Gernot Hoffmann

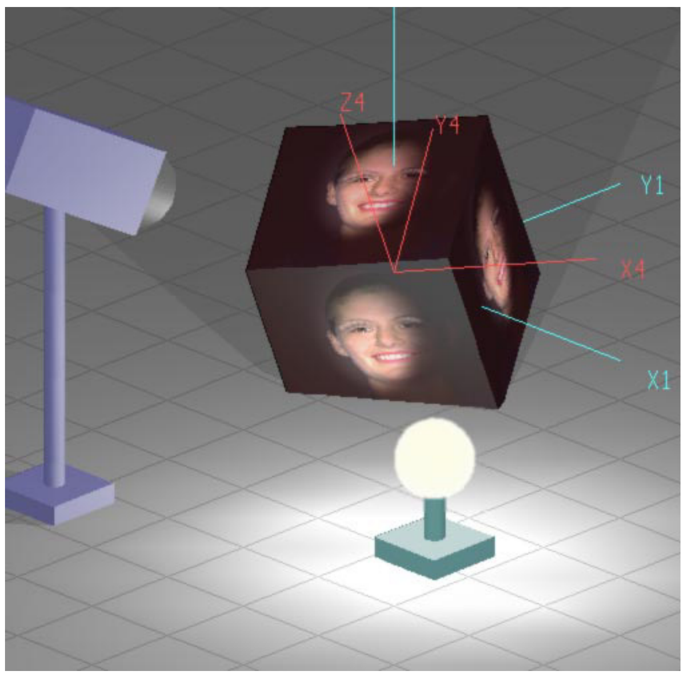
Euler Angles and Projections

Contents

Euler Angles

Coordinate Rotations

A rotational coordinate transformation delivers different column matrices for the same vector **x** .
Rotation about the x-axis from CS_1 to CS_2 : $\mathbf{x}_2 = \mathbf{X}_{21}\,\mathbf{x}_1$
Rotation about the y-axis from CS_1 to CS_2 : $\mathbf{x}_2 = \mathbf{Y}_{21}\,\mathbf{x}_1$
Rotation about the z-axis from CS_1 to CS_2 : $\mathbf{x}_2 = \mathbf{Z}_{21}\,\mathbf{x}_1$



$$\mathbf{X}_{21} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

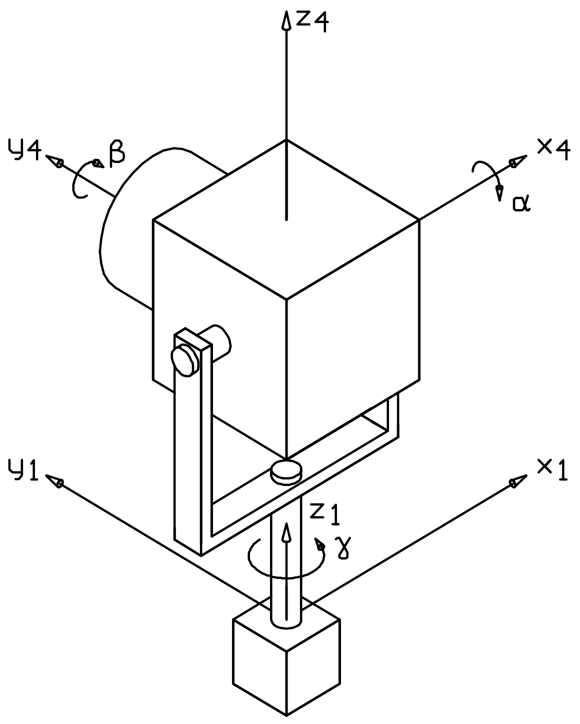
$$\mathbf{Y}_{21} = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix}$$

$$\mathbf{Z}_{21} = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The rotation about the y-axis has a different sign pattern. Compound matrix rotations about three axes depend on the sequence.

Camera Angles

The optical axis for cameras, may be real or fictitious for computer graphics, is here always aligned with the y-axis.
It does not seem natural to use the z-axis, because the view of a camera is nearly never vertical.
The image appears then in the z, x-plane.
The sequence is



$$\mathbf{x}_4 = \mathbf{Y}_{43}\,\mathbf{X}_{32}\,\mathbf{Z}_{21}\,\mathbf{x}_1 = \mathbf{C}_{41}\,\mathbf{x}_1 \,.$$

Turn first about the azimuth axis z by γ , then about the elevation axis x by α and finally about the roll axis y by β . For computer graphics we use $\beta = 0$ and for real cameras $\beta \approx 0$ is a parameter of the camera error model.

For this sequence we find the matrix $\mathbf{C}_{41}$:

$$\mathbf{C}_{41} = \begin{bmatrix} \cos(\beta)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma) & \cos(\beta)\sin(\gamma) + \sin(\alpha)\sin(\beta)\cos(\gamma) & -\cos(\alpha)\sin(\beta) \\ -\cos(\alpha)\sin(\gamma) & \cos(\alpha)\cos(\gamma) & \sin(\alpha) \\ \sin(\beta)\cos(\gamma) + \sin(\alpha)\cos(\beta)\sin(\gamma) & \sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\beta)\cos(\gamma) & \cos(\alpha)\cos(\beta) \end{bmatrix}$$

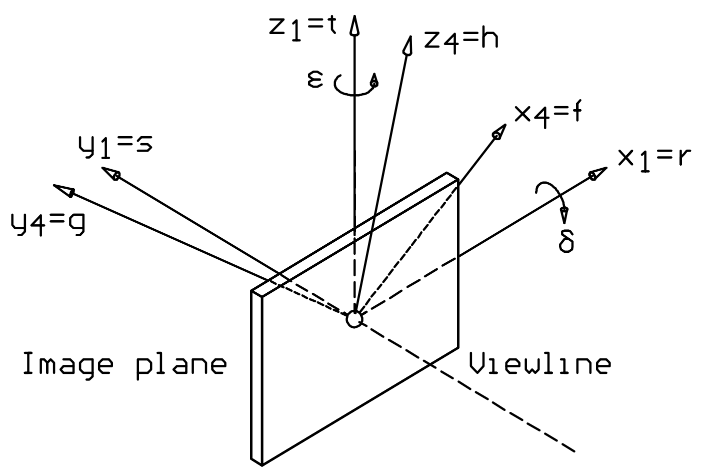
The case of $\mathbf{C}_{41}$ with $\beta = 0$ is given by $\mathbf{D}_{41}$:

$$\mathbf{D}_{41} = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 \\ -\cos(\alpha)\sin(\gamma) & \cos(\alpha)\cos(\gamma) & \sin(\alpha) \\ \sin(\alpha)\sin(\gamma) & -\sin(\alpha)\cos(\gamma) & \cos(\alpha) \end{bmatrix}$$

Tilted Image Angles

For the projection on tilted image planes we need additionally a sequence, where the rotation of the camera can be compensated by a rotation of the image plane deviating from the orthogonal orientation. This is necessary for special purposes in computer graphics, e.g. the rectification of verticals, and also for the error model of CCD-cameras.
Therefore **Z** and **X** are swapped:

$$\mathbf{x}_4 = \mathbf{Y}_{43}\,\mathbf{Z}_{32}\,\mathbf{X}_{21}\,\mathbf{x}_1 = \mathbf{T}_{41}\,\mathbf{x}_1 .$$



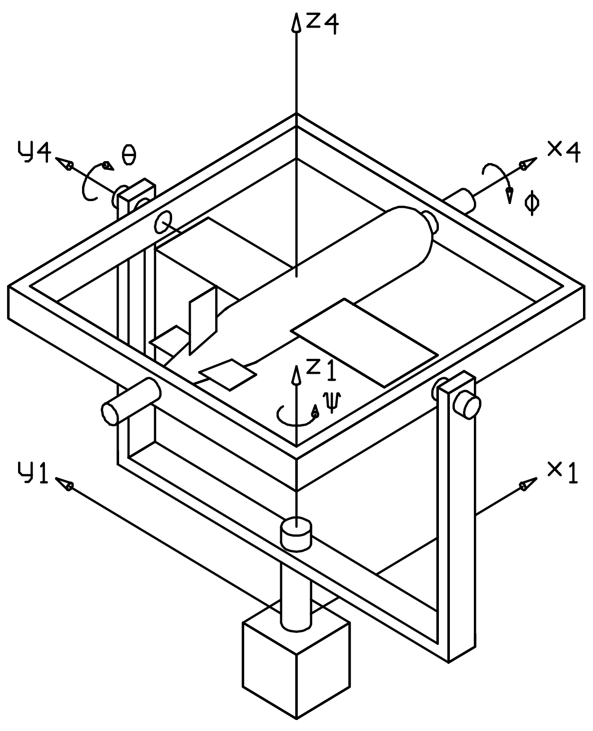
Turn first about the x-axis by δ and then about the z-axis by ε . The third rotation is actually not needed and can be replaced later by β in $\mathbf{C}_{41}$. Set $\mathbf{Y}_{43} = \mathbf{I}$.

$$\mathbf{T}_{41} = \begin{bmatrix} \cos(\varepsilon) & \cos(\delta)\sin(\varepsilon) & \sin(\delta)\sin(\varepsilon) \\ -\sin(\varepsilon) & \cos(\delta)\cos(\varepsilon) & \sin(\delta)\cos(\varepsilon) \\ 0 & -\sin(\delta) & \cos(\delta) \end{bmatrix}$$

Aircraft Angles

The next sequence is used for aircraft and other vehicles like cars:

$$\mathbf{x}_4 = \mathbf{X}_{43}\,\mathbf{Y}_{32}\,\mathbf{Z}_{21}\,\mathbf{x}_1 = \mathbf{A}_{41}\,\mathbf{x}_1 .$$



Rotate by the yaw angle ψ about the z-axis, by the pitch angle θ about the y-axis and the roll angle ϕ about the x-axis.

Here we have a minor problem: in flight mechanics, the nose of the plane is in x-direction, the right wing in y-direction and z points downwards (German standard LN9300).
For general applications the y-axis points to the left and the z-axis quite naturally upwards, but nose down is now a positive pitch.

Nevertheless the same Matrix $\mathbf{A}_{41}$ is valid for both cases, provided the axes of CS_1 and CS_4 are in the same orientation. Note: positive angles are always in right screw direction.

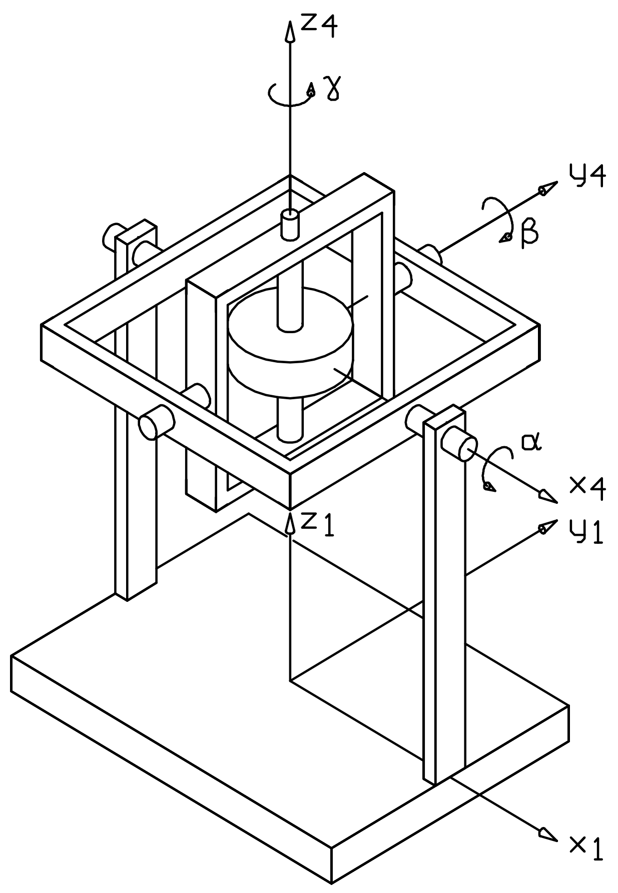
$$\mathbf{A}_{41} = \begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ -\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & \cos(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & \sin(\phi)\cos(\theta) \\ \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) & -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

Gyro Angles

In gyro instruments the last angle is assigned to the rotation of the flywheel.

$\mathbf{x}_4 = \mathbf{Z}_{43}\, \mathbf{Y}_{32}\, \mathbf{X}_{21}\, \mathbf{x}_1 = \mathbf{G}_{41}\, \mathbf{x}_1$

The first two angles: α about the x-axis and β about the y-axis. The third degree of freedom γ about the z-axis belongs to the rotation of the gyro flywheel, whereas the first two angles are fixed to the cardan frames.
The figure shows as usual the situation for zero angles. For non zero angles the axes of rotation are not orthogonal to each other.



$$\mathbf{G}_{41} = \begin{bmatrix} \cos(\beta)\cos(\gamma) & \cos(\alpha)\sin(\gamma) + \sin(\alpha)\sin(\beta)\cos(\gamma) & \sin(\alpha)\sin(\gamma) - \cos(\alpha)\sin(\beta)\cos(\gamma) \\ \cos(\beta)\sin(\gamma) & \cos(\alpha)\cos(\gamma) - \sin(\alpha)\sin(\beta)\sin(\gamma) & \sin(\alpha)\cos(\gamma) + \cos(\alpha)\sin(\beta)\sin(\gamma) \\ \sin(\beta) & -\sin(\alpha)\cos(\beta) & \cos(\alpha)\cos(\beta) \end{bmatrix}$$

Single Axis 3D-Rotation

Once an orthonormal rotation matrix is given, e.g. the Aircraft Matrix $\mathbf{A} = (a_{ik})$ with numbers only, the rotation can be described as a single axis rotation about an axis $\mathbf{n}$ with the angle η :

$$\eta = \arccos[\,0.5\cdot(a_{11} + a_{22} + a_{33} - 1)\,]$$

$$\mathbf{n} = [\,0.5/\sin(\eta)\,]\,(a_{32} - a_{23},\, a_{13} - a_{31},\, a_{21} - a_{12})^T$$

Equations found in : J. Hoscheck + D. Lasser:
Grundlagen der geometrischen Datenverarbeitung, B.G.Teubner Stuttgart, 1992

Rotation matrices have one eigenvalue $\lambda = 1$. The rotation axis $\mathbf{n}$ is the normalized eigenvector.
By $\mathbf{A\,x} = \lambda\,\mathbf{x}$ we get $\mathbf{A\,x} = \mathbf{x}$, $\mathbf{x} = \mathbf{A}^{-1}\,\mathbf{x} = \mathbf{A}^T\mathbf{x}$. Then we find $(\,\mathbf{A} - \mathbf{A}^T)\,\mathbf{x} = \mathbf{0}$. Now it can be shown easily, that $\mathbf{n} = (a_{32} - a_{23},\, a_{13} - a_{31},\, a_{21} - a_{12})^T$ is an eigenvector. The normalization could be done without $\sin(\eta)$.
The equation for η is a result of the invariance of the sum of the diagonal elements with respect to similarity transformations of the type $\mathbf{N}^T\mathbf{A\,N} = \mathbf{Z}$, using a $\mathbf{Z}$ rotation as mentioned in the first chapter .

Matrix Features

Single and compound rotation matrices are orthonormal: $\mathbf{C}^{-1} = \mathbf{C}^T$. Therefore it is not necessary to write down the inverse rotational transformation like $\mathbf{x}_1 = \mathbf{C}_{14}\,\mathbf{x}_4 = \mathbf{C}_{41}{}^{-1}\,\mathbf{x}_4 = \mathbf{C}_{41}{}^T\,\mathbf{x}_4$ explicitly, but it is worth to mention, that swapping the indices means the same as transposing the matrix.

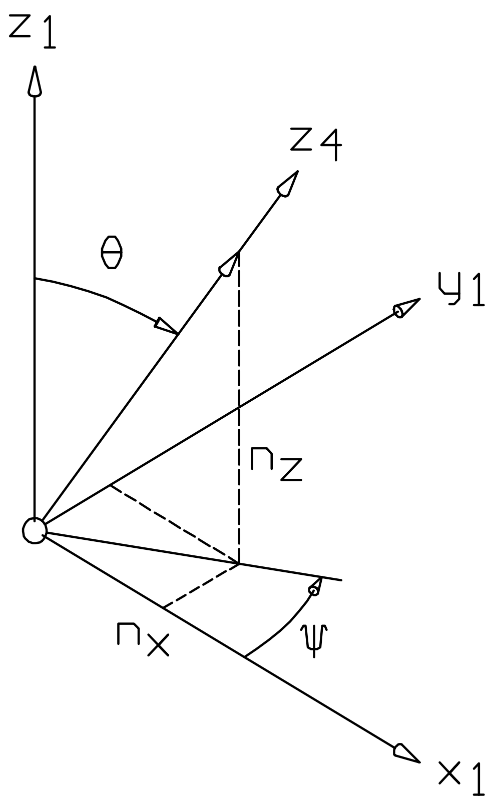
Linearized matrices are possible, if the angles are small. For example in the gyro matrix $\mathbf{G}_{41}$ we can sometimes assume $\sin(\alpha) \approx \alpha$, $\cos(\alpha) \approx 1$, $\sin(\beta) \approx \beta$, $\cos(\beta) \approx 1$. Furtheron products of small angles must be neglectable: $\alpha\,\beta \approx 0$. Products of completely linearized matrices are commutative.
Linearized matrices are not formalistically orthonormal, but this characteristic can be achieved by neglecting products of small angles.

Aligning a Body Axis

Sometimes a body coordinate system has to be aligned to a normal vector $\mathbf{n}=(n_x,n_y,n_z)^T$. Only two angles can be aligned, the third is free.
If we use the **A**-sequence, then a vehicle is properly aligned to a hill road.
Turn first the yaw angle into the desired azimuth direction and then the pitch angle in order to put the car´s z_4-axis along the normal direction.

```
eps = 1e-16;
r = Sqrt(nx·nx+ny·ny); the=0; psi=0;
phi=0;
If r>eps Then
Begin
   Atangens(ny,nx,psi,flag);
   Atangens(r, nz,the,flag);
End Else If nz<0 Then the=pi;
```

Atangens (atan2 in C/C++) is explained on the last page.



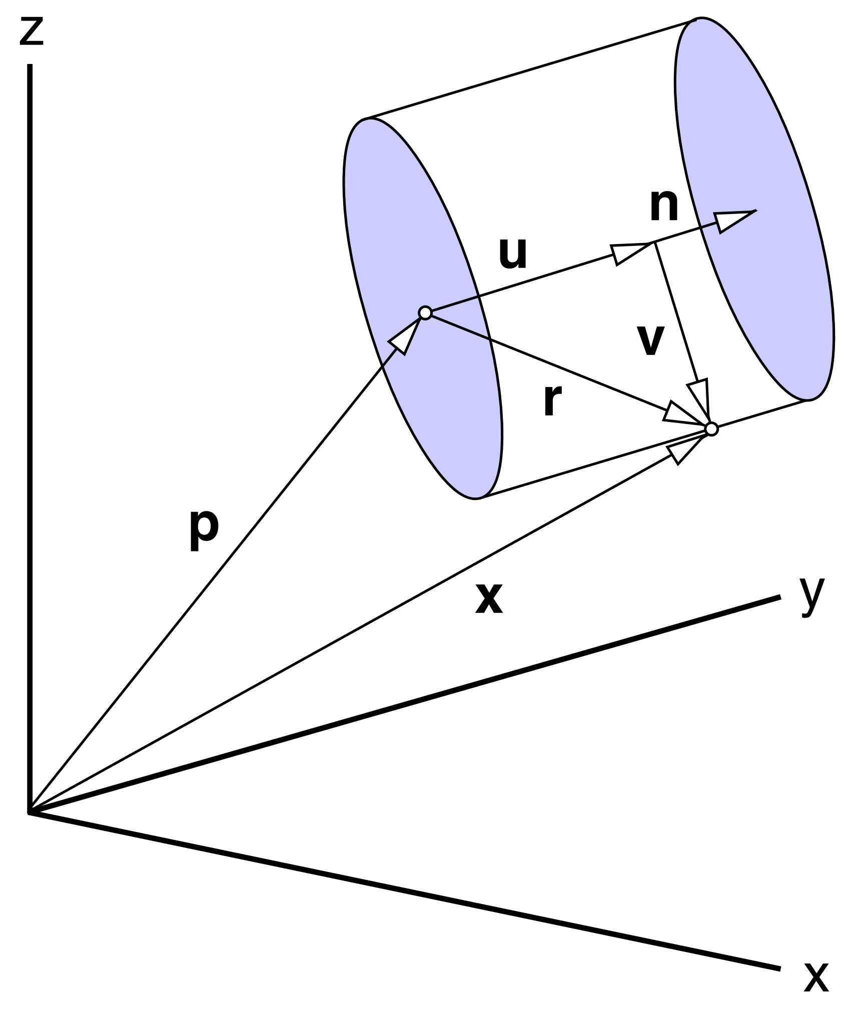
Body Rotation about an Axis

$\mathbf{x}_1$	Object point before rotation
$\mathbf{x}_2$	Object point after rotation
p	Reference point of axis
n	Direction vector, normalized
η	Rotation angle (right screw positive)
u	Projection of **r** on **n**
v	Orthogonal component of **r**
a,**b**	Orthogonal vector base, each length $\lvert\mathbf{v}\rvert$

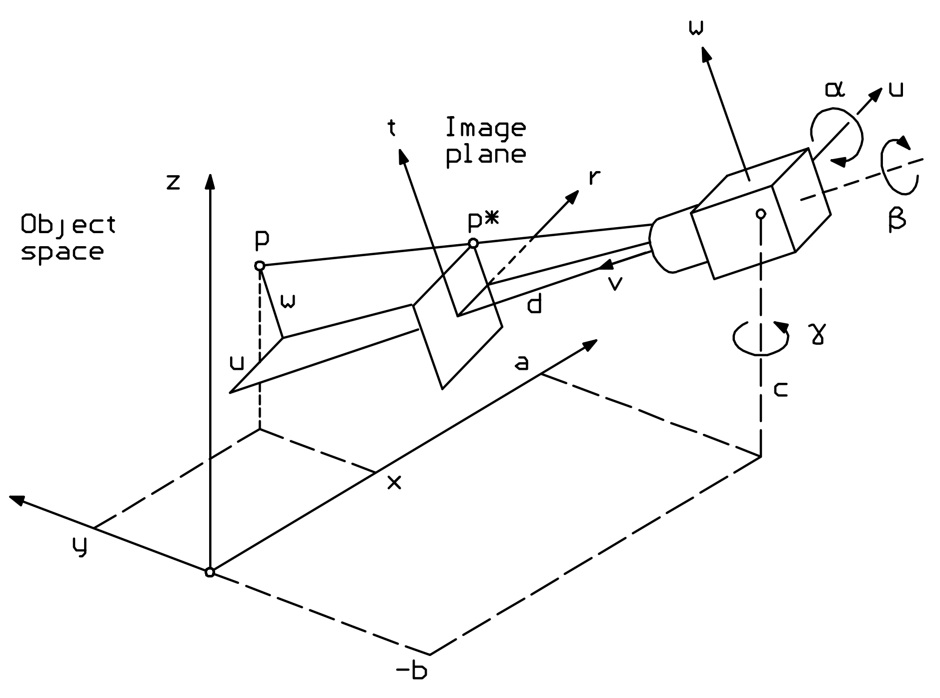
$$\mathbf{r} = \mathbf{x}_1 - \mathbf{p}$$
$$\mathbf{u} = (\mathbf{r}^T\mathbf{n})\mathbf{n}$$
$$\mathbf{v} = \mathbf{r} - \mathbf{u}$$
$$\mathbf{a} = \mathbf{v}$$
$$\mathbf{b} = \mathbf{n} \times \mathbf{a}$$
$$\mathbf{x}_2 = \mathbf{p} + \mathbf{u} + \mathbf{a}\cos(\eta) + \mathbf{b}\sin(\eta)$$

More about Object Rotations can be found here:

http://docs-hoffmann.de/rotate09072002.pdf



Orthogonal Projection



A camera is positioned in **a** = (a, b, c)^T in the object space **x**=(x, y, z)^T and rotated by α, β, γ, using the **C**-sequence with **C** = **C**$_{41}$ or **D**$_{41}$ for β = 0 .
In computer graphics the camera is focussed to the viewpoint. The angles are defined by the camera position **a** and the viewpoint **a**$_v$ = (a$_v$, b$_v$, c$_v$)^T.
The image plane or viewplane is centered in the viewpoint and orthogonal to the viewline.
Objects near to the viewplane are mapped by scale-factor one to the viewplane.
Once the image is created on the viewplane, it can be mapped to any viewport by a workstation transformation.
This means: We put a frame on the viewplane and the content is shown in the viewport.
The word *frame* is used instead of *window*. In GKS and PHIGS a *window* cuts a part of the real word, here the specific viewplane content. We use the word *window* for a viewport on a monitor.
The two angles α and γ can be calculated by the procedure **Atangens** on the last page.
The signs are relevant. Furtheron, d is the distance of the viewpoint from the center of projection.

$$\tan(g) = \frac{a - a_v}{-(b - b_v)}$$

$$\tan(a) = \frac{-(c - c_v)}{\sqrt{(a - a_v)^2 + (b - b_v)^2}}$$

$$d = \sqrt{(a - a_v)^2 + (b - b_v)^2 + (c - c_v)^2}$$

The camera itself is represented by coordinates **u** = (u, v, w)^T. The view plane r, t is part of the „image space“ **r** = (r, s, t)^T, which is for formalistical reasons sometimes defined in three dimensions with s = 0. The projection of an object point **x** = (x, y, z)^T onto the orthogonal viewplane is given by this set of equations:

$$\mathbf{u} = \mathbf{C}\,(\mathbf{x} - \mathbf{a})$$

$$r = d\,u / v$$
$$t = d\,w / v$$

For a parallel projection we have simply r = u and t = w .

A completely linear implicite formulation is found by using the matrix **A** :

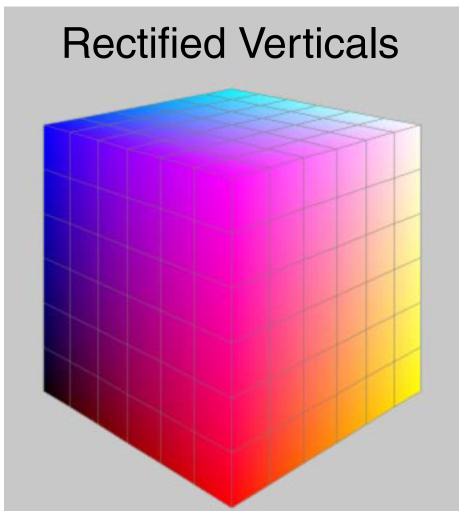
$$\mathbf{A} = \begin{bmatrix} -1 & r/d & 0 \\ 0 & t/d & -1 \end{bmatrix}$$

$$\mathbf{u} = \mathbf{C}\,(\mathbf{x} - \mathbf{a})$$

$$\mathbf{A}\,\mathbf{u} = \mathbf{0}$$

Non-Orthogonal Projection

For rectified verticals or undistorted front planes we have to tilt the viewplane. This is the same as tilting the film plane in a studio camera (e.g. SINAR). It is not simply a rotational transformation, but the rotation matrix $\mathbf{T}_{41}$ (page 2) is essential. $\mathbf{f} = (f, g, h)^T$ ist the new tilted viewplane with $g=0$..



$$D = 1 - (r/d)\cdot\tan(\varepsilon)/\cos(\delta) + (t/d)\cdot\tan(\delta)$$

$$f = [\ r/\cos(\varepsilon)\ ] / D$$

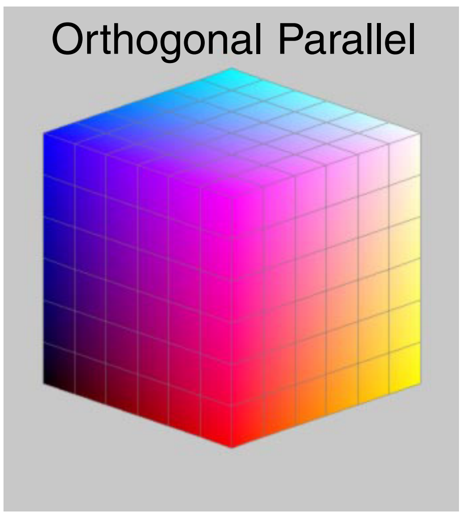
$$h = [\ t/\cos(\delta) - r\cdot\tan(\varepsilon)\cdot\tan(\delta)\ ] / D$$

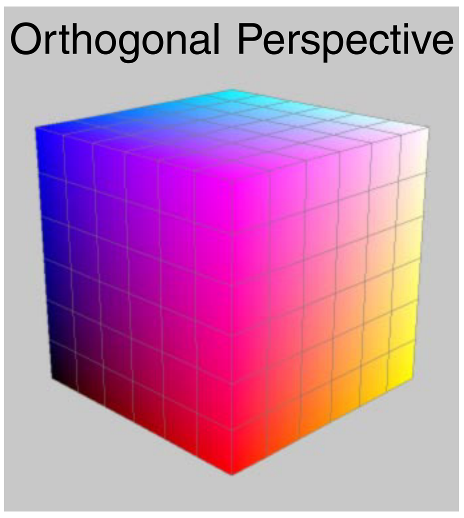
Inversion:

$$E = 1 + (f/d)\cdot\cos(\delta)\cdot\sin(\varepsilon) - (h/d)\cdot\sin(\delta)$$

$$r = [\ f\cdot\cos(\varepsilon)\ ] / E$$

$$t = [\ h\cdot\cos(\delta) + f\cdot\sin(\varepsilon)\cdot\sin(\delta)\ ] / E$$





Projective Mapping

A point $\mathbf{x} = (x, y, z)^T$ in 3D or $\mathbf{x} = (x, y)^T$ in 2D can be mapped to $\mathbf{f} = (f, h)^T$ by a general algorithm without any fictitious camera. Typical applications:
Map cube to image (drawing perspective, parallel, isometric, cabinet, and so on)
Map one quadriliteral to another (photogrammetric rectification for maps, for house façades, and so on)

$$D = [\ 1 + \mathbf{c}^T\mathbf{x}\ ]$$

$$f = [\ a_o + \mathbf{a}^T\mathbf{x}\ ] / D$$

$$h = [\ b_o + \mathbf{b}^T\mathbf{x}\ ] / D$$

The unknown parameters $\mathbf{a}, \mathbf{b}, \mathbf{c}, a_o, b_o$ can be determined by $q=4$ points $\mathbf{x}_i, \mathbf{f}_i$ for the 2D source or by $q=6$ points $\mathbf{x}_i, \mathbf{f}_i$ for the 3D source. Of course the points shouldn´t be collinear.
Multiply both sides by the denominator and rearrange.

$$\mathbf{a}^T\mathbf{x}_i + \mathbf{0}^T\mathbf{x}_i - f_i\,\mathbf{c}^T\mathbf{x}_i + a_o + 0 = f_i$$

$$\mathbf{0}^T\mathbf{x}_i + \mathbf{b}^T\mathbf{x}_i - h_i\,\mathbf{c}^T\mathbf{x}_i + 0 + b_o = h_i$$

Once rearranged in a big matrix $\mathbf{M}$, this is a linear equation system for the $p=8$ unknowns (2D source) or the $p=11$ unknowns (3D source) $\mathbf{p} = (\mathbf{a}^T, \mathbf{b}^T, \mathbf{c}^T, a_o, b_o)^T$ and the right side $\mathbf{q} = (f_1, h_1, ..., f_q, h_q)^T$.
It can be solved by the Gauss Transformation and Cholesky, because the final matrix is symmetric.

$$\mathbf{M}\,\mathbf{p} = \mathbf{q}$$

$$\mathbf{M}^T\mathbf{M}\,\mathbf{p} = \mathbf{M}^T\mathbf{q}$$

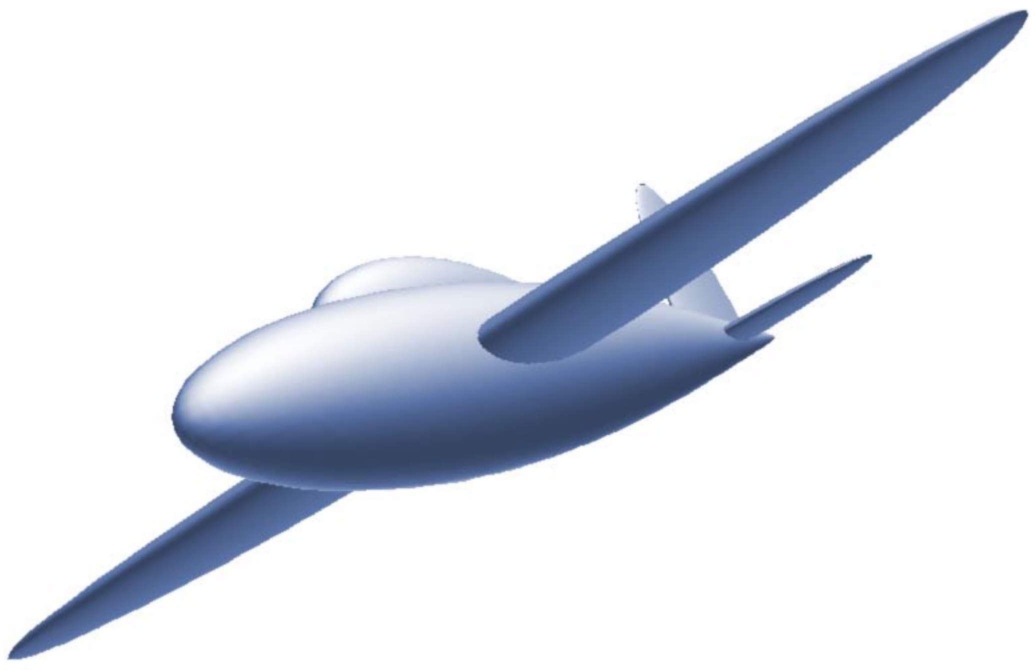
Phi = Arctan(n/d)

```
All Variables can be defined as Double instead of Single=Float

Function Atan2(n,d: Single): Single;
{   f/rad=arctan(n/d); any n,d; angle +-pi;  >=387 only;    }
{   This is the same as atan2 in C/C++
}
Assembler;
ASM FLD n; FLD d; FPATAN;
END;

Procedure Atangens (n,d: Single; Var phi: Single; Var flag: Integer);
{   phi/rad=arctan(n/d); xy-coord: n=y,d=x;  >=387 only
    flag=0: OK, flag=1: no solution                                }
Const eps = 1E-16;
Begin flag:=1; phi:=0;
 If (Abs(n)>eps) Or (Abs(d)>eps) Then
 Begin flag:=0; phi:=Atan2(n,d);
 End;
End;

Procedure Atangens (n,d: Single; Var phi: Single; Var flag: Integer);
{   phi/rad=arctan(n/d); xy-Koord.: n=y,d=x;
    flag=0: OK, flag=1: no solution
    Requires only a standard function arctan(z) for two quadrants   }
Const eps = 1E-16;
Begin
 flag:=1; phi:=0;
 If (abs(n)>eps) Or (abs(d)>eps) Then
 Begin
  flag:=0; If abs(d)>=abs(n) Then
  Begin     phi:=ArcTan( n/d);  If d <0 Then phi:=phi+pi;
  End Else
  Begin     phi:=ArcTan(-d/n);  If n>=0 Then phi:=phi+0.5*pi Else phi:=phi+1.5*p
i;
  End;
 End;
End;

Procedure Acosinus (x: Single; Var phi: Single; Var Flag: Integer);
{ phi/rad=arccos(x)
  flag=0: OK, flag=1: no solution                               }
Var x2: Single;
Begin flag:=1; phi:=0; x2:=sqr(x);
 If x2<=1 Then
 Begin flag:=0; Atangens (sqrt(1-x2),x,phi,flag);
 End;
End;

Procedure XPowerA (x,a: Single; Var y: Single; Var flag: Integer);
{ y=x^a for x>0 ; flag=0: OK, flag=1: no solution                   }
Const eps=1E-16;
Begin flag:=0;
```

Angle between two Vectors

```
Function Angle2 (A,B: XYZ): Single;
{ Angle 0..pi between vectors A and B    }
{ Requires Norm(A)>0 and Norm(B)>0       }
{ tan(a) = Norm(AxB)/(A.B)                    }
{ Don´t use y = arccos(x) !              }
Var  n,d: Single;
Begin
With A Do n:=Sqr(y*B.z-z*B.y)+Sqr(z*B.x-x*B.z)+Sqr(x*B.y-y*B.x);
With A Do d:=x*B.x+y*B.y+z*B.z;
Angle2:=atan2(Sqrt(n),d);
End;
```

Aircraft Simulation (1)



In fact it is only the simulation of a rigid body which rotates like an aircraft. Several features have to be considered:

Gimbal lock is a singularity which happens for the A-Sequence in positions with $\cos(\theta)=0$ and in the vicinity, e.g. during vertical climb or descent.
This problem is solved by introducing a second Euler angle sequence:

$\mathbf{x}_4 = \mathbf{Y}_{43}\, \mathbf{X}_{32}\, \mathbf{Z}_{21}\, \mathbf{x}_1 = \mathbf{B}_{41}\, \mathbf{x}_1$

Rotate by the yaw angle ψ about by the z-axis, by the roll angle Φ about the x-axis and the pitch angle θ about the y-axis. These angles differ from the A-Sequence, even for the yaw angle, but they can be mutually converted. The singularity is at $\cos(\Phi)=0$.

The A-Sequence and the B-Sequence are toggled appropriately at $\mathrm{Abs}(\sin(\theta)) = 0.7$.
If the A-Sequence is actually used for the integration, then the angles for the B-Sequence are calculated.
Instead of four quadrant z=Atan2(y/x), a Newton iteration z=Atan3(y/x) is used, where the initial value is the previous value. This should guarantee consistent angle descriptions without jumps.
If the B-Sequence is used for the integration, then the angles for the A-Sequence are calculated in the same similarly.

The matrix for the derivatives of the Euler angles as functions of the body fixed angular velocities has to be calculated for both sequences. The results can be taken from the source code.

The equations of motion are simplified. There are no products of inertia, the coordinates are in principal axes directions. No aerodynamic torques are introduced. The yaw rate depends on the roll angle, this looks plausible.

The control systems has simple rate dampers for all axes. The attitude controller can be operated in two modes: Pitch and roll or pitch and yaw. In the yaw mode, the roll angle depends on the heading error.
The motion looks rather natural.

Essential tests are made by applying torques about the body fixed axes. For one keystroke a torque is applied for one integration step. It can be shown, that the reaction happens always only about the respective axis, a good test for the correct definitions of the Euler angle mathematics.

The differential equations are integrated by 'False Euler' . The results for the angular velocity from the first step are immediately used as inputs for the integration of the Euler Angles. Standard Euler would use always old values on the right side.

Aircraft Simulation (2)

```
Function ATan2(n,d:Double): Double;
{    f/rad=arctan(n/d); any n,d; f=-pi..+pi; >=387 only;  }
Assembler;
ASM
 FLD n; FLD d; FPATAN;
END;

Procedure ATan3(n,d: Double; Var a: Double);
{    Newton Iteration }
Var si,co: Double;
Begin
 SicCoc(a,si,co);
 a:=a-(d*si-n*co)/(d*co+n*si);
 SicCoc(a,si,co);
 a:=a-(d*si-n*co)/(d*co+n*si);
 SicCoc(a,si,co);
 a:=a-(d*si-n*co)/(d*co+n*si);
End;

Procedure MatObj3Da;
{    Transposed A41 for object rotation }
Begin
SicCoc(Ph1,sPh1,cPh1);
SicCoc(Th1,sTh1,cTh1);
SicCoc(Ps1,sPs1,cPs1);
o11:= cTh1*cPs1;    o12:=-cPh1*sPs1+sPh1*sTh1*cPs1;    o13:= sPh1*sPs1+cPh1*sTh1*cPs1;
o21:= cTh1*sPs1;    o22:= cPh1*cPs1+sPh1*sTh1*sPs1;    o23:=-sPh1*cPs1+cPh1*sTh1*sPs1;
o31:=-sTh1;         o32:= sPh1*cTh1;                   o33:= cPh1*cTh1;
End;

Procedure MatObj3Db;
{    Transposed B41 for object rotation }
Begin
SicCoc(Ph2,sPh2,cPh2);
SicCoc(Th2,sTh2,cTh2);
SicCoc(Ps2,sPs2,cPs2);
o11:= cTh2*cPs2-sPh2*sTh2*sPs2;    o12:=-cPh2*sPs2;    o13:= sTh2*cPs2+sPh2*cTh2*sPs2;
o21:= cTh2*sPs2+sPh2*sTh2*cPs2;    o22:= cPh2*cPs2;    o23:= sTh2*sPs2-sPh2*cTh2*cPs2;
o31:=-cPh2*sTh2;                   o32:= sPh2;         o33:= cPh2*cTh2;
End;

Procedure Integ;
{    Rotational differential equations for a rigid body, like an aircraft
     Simplified attitude Controller }
Var       icTh1,icPh2: Double;
Const     Jx=0.6;   Jy=1;     Jz=1.5;
          dampP=2;  conPhi= 4;
          dampQ=3;  conThe=10;
          dampR=2;  conPsi= 1;
          ratePsi=1;
               dT=0.1;
Begin
{    Toggle Euler Angle Sequences if necessary }
EModeA:=Abs(sTh1)<0.7;
{    Roll damper, Pitch damper, Yaw damper
     Txc,Tyc,Tzz = Command Torques  Typical +-2 Puls }
     Tx:=-dampP*wx+Txc;
     Ty:=-dampQ*wy+Tyc;
     Tz:=-dampR*wz+Tzc;
{    Roll Controller, Pitch Controller, Yaw Controller
     Actually, not the angles but sines and cosines are Controlled
     Phc  =    Command   Roll      angle
     Thc  =    Command   Pitch     angle
     Psc  =    Command   Yaw       angle   }
```

Aircraft Simulation (3)

```
     If CCon Then { Roll + Pitch }
     Begin
     If EmodeA Then
     Begin
      If CPsi Then { Pitch + Yaw, Roll automatically  }
      Begin
          Phc:=+conPsi*(sPs1*cPsc-cPs1*sPsc);
          SicCoc(Phc,sPhc,cPhc);
      End;
      Tx:=Tx-conPhi*(sPh1*cPhc-cPh1*sPhc);
      Ty:=Ty-conThe*(sTh1*cThc-cTh1*sThc);
      Tz:=Tz-ratePsi*sPh1*cTh1;
   End Else
   Begin
      If CPsi Then { Pitch + Yaw, Roll automatically             }
      Begin
          Phc:=+conPsi*(sPs2*cPsc-cPs2*sPsc);
          SicCoc(Phc,sPhC,cPhc);
      End;
      Tx:=Tx-conPhi*(sPh2*cPhc-cPh2*sPhc);
      Ty:=Ty-conThe*(sTh2*cThc-cTh2*sThc);
      Tz:=Tz-ratePsi*sPh2*cTh2;
   End;
  End;
{ Body fixed coordinate system in principal axes direction   }
{ Integration of angular velocities }
wx:=wxo        +    dT*(wyo*wzo*(Jy-Jz)+Tx)/Jx;
wy:=wyo        +    dT*(wzo*wxo*(Jz-Jx)+Ty)/Jy;
wz:=wzo        +    dT*(wxo*wyo*(Jx-Jy)+Tz)/Jz;
wxo:=wx; wyo:=wy; wzo:=wz;
{ Integration of angle derivatives in Mode A=1 or Mode B=2  }
If EmodeA Then
Begin
icTh1:=1/cTh1;
Ph1:=Ph1  +    dT*( wx + wy*sPh1*sTh1*icTh1  +    wz*cPh1*sTh1*icTh1);
Th1:=Th1  +    dT*( wy*cPh1                  -    wz*sPh1           );
Ps1:=Ps1  +    dT*( wy*sPh1*icTh1            +    wz*cPh1*icTh1     );
MatObj3Da; { Elements oik, for object rotation Mode A=1 }
Atan3     (-o31,o33,Th2);
SicCoc    (Th2,sTh2,cTh2);
Atan3     (o32*cTh2,o33,Ph2);
SicCoc    (Ph2,sPh2,cPh2);
Atan3     (-o12,o22,Ps2);
SicCoc    (Ps2,sPs2,cPs2);
End Else
Begin
icPh2:=1/cPh2;
Ph2:=Ph2  +    dT*( wx*cTh2                  +    wz*sTh2           );
Th2:=Th2  +    dT*( wx*sTh2*sPh2*icPh2 + wy  -    wz*cTh2*sPh2*icPh2);
Ps2:=Ps2  +    dT*(-wx*sTh2*icPh2            +    wz*cTh2*icPh2     );
MatObj3Db; { Elements oik, for object rotation Mode B=2 }
Atan3     ( o32,o33,Ph1);
SicCoc    (Ph1,sPh1,cPh1);
Atan3     (-o31*cPh1,o33,Th1);
SicCoc(Th1,sTh1,cTh1);
Atan3     ( o21,o11,Ps1);
SicCoc    (Ps1,sPs1,cPs1);
End;
End;
```

Gernot Hoffmann,
November 26 / 2001 — February 14 / 2013
Website
Load browser, click here