欧拉角是通过依次相对旋转完成，也可以通过绕一个旋转轴一次旋转到相同的位置

欧拉角是指：绕坐标轴的旋转角

四元数：首先在坐标系下有一个单位向量（x,y,z）,绕该轴旋转θ角，则可以通过四元数表示这一旋转过程:(w,ix,jy,kz),其中w=cos(θ/2)，x=sin(θ/2)，y=sin(θ/2) ，z=sin(θ/2)

**四元数矩阵的乘法及其可易性**

http://pubimage.360doc.com/NewArticle/bgcolor.jpg

http://pubimage.360doc.com/NewArticle/fontSize.jpg

**四元数和旋转矩阵互相转换**

2018-04-07  [须臾oiekq...](http://www.360doc.com/userhome/54247461)  转自 [贤人好客](http://www.360doc.com/userhome/617416)

[修改](http://www.360doc.com/editarticle.aspx?articleid=743492756)

[微信](javascript:void(0);)分享：

|  |
| --- |
| 四元数的用途和好处我就不多说了。 之前在网上找旋转矩阵和四元数相互转换的代码，找了几个都不大对劲，正反算算不过来，最后还是从osg源码里贴出来的这个，应该没什么问题。 这里给一个链接，[Matrix and Quaternion FAQ](http://www.j3d.org/matrix_faq/matrfaq_latest.html) http://www.flipcode.com/documents/matrfaq.html  以下是源文件：  #include<iostream> #include<cmath> using namespace std;  typedef double ValType;  struct Quat; struct Matrix;  struct Quat { ValType \_v[4];//x, y, z, w  /// Length of the quaternion = sqrt( vec . vec ) ValType length() const {     return sqrt( \_v[0]\*\_v[0] + \_v[1]\*\_v[1] + \_v[2]\*\_v[2] + \_v[3]\*\_v[3]); }  /// Length of the quaternion = vec . vec ValType length2() const {     return \_v[0]\*\_v[0] + \_v[1]\*\_v[1] + \_v[2]\*\_v[2] + \_v[3]\*\_v[3]; } };  struct Matrix { ValType \_mat[3][3]; };  #define QX q.\_v[0] #define QY q.\_v[1] #define QZ q.\_v[2] #define QW q.\_v[3]  void Quat2Matrix(const Quat& q, Matrix& m) {     double length2 = q.length2();     if (fabs(length2) <= std::numeric\_limits<double>::min())     {         m.\_mat[0][0] = 0.0; m.\_mat[1][0] = 0.0; m.\_mat[2][0] = 0.0;         m.\_mat[0][1] = 0.0; m.\_mat[1][1] = 0.0; m.\_mat[2][1] = 0.0;         m.\_mat[0][2] = 0.0; m.\_mat[1][2] = 0.0; m.\_mat[2][2] = 0.0;     }     else     {         double rlength2;         // normalize quat if required.         // We can avoid the expensive sqrt in this case since all 'coefficients' below are products of two q components.         // That is a square of a square root, so it is possible to avoid that         if (length2 != 1.0)         {             rlength2 = 2.0/length2;         }         else         {             rlength2 = 2.0;         }                  // Source: Gamasutra, Rotating Objects Using Quaternions         //         //http://www.gamasutra.com/features/19980703/quaternions\_01.htm                  double wx, wy, wz, xx, yy, yz, xy, xz, zz, x2, y2, z2;                  // calculate coefficients         x2 = rlength2\*QX;         y2 = rlength2\*QY;         z2 = rlength2\*QZ;                  xx = QX \* x2;         xy = QX \* y2;         xz = QX \* z2;                  yy = QY \* y2;         yz = QY \* z2;         zz = QZ \* z2;                  wx = QW \* x2;         wy = QW \* y2;         wz = QW \* z2;                  // Note. Gamasutra gets the matrix assignments inverted, resulting         // in left-handed rotations, which is contrary to OpenGL and OSG's          // methodology. The matrix assignment has been altered in the next         // few lines of code to do the right thing.         // Don Burns - Oct 13, 2001         m.\_mat[0][0] = 1.0 - (yy + zz);         m.\_mat[1][0] = xy - wz;         m.\_mat[2][0] = xz + wy;                           m.\_mat[0][1] = xy + wz;         m.\_mat[1][1] = 1.0 - (xx + zz);         m.\_mat[2][1] = yz - wx;                  m.\_mat[0][2] = xz - wy;         m.\_mat[1][2] = yz + wx;         m.\_mat[2][2] = 1.0 - (xx + yy);     } }  void Matrix2Quat(const Matrix& m, Quat& q) {     ValType s;     ValType tq[4];     int    i, j;      // Use tq to store the largest trace     tq[0] = 1 + m.\_mat[0][0]+m.\_mat[1][1]+m.\_mat[2][2];     tq[1] = 1 + m.\_mat[0][0]-m.\_mat[1][1]-m.\_mat[2][2];     tq[2] = 1 - m.\_mat[0][0]+m.\_mat[1][1]-m.\_mat[2][2];     tq[3] = 1 - m.\_mat[0][0]-m.\_mat[1][1]+m.\_mat[2][2];      // Find the maximum (could also use stacked if's later)     j = 0;     for(i=1;i<4;i++) j = (tq[i]>tq[j])? i : j;      // check the diagonal     if (j==0)     {         /\* perform instant calculation \*/         QW = tq[0];         QX = m.\_mat[1][2]-m.\_mat[2][1];         QY = m.\_mat[2][0]-m.\_mat[0][2];         QZ = m.\_mat[0][1]-m.\_mat[1][0];     }     else if (j==1)     {         QW = m.\_mat[1][2]-m.\_mat[2][1];         QX = tq[1];         QY = m.\_mat[0][1]+m.\_mat[1][0];         QZ = m.\_mat[2][0]+m.\_mat[0][2];     }     else if (j==2)     {         QW = m.\_mat[2][0]-m.\_mat[0][2];         QX = m.\_mat[0][1]+m.\_mat[1][0];         QY = tq[2];         QZ = m.\_mat[1][2]+m.\_mat[2][1];     }     else /\* if (j==3) \*/     {         QW = m.\_mat[0][1]-m.\_mat[1][0];         QX = m.\_mat[2][0]+m.\_mat[0][2];         QY = m.\_mat[1][2]+m.\_mat[2][1];         QZ = tq[3];     }      s = sqrt(0.25/tq[j]);     QW \*= s;     QX \*= s;     QY \*= s;     QZ \*= s; }  void printMatrix(const Matrix& r, string name) { cout<<"RotMat "<<name<<" = "<<endl; cout<<"\t"<<r.\_mat[0][0]<<" "<<r.\_mat[0][1]<<" "<<r.\_mat[0][2]<<endl; cout<<"\t"<<r.\_mat[1][0]<<" "<<r.\_mat[1][1]<<" "<<r.\_mat[1][2]<<endl; cout<<"\t"<<r.\_mat[2][0]<<" "<<r.\_mat[2][1]<<" "<<r.\_mat[2][2]<<endl; cout<<endl; }  void printQuat(const Quat& q, string name) { cout<<"Quat "<<name<<" = "<<endl; cout<<"\t"<<q.\_v[0]<<" "<<q.\_v[1]<<" "<<q.\_v[2]<<" "<<q.\_v[3]<<endl; cout<<endl; }  int main() { ValType phi, omiga, kappa;  phi = 1.32148229302237 ; omiga = 0.626224465189316 ; kappa = -1.4092143985971;      ValType a1,a2,a3,b1,b2,b3,c1,c2,c3;      a1 = cos(phi)\*cos(kappa) - sin(phi)\*sin(omiga)\*sin(kappa);     a2 = -cos(phi)\*sin(kappa) - sin(phi)\*sin(omiga)\*cos(kappa);     a3 = -sin(phi)\*cos(omiga);      b1 = cos(omiga)\*sin(kappa);     b2 = cos(omiga)\*cos(kappa);     b3 = -sin(omiga);      c1 = sin(phi)\*cos(kappa) + cos(phi)\*sin(omiga)\*sin(kappa);     c2 = -sin(phi)\*sin(kappa) + cos(phi)\*sin(omiga)\*cos(kappa);     c3 = cos(phi)\*cos(omiga);          Matrix r;     r.\_mat[0][0] = a1; r.\_mat[0][1] = a2; r.\_mat[0][2] = a3;  r.\_mat[1][0] = b1; r.\_mat[1][1] = b2; r.\_mat[1][2] = b3;  r.\_mat[2][0] = c1; r.\_mat[2][1] = c2; r.\_mat[2][2] = c3;  printMatrix(r, "r");       //////////////////////////////////////////////////////////  Quat q; Matrix2Quat(r, q);  printQuat(q, "q");  Matrix \_r; Quat2Matrix(q, \_r);  printMatrix(\_r, "\_r");      system("pause"); return 0;  } |