Contents

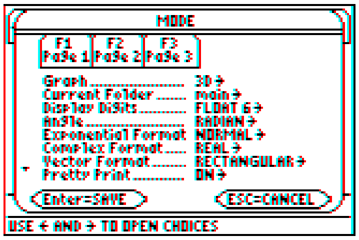
Fourth International Derive TI-89/92 Conference
Liverpool John Moores University, July 12 –15, 2000

Exploring 3D Rotations with the TI-89

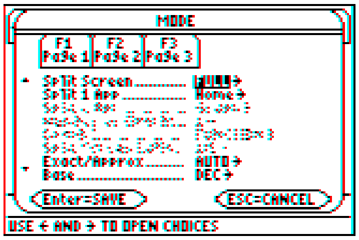
Dennis Pence
Western Michigan University, USA
Email: dennis.pence@wmich.edu

Introduction

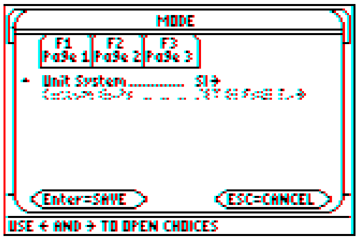
Because of its price, small size, and power, the TI-89 is quickly becoming the graphing calculator of choice for students majoring in applied mathematics, engineering, and science. All of the activities we describe can also be done on the TI-92 with a Plus Module installed, but the screen size is a little different. It is necessary to get all calculators in a workshop in the same mode settings. Please move to the MODE screen in the HOME application, and make your settings match those listed below.



```
Graph..................  3D →
Current Folder.......  main →
Display Digits.......  FLOAT 6 →
Angle..................  RADIAN →
Exponential Format  NORMAL →
Complex Format.....  REAL →
Vector Format.......  RECTANGULAR →
Pretty Print.........  ON →
```

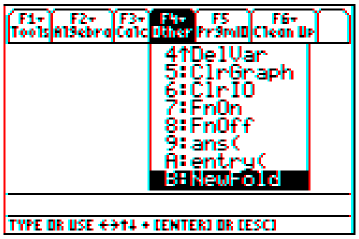


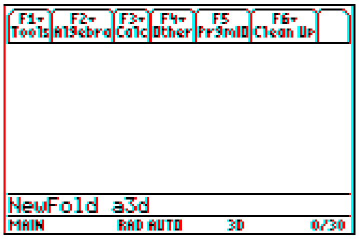
```
Split Screen.........  FULL →
Split 1 App..........  Home →
Exact/Approx........  AUTO →
Base....................  DEC →
```



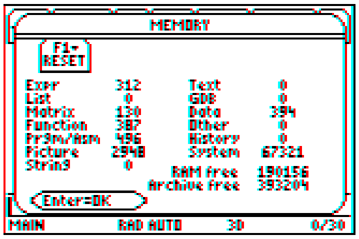
```
Unit System.........  SI →
```

Press ENTER enough times to save these settings. It is also suggested that you create a new folder to hold the work for this workshop. That way there will be no conflicts with variables you may have defined in your main folder. Of course if you are using a "loaner," a 2nd MEM F1 (RESET) might be more appropriate with no need for a new folder.





```
NewFold a3d
```



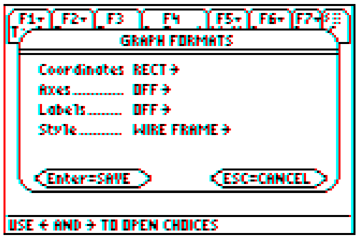
```
Expr        312      Text        0
List          0      GDB         0
Matrix      130      Data      394
Function    387      Other       0
Prgm/Asm    496      History     0
Picture    2948      System  67321
String        0
                RAM free    190156
            Archive free    393204
```

Basic 3D Graphing Features

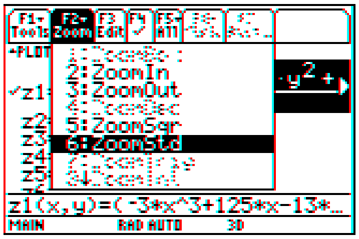
Move to the Y= Editor, and type in the following sample function of two variables.

```
z1(x,y) = (-3*x^3+125*x-13*y^2+750)/250
```

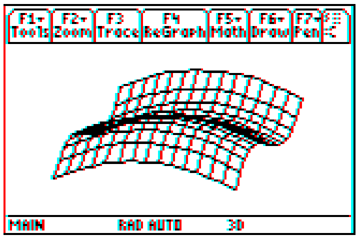
The strange coefficients are designed to give a "nice" plot in the standard viewing box. Press F1 (Tools) and make sure that you have the graph formatting options selected as below (axes OFF, labels OFF, and style WIRE FRAME). Then select ZoomStd.



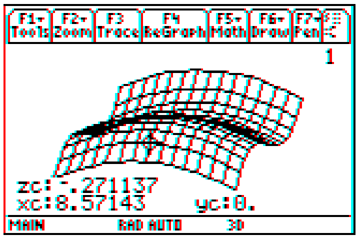
Coordinates RECT
Axes OFF
Labels OFF
Style WIRE FRAME



z1(x,y)=(-3*x^3+125*x-13*…

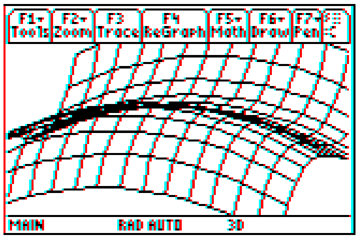


For the benefit of those who have not yet fully explored the 3D capabilities of the TI-89, we will quickly review these features. While looking at a 3D plot, press F3 (Trace) and move the cursor keys to see a blinking cursor point on the surface and the associated coordinates of that point. No matter what the orientation of the plot, you need to think that the cursor pad is orientated as a traditional *xy*-plane. Pressing the right cursor key increases the value of *x* regardless of what direction the blinking cursor moves on the screen. In our ZoomStd orientation, pressing the right cursor key moves the blinking point forward and to the left. Similarly, the up cursor key increases the value of *y*. Press ECS to turn this tracing feature off. Press the multiplication key to toggle back and forth between the regular view and a magnified view. With the trace off, pressing a cursor key will cause the plot to be rotated. Holding down a cursor key long enough, the rotation will continue even after you let up.

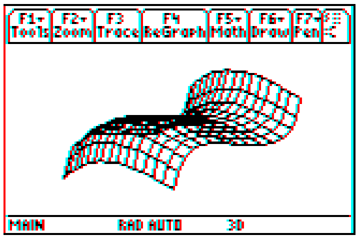


zc: -.271137
xc:8.57143 yc:0.

Trace (F3)

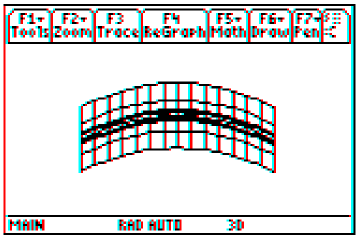


Magnified (×)

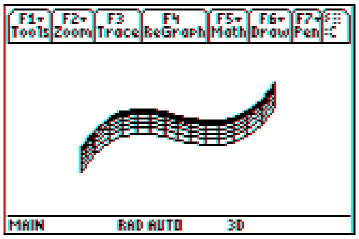


Rotated (≺, six times)

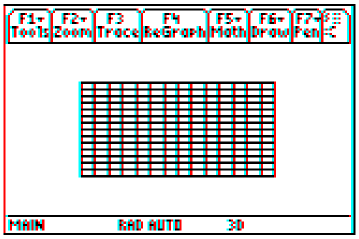
You can speed up or slow down the continuous rotation by pressing the addition key (+) or subtraction key (-). This will change the angle increments used in the rotation. To stop a continuously rotating plot, press any of ESC, ENTER, ON, or diamond ◊ (space). Press one of the keys labeled X, Y, or Z to have the plot rotated so that you look directly down the axis labeled with that letter. Finally, press zero (0) to have the orientation return to the settings before all of this interactive rotating (here ZoomStd).



X



Y

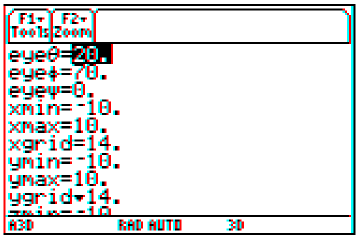


Z

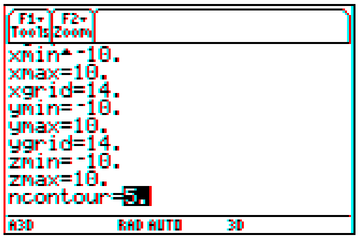
If time permits, explore the alternative 3D graph formatting options to have axes AXES or BOX, or to have the style with a HIDDEN SURFACE plot, a CONTOUR LEVELS plot, or both WIRE mesh and CONTOUR levels together in a plot.

Understanding the 3D Window Settings

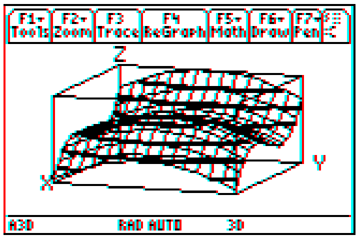
Return to the ZoomStd plot orientation. We will briefly explain the Window settings and explore how changing them affects the plot. The first three settings give the orientation of the viewer relative to the plot (or the "eye" location). We shall work with these angles in more detail later.



```
eyeθ=20.
eyeϕ=70.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.
zmin=-10
A3D        RAD AUTO     3D
```

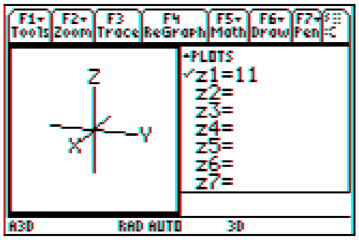


```
xmin▴-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid=14.
zmin=-10.
zmax=10.
ncontour=5.
A3D        RAD AUTO     3D
```

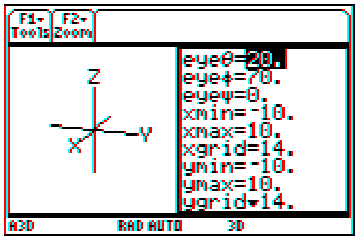


Obviously xmin, xmax, ymin, ymax, zmin, and zmax specify the viewing box. The variables xgrid and ygrid determine the number and spacing of the grid lines in each direction. For example, the spacing between grid lines along the *x*-axis is delta-*x* = (xmax - xmin)/xgrid. Thus one grid line corresponds to *x* = xmin, and there are 14 other grid lines for *x* = xmin + *j**delta-*x*, *j* = 1, 2, ..., 14, ending with *x* = xmax. The variable ncontour determines how many contour lines will be attempted in a contour plot. Of course the function many not take on values equal to all of the levels, resulting in less than ncontour curves, but here we do get five different contour lines for this function in the wire-mesh/contour plot above.

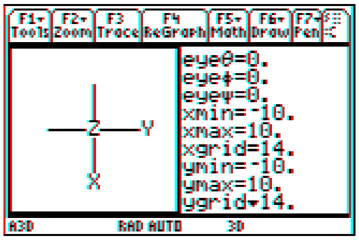
In the FULL screen plotting that we have been doing, the axes are not equally scaled. The plot is stretched horizontally to more fully use the space available. We get a more equally scaled plot if we change the MODE setting to a split screen LEFT-RIGHT way of looking at a plot. Also change the graphing format to axes AXES, labels ON, and style WIRE FRAME. This will speed thing up and show the coordinate axes. To further investigate the meaning of the "eye" angles, we wish to plot only the axes. To "fool" the calculator into doing this, we ask it to plot $z1(x, y) = 11$ in the standard viewing window. No part of the graph of this function will appear in the viewing box. Note that this standard window looks very much like the traditional orientation for the axes in textbooks. We toggle back and forth between the two parts of the split screen using 2nd __ (above the APPS key) so that we can see both the window settings and the 3D plot of the axes.



```
▸PLOTS
✓z1=11
 z2=
 z3=
 z4=
 z5=
 z6=
 z7=
A3D        RAD AUTO     3D
```



```
eyeθ=20.
eyeϕ=70.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.
A3D        RAD AUTO     3D
```



```
eyeθ=0.
eyeϕ=0.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.
A3D        RAD AUTO     3D
```

If we change to eye$\theta = 0$, eye$\phi = 0$, and eye$\psi = 0$ or where the eye angle triple $(\theta, \phi, \psi) = (0, 0, 0)$, we will find that the "neutral" orientation has us looking straight down the z-axis (with positive z up) with the positive x-axis down and the positive y-axis to the right. We will consider the eye angle triple (0, 0, 0) as the screen coordinate system, and label it the UVW system with unit vectors along each of the screen axes as **u**, **v**, **w**. The desired XYZ system will be thought of as being "rotated" from the UVW system. The word "rotated" implies movement (when there really isn't any here). Both the UVW and the XYZ can be still, but we visualize how one would need to "move" a set of axes from the UVW orientation into the XYZ orientation.

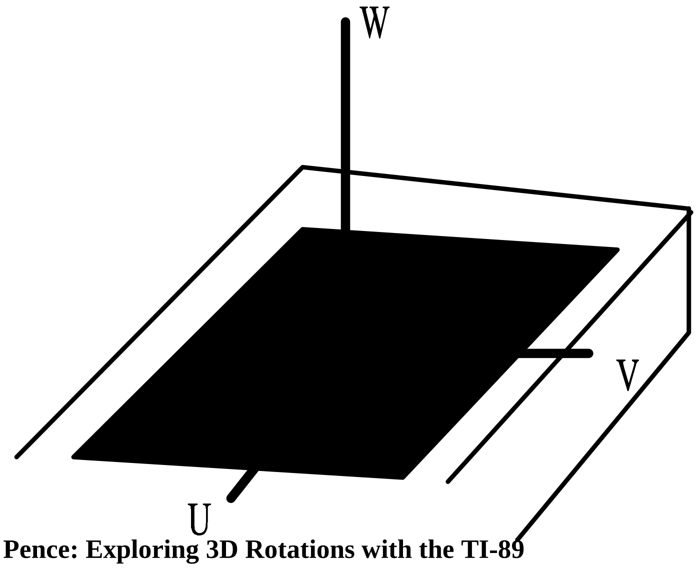
Suppose we have a three-vector $\mathbf{p} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$ (in the XYZ system) or $\mathbf{p} = r\mathbf{u} + s\mathbf{v} + t\mathbf{w}$ (in the UVW system). We seek a transition matrix to describe how to go from (a, b, c) to (r, s, t).

$$A\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} r \\ s \\ t \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} r \\ s \\ t \end{bmatrix}$$

where $\mathbf{i} = x_1\mathbf{u} + x_2\mathbf{v} + x_3\mathbf{w}$, $\mathbf{j} = y_1\mathbf{u} + y_2\mathbf{v} + y_3\mathbf{w}$, $\mathbf{k} = z_1\mathbf{u} + z_2\mathbf{v} + z_3\mathbf{w}$. We know that A is a proper orthogonal matrix, i.e. $A^{-1} = A^T$ and $\det(A) = 1$. The following special rotation matrices correspond to rotation about one of the UVW axes. The sign convention for the angle of rotation is that of a *right-hand system* in that if the thumb of the right hand points in the direction of the positive axis of rotation, the fingers of the right hand curl in the positive angle direction.

$$A_1(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \quad A_2(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}, \quad A_3(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

An arbitrary rotation can be represented as a sequence of three rotations about one of the UVW coordinate axes. The first rotation in the sequence can be about any axis, the second about a different axis, and the third about a axis different from the second. Since the order in which we multiply matrices matters, the order in which we perform the rotations matters. The angles are called an *Euler angle sequence*.



Claim: The eye angle triple (θ, ϕ, ψ) specified in the Window editor corresponds to the transition

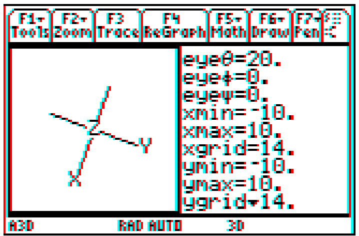
$$A = A_3(\psi)A_2(-\varphi)A_3(-\theta)$$

matrix
giving a 3-2-3 Euler angle sequence $(-\theta, -\phi, \psi)$.

This information can be used in the following way. If we know the coordinates (a, b, c) for a point on the surface of our 3D plot, then we can compute

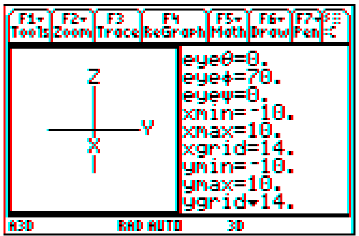
$$A\begin{bmatrix}a\\b\\c\end{bmatrix} = A_3(\psi)A_2(-\varphi)A_3(-\theta)\begin{bmatrix}a\\b\\c\end{bmatrix} = \begin{bmatrix}r\\s\\t\end{bmatrix}$$

and then plot (*r*, *s*) in the UV system in the plane of the screen to get the desired projection.



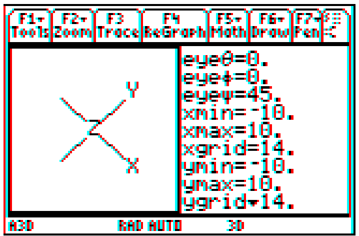
F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

eyeθ=20.
eyeφ=0.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.



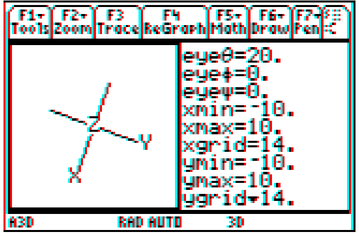
F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

eyeθ=0.
eyeφ=70.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.



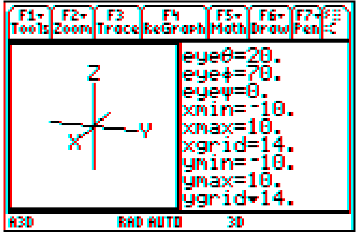
F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Y
Z
X
A3D RAD AUTO 3D

eyeθ=0.
eyeφ=0.
eyeψ=45.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.



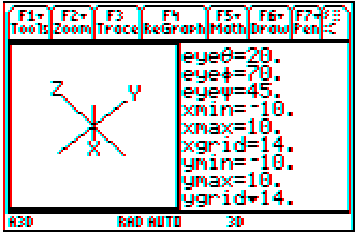
F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

eyeθ=20.
eyeφ=0.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.



F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

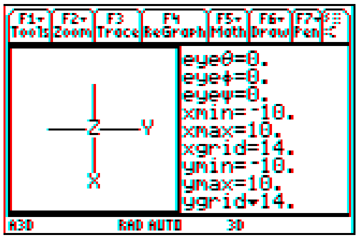
eyeθ=20.
eyeφ=70.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.



F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

eyeθ=20.
eyeφ=70.
eyeψ=45.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.

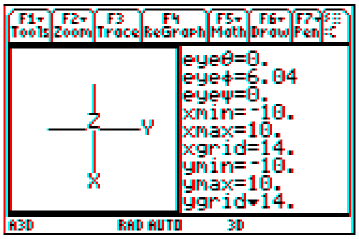
Further we can interpret the rotation of the XYZ system by a press of one of the cursor keys in the notation by investigating the same effect.



F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

eyeθ=0.
eyeφ=0.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.

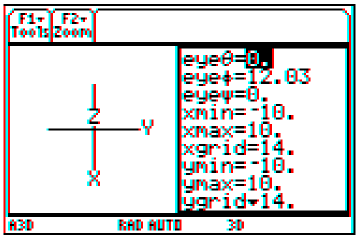
UVW system



F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

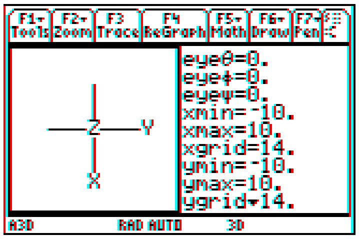
eyeθ=0.
eyeφ=6.04
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.

⮝ once, A_2(-6.04°)



eyeθ=0.
eyeφ=12.03
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.

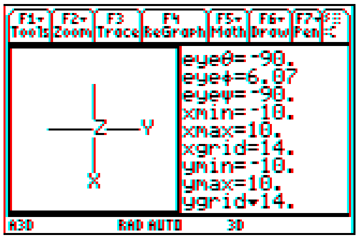
⮝ twice, A_2(-12.03°)



F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

eyeθ=0.
eyeφ=0.
eyeψ=0.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.

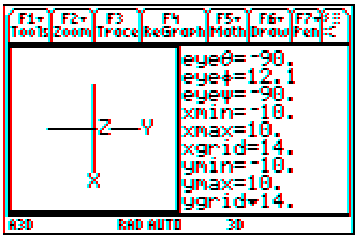
UVW system



F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

eyeθ=-90.
eyeφ=6.07
eyeψ=-90.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.

➤ once, A_1(-6.07°)



F1▾ Tools | F2▾ Zoom | F3 Trace | F4 ReGraph | F5▾ Math | F6▾ Draw | F7▾ Pen
Z
Y
X
A3D RAD AUTO 3D

eyeθ=-90.
eyeφ=12.1
eyeψ=-90.
xmin=-10.
xmax=10.
xgrid=14.
ymin=-10.
ymax=10.
ygrid▾14.

➤ twice, A_1(-12.1°)

We can check this by confirming that the rotation is recorded in the eye angles by (-90°, 6°, -90°).

$$\begin{aligned} A_3(-90^\circ)A_2(-6^\circ)A_3(90^\circ) &= \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 6^\circ & 0 & -\sin 6^\circ \\ 0 & 1 & 0 \\ \sin 6^\circ & 0 & \cos 6^\circ \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos 6^\circ & \sin 6^\circ \\ 0 & -\sin 6^\circ & \cos 6^\circ \end{bmatrix} = A_1(-6^\circ) \end{aligned}$$

It is also easy to define functions a1, a2, and a3 on the TI-89 to give these special rotation matrices. That makes the checking straightforward. For example,

```
Define a3(x)=[cos(x),-sin(x),0;sin(x),cos(x),0;0,0,1]
```

gives one of these matrix functions.

The three eye angles we set in the Window can give us any rotation as a 3-2-3 Euler angle sequence. We can also move to any rotational orientation using the cursors keys by effectively using a 2-1-2 Euler angle sequence or a 1-2-1 Euler angle sequence. For more information about rotations, the UMAP Unit 652, “Spacecraft Attitude, Rotations and Quaternions,” by Dennis Pence (1984) is recommended. Given how the eye angle settings change to some standard (no matter what you set them), I strongly suspect that the machine is using a different representation for the rotation internally. As described in this UMAP Unit, an excellent choice would be the quaternion representation. [Obviously this rather dated UMAP module needs to be rewritten with activities for a TI-89!] We give a brief introduction to the topic of quaternions next.

Beginning Quaternions

The CAS capabilities can be used to study the topic of quaternions. A quaternion has the general form

$$\tilde{q} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4$$

where

$$V(\tilde{q}) = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

is the vector part and

$$S(\tilde{q}) = q_4$$

is the scalar part.

You can add (or subtract) quaternions by adding (or subtracting) like components. Two quaternions can be multiplied in the following fashion.

$$\begin{aligned} \tilde{p}\tilde{q} &= \left(p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k} + p_4\right)\left(q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4\right) \\ &= p_1q_1\mathbf{ii} + p_1q_2\mathbf{ij} + p_1q_3\mathbf{ik} + p_1q_4\mathbf{i} + p_2q_1\mathbf{ji} + p_2q_2\mathbf{jj} + p_2q_3\mathbf{jk} + p_2q_4\mathbf{j} \\ &\quad + p_3q_1\mathbf{ki} + p_3q_2\mathbf{kj} + p_3q_3\mathbf{kk} + p_3q_4\mathbf{k} + p_4q_1\mathbf{i} + p_4q_2\mathbf{j} + p_4q_3\mathbf{k} + p_4q_4 \end{aligned}$$

where

$$\mathbf{ii} = \mathbf{jj} = \mathbf{kk} = -1, \quad \mathbf{ij} = \mathbf{k}, \quad \mathbf{jk} = \mathbf{i}, \quad \mathbf{ki} = \mathbf{j},$$
$$\mathbf{ji} = -\mathbf{k}, \quad \mathbf{kj} = -\mathbf{i}, \quad \mathbf{ik} = -\mathbf{j}$$

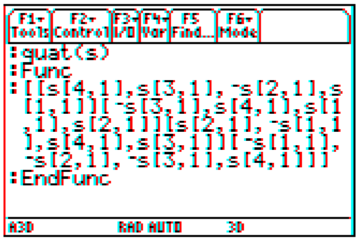
Thus

$$\begin{aligned}\tilde{p}\tilde{q} &= (p_1q_4 + p_2q_3 - p_3q_2 + p_4q_1)\,\mathbf{i} + (-p_1q_3 + p_2q_4 + p_3q_1 + p_4q_2)\,\mathbf{j} \\ &\quad + (p_1q_2 - p_2q_1 + p_3q_4 + p_4q_3)\,\mathbf{k} + (-p_1q_1 - p_2q_2 - p_3q_3 + p_4q_4)\end{aligned}$$

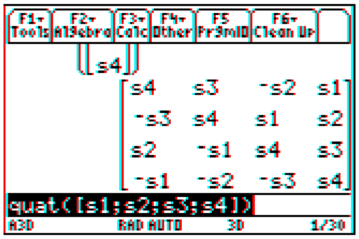
.

On the TI-89, we will store the four components of a quaternion as a matrix with four rows and one column. Unfortunately the variable names *p1* and *q1* have other meanings on this calculator, so we avoid these. There are several ways that we can implement quaternion multiplication. One of the more interesting ways is the following. We first form a special matrix using the components of a quaternion.

$$\tilde{Q}(\tilde{s}) = \begin{bmatrix} s_4 & s_3 & -s_2 & s_1 \\ -s_3 & s_4 & s_1 & s_2 \\ s_2 & -s_1 & s_4 & s_3 \\ -s_1 & -s_2 & -s_3 & s_4 \end{bmatrix}$$

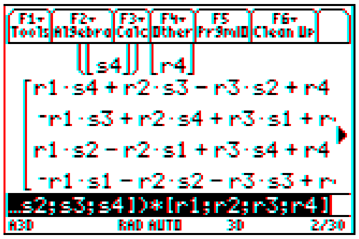


```
:quat(s)
:Func
:[[s[4,1],s[3,1],⁻s[2,1],s
 [1,1]][⁻s[3,1],s[4,1],s[1
 ,1],s[2,1]][s[2,1],⁻s[1,1
 ],s[4,1],s[3,1]][⁻s[1,1],
 ⁻s[2,1],⁻s[3,1],s[4,1]]]
:EndFunc
```

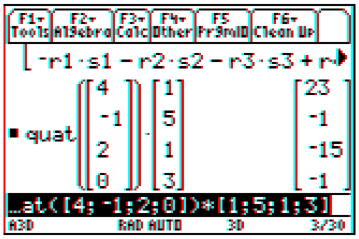


```
  ([s4])
         [s4   s3   ⁻s2  s1]
         [⁻s3  s4   s1   s2]
         [s2   ⁻s1  s4   s3]
         [⁻s1  ⁻s2  ⁻s3  s4]
quat([s1;s2;s3;s4])
```

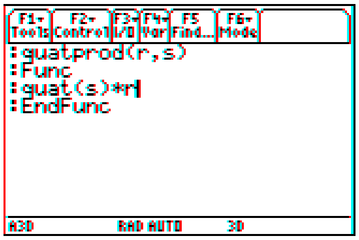
We can compute $\tilde{r}\tilde{s} = \tilde{Q}(\tilde{s})\tilde{r}$ using a matrix multiplication.



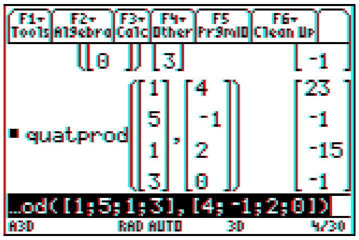
```
    ([s4]) [r4]
[r1·s4 + r2·s3 − r3·s2 + r4
 ⁻r1·s3 + r2·s4 + r3·s1 + r
 r1·s2 − r2·s1 + r3·s4 + r4
[⁻r1·s1 − r2·s2 − r3·s3 + r
…s2;s3;s4])*[r1;r2;r3;r4]
```



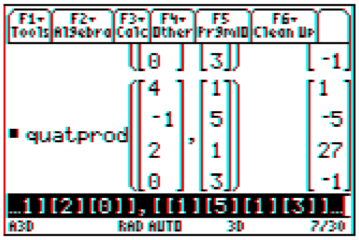
```
[⁻r1·s1 − r2·s2 − r3·s3 + r
      ([4 ]) [1]        [23 ]
      ( ⁻1 ) [5]        [⁻1 ]
■ quat( 2  )·[1]        [⁻15]
      ([0 ]) [3]        [⁻1 ]
…at([4;⁻1;2;0])*[1;5;1;3]
```



```
:quatprod(r,s)
:Func
:quat(s)*r
:EndFunc
```

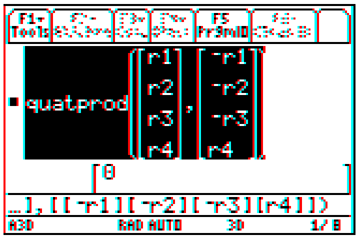


```
          ([0 ]) [3]        [⁻1 ]
          ([1] [4 ])        [23 ]
          ( 5   ⁻1 )        [⁻1 ]
■ quatprod( 1 , 2  )        [⁻15]
          ([3] [0 ])        [⁻1 ]
…od([1;5;1;3],[4;⁻1;2;0])
```

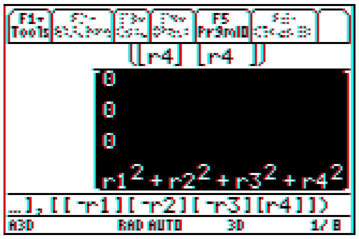


```
          ([0 ] [3])        [⁻1]
          ([4 ] [1])        [1 ]
          ( ⁻1   5 )        [⁻5]
■ quatprod( 2  , 1 )        [27]
          ([0 ] [3])        [⁻1]
…1][2][0]],[[1][5][1][3]]…
```

The norm of a quaternion, denoted by $\tilde{q}$ $|\tilde{q}|$, is the same as the norm of a 4-vector, so we can use the matrix norm command. The conjugate of a quaternion is $\tilde{q}^* = -V(\tilde{q}) + S(\tilde{q})$, very much like the conjugate of a complex number. This could be programmed as a function. The quaternion product of a quaternion $\tilde{r}$ and its conjugate is the square of its norm, also much like with a complex number.



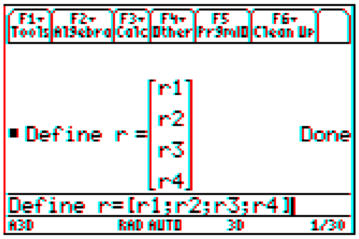
```
■ quatprod([r1;r2;r3;r4], [⁻r1;⁻r2;⁻r3;r4])
[0
…],[[⁻r1][⁻r2][⁻r3][r4]])
A3D          RAD AUTO        3D          1/8
```



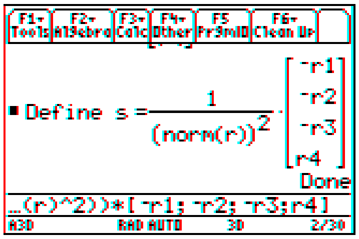
```
([r4] [r4 ])
[0
 0
 0
 r1^2 + r2^2 + r3^2 + r4^2]
…],[[⁻r1][⁻r2][⁻r3][r4]])
A3D          RAD AUTO        3D          1/8
```

We can confirm that any non-zero quaternion has a multiplicative inverse. $\tilde{r}^{-1} = \left(\frac{1}{|\tilde{r}|^2}\right)\tilde{r}^*$

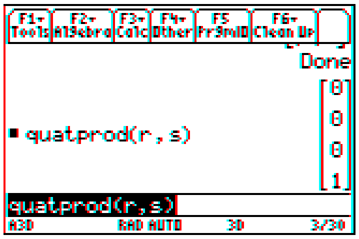
In particular, for a unit quaternion (one with norm 1), the conjugate is the inverse.



```
■ Define r = [r1;r2;r3;r4]                Done
Define r=[r1;r2;r3;r4]
A3D          RAD AUTO        3D          1/30
```



```
■ Define s = 1/(norm(r))^2 · [⁻r1;⁻r2;⁻r3;r4]
                                          Done
…(r)^2))*[⁻r1;⁻r2;⁻r3;r4]
A3D          RAD AUTO        3D          2/30
```



```
                                          Done
■ quatprod(r,s)                      [0;0;0;1]
quatprod(r,s)
A3D          RAD AUTO        3D          3/30
```

The Quaternion Rotation Operator

We define the following family of operators on the space of all quaterions.

$$\mathbf{M}_{\tilde{q}}(\tilde{p}) = \tilde{q}^{-1}\tilde{p}\tilde{q} = \left(\frac{1}{|\tilde{q}|^2}\right)\tilde{q}^*\tilde{p}\tilde{q} = \left(\frac{\tilde{q}^*}{|\tilde{q}|}\right)\tilde{p}\left(\frac{\tilde{q}}{|\tilde{q}|}\right)$$

Although we can index this family with any quaternion, we get exactly the same operator using a unit quaternion "in the same direction" or using the negative of that quaternion. It is easy to verify the following properties for this family.

$$\mathbf{M}_{\tilde{q}}(a\tilde{r} + b\tilde{s}) = a\mathbf{M}_{\tilde{q}}(\tilde{r}) + b\mathbf{M}_{\tilde{q}}(\tilde{s}), \quad \text{(linearity)}$$

$$\mathbf{M}_{\tilde{p}} \circ \mathbf{M}_{\tilde{q}} = \mathbf{M}_{\tilde{q}\tilde{p}}, \quad \text{(composition)}$$

$$\left(\mathbf{M}_{\tilde{q}}\right)^{-1} = \mathbf{M}_{\left(\tilde{q}^{-1}\right)}, \quad \text{(inverse operator)}$$

A quaternion with zero scalar part is called a pure vector. Restricting this operator to the subspace of pure vectors, we find that the image space is also the space of pure vectors. In fact,

$$\mathbf{M}_{\tilde{q}}(a\mathbf{i}+b\mathbf{j}+c\mathbf{k}) = A\begin{pmatrix} a \\ b \\ c \end{pmatrix} \text{ for some rotation matrix } A.$$

Thus, a quaternion (in particular, a unit quaternion) becomes another way to parameterize a rotation via this operator. For a general rotation about an axis specified by a unit vector $\mathbf{n} = n_1\mathbf{i} + n_2\mathbf{j} + n_3\mathbf{k}$ through an angle α measured by the right-hand orientation, the operator $M_{\tilde{q}}$ gives the same rotation when the unit quaternion is

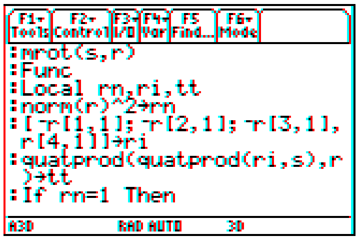
$$\tilde{q} = \sin\left(\frac{\alpha}{2}\right)(n_1\mathbf{i}+n_2\mathbf{j}+n_3\mathbf{k}) + \cos\left(\frac{\alpha}{2}\right).$$

Using the composition property, the unit quaternion corresponding to the eye angle triple (θ, ϕ, ψ) in the 3D window setting on the TI-89 will be

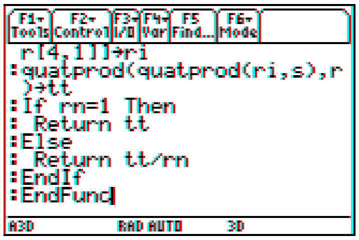
$$\tilde{q} = \left(\sin\frac{-\theta}{2}\mathbf{k} + \cos\frac{-\theta}{2}\right)\left(\sin\frac{-\varphi}{2}\mathbf{j} + \cos\frac{-\varphi}{2}\right)\left(\sin\frac{\psi}{2}\mathbf{k} + \cos\frac{\psi}{2}\right)$$

.

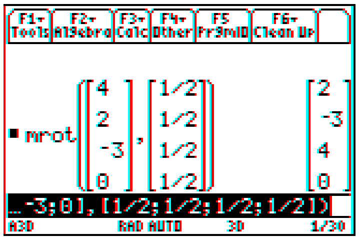
The following function-type program implements this quaternion rotation operator .



```
:mrot(s,r)
:Func
:Local rn,ri,tt
:norm(r)^2→rn
:[⁻r[1,1];⁻r[2,1];⁻r[3,1],
 r[4,1]]→ri
:quatprod(quatprod(ri,s),r
 )→tt
:If rn=1 Then
```



```
 r[4,1]]→ri
:quatprod(quatprod(ri,s),r
 )→tt
:If rn=1 Then
:  Return tt
:Else
:  Return tt/rn
:EndIf
:EndFunc
```



$$\blacksquare\ \text{mrot}\left(\begin{bmatrix} 4 \\ 2 \\ ^-3 \\ 0 \end{bmatrix}, \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}\right) \qquad \begin{bmatrix} 2 \\ ^-3 \\ 4 \\ 0 \end{bmatrix}$$

```
…⁻3;0],[1/2;1/2;1/2;1/2])
```