Programming Rotations in Gizmo

The orientation of the Gizmo plot is stored internally as a quaternion. A quaternion is analogous to a complex number but extended to 3 dimensions.

When you manually rotate the plot you are changing the internal quaternion. You can query it using **GetGizmo** curQuaternion.

There are a number of **ModifyGizmo** keywords that programmatically set the orientation. The setQuaternion, setRotationMatrix, and euler keywords set the orientation in absolute terms and take a quaternion parameter, a transformation matrix, or a set of Euler angles, respectively. The appendRotation keyword applies a rotation specified by a quaternion to the current orientation. The goHome keyword goes to the home orientation. The idleEventQuaternion and idleEventRotation keywords change the orientation periodically. The matchRotation keyword sets the orientation to match another Gizmo window. The syncRotation keyword syncs one Gizmo window's rotation to that of another. The stopRotation keyword stops rotation.

No matter how you set the orientation it is stored internally as a quaternion.

If you want to rotate Gizmo's display so that the X axis points to the right, the Y axis points away from you, and the Z axis points up, you need a quaternion for rotation of 90 degrees about the X axis. This can be accomplished using the command

```
ModifyGizmo setQuaternion={sin(pi/4),0,0,cos(pi/4)}
```

If you want to rotate the plot to the orientation specified by an axis of rotation and an angle about that axis, you first need to convert those inputs into a quaternion. For an axis of rotation given by Ax, Ay, Az and an angle theta in radians, the rotation quaternion consists of the four elements:

```
Qx = Ax*sin(theta/2)/N
Qy = Ay*sin(theta/2)/N
Qz = Az*sin(theta/2)/N
Qw = cos(theta/2)
```

where we normalized the rotation vector using N=sqrt(Ax^2+Ay^2+Az^2).

To compute a rotation quaternion that represents two consecutive rotations, i.e., a rotation specified by quaternion q1 followed by a rotation specified by quaternion q2, we need to compute the product quaternion qr=q2*q1 using quaternion multiplication, which is not commutative. This can be computed using the following function:

```
// q1 and q2 are 4 elements waves corresponding to {x,y,z,w} quaternions.
// The function computes a new quaternion in qr which represents quaternion
// product q2*q1.
Function MultiplyQuaternions(q2,q1,qr)
   Wave q2,q1,qr

   Variable w1=q1[3]
   Variable w2=q2[3]
   qr[3]=w1*w2-(q1[0]*q2[0]+q1[1]*q2[1]+q1[2]*q2[2])
   Make/N=4/FREE vcross=0
   vcross[0]=(q2[1]*q1[2])-(q2[2]*q1[1])
   vcross[1]=(q2[2]*q1[0])-(q2[0]*q1[2])
   vcross[2]=(q2[0]*q1[1])-(q2[1]*q1[0])
   MatrixOP/FREE aa=w1*q2+w2*q1+vcross
   qr[0]=aa[0]
   qr[1]=aa[1]
   qr[2]=aa[2]
   Variable NN=norm(qr)
   qr/=NN
End
```