^2) & u_y u_z \\ u_x u_z & u_y u_z & -(u_x^2 + u_y^2) \end{bmatrix}$$

$$= \begin{bmatrix} 1 - (u_y^2 + u_z^2)(1 - cos\theta) & 0 - sin\theta u_z + (1 - cos\theta)u_x u_y & 0 + sin\theta u_y + (1 - cos\theta)u_x u_z \\ 0 + sin\theta u_z + (1 - cos\theta)u_x u_y & 1 - (1 - cos\theta)(u_x^2 + u_z^2) & 0 - sin\theta u_x + (1 - cos\theta)u_y u_z \\ 0 - sin\theta u_y + (1 - cos\theta)u_x u_z & 0 + sin\theta u_x + (1 - cos\theta)u_y u_z & 1 - (1 - cos\theta)(u_x^2 + u_y^2) \end{bmatrix}$$

$$= \begin{bmatrix} cos\theta + u_x^2(1 - cos\theta) & -sin\theta u_z + (1 - cos\theta)u_x u_y & sin\theta u_y + (1 - cos\theta)u_x u_z \\ sin\theta u_z + (1 - cos\theta)u_x u_y & cos\theta + u_y^2(1 - cos\theta) & -sin\theta u_x + (1 - cos\theta)u_y u_z \\ -sin\theta u_y + (1 - cos\theta)u_x u_z & sin\theta u_x + (1 - cos\theta)u_y u_z & cos\theta + u_z^2(1 - cos\theta) \end{bmatrix}$$

# 引用


[1]Henderson, D.M.. Euler angles, quaternions, and transformation matrices for space shuttle analysis[C]//NASA, Jun 09, 1977.

[2] https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles#Tait%E2%80%93Bryan_angles

[3] https://en.wikipedia.org/wiki/Euler_angles

[4] https://en.wikipedia.org/wiki/Rotation_matrix

[5] Slabaugh G G. Computing Euler angles from a rotation matrix[J]. 1999.

[6] Mike Day, Converting a Rotation Matrix to a Quaternion. https://d3cw3dd2w32x2b.cloudfront.net/wp-content/uploads/2015/01/matrix-to-quat.pdf

[7] Tomas K.M. , Eric H., Naty H.. Real Time Rendering 3rd Edition , p68-p69, 2008.

[8] Tomas K.M. , Eric H., Naty H.. Real Time Rendering 3rd Edition , p76-p77, 2008.

[9] https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

[10] https://en.wikipedia.org/wiki/Cross_product#Conversion_to_matrix_multiplication

[11] http://mathworld.wolfram.com/RodriguesRotationFormula.html

[12] https://zh.wikipedia.org/wiki/%E5%9B%9B%E5%85%83%E6%95%B8

[13] https://blog.csdn.net/silangquan/article/details/39008903

[14] Quaternion and Rotations, http://run.usc.edu/cs520-s12/quaternions/quaternions-cs520.pdf