1.轴、角转四元数公式：

q = [cos(Q/2), sin(Q /2)v] v 是旋转轴矢量，Q 是旋转角度

通过旋转轴和绕该轴旋转的角度可以构造一个四元数：

$$w = \cos(\alpha/2)$$
$$x = \sin(\alpha/2)\cos(\beta_x)$$
$$y = \sin(\alpha/2)\cos(\beta_y)$$
$$z = \sin(\alpha/2)\cos(\beta_z)$$

其中 α 是绕旋转轴旋转的角度，$\cos(\beta_x), \cos(\beta_y), \cos(\beta_z)$ 为旋转轴在x,y,z方向的分量（由此确定了旋转轴）。

2.欧拉角转四元数:

qroll = [cos (y/2), (sin(y/2), 0, 0)]

qpitch = [cos (q/2), (0, sin(q/2), 0)]

qyaw = [cos(f /2), (0, 0, sin(f /2)]

pitch(俯仰），yaw(偏航），roll（滚转）

定义ψ、θ、φ分别为绕Z轴、Y轴、X轴的旋转角度，如果用Tait-Bryan angle表示，分别为Yaw、Pitch、Roll。

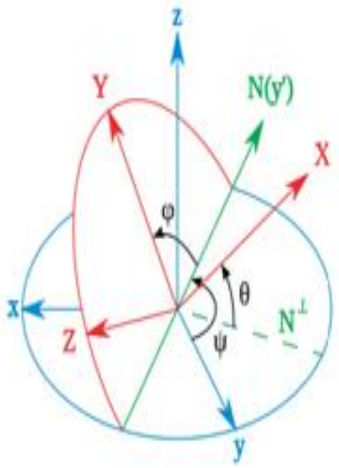

图2 Tait-Bryan angles (from wikipedia)

欧拉角到四元数的转换

$$q=\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}=\begin{bmatrix} \cos(\varphi/2)\cos(\theta/2)\cos(\psi/2)+\sin(\varphi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\varphi/2)\cos(\theta/2)\cos(\psi/2)-\cos(\varphi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\varphi/2)\sin(\theta/2)\cos(\psi/2)+\sin(\varphi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\varphi/2)\cos(\theta/2)\sin(\psi/2)-\sin(\varphi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix}$$

四元数到欧拉角的转换

$$\begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix}=\begin{bmatrix} \arctan\dfrac{2(wx+yz)}{1-2(x^2+y^2)} \\ \arcsin(2(wy-zx)) \\ \arctan(\dfrac{2(wz+xy)}{1-2(y^2+z^2)}) \end{bmatrix}$$

*arctan*和*arcsin*的结果是$\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$，这并不能覆盖所有朝向(对于$\theta$角$\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$的取值范围已经满足)，因此需要用*atan2*来代替*arctan*。

$$\begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix}=\begin{bmatrix} atan2(2(wx+yz),1-2(x^2+y^2)) \\ arcsin(2(wy-zx)) \\ atan2(2(wz+xy),1-2(y^2+z^2)) \end{bmatrix}$$

在其他坐标系下，需根据坐标轴的定义，调整一下以上公式。如在Direct3D中，笛卡尔坐标系的X轴变为Z轴，Y轴变为X轴，Z轴变为Y轴（无需考虑方向）。

$$q=\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}=\begin{bmatrix} \cos(\varphi/2)\cos(\theta/2)\cos(\psi/2)+\sin(\varphi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\varphi/2)\sin(\theta/2)\cos(\psi/2)+\sin(\varphi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\varphi/2)\cos(\theta/2)\sin(\psi/2)-\sin(\varphi/2)\sin(\theta/2)\cos(\psi/2) \\ \sin(\varphi/2)\cos(\theta/2)\cos(\psi/2)-\cos(\varphi/2)\sin(\theta/2)\sin(\psi/2) \end{bmatrix}$$

$$\begin{bmatrix} \varphi \\ \theta \\ \psi \end{bmatrix}=\begin{bmatrix} atan2(2(wz+xy),1-2(z^2+x^2)) \\ arcsin(2(wx-yz)) \\ atan2(2(wy+zx),1-2(x^2+y^2)) \end{bmatrix}$$

两个四元数相乘q" = q * q'的矩阵表示法如下所示：

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ w'' \end{bmatrix} = \begin{bmatrix} w & -z & y & x \\ z & w & -x & y \\ -y & x & w & z \\ -x & -y & -z & w \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

若令q = [S, V] = [cosθ, u*sinθ]，其中u为单位向量，而令q'= [S', V']为一四元数，则经过导证，可以得出q * q' * q^(-1)会使得q'绕着u轴旋转2θ。

由四元数的矩阵乘法与四元数的旋转，可以导证出上面的旋转公式可以使用以下的矩阵乘法来达成：

$$\begin{bmatrix} x'' \\ y'' \\ z'' \\ w'' \end{bmatrix} = \begin{bmatrix} 1-2*(y^2+z^2) & 2*(x*y-w*z) & 2*(w*y+x*z) & 0 \\ 2*(x*y+w*z) & 1-2*(x^2+z^2) & 2*(y*z-w*x) & 0 \\ 2*(x*z-w*y) & 2*(y*z+w*x) & 1-2*(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

讲了这么多，其实就是要引出上面这个矩阵乘法，也就是说如果您要让向量(x', y', z')（w'为0）对某个单位向量轴u(x, y, z)旋转角度2θ，则w = cosθ，代入以上的矩阵乘法，即可得旋转后的(x", y", z")，如果为了方便，转换矩阵的最下列与最右行会省略不写出来，而如下所示：

$$\begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} 1-2*(y^2+z^2) & 2*(x*y-w*z) & 2*(w*y+x*z) \\ 2*(x*y+w*z) & 1-2*(x^2+z^2) & 2*(y*z-w*x) \\ 2*(x*z-w*y) & 2*(y*z+w*x) & 1-2*(x^2+y^2) \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

3.四元数转旋转矩阵：

$$Rm = \begin{vmatrix} 1 - 2y2 - 2z2 & 2yz + 2wx & 2xz - 2wy \\ 2xy - 2wz & 1 - 2x2 - 2z2 & 2yz - 2wx \\ 2xz + 2wy & 2yz - 2wx & 1 - 2x2 - 2y2 \end{vmatrix}$$

4.矩阵转四元数代码：

```
MatToQuat(float m[4][4], QUAT * quat)
{
  float  tr, s, q[4];
```

```
  int    i, j, k;

  int nxt[3] = {1, 2, 0};

  tr = m[0][0] + m[1][1] + m[2][2];

  // check the diagonal
  if (tr > 0.0)
  {
    s = sqrt (tr + 1.0);
    quat->w = s / 2.0;
    s = 0.5 / s;
    quat->x = (m[1][2] - m[2][1]) * s;
    quat->y = (m[2][0] - m[0][2]) * s;
    quat->z = (m[0][1] - m[1][0]) * s;
  }
else
  {
    // diagonal is negative
          i = 0;
    if (m[1][1] > m[0][0]) i = 1;
    if (m[2][2] > m[i][i]) i = 2;
    j = nxt[i];
    k = nxt[j];

    s = sqrt ((m[i][i] - (m[j][j] + m[k][k])) + 1.0);

    q[i] = s * 0.5;

    if (s != 0.0) s = 0.5 / s;
```

```
    q[3] = (m[j][k] - m[k][j]) * s;
    q[j] = (m[i][j] + m[j][i]) * s;
    q[k] = (m[i][k] + m[k][i]) * s;

    quat->x = q[0];
    quat->y = q[1];
    quat->z = q[2];
    quat->w = q[3];
  }
}
```

5.四元数转矩阵代码：

```
QuatToMatrix(QUAT * quat, float m[4][4]){
float wx, wy, wz, xx, yy, yz, xy, xz, zz, x2, y2, z2;

// calculate coefficients
x2 = quat->x + quat->x;
y2 = quat->y + quat->y;
z2 = quat->z + quat->z;
xx = quat->x * x2;
xy = quat->x * y2;
xz = quat->x * z2;
yy = quat->y * y2;
yz = quat->y * z2;
zz = quat->z * z2;
wx = quat->w * x2;
wy = quat->w * y2;
wz = quat->w * z2;

m[0][0] = 1.0 - (yy + zz);
```

```
m[1][0] = xy - wz;
m[2][0] = xz + wy;
m[3][0] = 0.0;

m[0][1] = xy + wz;
m[1][1] = 1.0 - (xx + zz);
m[2][1] = yz - wx;
m[3][1] = 0.0;

m[0][2] = xz - wy;
m[1][2] = yz + wx;
m[2][2] = 1.0 - (xx + yy);
m[3][2] = 0.0;

m[0][3] = 0;
m[1][3] = 0;
m[2][3] = 0;
m[3][3] = 1;
}
```

6.欧拉角转四元数代码：

```
EulerToQuat(float roll, float pitch, float yaw, QUAT * quat)
{
	float cr, cp, cy, sr, sp, sy, cpcy, spsy;

// calculate trig identities
cr = cos(roll/2);

	cp = cos(pitch/2);
	cy = cos(yaw/2);
```

```
        sr = sin(roll/2);
        sp = sin(pitch/2);
        sy = sin(yaw/2);

        cpcy = cp * cy;
        spsy = sp * sy;


        quat->w = cr * cpcy + sr * spsy;
        quat->x = sr * cpcy - cr * spsy;
        quat->y = cr * sp * cy + sr * cp * sy;
        quat->z = cr * cp * sy - sr * sp * cy;
}
```

四元数乘法

```
QuatMul(QUAT *q1, QUAT *q2, QUAT *res){


float A, B, C, D, E, F, G, H;


A = (q1->w + q1->x)*(q2->w + q2->x);
B = (q1->z - q1->y)*(q2->y - q2->z);
C = (q1->w - q1->x)*(q2->y + q2->z);
D = (q1->y + q1->z)*(q2->w - q2->x);
E = (q1->x + q1->z)*(q2->x + q2->y);
F = (q1->x - q1->z)*(q2->x - q2->y);
G = (q1->w + q1->y)*(q2->w - q2->z);
H = (q1->w - q1->y)*(q2->w + q2->z);
```

```
res->w = B + (-E - F + G + H) /2;
res->x = A - (E + F + G + H)/2;
res->y = C + (E - F + G - H)/2;
res->z = D + (E - F - G + H)/2;
}
```

四元数插值

```
QuatSlerp(QUAT * from, QUAT * to, float t, QUAT * res)
    {
      float          to1[4];
      double         omega, cosom, sinom, scale0, scale1;


      // calc cosine
      cosom = from->x * to->x + from->y * to->y + from->z * to->z
                        + from->w * to->w;


      // adjust signs (if necessary)
      if ( cosom <0.0 ){ cosom = -cosom; to1[0] = - to->x;
            to1[1] = - to->y;
            to1[2] = - to->z;
            to1[3] = - to->w;
      } else  {
            to1[0] = to->x;
            to1[1] = to->y;
            to1[2] = to->z;
            to1[3] = to->w;
      }
```

```
        // calculate coefficients


       if ( (1.0 - cosom) > DELTA ) {
                // standard case (slerp)
                omega = acos(cosom);
                sinom = sin(omega);
                scale0 = sin((1.0 - t) * omega) / sinom;
                scale1 = sin(t * omega) / sinom;


        } else {
    // "from" and "to" quaternions are very close
            //  ... so we can do a linear interpolation
                scale0 = 1.0 - t;
                scale1 = t;
        }
        // calculate final values
        res->x = scale0 * from->x + scale1 * to1[0];
        res->y = scale0 * from->y + scale1 * to1[1];
        res->z = scale0 * from->z + scale1 * to1[2];
        res->w = scale0 * from->w + scale1 * to1[3];
}
```