◊ III.5

Euler Angle Conversion

Ken Shoemake
University of Pennsylvania
Philadelphia, PA
shoemake@graphics.cis.upenn.edu

◊ Introduction ◊

All modern computer graphics systems use homogeneous matrices internally, and most use quaternions (Foley *et al.* 1990). Many, however, retain a text-based interface using Euler angles and need to convert to and from their internal representations. But to exchange data with other systems, an interface may need to handle all of the 24 different ways of specifying rotations with a triple of angles (Craig 1989, Appendix B). The purpose of this Gem is to show how a few lines of code can convert any of these varieties of Euler angles to and from matrices and quaternions, with the choice of variety given as a parameter.

Recall that a triple of Euler angles $[\theta_1, \theta_2, \theta_3]$ describes how a coordinate frame r rotates with respect to a static frame s. The triple is interpreted as a rotation by θ_1 around an axis $\mathbf{A}_1$, then a rotation by θ_2 around an axis $\mathbf{A}_2$, and finally a rotation by θ_3 around an axis $\mathbf{A}_3$, with $\mathbf{A}_2$ different from both $\mathbf{A}_1$ and $\mathbf{A}_3$. The axes are restricted to the coordinate axes, X, Y, and Z, giving 12 possibilities: XYZ, XYX, YZX, YZY, ZXY, ZXZ, XZY, XZX, YXZ, YXY, ZYX, ZYZ. The jump to 24 comes from the choice of using axes from either the static frame s or the rotating frame r. Equivalently, the rotations can be listed right to left or left to right.[1]

◊ Combinatorial Collapse ◊

It will be helpful to designate a convention with a 4-tuple: inner axis, parity, repetition, and frame. The inner axis will be the axis of the first standard matrix to multiply a vector. Since we are assuming column vectors, the inner axis is the axis of the rightmost matrix. Parity is even if the inner axis X is followed by the middle axis Y, or Y is followed by Z, or Z is followed by X; otherwise parity is odd. Repetition means whether the first and last axes are the same or different. Frame refers to the choice of either the static or the rotating frame, and applies to all three axes. With static frame axes the

[1]See (Craig 1989, Section 2.8) for a more leisurely discussion.

IBM ISBN 0-12-336155-9
Macintosh ISBN 0-12-336156-7

inner axis is the first axis, while with rotating frame axes the inner axis is the last axis.

Define the standard rotation matrices $\mathbf{R}_x(\theta)$, $\mathbf{R}_y(\theta)$, and $\mathbf{R}_z(\theta)$ as

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin -\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ \sin -\theta & 0 & \cos\theta \end{bmatrix} \quad \begin{bmatrix} \cos\theta & \sin -\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then $\mathbf{R}_x(\theta_3)\mathbf{R}_y(\theta_2)\mathbf{R}_x(\theta_1)$ is [X,Even,Same,S-frame], which we can abbreviate as XESS, while $\mathbf{R}_z(\theta_1)\mathbf{R}_x(\theta_2)\mathbf{R}_y(\theta_3)$ is [Y,Odd,Diff,R-frame], which we can abbreviate as YODR. Since each of the last three choices in the tuple can be encoded in a single bit, the whole tuple compactly encodes as an integer between 0 and 23. From a human factors perspective, the tuple notation makes it impossible to refer to nonsense conventions like XYY, and the integer values can be given meaningful names (see the code).

From a programming perspective, the tuple helps us collapse 24 cases to 2. Suppose we have code to convert a rotation matrix to XEDS angles, $\mathbf{R} = \mathbf{R}_z(\theta_3)\mathbf{R}_y(\theta_2)\mathbf{R}_x(\theta_1)$. If we are asked to extract XED$\underline{\text{R}}$ angles, $\mathbf{R} = \mathbf{R}_z(\theta_1)\mathbf{R}_y(\theta_2)\mathbf{R}_x(\theta_3)$, we use our code as is, and simply swap θ_1 and θ_3 afterwards.

We can also accomodate $\underline{\text{Y}}$EDS angles, $\mathbf{R} = \mathbf{R}_x(\theta_3)\mathbf{R}_z(\theta_2)\mathbf{R}_y(\theta_1)$, by first changing the basis of $\mathbf{R}$ by a permutation matrix $\mathbf{P}$ which converts (Y, Z, X) to (X, Y, Z). Now applying our old XEDS code to $\mathbf{PRP}^{\mathrm{T}}$ extracts the YEDS angles we want from $\mathbf{R}$. In fact, we can extract any permutation we like by a suitable choice of $\mathbf{P}$, with one caveat. When the permutation is odd (X$\underline{\text{O}}$DS), we are switching to a left-handed coordinate frame, and the sense of rotation is reversed. The fix is simple: negate the angles.

No permutation, however, can turn (X, Y, Z) into (X, Y, X); we need new code for XE$\underline{\text{S}}$S angles. But the XEDS and XESS archetypes, coupled with permuting, negating, and swapping, are all we need. For efficiency, we can permute as we access the matrix entries during extraction. And while the discussion so far has focused on matrix-to-angle extraction, angle-to-matrix conversion permits the same economies. A little more thought shows quaternion conversions can also be collapsed.

◇ Archetypes ◇

Now we need archetypes for the conversions. The XEDS and XESS choices list fixed axis rotations in the order they are applied, and yield the matrices

$$\mathbf{R}_{xyz} = \begin{bmatrix} c_2c_3 & s_2s_1c_3 - c_1s_3 & s_2c_1c_3 + s_1s_3 \\ c_2s_3 & s_2s_1s_3 + c_1c_3 & s_2c_1s_3 - s_1c_3 \\ -s_2 & c_2s_1 & c_2c_1 \end{bmatrix}$$

and

$$\mathbf{R}_{xyx} = \begin{bmatrix} c_2 & s_2s_1 & s_2c_1 \\ s_2s_3 & -c_2s_1s_3 + c_1c_3 & -c_2c_1s_3 - s_1c_3 \\ -s_2c_3 & c_2s_1c_3 + c_1s_3 & c_2c_1c_3 - s_1s_3 \end{bmatrix}$$

where c_2 is $\cos\theta_2$, s_1 is $\sin\theta_1$, and so on. The corresponding quaternions are

$$q_{xyz} = [c_2 s_1 c_3 - s_2 c_1 s_3,\ c_2 s_1 s_3 + s_2 c_1 c_3,\ c_2 c_1 s_3 - s_2 s_1 c_3,\ c_2 c_1 c_3 + s_2 s_1 s_3]$$

and

$$q_{xyx} = [c_2(c_1 s_3 + s_1 c_3),\ s_2(c_1 c_3 + s_1 s_3),\ s_2(c_1 s_3 - s_1 c_3),\ c_2(c_1 c_3 - s_1 s_3)]$$

Note that the quaternion w component is given last, not first.

Conversion from Euler angles to quaternions or matrices can take advantage of common subexpressions, but is otherwise obvious. Converting a quaternion to Euler angles is easiest if we first convert to a matrix. Matrix conversion extracts sine and cosine of θ_2, then divides by the results to obtain sine and cosine of θ_1 and θ_3. When $\sin\theta_2 = 0$ (XESS) or $\cos\theta_2 = 0$ (XEDS), an alternate strategy must be used to avoid dividing by zero. In any scheme of Euler angles there are many triples that can describe the same matrix, which is a particularly bad problem when the alternate strategy is needed. The conversion routine cannot avoid this problem, but makes a conservative choice. In either situation atan2 is used to compute each angle from its sine and cosine to obtain quadrant information and good accuracy.

The HMatrix data type in the code represents 4-by-4 homogeneous transforms for right-handed rotations in right-handed coordinates applied to column vectors. If you are using row transforms (such as the Silicon Graphics GL library's Matrix type), you will need to transpose the matrix accesses. Free storage management is much easier if common data types are (multiples of) the same size, so I have chosen to use the Quat data type to hold Euler angles as well. A variety of construction and extraction macros are defined, but most users will only need the constants encoding order, such as EulOrdXYZs. For example, the following program reads ϕ, θ, and ψ and converts them to a matrix using the quantum mechanics convention $\mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_x(\psi)$ (Goldstein 1980, Appendix B). It then converts the matrix to roll, pitch, and yaw angles, and prints them.

```
/* EulerSample.c - Read angles as quantum mechanics, write as aerospace */
#include <stdio.h>
#include "EulerAngles.h"
void main(void)
{
    EulerAngles outAngs, inAngs = {0,0,0,EulOrdXYXr};
    HMatrix R;
    printf("Phi Theta Psi (radians): ");
    scanf("%f %f %f",&inAngs.x,&inAngs.y,&inAngs.z);
    Eul_ToHMatrix(inAngs, R);
    outAngs = Eul_FromHMatrix(R, EulOrdXYZs);
    printf(" Roll   Pitch  Yaw    (radians)\n");
    printf("%6.3f %6.3f %6.3f\n", outAngs.x, outAngs.y, outAngs.z);
}
```

◇ Code ◇

Headers

```
/**** QuatTypes.h - Basic type declarations ****/
#ifndef _H_QuatTypes
#define _H_QuatTypes
/*** Definitions ***/
typedef struct {float x, y, z, w;} Quat; /* Quaternion */
enum QuatPart {X, Y, Z, W};
typedef float HMatrix[4][4]; /* Right-handed, for column vectors */
typedef Quat EulerAngles;    /* (x,y,z)=ang 1,2,3, w=order code  */
#endif
/**** EOF ****/

/**** EulerAngles.h - Support for 24 angle schemes ****/
/* Ken Shoemake, 1993 */
#ifndef _H_EulerAngles
#define _H_EulerAngles
#include "QuatTypes.h"
/*** Order type constants, constructors, extractors ***/
    /* There are 24 possible conventions, designated by:    */
    /*    o EulAxI = axis used initially                    */
    /*    o EulPar = parity of axis permutation             */
    /*    o EulRep = repetition of initial axis as last     */
    /*    o EulFrm = frame from which axes are taken        */
    /* Axes I,J,K will be a permutation of X,Y,Z.           */
    /* Axis H will be either I or K, depending on EulRep.   */
    /* Frame S takes axes from initial static frame.        */
    /* If ord = (AxI=X, Par=Even, Rep=No, Frm=S), then      */
    /* {a,b,c,ord} means Rz(c)Ry(b)Rx(a), where Rz(c)v      */
    /* rotates v around Z by c radians.                     */
#define EulFrmS      0
#define EulFrmR      1
#define EulFrm(ord)  ((unsigned)(ord)&1)
#define EulRepNo     0
#define EulRepYes    1
#define EulRep(ord)  (((unsigned)(ord)>>1)&1)
#define EulParEven   0
#define EulParOdd    1
#define EulPar(ord)  (((unsigned)(ord)>>2)&1)
#define EulSafe      "\000\001\002\000"
#define EulNext      "\001\002\000\001"
#define EulAxI(ord)  ((int)(EulSafe[(((unsigned)(ord)>>3)&3)]))
#define EulAxJ(ord)  ((int)(EulNext[EulAxI(ord)+(EulPar(ord)==EulParOdd)]))
#define EulAxK(ord)  ((int)(EulNext[EulAxI(ord)+(EulPar(ord)!=EulParOdd)]))
#define EulAxH(ord)  ((EulRep(ord)==EulRepNo)?EulAxK(ord):EulAxI(ord))
    /* EulGetOrd unpacks all useful information about order simultaneously. */
#define EulGetOrd(ord,i,j,k,h,n,s,f) {unsigned o=ord;f=o&1;o>>=1;s=o&1;o>>=1;\
    n=o&1;o>>=1;i=EulSafe[o&3];j=EulNext[i+n];k=EulNext[i+1-n];h=s?k:i;}
    /* EulOrd creates an order value between 0 and 23 from 4-tuple choices. */
#define EulOrd(i,p,r,f)     (((((((i)<<1)+(p))<<1)+(r))<<1)+(f))
```

```
    /* Static axes */
#define EulOrdXYZs    EulOrd(X,EulParEven,EulRepNo,EulFrmS)
#define EulOrdXYXs    EulOrd(X,EulParEven,EulRepYes,EulFrmS)
#define EulOrdXZYs    EulOrd(X,EulParOdd,EulRepNo,EulFrmS)
#define EulOrdXZXs    EulOrd(X,EulParOdd,EulRepYes,EulFrmS)
#define EulOrdYZXs    EulOrd(Y,EulParEven,EulRepNo,EulFrmS)
#define EulOrdYZYs    EulOrd(Y,EulParEven,EulRepYes,EulFrmS)
#define EulOrdYXZs    EulOrd(Y,EulParOdd,EulRepNo,EulFrmS)
#define EulOrdYXYs    EulOrd(Y,EulParOdd,EulRepYes,EulFrmS)
#define EulOrdZXYs    EulOrd(Z,EulParEven,EulRepNo,EulFrmS)
#define EulOrdZXZs    EulOrd(Z,EulParEven,EulRepYes,EulFrmS)
#define EulOrdZYXs    EulOrd(Z,EulParOdd,EulRepNo,EulFrmS)
#define EulOrdZYZs    EulOrd(Z,EulParOdd,EulRepYes,EulFrmS)
    /* Rotating axes */
#define EulOrdZYXr    EulOrd(X,EulParEven,EulRepNo,EulFrmR)
#define EulOrdXYXr    EulOrd(X,EulParEven,EulRepYes,EulFrmR)
#define EulOrdYZXr    EulOrd(X,EulParOdd,EulRepNo,EulFrmR)
#define EulOrdXZXr    EulOrd(X,EulParOdd,EulRepYes,EulFrmR)
#define EulOrdXZYr    EulOrd(Y,EulParEven,EulRepNo,EulFrmR)
#define EulOrdYZYr    EulOrd(Y,EulParEven,EulRepYes,EulFrmR)
#define EulOrdZXYr    EulOrd(Y,EulParOdd,EulRepNo,EulFrmR)
#define EulOrdYXYr    EulOrd(Y,EulParOdd,EulRepYes,EulFrmR)
#define EulOrdYXZr    EulOrd(Z,EulParEven,EulRepNo,EulFrmR)
#define EulOrdZXZr    EulOrd(Z,EulParEven,EulRepYes,EulFrmR)
#define EulOrdXYZr    EulOrd(Z,EulParOdd,EulRepNo,EulFrmR)
#define EulOrdZYZr    EulOrd(Z,EulParOdd,EulRepYes,EulFrmR)

EulerAngles Eul_(float ai, float aj, float ah, int order);
Quat Eul_ToQuat(EulerAngles ea);
void Eul_ToHMatrix(EulerAngles ea, HMatrix M);
EulerAngles Eul_FromHMatrix(HMatrix M, int order);
EulerAngles Eul_FromQuat(Quat q, int order);
#endif
/**** EOF ****/
```

Routines

```
/**** EulerAngles.c - Convert Euler angles to/from matrix or quat ****/
/* Ken Shoemake, 1993 */
#include <math.h>
#include <float.h>
#include "EulerAngles.h"

EulerAngles Eul_(float ai, float aj, float ah, int order)
{
    EulerAngles ea;
    ea.x = ai; ea.y = aj; ea.z = ah;
    ea.w = order;
    return (ea);
}
```

```
/* Construct quaternion from Euler angles (in radians). */
Quat Eul_ToQuat(EulerAngles ea)
{
    Quat qu;
    double a[3], ti, tj, th, ci, cj, ch, si, sj, sh, cc, cs, sc, ss;
    int i,j,k,h,n,s,f;
    EulGetOrd(ea.w,i,j,k,h,n,s,f);
    if (f==EulFrmR) {float t = ea.x; ea.x = ea.z; ea.z = t;}
    if (n==EulParOdd) ea.y = -ea.y;
    ti = ea.x*0.5; tj = ea.y*0.5; th = ea.z*0.5;
    ci = cos(ti);  cj = cos(tj);  ch = cos(th);
    si = sin(ti);  sj = sin(tj);  sh = sin(th);
    cc = ci*ch; cs = ci*sh; sc = si*ch; ss = si*sh;
    if (s==EulRepYes) {
        a[i] = cj*(cs + sc);    /* Could speed up with */
        a[j] = sj*(cc + ss);    /* trig identities. */
        a[k] = sj*(cs - sc);
        qu.w = cj*(cc - ss);
    } else {
        a[i] = cj*sc - sj*cs;
        a[j] = cj*ss + sj*cc;
        a[k] = cj*cs - sj*sc;
        qu.w = cj*cc + sj*ss;
    }
    if (n==EulParOdd) a[j] = -a[j];
    qu.x = a[X]; qu.y = a[Y]; qu.z = a[Z];
    return (qu);
}

/* Construct matrix from Euler angles (in radians). */
void Eul_ToHMatrix(EulerAngles ea, HMatrix M)
{
    double ti, tj, th, ci, cj, ch, si, sj, sh, cc, cs, sc, ss;
    int i,j,k,h,n,s,f;
    EulGetOrd(ea.w,i,j,k,h,n,s,f);
    if (f==EulFrmR) {float t = ea.x; ea.x = ea.z; ea.z = t;}
    if (n==EulParOdd) {ea.x = -ea.x; ea.y = -ea.y; ea.z = -ea.z;}
    ti = ea.x;    tj = ea.y;    th = ea.z;
    ci = cos(ti); cj = cos(tj); ch = cos(th);
    si = sin(ti); sj = sin(tj); sh = sin(th);
    cc = ci*ch; cs = ci*sh; sc = si*ch; ss = si*sh;
    if (s==EulRepYes) {
        M[i][i] = cj;     M[i][j] =  sj*si;    M[i][k] =  sj*ci;
        M[j][i] = sj*sh;  M[j][j] = -cj*ss+cc; M[j][k] = -cj*cs-sc;
        M[k][i] = -sj*ch; M[k][j] =  cj*sc+cs; M[k][k] =  cj*cc-ss;
    } else {
        M[i][i] = cj*ch; M[i][j] = sj*sc-cs; M[i][k] = sj*cc+ss;
        M[j][i] = cj*sh; M[j][j] = sj*ss+cc; M[j][k] = sj*cs-sc;
        M[k][i] = -sj;   M[k][j] = cj*si;    M[k][k] = cj*ci;
    }
    M[W][X]=M[W][Y]=M[W][Z]=M[X][W]=M[Y][W]=M[Z][W]=0.0; M[W][W]=1.0;
}
```

```
/* Convert matrix to Euler angles (in radians). */
EulerAngles Eul_FromHMatrix(HMatrix M, int order)
{
    EulerAngles ea;
    int i,j,k,h,n,s,f;
    EulGetOrd(order,i,j,k,h,n,s,f);
    if (s==EulRepYes) {
        double sy = sqrt(M[i][j]*M[i][j] + M[i][k]*M[i][k]);
        if (sy > 16*FLT_EPSILON) {
            ea.x = atan2(M[i][j], M[i][k]);
            ea.y = atan2(sy, M[i][i]);
            ea.z = atan2(M[j][i], -M[k][i]);
        } else {
            ea.x = atan2(-M[j][k], M[j][j]);
            ea.y = atan2(sy, M[i][i]);
            ea.z = 0;
        }
    } else {
        double cy = sqrt(M[i][i]*M[i][i] + M[j][i]*M[j][i]);
        if (cy > 16*FLT_EPSILON) {
            ea.x = atan2(M[k][j], M[k][k]);
            ea.y = atan2(-M[k][i], cy);
            ea.z = atan2(M[j][i], M[i][i]);
        } else {
            ea.x = atan2(-M[j][k], M[j][j]);
            ea.y = atan2(-M[k][i], cy);
            ea.z = 0;
        }
    }
    if (n==EulParOdd) {ea.x = -ea.x; ea.y = - ea.y; ea.z = -ea.z;}
    if (f==EulFrmR) {float t = ea.x; ea.x = ea.z; ea.z = t;}
    ea.w = order;
    return (ea);
}

/* Convert quaternion to Euler angles (in radians). */
EulerAngles Eul_FromQuat(Quat q, int order)
{
    HMatrix M;
    double Nq = q.x*q.x+q.y*q.y+q.z*q.z+q.w*q.w;
    double s = (Nq > 0.0) ? (2.0 / Nq) : 0.0;
    double xs = q.x*s,    ys = q.y*s,    zs = q.z*s;
    double wx = q.w*xs,   wy = q.w*ys,   wz = q.w*zs;
    double xx = q.x*xs,   xy = q.x*ys,   xz = q.x*zs;
    double yy = q.y*ys,   yz = q.y*zs,   zz = q.z*zs;
    M[X][X] = 1.0 - (yy + zz); M[X][Y] = xy - wz; M[X][Z] = xz + wy;
    M[Y][X] = xy + wz; M[Y][Y] = 1.0 - (xx + zz); M[Y][Z] = yz - wx;
    M[Z][X] = xz - wy; M[Z][Y] = yz + wx; M[Z][Z] = 1.0 - (xx + yy);
    M[W][X]=M[W][Y]=M[W][Z]=M[X][W]=M[Y][W]=M[Z][W]=0.0; M[W][W]=1.0;
    return (Eul_FromHMatrix(M, order));
}
/**** EOF ****/
```

◇ Bibliography ◇

(Craig 1989) John J. Craig. *Introduction to Robotics: Mechanics and Control*, 2nd edition. Addison-Wesley, Reading, MA, 1989.

(Foley *et al.* 1990) James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*, 2nd edition. Addison-Wesley, Reading, MA, 1990.

(Goldstein 1980) Herbert Goldstein. *Classical Mechanics*, 2nd edition. Addison-Wesley, Reading, MA, 1980.