Rotation Matrices to Rotate an Object around the Axes of a Three-dimensional Right-handed Coordinate System.

An ordered pair \((x,y)\) of non-parallel vectors is said to have a positive orientation, if the smallest rotation that gives \(x\) the same direction as \(y\) is positive. Otherwise \((x,y)\) has a negative orientation.

\[\begin{align*}
\text{positive oriented pair} &\quad \text{negative oriented pair} \\
\end{align*}\]

A triplet \((x,y,z)\) of linearly independent vectors is said to have a positive orientation, if \((x,y)\) has a positive orientation when viewed from the endpoint of \(z\). Otherwise \((x,y,z)\) has a negative orientation.

\[\begin{align*}
z &\quad \text{positive oriented triplet} \\
\end{align*}\]

\[\begin{align*}
\text{negative oriented triplet} \\
\end{align*}\]

If one spreads out the thumb, index finger and middle finger of the right hand, one gets a triplet with positive orientation. Similarly the corresponding fingers of the left hand gives a triplet with negative orientation. The coordinate system defined by \((x,y,z)\) is therefore called a right-handed coordinate system if the triplet has positive orientation, and left-handed otherwise.
The triplet \((x,y,z)\) has six permutations, three of which have the same orientation: \((y,z,x)\) and \((z,x,y)\). Note that these two triplets are cyclic permutations of the first. From now on we will only discuss right-handed coordinate systems.

A rotation of a vector around the axis \(z\) in the coordinate system \((x,y,z)\) is positive, if the projection vector in the \((x,y)\) plane rotates in a positive direction when viewed from the endpoint of \(z\).

In a two-dimensional system \((x,y)\) a vector is rotated by \(\alpha\) degrees (\(\alpha\) positive) around the origin by multiplication by the matrix:

\[
\begin{pmatrix}
\cos \alpha & -\sin \alpha \\
\sin \alpha & \cos \alpha \\
\end{pmatrix}
\]

Similarly in the system \((x,y,z)\) a vector is rotated by \(\alpha\) degrees around the \(z\)-axis by multiplication by the matrix:

\[
R = 
\begin{pmatrix}
\cos \alpha & -\sin \alpha & 0 \\
\sin \alpha & \cos \alpha & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

How do we rotate an object around the \(y\)-axis in \((x,y,z)\) without changing the orientation of the system? Since \((z,x,y)\) has the same orientation as \((x,y,z)\) this is done by first permuting the system \((x,y,z)\) to \((z,x,y)\), then applying \(R\) and finally by permuting the system back again. The permutation matrix looks like:

\[
P = 
\begin{pmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{pmatrix}
\]

and the desired rotation matrix is:

\[
P^RP = 
\begin{pmatrix}
\cos \alpha & 0 & \sin \alpha \\
0 & 1 & 0 \\
-\sin \alpha & 0 & \cos \alpha \\
\end{pmatrix}
\]
Problem: Derive the rotation matrix for rotating a vector around the x-axis in coordinate system $(x, y, z)$.

Solution:

We are given the matrix $R = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$ which rotates around the z-axis. We want to rotate around the x, so the defines a permutation of coordinates so that the z-axis becomes the x. That is $(x, y, z) \rightarrow (y, z, x)$.

The permutation matrix that accomplishes this is $P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

$P^T$ (P transpose) is the matrix that undos what P does.

$P^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Note that $PP^T = P^TP = I$.

So what we want to do is rotate the z into the x, rotate around the x (really the y) and rotate the x back to the z. In mathematical notation, this is:

$$R_x = P^T R_z P.$$ So we just multiply them out.

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{pmatrix}$$

is the matrix that rotates $\alpha$ degrees about the x-axis.
Problem: To build a 3 x 3 rotation matrix which will rotate $E^3$ (i.e. any rigid body in three space) through an angle $\theta$ about an arbitrary unit vector given by $\vec{a} = (a_1, a_2, a_3)$, (where $a_i$ are the direction cosines, i.e. the normalized projections) through the origin. Note that an arbitrary vector $\vec{A}$ is transformed to a unit vector $\vec{a}$ with respect to its length by normalizing.

\[ \vec{a} = \frac{\vec{A}}{||\vec{A}||} \quad ||\vec{A}|| = \sqrt{\sum a_i^2} \]

The trick is to rotate $\vec{a}$ so that it points along $x_3$, shown below, then rotate 0 about $x_3$, then rotate back so it is pointing along $\vec{a}$ again. (ref. Euler angles in Goldstein's Classical Mechanics)

step 1. rotate $-\alpha^0$ about $x_3$, swinging $\vec{a}$ into the $(x_1, x_3)$ plane.

\[ R_1 = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \]

where

\[ \cos \alpha = \frac{a_1}{\sqrt{a_1^2 + a_2^2}} \]

\[ \sin \alpha = \frac{a_2}{\sqrt{a_1^2 + a_2^2}} \]

Note that if the denominator is 0, $\vec{a}$ already points along $x_3$ so that the total rotation matrix is $R_3$. 

(I assume $\alpha$ has been normalized in my diagrams)
step 2. rotate $\phi$ about $x_2$, swinging $\bar{a}$ into $x_3$.

$$R_2 = \begin{bmatrix}
\cos \phi & 0 & -\sin \phi \\
0 & 1 & 0 \\
\sin \phi & 0 & \cos \phi
\end{bmatrix}$$

(2) where $\cos \phi = a_3$

$\sin \phi = \sqrt{1 - a_3^2}$

step 3. rotate $\theta^0$ about $x_3$

$$R_3 = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}$$

(3) step 4. rotate $-\theta^0$ about $x_2$.

$$R_4 = R_2^T$$

step 5. rotate $\phi^0$ about $x_3$.

$$R_5 = R_1^T$$

where the transpose of a matrix is obtained by interchanging rows and columns.

The single rotation matrix $R$ which will do all this is:

$$R = R_5 R_4 R_3 R_2 R_1$$

$$= (R_2 R_1)^T R_3 R_2 R_1$$

CLASS EXERCISE: Derive the analogous procedure for four-space rotations.

Algorithm:

1. Normalize $\mathbf{a}$, using formula set 1, to get $(a_1, a_2, a_3)$

2. Use formula set 1 on $a_1 + a_2$ to get $R_1$

3. Use set 2 on $a_3$ to get $R_2$

4. Compute $R_3$ by taking the square of $\phi$ (the movement) and applying formula 3

5. Get $R_4 = R_2^T + R_5 = R_2^T$ by interchanging the rows and columns of $R_1 + R_2$

6. Get the single rotation matrix $R$ by performing some tedious matrix multiplications [Use APL!] from the formula.

$$R = (R_1^T (R_3^T (R_3 (R_2 R_1))))$$
Using BUGS write a program that

1. Displays a set of x, y and z axes and a cube (draw all faces of the cube, i.e. do not remove hidden lines). Show the axes and cube rotated 30° around the x axis of the Vector General coordinate system and -30° around the y axis of the Vector General coordinate system. Like this for joystick x,y,z = 0:

   ![Cube Diagram]

2. Have the joystick move the cube in the positive and negative directions with respect to the axes.

3. Now imagine a main diagonal of the cube from the lower rear left corner to the top front right corner (i.e. from (0,0,0) to (1,1,1) on a unit cube at position zero). Rotate the cube around that diagonal in real time, using a control dial to determine the speed of the rotation. (+ direction!)

4. Program should terminate when function key 31 is pressed. The description and macro flowchart should contain descriptions of the main line program, subroutines, interrupt routines, and the rotation matrices.

On the final due date hand in a well commented listing, a macro flowchart and a description of your program.

5. Scale the cube uniformly (from a dial).
Notes + Hints

1. Remember matrix multiplication is not commutative. Do the innermost, rightmost one first.

2. When you multiply signed fractions, your result must be doubled (because of the sign bits). Example: (.5)(.5) = .25, but (.5)(.5) = 1, if you multiply too much. The best way to accomplish this is to do a LSHIFT Rn,2 immediately after your multiply.

3. You can check your matrix by making sure it keeps points on the line between (0,0,0) & (1,1,1) constant. This should suggest to you that perhaps not all 8 vertices of the cube have to be rotated. (which ones don't?)

4. Scaling & translation can be accomplished using the V6 registers, but be careful. The translation must first be multiplied by the change-of-coordinate matrix (or else z translation would do anything)

5. Use orthographic projection (i.e., just ignore z). You can do perspective if you want, but it's not worth it.

6. The general formula for transforming a vertex $v = (v_1, v_2, v_3)$ is:

   \[ M(s R(z + \Delta z)) \]

   where $s$ is the scale factor gotten from a dial and $\Delta z$ is the displacement vector obtained from the joystick. Note that this will cause the displacement to be scaled. If you don't want this (I don't care) use:

   \[ v' = M(s R(v) + \Delta z) \]

   where, as above, $R$ is the incremental rotation matrix about the diagonal and $M$ is the change-of-basis matrix about the axes.

7. It's best to rotate incrementally so you don't need sine tables. But this causes degeneration due to the fixed point arithmetic truncation. So you ought to reset the box taking advantage of symmetry to do so as frequently as possible. Also, use 16 bit arithmetic, even though display orders only have 1a bits.
SOME SIMPLE BUT USEFUL RESULTS IN ANALYTIC GEOMETRY

In an n-dimensional Euclidean space with a fixed reference frame, a vector can be represented by an n-tuple of numbers. We shall put a bar over symbols denoting n-tuples, e.g.,

\[ \overline{r} = (r_1, r_2, \ldots, r_n). \]

1. The length \( |\overline{r}| \) of a vector \( \overline{r} \) is

\[ |\overline{r}| = \sqrt{r_1^2 + r_2^2 + \ldots + r_n^2}. \]

2. The direction cosines of a vector \( \overline{r} \) are

\[ \frac{r_1}{|\overline{r}|}, \frac{r_2}{|\overline{r}|}, \ldots, \frac{r_n}{|\overline{r}|}. \]

3. The dot product of vectors \( \overline{r} \) and \( \overline{s} \) is \( \overline{r} \cdot \overline{s} = r_1s_1 + r_2s_2 + \ldots + r_ns_n. \)

4. The cross product of 3-vectors \( \overline{r} \) and \( \overline{s} \) is \( \overline{r} \times \overline{s} = (r_2s_3 - r_3s_2, r_3s_1 - r_1s_3, r_1s_2 - r_2s_1). \)

**Solid Analytic Geometry**

1. The length of a vector \( \overline{r} \) is \( \sqrt{\overline{r} \cdot \overline{r}}. \)
2. The vector directed from point \( \overline{r} \) to point \( \overline{s} \) is \( \overline{s} - \overline{r}. \)
3. The angle \( \theta \) between non-zero vectors \( \overline{r} \) and \( \overline{s} \) is given by

\[ \cos \theta = \frac{\overline{r} \cdot \overline{s}}{|\overline{r}| |\overline{s}|}. \]
4. \( \overline{r} \) is perpendicular to \( \overline{s} \) iff \( \overline{r} \cdot \overline{s} = 0. \)
5. \( \overline{r} \) is parallel to \( \overline{s} \) iff \( \overline{r} \times \overline{s} = 0. \)
6. If \( \overline{r} \times \overline{s} \neq \overline{0} \), then the vector \( \overline{r} \times \overline{s} \) is perpendicular both to \( \overline{r} \) and to \( \overline{s} \).
7. The plane through point \( \vec{P} \) with direction \( \vec{a} \) for its normal has the equation 
\[(\vec{x} - \vec{P}) \cdot \vec{a} = 0.\]

8. The distance \( D \) from point \( \vec{s} \) to the plane defined in \( F_1 \) is
\[D = \frac{(\vec{s} - \vec{P}) \cdot \vec{a}}{\vec{a}}.\]

9. The line through point \( \vec{P} \) with direction \( \vec{a} \) has the equation
\[\vec{x} = \vec{P} + t \vec{a}, \text{ where } t \text{ is a scalar parameter.}\]

10. The distance \( D \) from point \( \vec{s} \) to the line defined in 9. is
\[D = \frac{(\vec{s} - \vec{P}) \times \vec{a}}{\vec{a}}.\]

**Line through points \( \vec{a} \) and \( \vec{b} \):**

Any point \( \vec{x} \) on the infinite line passing through distinct points \( \vec{a} \) and \( \vec{b} \) has the form
\[\vec{x} = t \vec{b} + (1-t)\vec{a}, \text{ } t \in (-\infty, \infty).\]

If \( t = 0 \), \( \vec{x} = \vec{a} \);
If \( t = 1 \), \( \vec{x} = \vec{b} \);
if \( t \in (0,1) \), \( \vec{x} \) is on that part of the line which lies between \( \vec{a} \) and \( \vec{b} \);
if \( t \notin [0,1] \), \( \vec{x} \) is on that part of the line which does not lie between \( \vec{a} \) and \( \vec{b} \).

**Right-Hand Rule:** To determine the 'handedness' of a 3-space Cartesian reference frame with axes \( (\vec{e}_1, \vec{e}_2, \vec{e}_3) \), place your right hand on the \( (\vec{e}_1, \vec{e}_2) \) plane with your fingers curling from \( \vec{e}_1 \) towards \( \vec{e}_2 \). Your thumb will then point in the direction of \( \vec{e}_3 \) if the reference frame is right-handed, and in the direction of \(-\vec{e}_3 \) if the reference frame is left-handed.
left-handed reference frame

The matrix $A$ which will transform a set of 3 mutually orthogonal unit vectors $\vec{e}_1, \vec{e}_2, \vec{e}_3$ into another set of 3 mutually orthogonal unit vectors $\vec{e}_1', \vec{e}_2', \vec{e}_3'$, can be found by letting

$$A = \begin{bmatrix} \vec{e}_1 \cdot \vec{e}_1' & \vec{e}_2 \cdot \vec{e}_1' & \vec{e}_3 \cdot \vec{e}_1' \\ \vec{e}_1 \cdot \vec{e}_2' & \vec{e}_2 \cdot \vec{e}_2' & \vec{e}_3 \cdot \vec{e}_2' \\ \vec{e}_1 \cdot \vec{e}_3' & \vec{e}_2 \cdot \vec{e}_3' & \vec{e}_3 \cdot \vec{e}_3' \end{bmatrix} = [\vec{e}_i' \cdot \vec{e}_j']$$

If both the $\vec{e}_i$ and the $\vec{e}_i'$ systems have the same 'handedness' or 'orientation', $\det(A) = +1$, i.e. $A$ is a pure rotation. If $\{\vec{e}_i\}$ and $\{\vec{e}_i'\}$ have different handedness, $\det(A) = -1$ and $A$ is a combination of a reflection and a rotation.

**Cross Product as a Matrix**

Let

$$X_p = \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix},$$

where $\vec{r}$ is a given 3-vector. Then if $\vec{s}$ is any 3-vector,

$$X_p \vec{s} = \vec{r} \times \vec{s}.$$
A 2-D window usually is defined by a max and min value for x and y or by a center of the window and maximum relative x and y values.

Simple subtractions or comparisons suffice to find out whether or not a point is in view.

3-D window may be cube (if an orthogonal projection is involved) or an infinite pyramid (if a perspective projection is involved).
1. Standard Window

If a perspective projection is involved, we may, by suitable translations, scales and rotations, place the point of view at the origin and force the viewport to be the square in the $Z = 1$ plane bounded by $X = \pm 1$, and $Y = \pm 1$.

Then the viewing volume may be described by $Z \geq X$, $Z \geq -X$, $Z \geq Y$, and $Z \geq -Y$.

2. Point

Thus, to window a point, form the differences (and sums) $Z-X$, $Z+X$, $Z-Y$, $Z+Y$.

If any are negative, the point is out of view.

3. Line

A line segment to be displayed may be categorized according to the number of its endpoints which lie within the window.

2  line is totally visible
1  line is partially visible
0  doubtful (i.e. both endpoints outside, further tests necessary)
BLANKING One scheme of handling lines in which, if any part of the line is not visible, the whole line is not displayed. This is very simple and easy to do, both in the hardware and in the software.

CLIPPING The alternative to blanking. Any part of a line within the window is displayed. If one of the endpoints lies in view, finding the other point is easy enough. However, if both points are out of view, further tests must be made. One method for deciding would be to solve for points of intersection of the line containing the line segment with the lines forming the edges of the window.

Analytic Algorithm
In 2 Dimensions:

Using notation from linear algebra, find \( t \) and \( s \) such that
\[
t \vec{P}_1 + (1-t) \vec{P}_2 = s \vec{C}_1 + (1-s) \vec{C}_2.
\]
(Note that points of the form \( t \vec{X}_1 + (1-t) \vec{X}_2 \) for real \( t \) are on the line through the points \( \vec{X}_1 \) and \( \vec{X}_2 \). Also when \( t \in [0,1] \) the point is on the line segment from \( \vec{X}_2 \) to \( \vec{X}_1 \).) If both \( t \) and \( s \) are between 0 and 1, then the point of intersection is both.
between $\vec{P}_1$ and $\vec{P}_2$ (so it could be displayed) and between $\vec{C}_1$ and $\vec{C}_2$ (so it is in view).

As soon as one point to be viewed has been found the line segment may be treated as two separate line segments each of which has one endpoint in view. (One of these segments may be rejected completely, as shown below.)

In 3D one must find the intersection of the $\vec{P}_1\vec{P}_2$ line with the planes that define the viewing volume, determining whether each point is part of the line segment $\vec{P}_1\vec{P}_2$ and if so whether it is part of the viewing volume.

A different method, which works in 2D and in 3D, was devised by Sproull and Sutherland for use in their (hardware) Clipping Divider (see reference no. 40). For each point, they form the "window coordinates"

\[
\begin{align*}
(y-y_{\text{min}}, & \quad x-x_{\text{min}}, \quad y_{\text{max}}-y, \quad x_{\text{max}}-x) \quad (2-D \text{ Arbitrary rectangular window}) \\
& \quad (y+Z, \quad x+Z, \quad z-y, \quad z-x) \quad (3-D \text{ Square window centered at origin})
\end{align*}
\]

The $Z$ coordinate defines the $X$ max and min and $Y$ max and min.
They then form the "out code" for the point, a 4 bit number consisting of the sign bits of each of the 4 coordinates.

If the out code is 0000 the point is in view. If a line segment is "doubtful", the out codes for its endpoints are combined with a logical "and" operation. If the result is non-zero, the points are both to the right or both above etc. and the line may be rejected. If the result is zero, the mid point of the line segment is found, its screen coordinates calculated and its outcode generated and compared to the out codes of both endpoints. It is always possible to reject one half of the line segment this way. (see figure below) If one endpoint is in, half is rejected or accepted.

If both halves may be rejected, the whole line is; if only one half, the algorithm iterates.
For a single point—
Values of the "out code" in and around a 3 D
window for positive Z (left) and negative Z (right).
Current Hardware Techniques for Windowing, Blanking and Clipping

Some display devices have no facilities for windowing. On the IBM 2250 Mod I, for example, the coordinate space is (0-4095) X (0-4095) and any coordinate outside of this range is adjusted by reducing it modulo 4096. This (probably undesired) effect is called wrap-around.

Some devices, like the IBM 2250 Mod IV and the DEC 338 provide windowing features by allowing extra precision high-order bits in the X and Y registers (one bit in the mod IV; 0,1,2 or 3 under program control in the 338). A point will be displayed in these machines if its high order bits in both X and Y are zero. The Mod IV (having an automatic vector generator) blanks line segments which are partly off the screen. The 338 (having a point plotting vector generator) does clipping by not intensifying points along the line which are out of bounds. If you go beyond full scale space, you wrap-around.

The Adage Graphics Terminal provides dynamic windowing in three dimensional rectangular coordinates. Digital maximum and minimum X, Y and Z values may be specified within the display file. A vector is blanked whenever it goes beyond one of these limits.

Ivan Sutherland's Head Mounted Display (see reference no. 43) (which presents perspective views in stereo to each eye) has three-dimensional perspective clipping. The hardware for this, the Clipping Divider, uses the midpoint algorithm mentioned above. Like the Adage graphics terminal, windowing values are specified by commands from within the display file.
LINE

CLASSIC: 1. BOTH POINTS INSIDE?
          YES, THEN IN
          2. SOLVE FOR INTERSECTION
             POINTS
          3. IF POINT ON WINDOW, CLIP

CLIPPING DIVIDER:

2 DIMENSIONAL WINDOW COORDINATES:

\[
\begin{bmatrix}
Y - Y_{\text{MIN}}, X - X_{\text{MIN}}, Y_{\text{MAX}} - Y, X_{\text{MAX}} - X
\end{bmatrix}
\]

3 DIMENSIONAL WINDOW COORDINATES:

\[
\begin{bmatrix}
\end{bmatrix}
\]