NORRIS RESEARCH™
ENGINEERING ANALYSIS AND SIMULATION
Huntsville Operations • (256) 883-8626 • www.norris-research.com

QuatView™

Quaternion Transformation and Visualization

User Guide

Release 3.0.0

March 4, 2016

Contents

Program Overview

QuatView™ is a program for computing and displaying quaternions, Euler angles, and rotation matrices. These are all methods for mathematically specifying the orientation of an object in 3D space. *QuatView* includes the following components:

1. A display tool that shows how the Euler angles, quaternion component values, and rotation matrix components all vary as the 3D orientation varies.
2. A conversion tool that can convert 3D orientation parameters to and from Euler angles, quaternion, rotation matrix, and rotation axis/angle.
3. A quaternion player that reads a formatted text file containing quaternion and time values and then displays the object orientation as a function of time with a user-selected frame rate or in real-time.
4. A network server that can receive and display quaternion data that is transmitted from client applications running on remote computers.
5. A 3D viewer that displays the object orientation in window that can be resized up to full-screen.

Quaternions and their relationship to 3D orientation can be difficult to understand and visualize. The same can be said of 3D rotational transformations in general. Some of the confusion arises from the different conventions that are commonly used for mathematically describing rotation. These conventions include the 3D coordinate frame handedness (left or right handed), choice of reference frame (fixed or rotating), and the rotation matrix storage scheme (row major or column major). The purpose of *QuatView* and the associated documentation is to reduce this confusion and thereby promote greater understanding of quaternions and the mathematics of 3D rotation.

QuatView is a multi-threaded Windows application. It is written in C# and uses the Microsoft dot net runtime framework (version 3.5). It also uses OpenGL and requires a 3D graphics card for optimal performance. *QuatView* is object oriented and built around a general purpose quaternion class that encapsulates all the mathematical properties and conversion calculations. This quaternion class source code is available by email request (Email address is codeman@TobyNorris.com).

QuatView is freeware and can be freely used and distributed according to the license agreement given here. The author does reserve all rights to the program and documentation in accordance with international copyright law.

Feature Summary

QuatView has the following features and capabilities:

1. Provides an interactive GUI that allows the user to vary the Euler angles or quaternion components while continually displaying the corresponding Euler Angles, quaternion components, and rotation matrix components.
2. Provides a 3D graphical display of the orientation as the Euler angles or quaternion components are varied.
3. Allows the user to input any arbitrary vector and then observe how the vector components vary with changes in coordinate system orientation.
4. Provides an option to specify the input vector in the fixed frame coordinates (Frame A) or the rotating frame coordinates (Frame B).
5. Allows the user to convert an object orientation between any of the following four representations: Euler angles, quaternion, rotation axis and angle, and a 3x3 rotation matrix.
6. Supports the following 6 Euler angle rotation sequences which represent all the possible permutations with no repeated axes: 123,231,312,321,213,132.
7. Supports the following 6 rotation sequences that have one non-sequential repeated axis: 121,131,212,232,313,323.
8. Converts between any two different Euler angle rotation sequences.
9. Provides the following quaternion operations: complex conjugate, negate, normalize, and identity.
10. Provides the following rotation matrix operations: negate, transpose, orthonormalize, and identity.
11. Provides the following Euler angle operations: negate, reverse direction, and reset to zero.
12. Provides 3D animation of the orientation by automatically varying each Euler angle at a user-selectable speed.
13. Provides copy and paste of input data between the various GUI controls.
14. Provides selectable number of decimal places for output display.
15. Includes option to keep the Euler angles constant or keep the quaternion constant when changing the rotation sequence.
16. Includes option to plot the input vector and select the vector color.
17. Includes option to select the 3D plot object shape: cone, cylinder, cube, and sphere.
18. Includes option to select the major axis for cone and cylinder.
19. Includes option to enter the L/D ratio for cone and cylinder.
20. Includes option to display DCM orthonormality error.
21. Provides quaternion file playback capability with user-defined file format and customizable playback speed up to 60 Hz.
22. Provides real time quaternion file playback capability with the frame rate

determined from time stamps in the playback file.

23. Provides interactive quaternion file playback with pause, forward, and reverse capabilities controlled by a slider trackbar.
24. Includes a TCP server to receive and display streaming quaternion data sent from a remote client.
25. Includes a TCP client class and DLL interface that can be easily added to any dot net application to provide the capability to send quaternion data to QuatView over the internet.
26. Includes a TCP client test application that can read and transmit quaternion history files from remote computers to the *QuatView* server.
27. Provides a scalable 3D graphics window to display the rotating body orientation at sizes up to full-screen.

Future Capabilities

QuatView has the following list of planned future capabilities:

1. Provide a non-interactive command line interface to convert between any two orientation representations.
2. Provide an option to read binary playback files.
3. Include an IGES or ACIS file reader so that complex 3D surface models can be imported and displayed.
4. Provide option to switch between left and right handed coordinate system conventions.

Provide option to switch between clockwise and counterclockwise rotation conventions.

Assumptions and Conventions

QuatView uses the following rotation conventions:

- **Coordinate System Type:** right handed
- **Rotation Direction:** counterclockwise is positive
- **Transformation Direction:** from reference Frame A (fixed) to Frame B (rotating)
- **Rotation Matrix Notation:** row major with one row for each Frame B axis unit vector. Rotation matrix is post multiplied by Frame A column vector to get the rotated Frame B column vector.
- **Quaternion Notation:** [q0, q1**i**, q2**j**, q3**k**] (first component is scalar)

Other conventions in common use are left handed coordinate systems, clockwise positive rotation, column major rotation matrix format, and quaternion numbering with the scalar component last. The user must exercise care in applying the results from *QuatView* to ensure that the above conventions are consistent with the user's analysis conventions.

The relationships between the Euler angles, quaternion, and rotation matrix generally depend on the choice of transformation direction and the coordinate system type. Transformation direction really refers to the direction of the angle measurements which can be referenced to the initial frame (Frame A) or the final frame (Frame B). That is, the angles can be measured from Frame A to Frame B or from Frame B to Frame A. The rotation matrix for one choice is simply the transpose of the other choice. The quaternion for one choice is simply the complex conjugate of the other. Choice of reference frame is described further in the quaternion overview section here.

The two different rotation matrix formats (row major or column major) are the transpose of each other. The row major format is most commonly used in dynamic simulation, while the column major format is most commonly used in computer graphics. The row major format is meant to be post-multiplied by a column vector, while the column major format is meant to be pre-multiplied by a row vector as shown below where R_R is the row major form and R_C is the column major form.

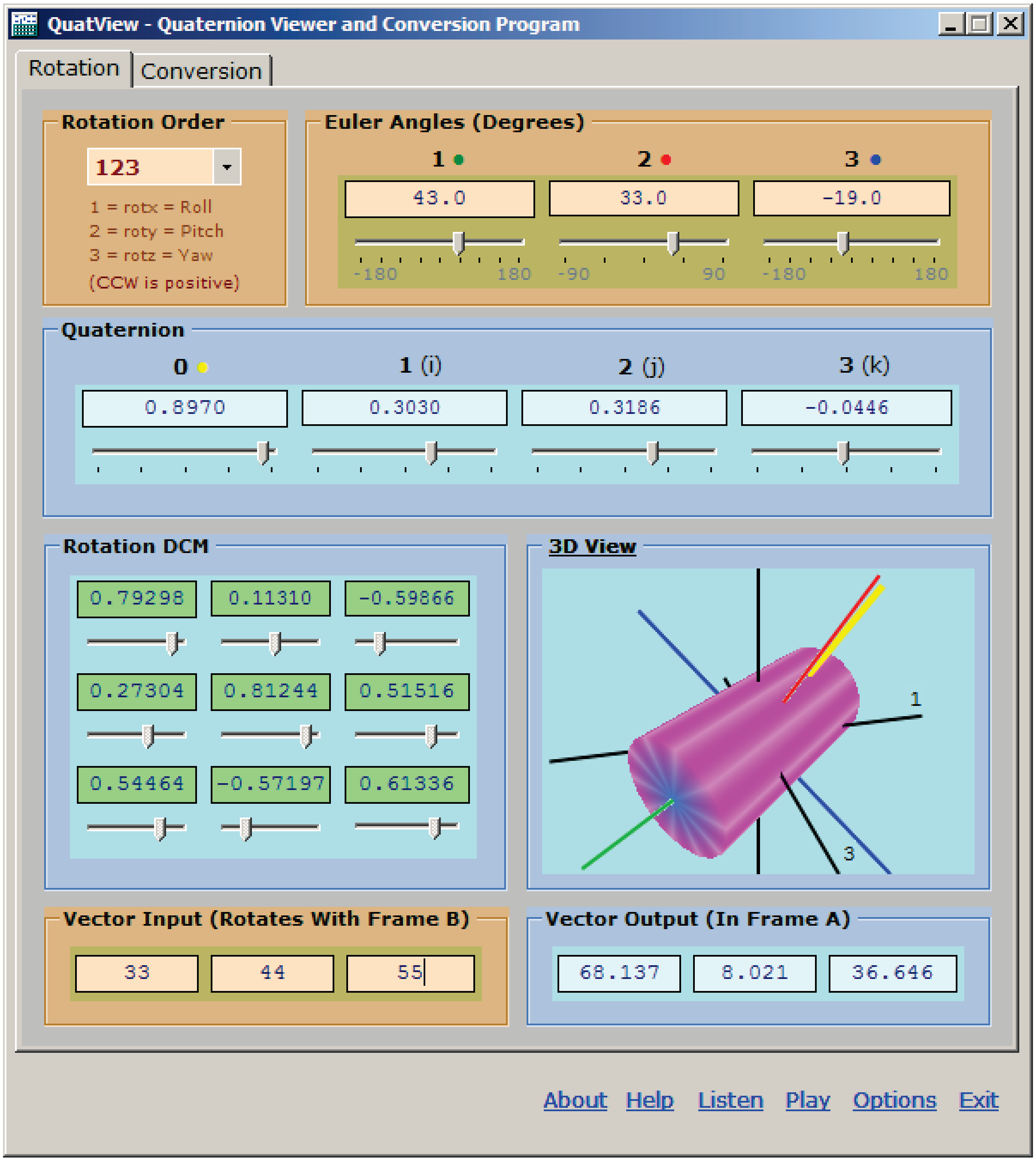
$V_F = [R_R] V_I$, row major rotation matrix (used by *QuatView* and Direct3D)

$V_F = V_I [R_C]$, column major rotation matrix (used by OpenGL)

V_I has the vector components referenced to the initial unrotated frame (Frame A), and V_F has the vector components referenced to the final rotated frame (Frame B).

GUI Description

The *QuatView* GUI has a tabbed interface with two tabs as shown in Figure 1 below. The first tab provides access to the rotation features and is described here. The second tab provides access to the conversion features and is described here.



Rotation Order

123

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees)

1 ●	2 ●	3 ●
43.0	33.0	-19.0

Quaternion

0 ●	1 (i)	2 (j)	3 (k)
0.8970	0.3030	0.3186	-0.0446

Rotation DCM

0.79298	0.11310	-0.59866
0.27304	0.81244	0.51516
0.54464	-0.57197	0.61336

Vector Input (Rotates With Frame B)

33	44	55

Vector Output (In Frame A)

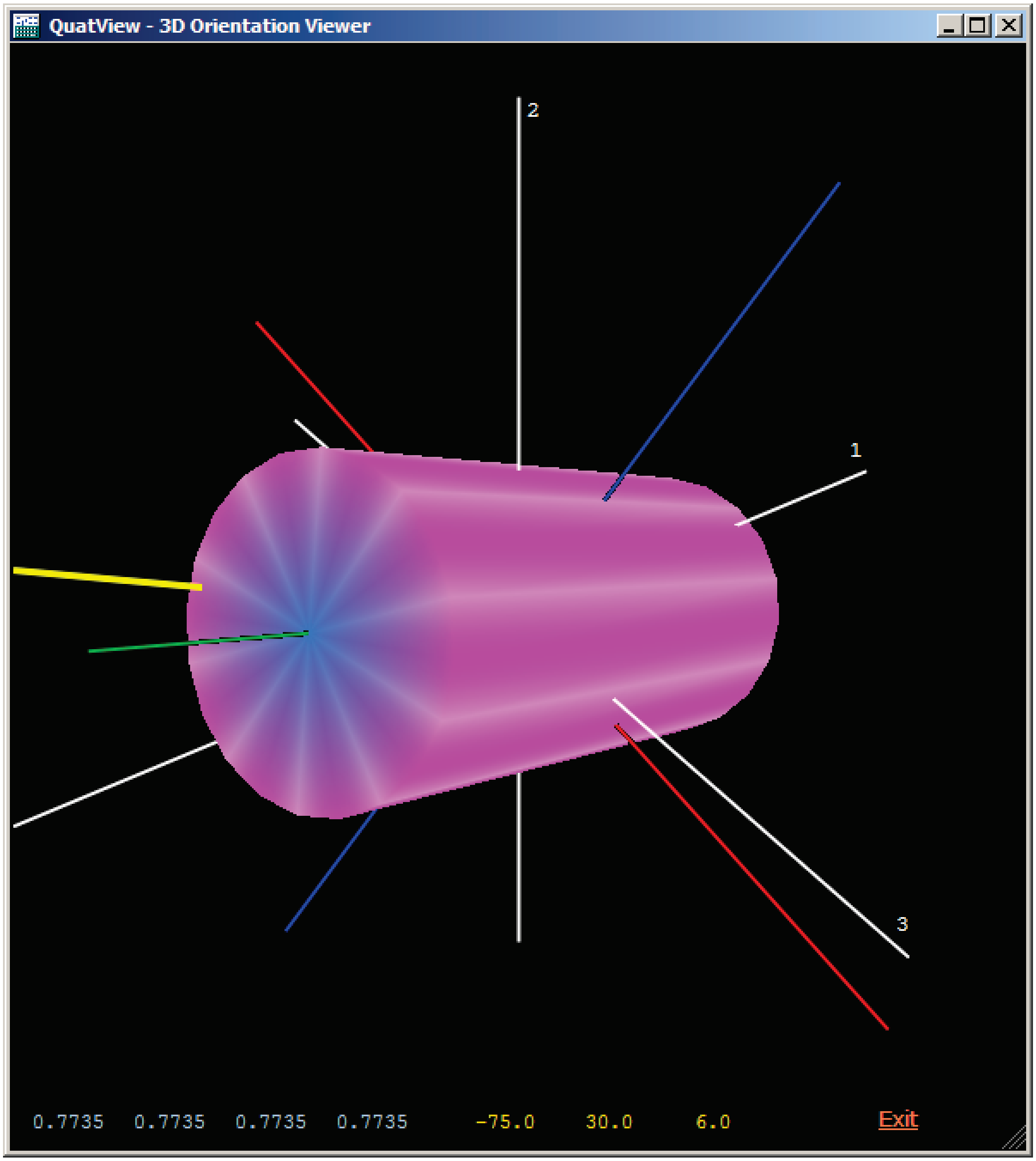
68.137	8.021	36.646

Figure 1. QuatView Main GUI

In addition to the rotation and conversion tabs, the *QuatView* GUI also has a row of link buttons across the bottom. These link buttons provide access to the following additional program features:

- **Listen** - provides TCP server capability to listen for and display quaternion data from remote clients.
- **Play** - provides playback capability for displaying a quaternion history data file interactively or in real time.
- **Options** - provides access to all program settings including animation options, numerical precision display options, 3D display options, tolerances, and choice of reference frame.
- **Help** - displays the program help file that you are currently reading.
- **About** - provides program version information and access to the *QuatView* website for updates.
- **Exit** - stops all program operations and closes the main window.

The small 3D View control on the rotation tab GUI also has a link button that, when clicked, will display a larger 3D viewer that can resized up to full screen. This larger 3D orientation viewer overlays the main GUI as shown in Figure 2 below. The features of the 3D viewer are described here.

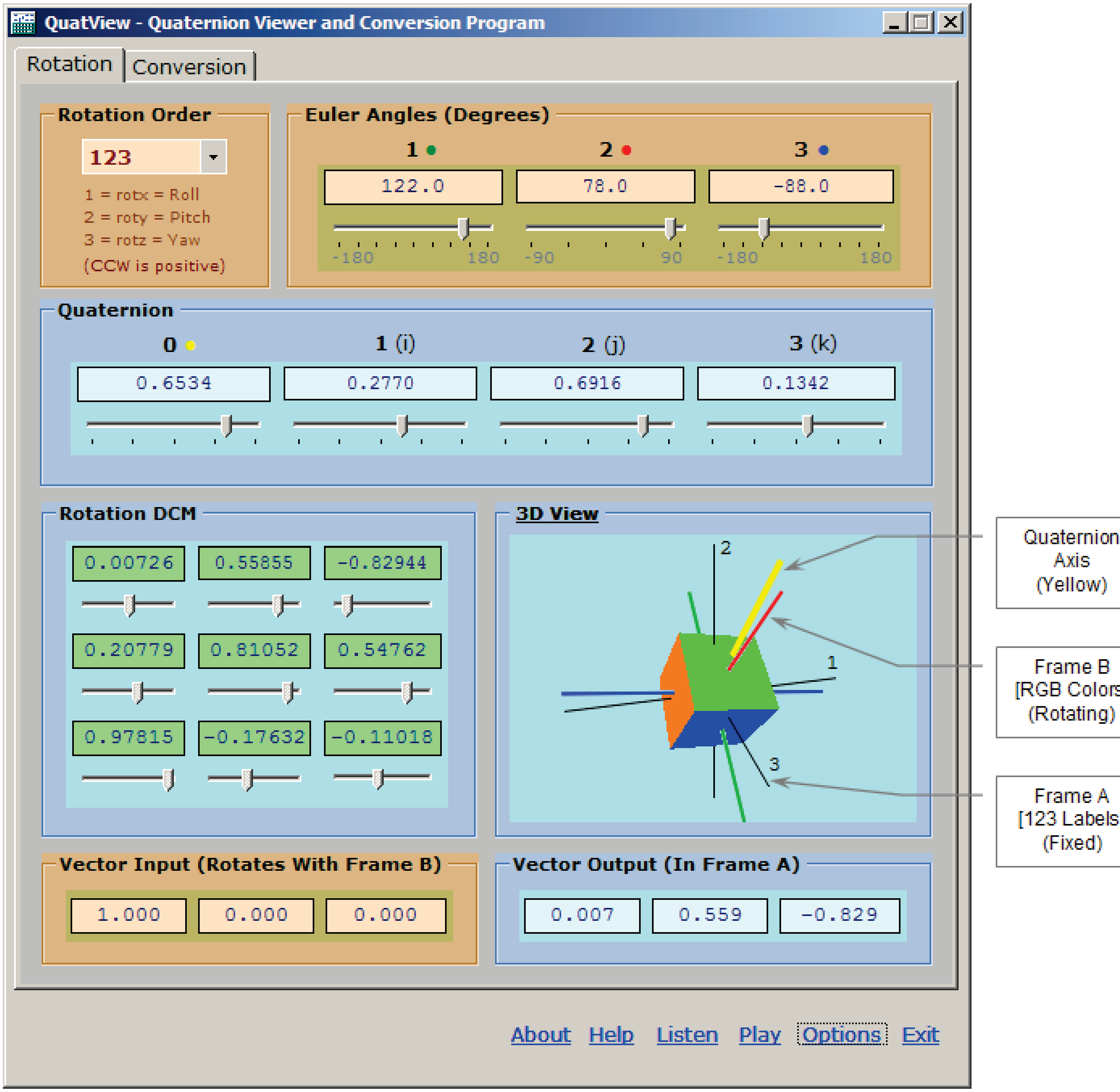


```
0.7735    0.7735    0.7735    0.7735       -75.0     30.0      6.0
```

Figure 2. QuatView 3D Viewer

Rotation Tab

Figure 3 shows the rotation tab GUI. The rotation interface has several sets of text boxes and slider controls organized into colorized groups based on whether the data is input or output. The input groups are colored brown, while the output groups are colored blue. The only input text boxes are for the Euler angles and for the vector to



Rotation Order

123

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees)

1 ●	2 ●	3 ●
122.0	78.0	-88.0

Quaternion

0 ●	1 (i)	2 (j)	3 (k)
0.6534	0.2770	0.6916	0.1342

Rotation DCM

0.00726	0.55855	-0.82944
0.20779	0.81052	0.54762
0.97815	-0.17632	-0.11018

3D View

Vector Input (Rotates With Frame B)

1.000	0.000	0.000

Vector Output (In Frame A)

0.007	0.559	-0.829

Quaternion Axis (Yellow)

Frame B [RGB Colors (Rotating)

Frame A [123 Labels (Fixed)

Figure 3. Rotation Tab GUI

be rotated. All other text boxes are read-only for displaying output. Changing the quaternion components can be done using the quaternion slider controls. Each Euler angle also has a slider control that can be used to vary the input angle. The nine rotation matrix sliders are read-only. All of the GUI slider controls automatically update to indicate the relative value of the quantity being displayed in the associated text box. The rotation quaternion is continuously normalized with any changes in the GUI input.

The upper and lower limits for the Euler angle sliders also vary depending on the rotation order choice. Those limits are set to ensure that there are no ambiguities when converting from the quaternion back to Euler angles. Euler angle singularities (i.e., gimbal lock) is properly handled by *QuatView*, and a red text box indicator is displayed whenever singularities occur as shown in Figure 4. When a singular condition does occur, then rotations about one Euler axis are frozen and the rotations about the remaining two Euler axes have an identical effect. This condition is commonly referred to as "gimbal lock", and is one major reason to avoid using Euler angles if the second angle is near ±90°. Note that the quaternion is not affected by the Euler angle singularity and all of the quaternion components can freely vary between -1 and +1 without any numerical instabilities.

The 3D orientation display shows a selectable 3D object with three colored lines that represent the rotated body (Frame B) coordinate axes. The body direction 1 axis is colored green, direction 2 is colored red, and direction 3 is colored blue. The quaternion axis is also displayed as a yellow line. The reference or fixed frame (Frame A) is displayed using black lines with the axes labeled 1, 2, and 3. The orientation of the 3D object and its Frame B axes changes as the Euler angles or quaternion components are varied by the user. The available 3D object geometries include a cube, cylinder, cone, and sphere.

The bottom left of the rotation GUI has a set of edit boxes where the user can enter a vector. This input vector is tied to Frame B by default. So as Frame B rotates the components of the vector as seen from Frame A will change. The group labeled "Vector Output (...)" then displays the rotated vector components from the perspective of the unrotated frame (Frame A). The user can also optionally specify the input vector to be tied to Frame A in which case the "Vector Output (...)" group will display the fixed vector components from the perspective of the rotated frame (Frame B). Optionally, the input vector can also be plotted in the 3D display.

The rotation tab GUI responds to the following mouse events: left mouse click, left mouse double click, right mouse click, and right mouse double click. The effects of these mouse events vary depending on the mouse location as described below.

Left Mouse Double Click

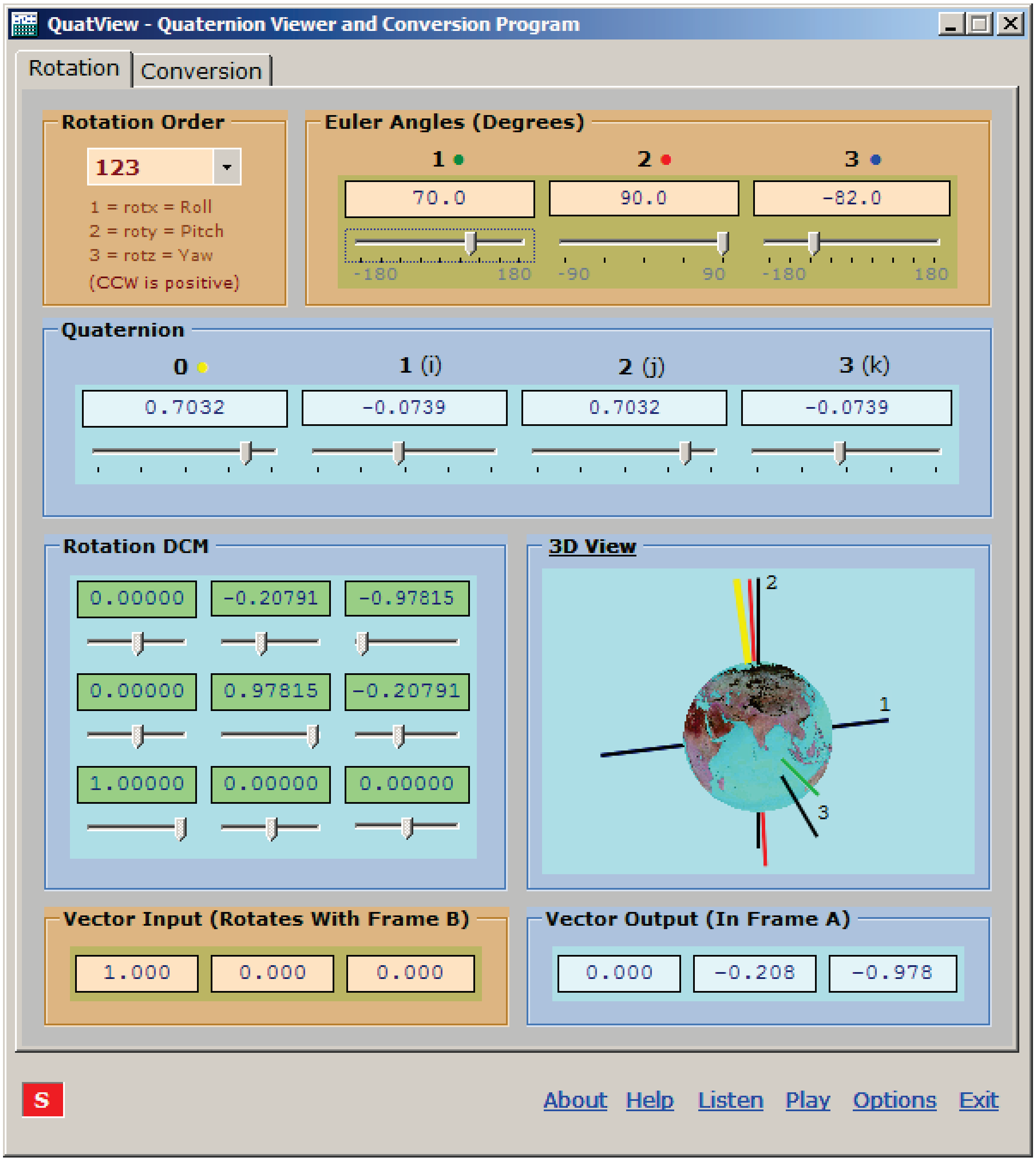
Double clicking the left mouse button over the Euler angle input group results in all three angles being reset to zero. Likewise, double clicking over the quaternion input group results in the quaternion being reset to [1,0,0,0]. Double clicking over the 3D display group resets the reference coordinate system back to its default orientation.

Right Mouse Context Menus

A right mouse context menu provides copy and paste commands for any of the input or output groups. These copy and paste commands process all of the separate text box values at once. This allows the user to copy and paste a complete quaternion, rotation matrix, or set of Euler angles at once, instead of having to copy and paste the data from each individual text box.

The right mouse context menu also provides options for performing mathematical operations on the Euler angles, quaternion, and rotation matrix. These operations include negation, transpose, complex conjugate, reverse order, animation, and reset. Figure 5 shows the Euler angle context menu, and Figure 6 shows the DCM context menu.

A main context menu is displayed if the cursor is located outside any of the input or output groups when the right mouse button is pressed. The main context menu duplicates the six link buttons along the lower right of the GUI. This can be useful if on systems with small screen displays that are too small vertically to display all the link buttons along the bottom of the GUI.



Rotation Order

123

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees)

1	2	3
70.0	90.0	-82.0

Quaternion

0	1 (i)	2 (j)	3 (k)
0.7032	-0.0739	0.7032	-0.0739

Rotation DCM

0.00000	-0.20791	-0.97815
0.00000	0.97815	-0.20791
1.00000	0.00000	0.00000

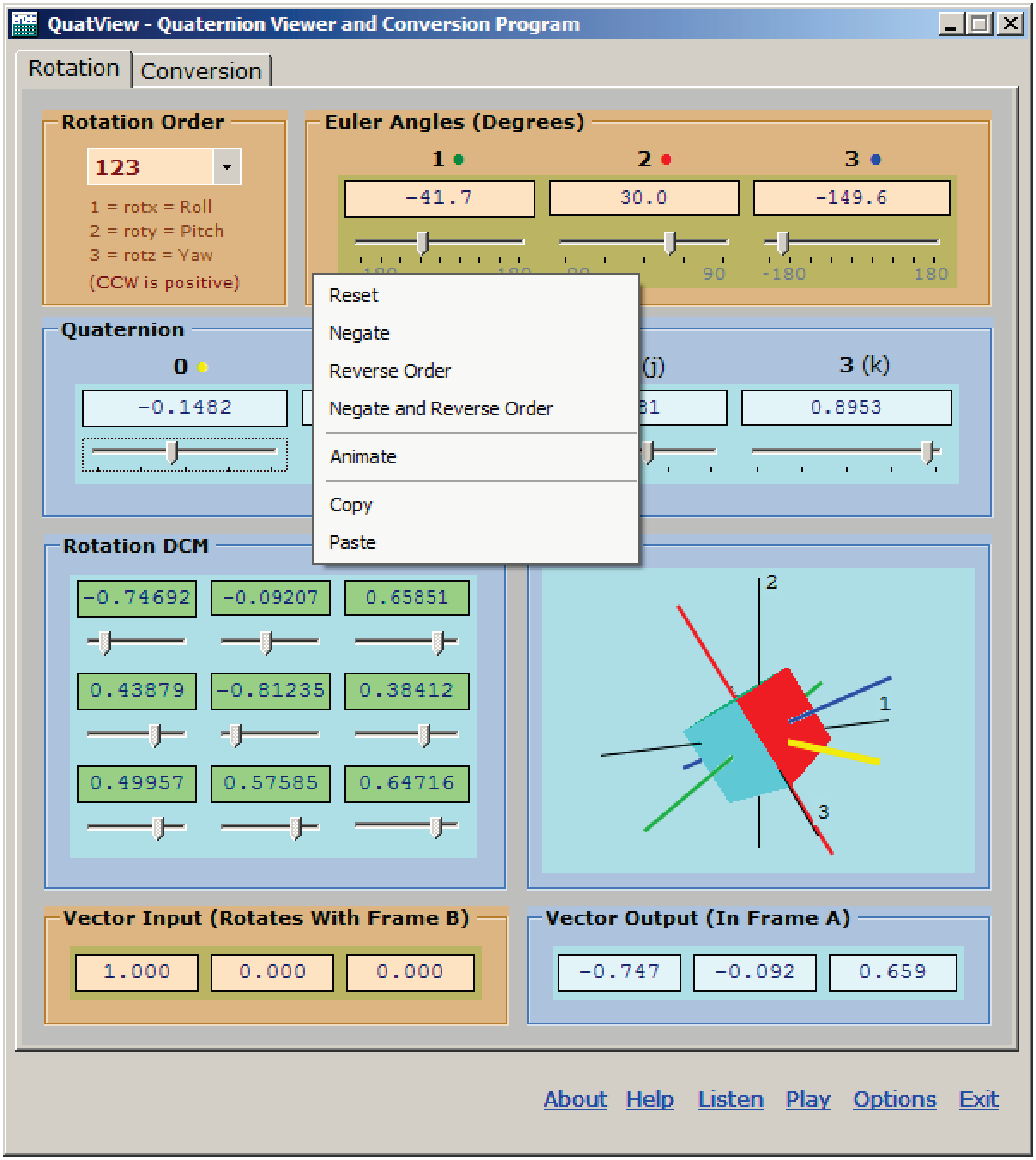
Vector Input (Rotates With Frame B)

1.000	0.000	0.000

Vector Output (In Frame A)

0.000	-0.208	-0.978

Figure 4. Rotation Tab Singularity Display



Rotation Order: 123

Euler Angles (Degrees): 1 = -41.7, 2 = 30.0, 3 = -149.6

Quaternion: 0 = -0.1482, (j) = …31, 3 (k) = 0.8953

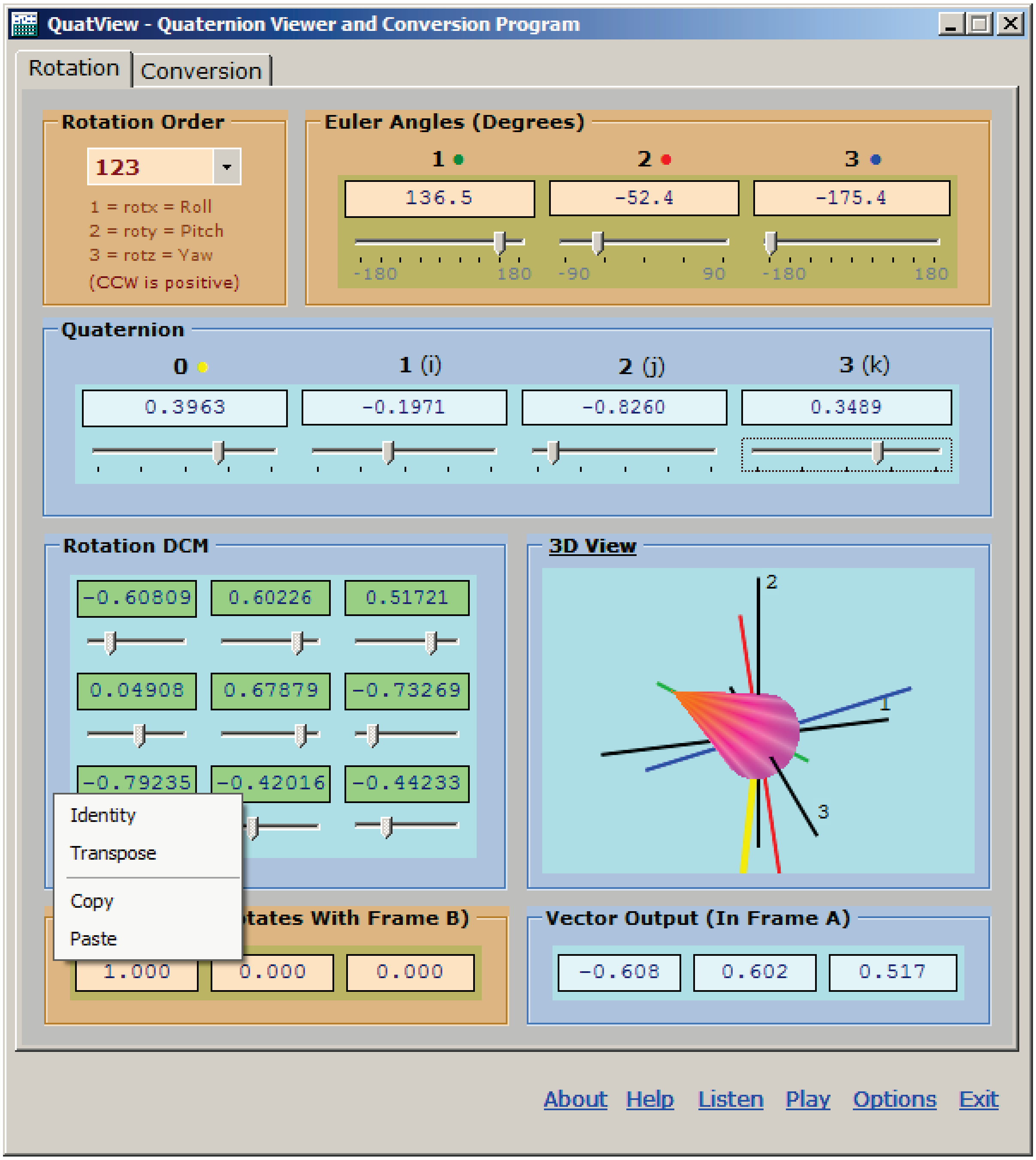
Rotation DCM:

-0.74692	-0.09207	0.65851
0.43879	-0.81235	0.38412
0.49957	0.57585	0.64716

Vector Input (Rotates With Frame B): 1.000 0.000 0.000

Vector Output (In Frame A): -0.747 -0.092 0.659

Figure 5. Euler Angle Right Mouse Context Menu



Rotation Order: 123

Euler Angles (Degrees)

1	2	3
136.5	-52.4	-175.4

Quaternion

0	1 (i)	2 (j)	3 (k)
0.3963	-0.1971	-0.8260	0.3489

Rotation DCM

-0.60809	0.60226	0.51721
0.04908	0.67879	-0.73269
-0.79235	-0.42016	-0.44233

tates With Frame B)

1.000	0.000	0.000

Vector Output (In Frame A)

-0.608	0.602	0.517

Figure 6. DCM Right Mouse Context Menu

3D Mouse Events

The 3D orientation display responds to the left mouse move and right mouse move commands. The left mouse move rotates the reference coordinate system (Frame A) with axes labeled as 1, 2, and 3. The right mouse move scales the display. The mouse wheel, if available, also scales the 3D display. Double clicking restores the orientation and scale factor back to their default values.

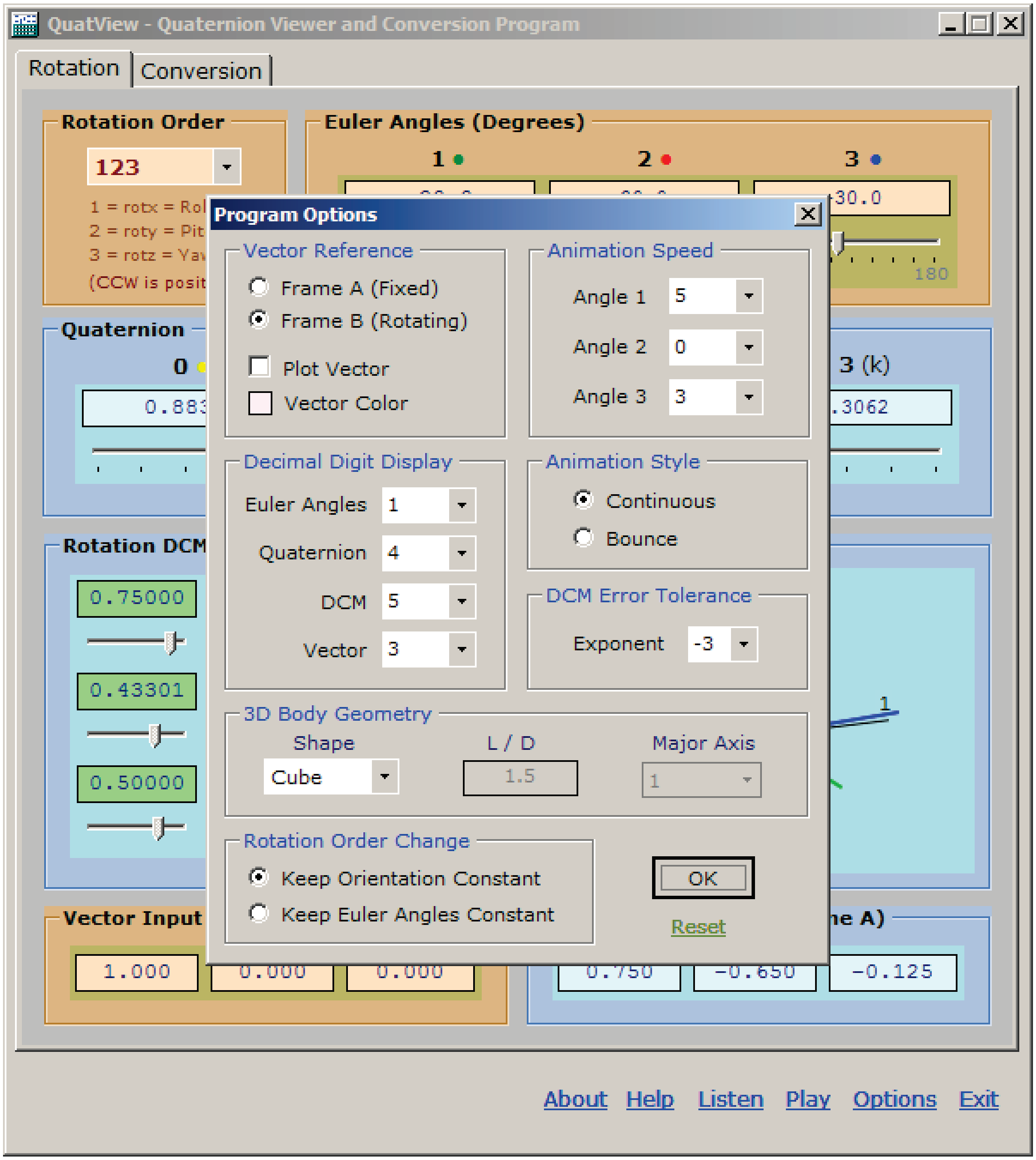
Animation

Animation is activated using the Euler angle right mouse menu. This same menu starts and stops the animation. The animation can also be stopped by clicking "Stop Animation" link on the bottom portion of the GUI outside of the two tabbed regions. The animation speed for each of the three Euler axes is selected using the Options dialog shown in Figure 7. Setting the speed to zero for a given axis disables rotation about that axis. Two animation styles are available: continuous and bounce.

Options Dialog

Pressing the Options link button will display the Options dialog as shown in Figure 7. This dialog allows the user to select the number of decimal digits to display, the animation speed for each Euler axis, the vector initial reference frame, the 3D object geometry, and whether or not to keep the Euler angles constant when changing the rotation order. An option is also available to plot the input vector and select a plot color for the vector.

All of the program options are automatically saved in a standard user configuration file so that the options are restored between program sessions. A reset link is provided just below the OK button so that the user can reset all the options to their default values.



Rotation Order: 123

1 = rotx = Rol

2 = roty = Pit

3 = rotz = Yav

(CCW is posit

-30.0

0.883

3 (k)

.3062

0.75000

0.43301

0.50000

1.000 0.000 0.000

ne A)

0.750 -0.650 -0.125

Program Options

Vector Reference

○ Frame A (Fixed)

● Frame B (Rotating)

☐ Plot Vector

☐ Vector Color

Animation Speed

Angle 1: 5

Angle 2: 0

Angle 3: 3

Decimal Digit Display

Euler Angles: 1

Quaternion: 4

DCM: 5

Vector: 3

Animation Style

● Continuous

○ Bounce

DCM Error Tolerance

Exponent: -3

3D Body Geometry

Shape: Cube

L / D: 1.5

Major Axis: 1

Rotation Order Change

● Keep Orientation Constant

○ Keep Euler Angles Constant

Figure 7. Options Dialog

Conversion Tab

Figure 8 shows the conversion tab GUI. The conversion interface has several sets of text boxes organized into colorized groups just like the rotation tab GUI. The input groups are colored brown, while the output groups are colored blue. The user selects what to convert from and what to convert to using the two combo boxes at the top left and right respectively. Once the necessary input values have been entered, then the user can press the convert button to see the conversion results. A right mouse context menu is also provided with copy and paste commands.

The conversion process works by first converting the requested input parameters into a quaternion and then converting the quaternion to the requested output parameters. This intermediate quaternion is always displayed at the bottom of the GUI as shown in Figure 8. This figure shows how the GUI is configured for computing the Eigen axis and angle from a quaternion. Many other conversion combinations are possible as shown in Figures 9 to 12.

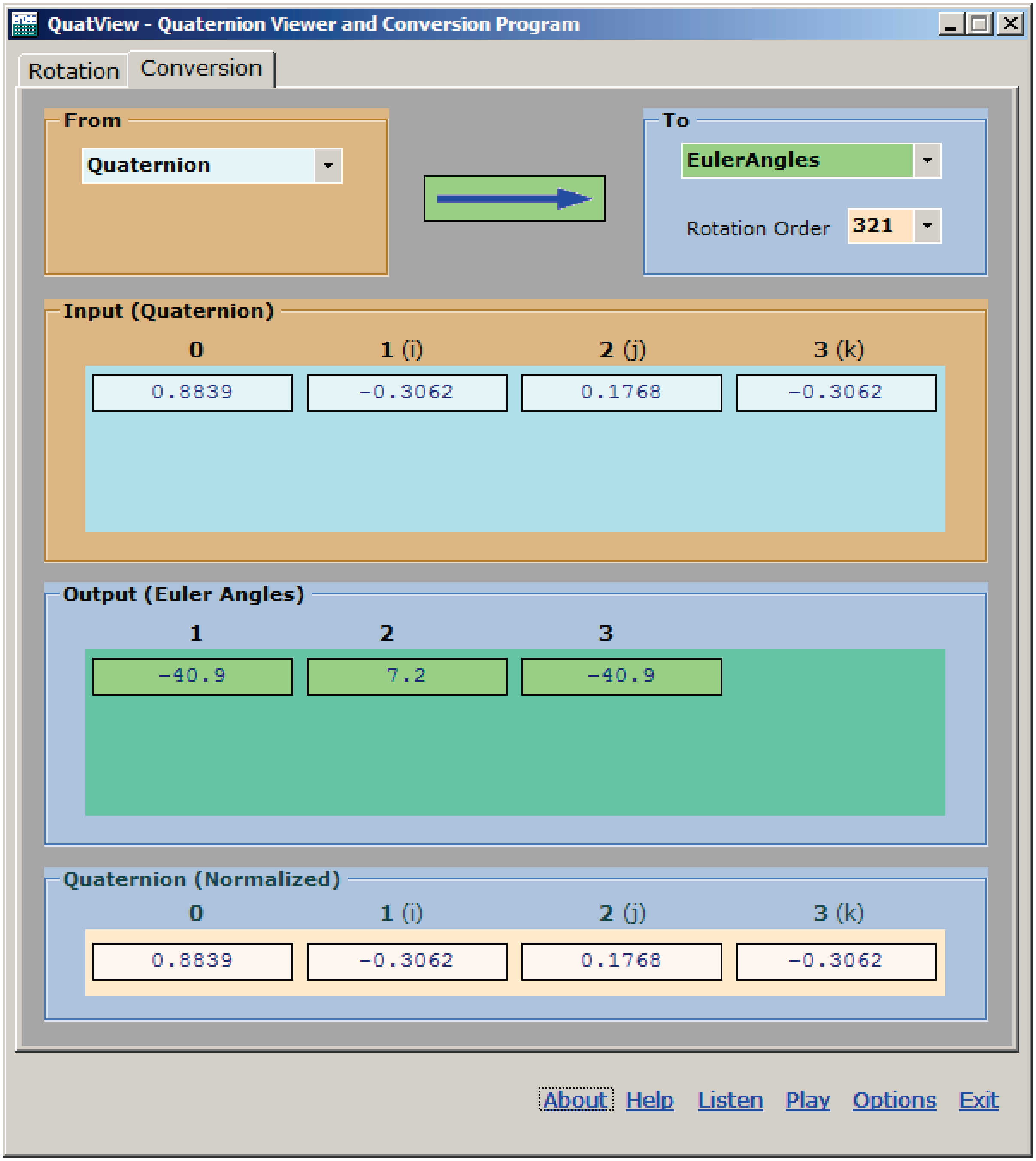
QuatView includes the capability to convert sets of Euler angles with different rotation orders as shown in Figure 9. This capability allows the user to answer questions like "What three Euler angles are required with a "121" rotation order to give me the same orientation as a given set of angles with a "321" rotation order?"

When converting from a DCM, the user must be make sure that the input DCM is indeed a valid rotation matrix. This means that the input matrix must be orthonormal and have a determinant of +1. *QuatView* will check the input matrix and report an error message if the input matrix does not satisfy these requirements.

Right Mouse Menu

A right mouse context menu provides copy and paste commands for the input or output groups. These copy and paste commands process all of the separate text box values at once. This allows the user to copy and paste a complete quaternion, rotation matrix, or set of Euler angles at once, instead of having to copy and paste the data from each individual text box.

If the DCM is being input then a special DCM context menu is available with options to display the DCM orthonormalization error and also orthonormalize the DCM components. Figure 10 shows the DCM context menu, and Figure 11 shows the orthonormality error display.



From: Quaternion

To: EulerAngles

Rotation Order: 321

Input (Quaternion)

0	1 (i)	2 (j)	3 (k)
0.8839	-0.3062	0.1768	-0.3062

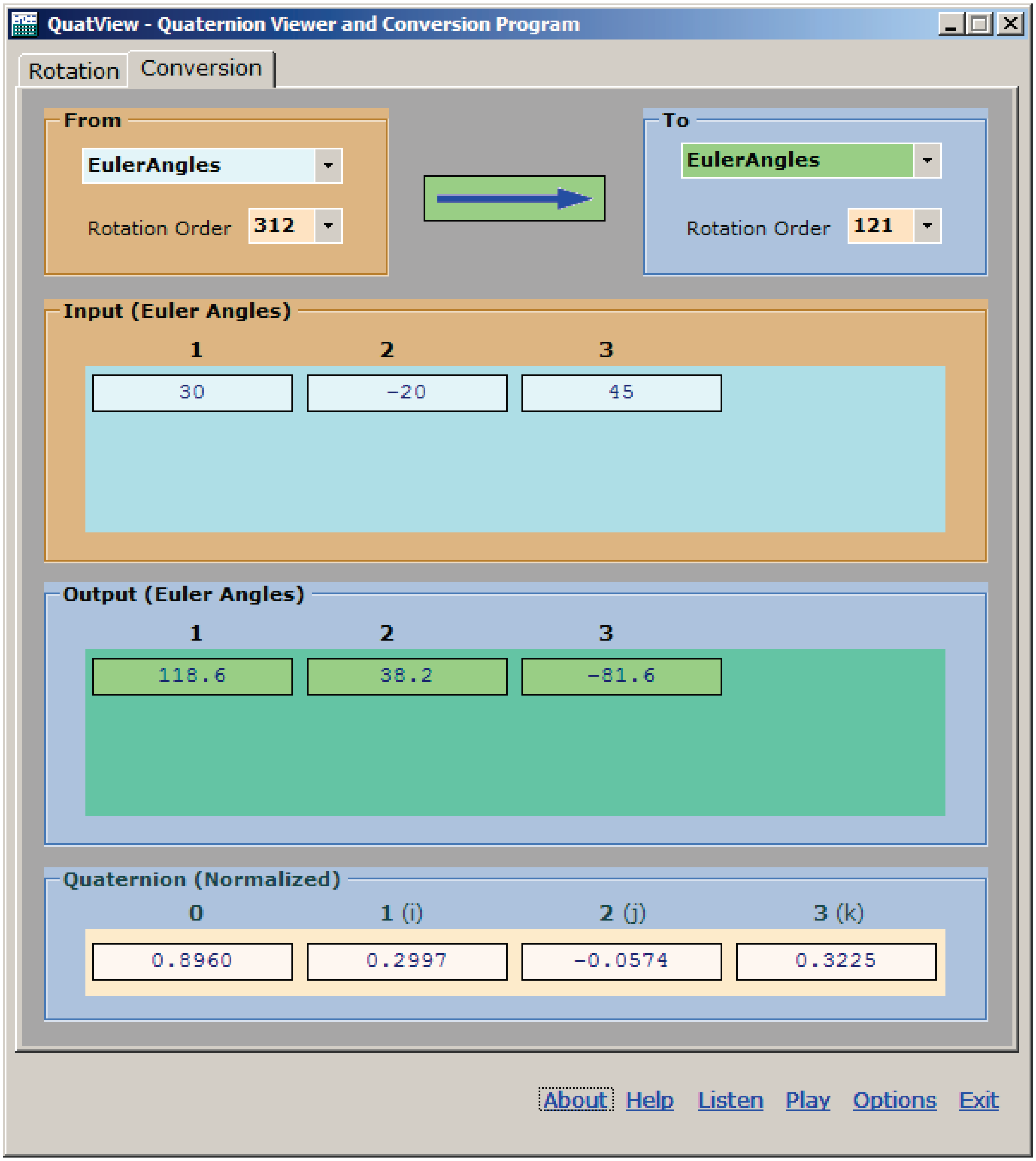
Output (Euler Angles)

1	2	3
-40.9	7.2	-40.9

Quaternion (Normalized)

0	1 (i)	2 (j)	3 (k)
0.8839	-0.3062	0.1768	-0.3062

Figure 8. Conversion Tab GUI (Quaternion to Euler Angle)



From
EulerAngles
Rotation Order 312

To
EulerAngles
Rotation Order 121

Input (Euler Angles)

1	2	3
30	-20	45

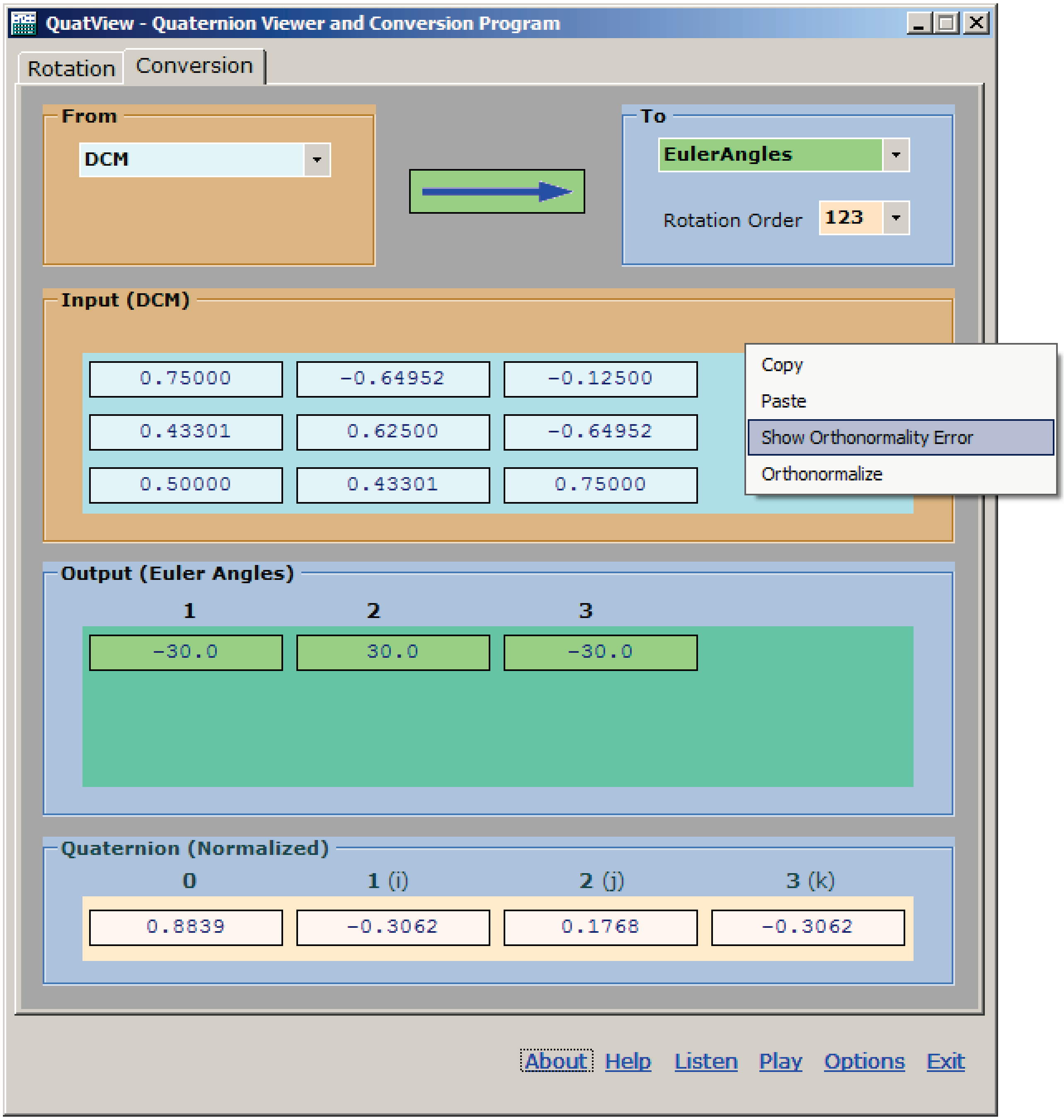
Output (Euler Angles)

1	2	3
118.6	38.2	-81.6

Quaternion (Normalized)

0	1 (i)	2 (j)	3 (k)
0.8960	0.2997	-0.0574	0.3225

Figure 9. Conversion Tab GUI (Euler Angle Rotation Order Conversion)



From: DCM

To: EulerAngles

Rotation Order 123

Input (DCM)

0.75000	-0.64952	-0.12500
0.43301	0.62500	-0.64952
0.50000	0.43301	0.75000

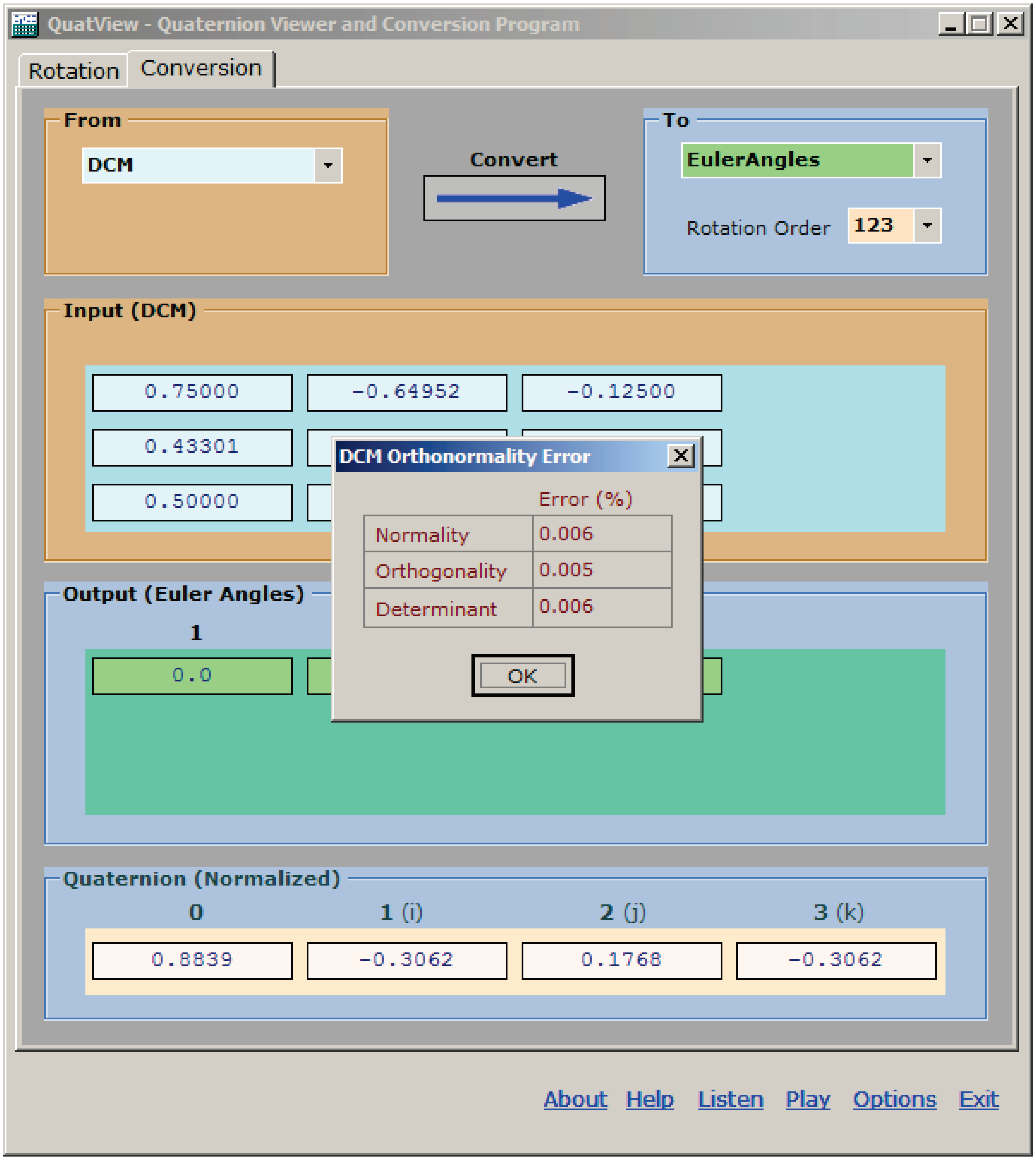
Output (Euler Angles)

1	2	3
-30.0	30.0	-30.0

Quaternion (Normalized)

0	1 (i)	2 (j)	3 (k)
0.8839	-0.3062	0.1768	-0.3062

Figure 10. DCM Context Menu For Conversion Tab



From: DCM

To: EulerAngles

Rotation Order: 123

Input (DCM)

0.75000	-0.64952	-0.12500
0.43301		
0.50000		

Output (Euler Angles)

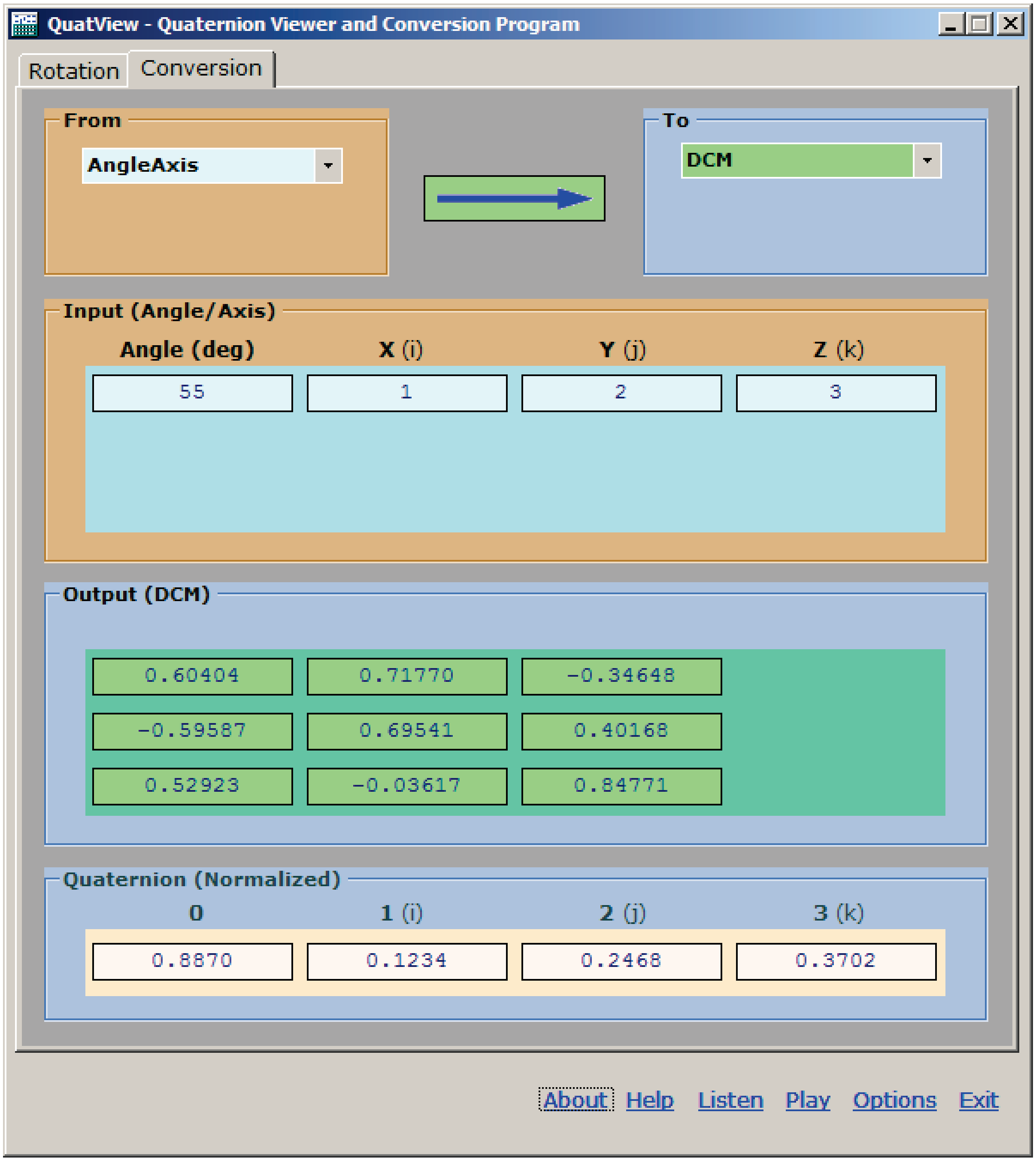
1
0.0

Quaternion (Normalized)

0	1 (i)	2 (j)	3 (k)
0.8839	-0.3062	0.1768	-0.3062

	Error (%)
Normality	0.006
Orthogonality	0.005
Determinant	0.006

Figure 11. DCM Orthonormality Error Dialog



From: AngleAxis

To: DCM

Input (Angle/Axis)

Angle (deg)	X (i)	Y (j)	Z (k)
55	1	2	3

Output (DCM)

0.60404	0.71770	-0.34648
-0.59587	0.69541	0.40168
0.52923	-0.03617	0.84771

Quaternion (Normalized)

0	1 (i)	2 (j)	3 (k)
0.8870	0.1234	0.2468	0.3702

Figure 12. Conversion Tab GUI (Eigen Angle/Axis to DCM)

Playback Mode

QuatView provides the capability to read a quaternion history file and display the contents of that file. This playback display capability allows the user to visualize the quaternion components, Euler Angles, DCM components, and object orientation as a function of time. The quaternion time history file that *QuatView* reads can be generated by a 6 DOF simulation program or by actual measurements.

Most 6 DOF simulation programs can provide an output file that contains all of the kinematic results from the simulation including the position, velocity, acceleration, orientation, rotational velocity, and rotational acceleration at each simulated time step. Certain fields in this output file will contain the four quaternion components and another field will have the time. *QuatView* allows the user to specify which fields have the quaternion components and the time. Note that the time is optional and is only required for real-time playback. *QuatView* currently requires that the time, if used, must be in units of seconds. Support for other time formats can be added in the future if needed.

Playback Dialog

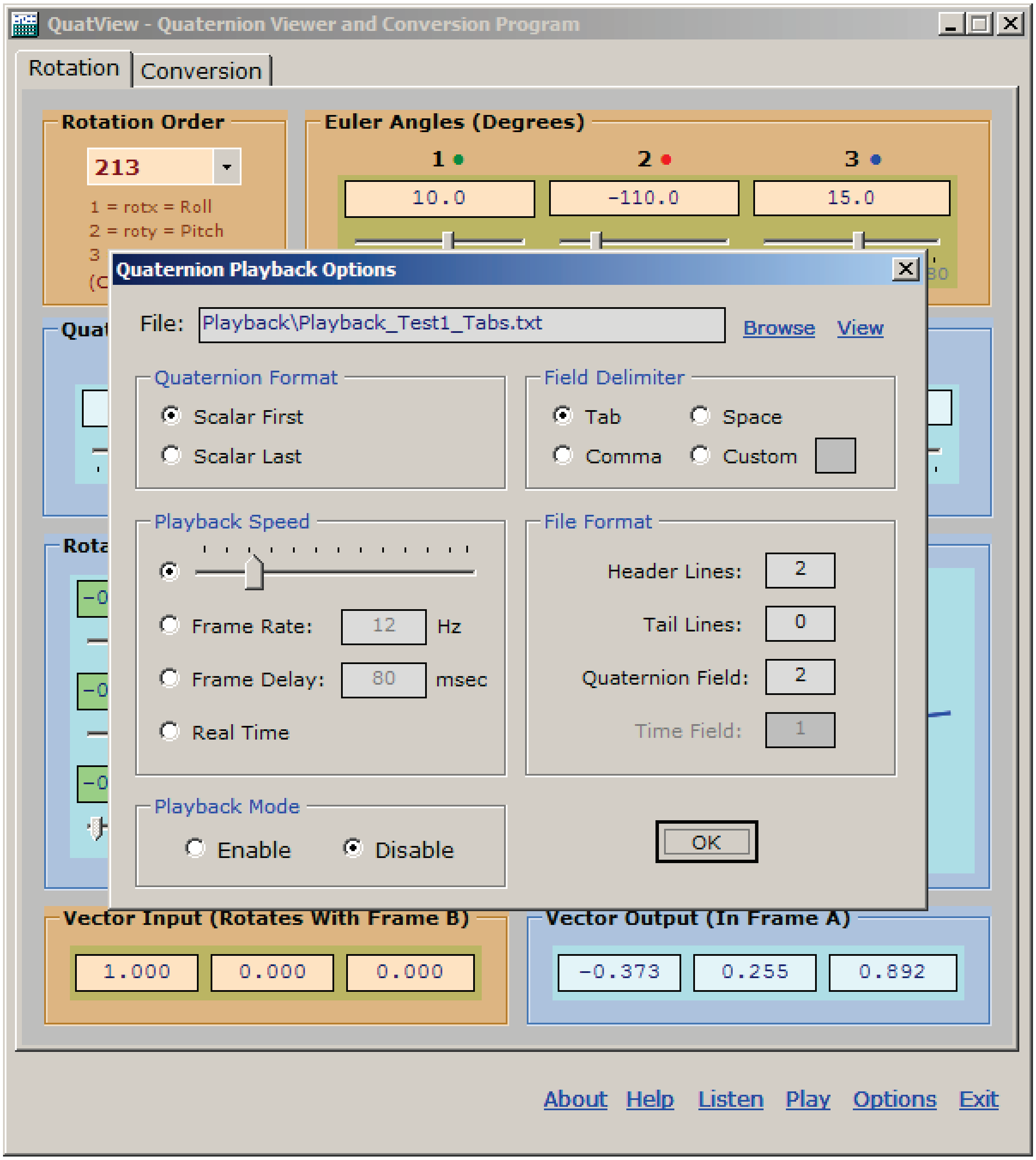
Pressing the Play link will display the Playback dialog as shown in Figure 13. This dialog allows the user to enable or disable the player. It also allows the user to specify the playback file path, the playback file format, the playback speed, and the quaternion format.

The playback file format options include the field delimiter character and the field index for the first quaternion component. Note that the field index numbering starts at 1. The number of header and tail lines in the file can also be specified. These header and tail lines will be ignored when the file is processed.

The playback speed can be entered as a frame rate, a frame delay time, or as "real time". When playing in non real-time mode, the frame rate time is constant. When playing in real-time mode, the frame rate is a variable that is determined from the time field that is read from the playback file. So a playback file that is suitable for real-time playback does require a minimum of 5 fields: four for the quaternion and one for time. A non-realtime playback file, on the other hand, requires only four fields for the quaternion.

The Playback dialog has two radio buttons to specify the quaternion format. The format can be specified as either of the two most common conventions: scalar first or scalar last.

The Playback dialog also includes two radio buttons for enabling or disabling the player. Once playback mode has been selected, *QuatView* will attempt to read the playback file and load the quaternion history into memory. If this read attempt fails for any reason, then an error message will be displayed and playback mode will be disabled. If the quaternion file is successfully loaded, then playback mode is enabled and the GUI is ready for player operation.


QuatView - Quaternion Viewer and Conversion Program
Rotation
Conversion
Quaternion Playback Options
Browse
View
OK
About
Help
Listen
Play
Options
Exit

Rotation Order: 213
1 = rotx = Roll
2 = roty = Pitch

Euler Angles (Degrees)

1	2	3
10.0	-110.0	15.0

File: Playback\Playback_Test1_Tabs.txt

Quaternion Format
● Scalar First
○ Scalar Last

Field Delimiter
● Tab ○ Space
○ Comma ○ Custom

Playback Speed
●
○ Frame Rate: 12 Hz
○ Frame Delay: 80 msec
○ Real Time

File Format
Header Lines: 2
Tail Lines: 0
Quaternion Field: 2
Time Field: 1

Playback Mode
○ Enable ● Disable

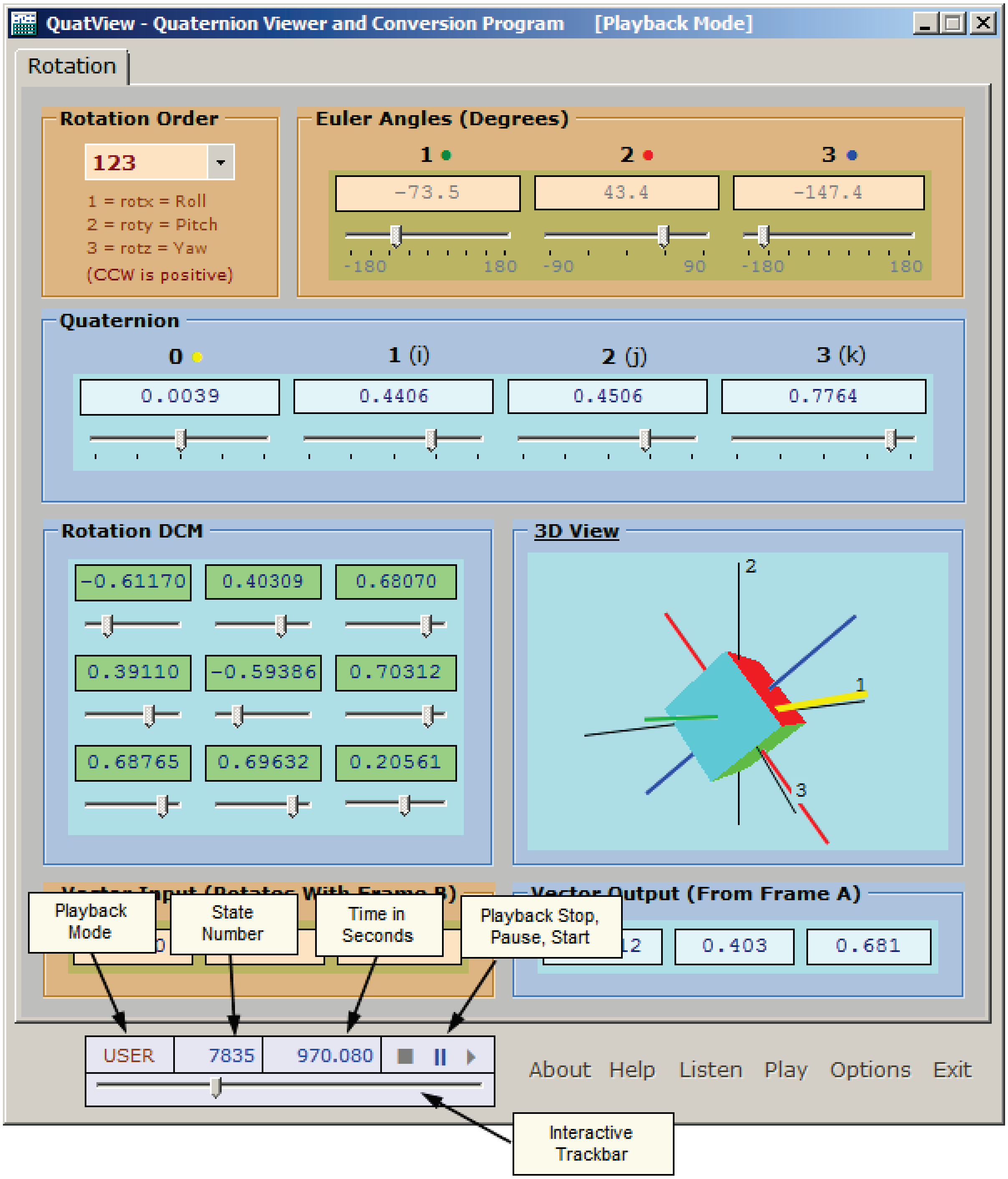
Vector Input (Rotates With Frame B): 1.000 0.000 0.000

Vector Output (In Frame A): -0.373 0.255 0.892

Figure 13. Playback Dialog

Player Operation

Once playback mode has been enabled, the GUI becomes read-only and is ready to start displaying the quaternion file contents. This means that no user input is allowed for the Euler angles or the quaternion components since the quaternion input data will come from the playback file. The conversion tab is also disabled so that no other GUI input can interfere with the playback. The only user input allowed during playback mode is from the three playback control buttons and the associated trackbar that appear at the bottom of the GUI. These playback control buttons consist of a Play button, a Pause button, and a Stop button. An elapsed time display also becomes visible along with a frame counter. Figure 14 shows how the GUI looks in playback mode.


QuatView - Quaternion Viewer and Conversion Program [Playback Mode]

Rotation Order: 123

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees)

1	2	3
-73.5	43.4	-147.4

Quaternion

0	1 (i)	2 (j)	3 (k)
0.0039	0.4406	0.4506	0.7764

Rotation DCM

-0.61170	0.40309	0.68070
0.39110	-0.59386	0.70312
0.68765	0.69632	0.20561

Vector Output (From Frame A): 0.403, 0.681

USER | 7835 | 970.080

Figure 14. GUI With Playback Mode Enabled

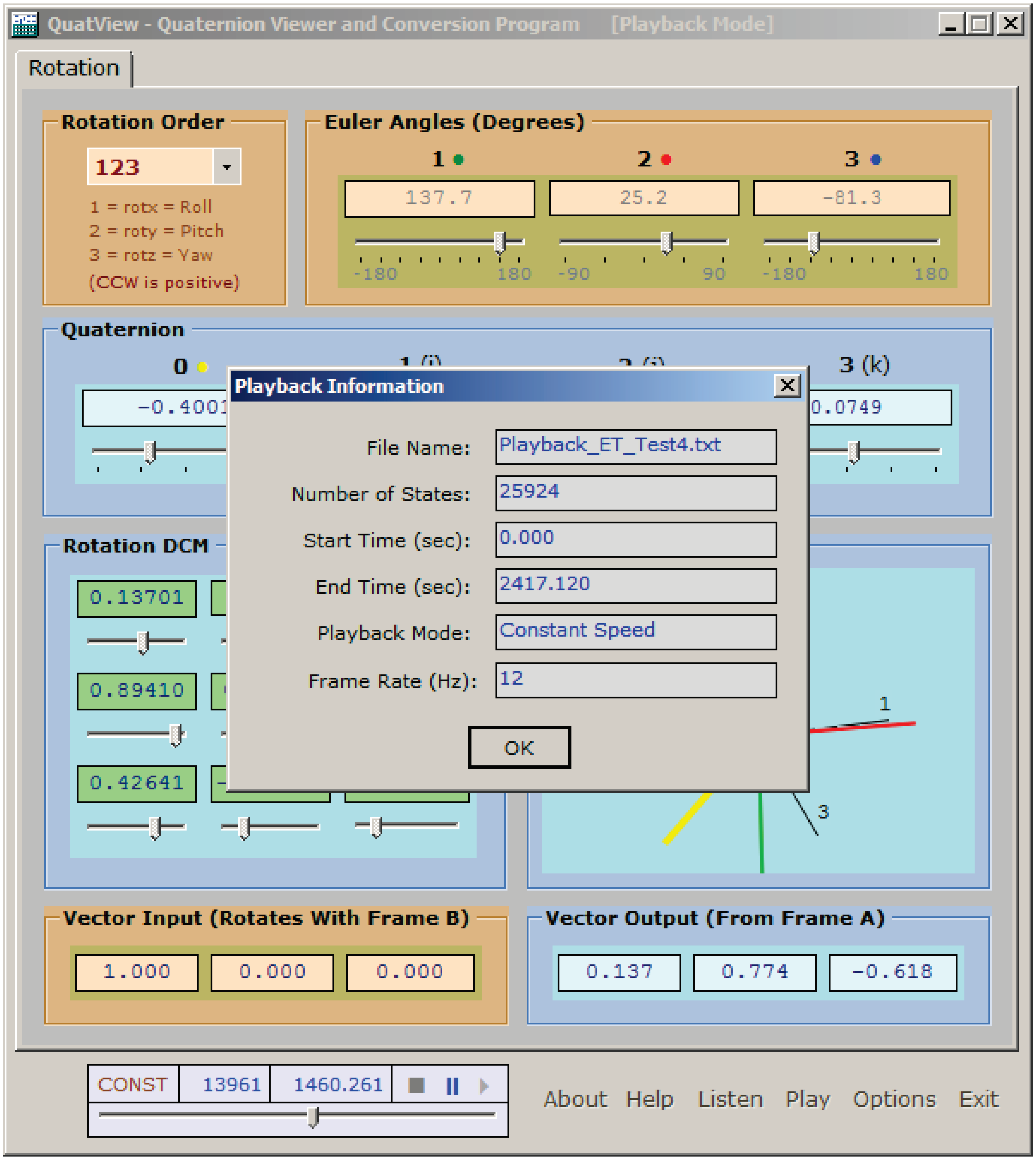
Playback begins once the Play button is pressed. During playback the overall progress is shown graphically by the trackbar and the frame counter. The 3D orientation display gets updated with each playback frame along with all the quaternion, Euler Angle, and DCM displays. Pressing the Stop button stops the playback and resets the elapsed time counter back to zero. Pressing the Pause button stops the playback but does not reset the time counter back to zero. Once paused, any subsequent press of the play button will resume playback from the paused frame. Note that during playback the elapsed time display will always show the total elapsed time which includes any pause time.

QuatView keeps track of the timing error during playback, and if this error gets above 50 milliseconds, then the elapsed time display will turn red. This means one of two things depending on whether the playback is at a constant frame rate or the playback is in real-time. If the playback is real-time, then a red time display means that the playback time is behind the desired elapsed time. In this case the player will speed up in an attempt to "catch up" and re-synchronize with real time. If the player does catch back up to real time, then the display goes back to its normal color. If the playback is at a constant frame rate, then a red display means that the desired frame delay time was not achieved for that particular frame. In this case no timing changes are made, and if the next frame delay time is back within the 50 ms error tolerance, then the display color will go back to normal. So for constant frame rate playback, only individual frames can be in error, whereas for real-time playback it is the cumulative elapsed time that can be in error.

QuatView is currently limited to maximum a playback speed of 60 Hz which corresponds to a frame time of 0.017 sec. This limitation is an operating system limitation due to the resolution of the Windows system timer. Higher playback speeds can be achieved by using the low level Windows multimedia timer which can deliver resolutions as low as 10 microseconds. This high speed playback capability may be added in the future, if necessary. However, no matter how low the timer resolution is, if the operating system is not a real-time operating system, then there is no guarantee that a given frame will execute on time.

The quaternion player also provides an interactive capability that allows the user to select any particular state by manually sliding the trackbar. With this capability, the user can take control of the player at any time, whether the player is stopped, paused, or playing. Once the user clicks on the slider control, the player will pause and give control to the user who can then move back and forth between the frames interactively. The user can then restart the player from this paused state and the player will continue to play at the proper frame rate. Note that once a real-time playback has been interrupted by the user, it must restart from frame 1. However, if a constant rate playback gets interrupted then it can restart from the current selected frame.

In summary, the quaternion player operates in one of three modes: constant rate, real time, or user. As Figure 14 shows, the current playback mode is displayed in the bottom left corner of the GUI as *CONST*, *REAL*, or *USER* respectively. Double clicking on this mode indicator will display a dialog box that summarizes all of the playback settings as shown in Figure 15. The constant and real-time modes are selected using the Playback dialog, while the user mode becomes active any time the user takes control of the player by clicking on or sliding the trackbar.



Rotation Order: 123

Euler Angles (Degrees): 1: 137.7 2: 25.2 3: -81.3

Quaternion: 0: -0.4001 3 (k): 0.0749

Rotation DCM: 0.13701 0.89410 0.42641

Vector Input (Rotates With Frame B): 1.000 0.000 0.000

Vector Output (From Frame A): 0.137 0.774 -0.618

CONST 13961 1460.261

Playback Information

File Name: Playback_ET_Test4.txt

Number of States: 25924

Start Time (sec): 0.000

End Time (sec): 2417.120

Playback Mode: Constant Speed

Frame Rate (Hz): 12

Figure 15. Playback Information Dialog

Server Mode

QuatView includes a TCP server mode that provides the capability to communicate with client programs running on remote computers. The *QuatView* server can receive and display a serialized binary data stream of quaternion history data from any client program that transmits the data in the recognized format. Server mode is similar to playback mode except that playback mode reads and displays the quaternion data from a local file while server mode reads and displays the quaternion data from a remote computer via a TCP network connection.

Server Dialog

Pressing the Listen link will display the Server dialog as shown in Figure 16. This dialog allows the user to enable or disable the server. It also allows the user to specify the server IP address and port, the IO timeout, and a directory for the optional log file.

IO Timeout

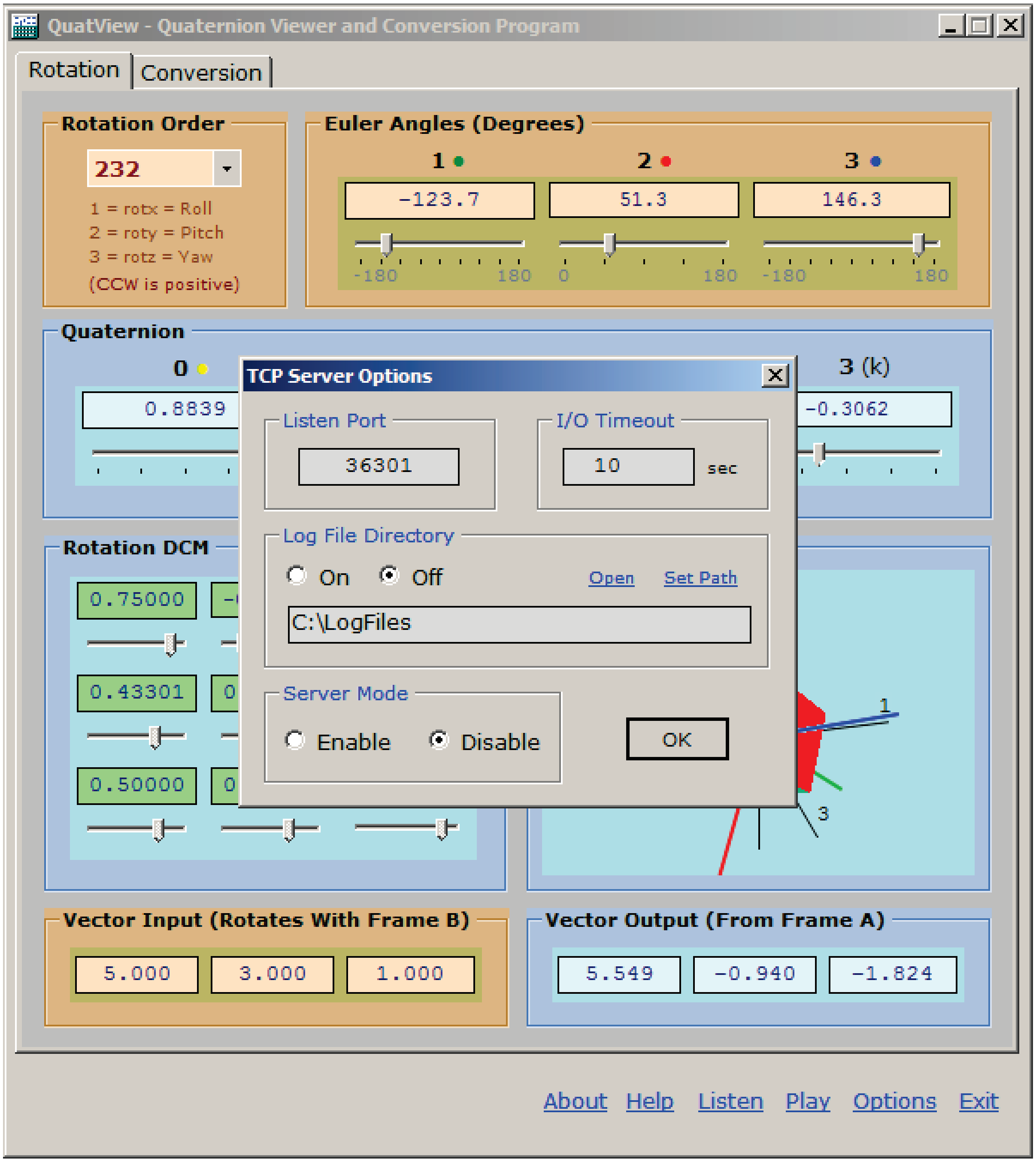
The IO timeout is the amount of time that the server will wait between quaternion messages before giving up and assuming a bad connection. This timeout only applies after the first successful quaternion message is received. The *QuatView* server will wait indefinitely for a client connection, and it will wait indefinitely for the first message from a connected client. However, after the first message is received, it will wait no longer than the IO timeout value before subsequent messages. The default timeout value is 10 seconds. This timeout value can be decreased to as low as 1 second or increased to any higher value. When the timeout between quaternion messages is exceeded the server will behave as if it received a stop message from the client. It will then reset its message counter to zero, close the optional log file, and then wait again indefinitely to receive another first message from the client.

Log File

The On/Off radio buttons on the Server dialog allow the user to request that a log file be written to record all of quaternion data received by the Server. The user can specify the directory for the log file by typing in a directory path or pressing the "Set Path" link button. The log file itself will be automatically named according to the following naming convention:

QuatView_YYYYMMDD_HHMMSS.log

where YYYY, MM, and DD are the year, month, and day respectively, and HH, MM, and SS are the hour minute and second respectively in universal coordinated time. The log file is formatted as 5 tab-delimited columns with a single header line that specifies the *QuatView* version along with the date and time stamp for the file creation.



Rotation Order: 232

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees): 1 ● -123.7, 2 ● 51.3, 3 ● 146.3

Quaternion: 0 ● 0.8839, 3 (k) -0.3062

Rotation DCM: 0.75000, 0.43301, 0.50000

Listen Port: 36301

I/O Timeout: 10 sec

Log File Directory: ○ On ● Off Open Set Path

C:\LogFiles

Server Mode: ○ Enable ● Disable

Vector Input (Rotates With Frame B): 5.000 3.000 1.000

Vector Output (From Frame A): 5.549 -0.940 -1.824

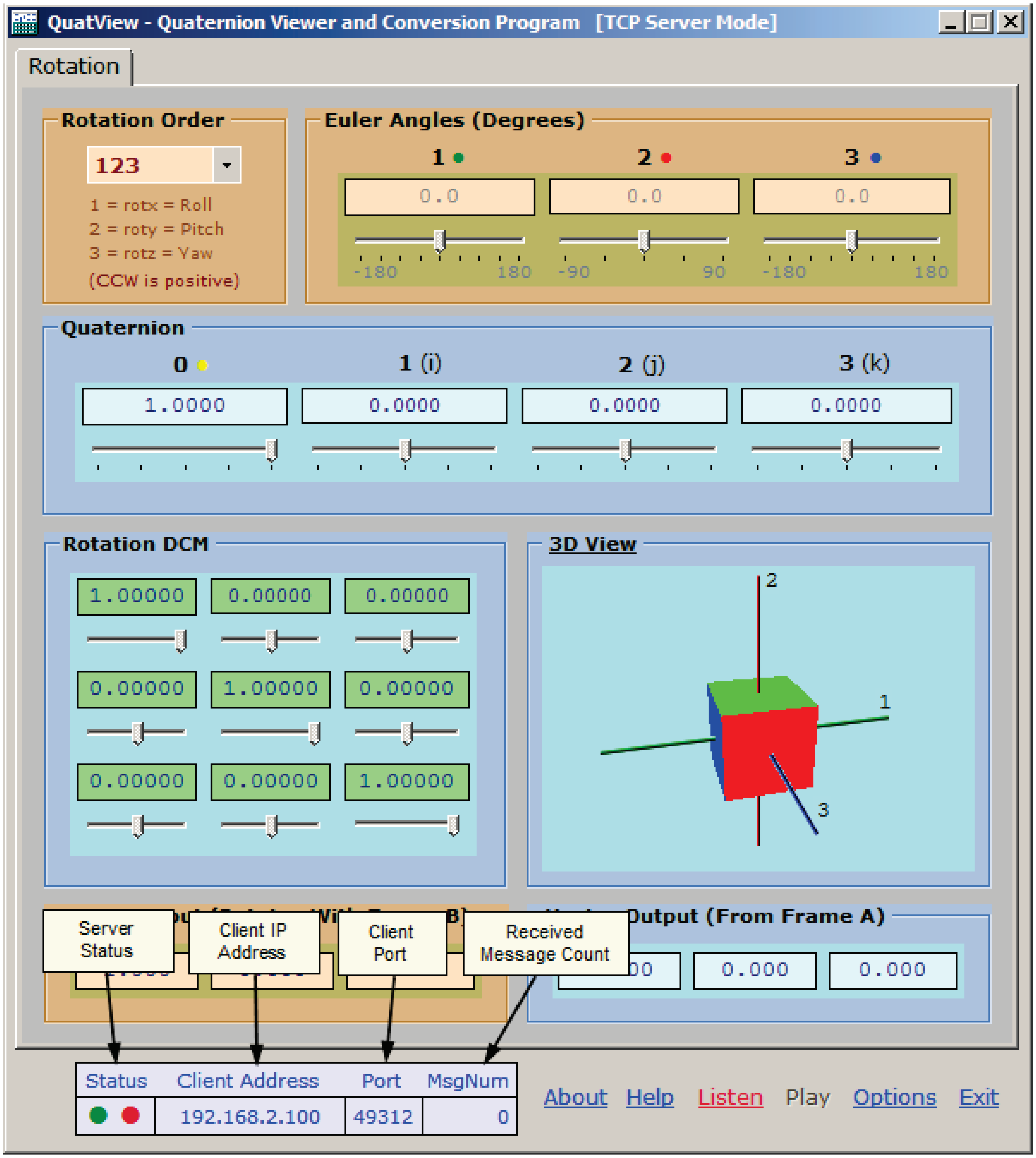
Figure 16. Server Dialog

Server Operation

Once server mode has been enabled, the GUI becomes read-only and is ready to start displaying the quaternion data received from the client application. This means that no user input is allowed for the Euler angles or the quaternion components since the quaternion input data will come from the client network stream The conversion tab is also disabled so that no other GUI input can interfere with the server. Figure 17 shows how the GUI looks in server mode.

The server mode GUI has four new controls at the bottom left that display the server state, client IP address and port, and a running count of the number of messages received.

The server state is indicated by the two lights at the lower left. The left light indicates the connection status and the right light indicates the reception status. If the connection light is green then an active client connection exists. If the reception light is green, then the server is actively receiving quaternion messages. The message format is defined by the QuatMessage class that is included with the client class library described here.



Rotation Order

123

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees)

1	2	3
0.0	0.0	0.0
-180 … 180	-90 … 90	-180 … 180

Quaternion

0	1 (i)	2 (j)	3 (k)
1.0000	0.0000	0.0000	0.0000

Rotation DCM

1.00000	0.00000	0.00000
0.00000	1.00000	0.00000
0.00000	0.00000	1.00000

Output (From Frame A): 0.000 0.000

Status	Client Address	Port	MsgNum
● ●	192.168.2.100	49312	0

Figure 17. GUI With Server Mode Enabled

Client Class

QuatView includes a TCP client class that can be used by other dot net applications to send quaternion data to the *QuatView* server. This class encapsulates all the functionality that is necessary for communicating with *QuatView* over a TCP network connection. The class is named *QuatViewClient*, and it is provided in the DLL file named QuatViewClient.dll that is located in the *QuatView* installation directory. The user simply adds a reference to this DLL file to use this class in a Visual Studio project. The class is contained within the namespace called NRC_QuatViewClient. So any C# client application will also need a corresponding "using NRC_QuatViewClient" statement to make use of the class. A description of the class properties, methods, and events is given below.

Properties

`IPEndPoint ServerIP`	Stores server IP address and port number
`string TransmitFilePath`	Path to the quaternion file to be transmitted
`int TransmitDelay`	The delay time between transmissions
`string ErrorMsg`	Readonly. Client error message.
`string StatusMsg`	Readonly. Client status message
`QuatMessage CurrentMessage`	Readonly. Current message being transmitted.
`int NumMessagesSent`	Readonly. Value that indicates the number of messages transmitted.
`bool IsConnected`	Readonly. Flag that indicates if a connection has been made.
`bool IsTransmitting`	Readonly. Flag that indicates if transmission is in progress.

Methods

`bool Connect(IPEndPoint serverAddress)`
Establishes connection to QuatView server. Returns true on success.
`bool Connect()`
Establishes connection to QuatView server. Returns true on success.
`bool Disconnect()`
Disconnects from QuatView server. Returns true on success.
`bool TransmitFileStart(string FilePath)`
Begins quaternion file transmission. Returns true on success.
`bool TransmitFileStart()`
Begins quaternion file transmission. Returns true on success.
`bool TransmitFileStop()`
Stops quaternion file transmission. Returns true on success.
`bool TransmitQuaternion(cQuaternion quatOut)`

```
Transmits a single quaternion object.  Returns true on success.
```

Events

```
event TransmitEvent(object sender, TransmitEventArgs eventArgs)
   This event fires after every quaternion message transmission.
```

The QuatViewClient class incorporates several other classes and enumerations that warrant further description. Those are the QuatMessage, cQuaternion, and TransmitEventArgs classes and the TransmitMsgType enumeration. The QuatMessage class is used to store the quaternion messages that are sent to the *QuatView* server. The TransmitEventArgs class is used to store the event data for the TransmitEvent event. This class has a single field that stores the message type. The TransmitMsgType enumeration listed below gives the various message types. The message type is transmitted in the state number field of the QuatMessage class. If the state number field has a positive number then that field indicates the state number. But if this field is negative then it indicates the message type. This is how commands such as stop and disconnect as well as error conditions are sent to the server.

The cQuaternion class is the class that stores the quaternion components and does all the mathematical calculations related to quaternions, vectors, Euler angles, and rotation matrices. This class source code and documentation are available separately upon request as described here.

QuatMessage Class Listing

```
[Serializable]
public class QuatMessage
{
      public uint Num;   // state number or message code if negative
      public double Time;  // time
      public double q0,q1,q2,q3; // quaternion

      public QuatMessage()
      {
            Num = 0;
            Time = 0.0;
            q0 = 1.00;
            q1 = 0.00;
            q2 = 0.00;
            q3 = 0.00;
      }
}
```

TransmitEventArgs Class Listing

```
public class TransmitEventArgs : EventArgs
{
   public TransmitMsgType MsgType;

   public TransmitEventArgs()
   {
      // set the default type
      MsgType = TransmitMsgType.Sent;
   }

   public TransmitEventArgs(TransmitMsgType msgType)
   {
      MsgType = msgType;
   }

}
```

TransmitMsgType Enumeration

```
public enum TransmitMsgType
{
      Sent       =  0,  // message was sent
      StopUser   = -1,  // premature stop (user commanded)
      StopError  = -2,  // premature stop (error condition)
      StopEnd    = -3,  // normal stop at end of playback file
      Disconnect = -4,  // disconnect

};
```

The procedure for using the QuatViewClient class is to first call one of the connect methods to establish the connection and then call one of the transmit start methods to transmit a quaternion history file or a single quaternion. The user can subscribe to the TransmitEvent event to receive notification whenever a transmission has occurred. When the input file or list of quaternions to transmit has ended the user can then call the TransmitFileStop method. This leaves the *QuatView* server in a connected state and ready for another new quaternion data stream to start. Or, if there is no more data to transmit, the user can call the Disconnect method which will terminate the connection to the *QuatView* server so that other clients may then connect.

The QuatViewClient class currently supports only a single file format: five tab delimited columns with the time in column 1 and the quaternion in columns 2 through 4. Other file formats can be easily added in the future, if needed.

3D Viewer Mode

In addition to the small 3D orientation display on the rotation tab, *QuatView* also provides a larger 3D viewer window that can be scaled to any size up to full screen. This larger 3D viewer mode is enabled by clicking on the 3D View link on the rotation tab GUI.

Figure 18 shows how QuatView looks in the 3D viewer mode. The local and global axes are labeled and colored just like the smaller 3D view on the rotation tab. The background color is hardwired to black. The current quaternion components are displayed along the lower left, and the current Euler angles are displayed along the lower right. The exit link button at the lower right turns off the 3D viewer mode and returns to the standard rotation tab GUI.

Once the 3D viewer mode is enabled, the user can rotate and scale the image using the left mouse button and mouse wheel respectively. Pressing the right mouse button displays the context menu shown in Figure 19. This context menu allows the user to turn animation on and off, access the options dialog, and copy the current 3D image to the Windows clipboard.

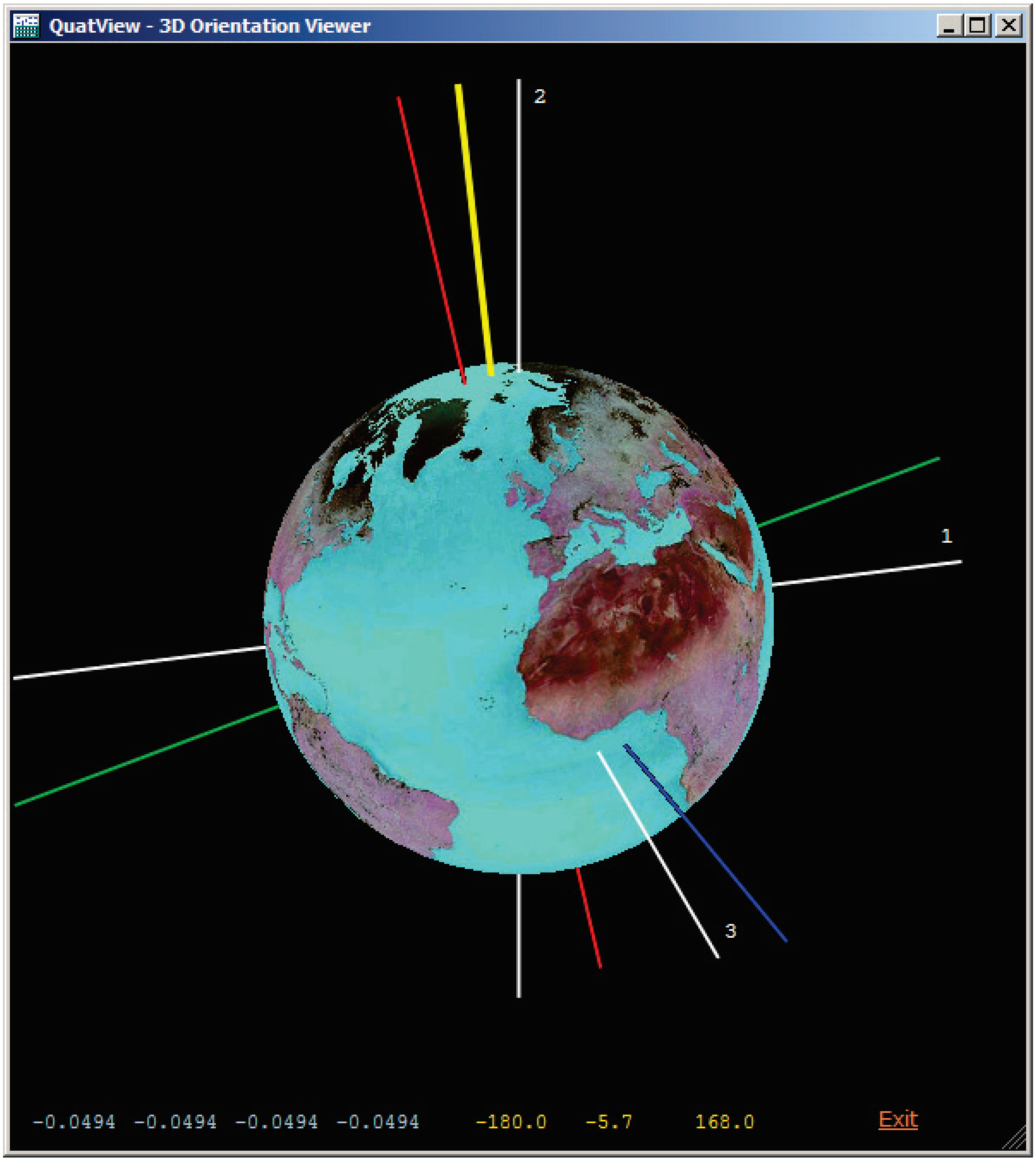


Figure 18. 3D Viewer Mode

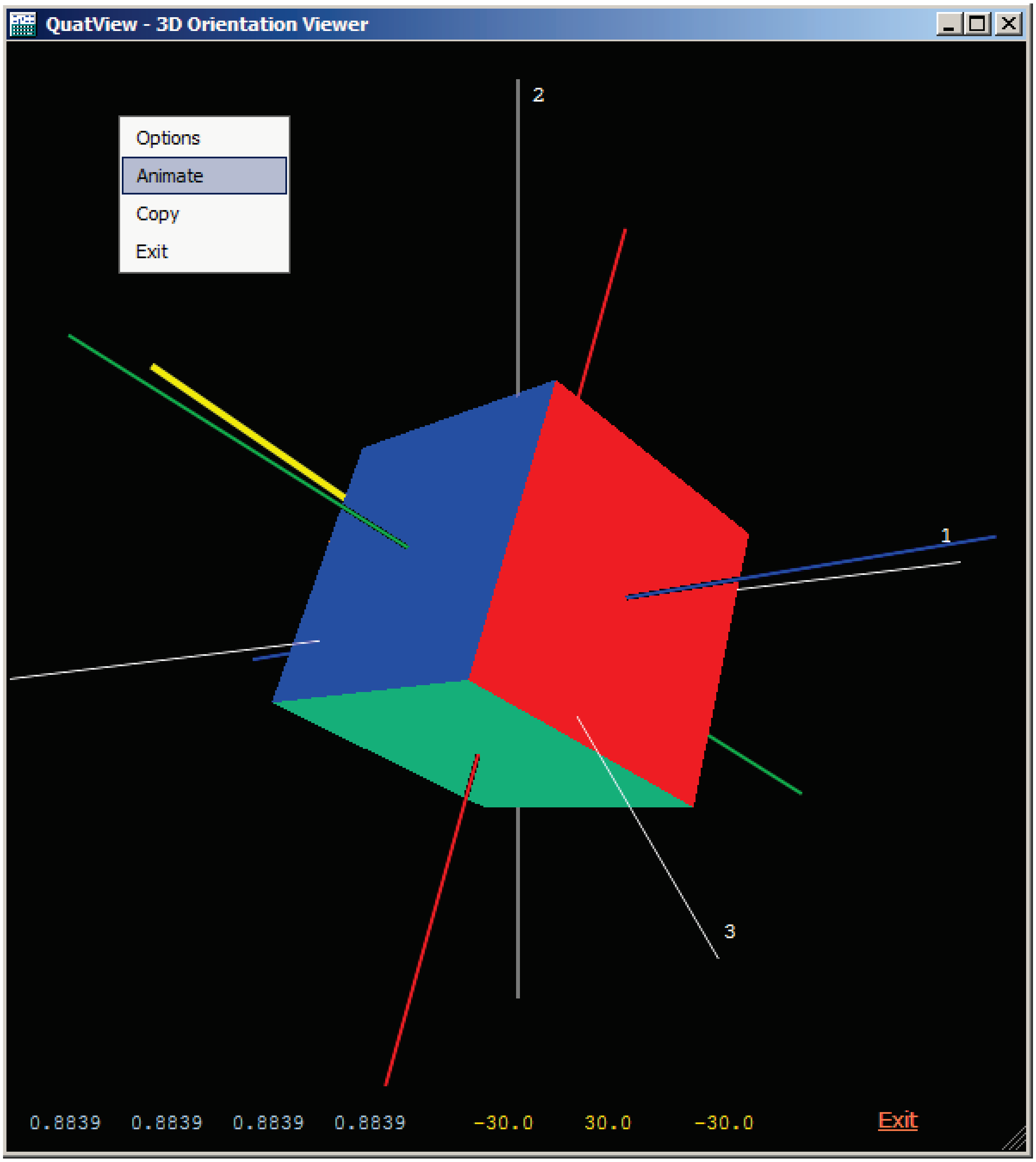


Figure 19. 3D Viewer Mode Right Mouse Context Menu

Quaternion Background

A quaternion is a mathematical abstraction just like a real number, complex number, tensor, or manifold. A quaternion is basically a type of complex number that consists of four values, one of which is real, while the other three are imaginary. As such, quaternions are referred to as hypercomplex numbers. So, just like a complex number consists of two parts (real and imaginary), the hypercomplex number consists of four parts (1 real and 3 imaginary). The mathematical properties of hypercomplex numbers are such that quaternions, when normalized, are ideally suited for representing the orientation of an object in 3D space and for transforming the orientation coordinates between any two 3D reference frames. Another name for the normalized quaternion is the versor.

Several textbooks and internet references are available that provide detailed descriptions of quaternions, including mathematically rigorous derivations of all the relevant quaternion rules and formulas. A short list of quaternion references is given here.

There are two common conventions for representing a quaternion: one has the real component first and the other has the real component last. QuatView uses the former convention with the following notation for the quaternion, q.

$$q = q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k} \equiv [q_0, q_1, q_2, q_3]$$

where,

q_0 = quaternion real component

q_1, q_2, q_3 = quaternion imaginary components

$\hat{i}, \hat{j}, \hat{k}$ = imaginary basis vectors with $\hat{i}^2 = \hat{j}^2 = \hat{k}^2 = -1$

A quaternion can be normalized just like a four dimensional vector by dividing each of the four components by the Euclidean norm. The Euclidean norm is defined as the square root of the sum of the squares of the four quaternion components.

Physical Interpretation

The normalized quaternion can be interpreted physically as an angle combined with a 3D axis about which a single rotation defines the final orientation of a frame relative to some initial reference frame. According to Euler's rotation theorem, the orientation of any object in 3D space can be defined by a single rotation about a single axis. Figure 20 shows a rotation axis, $\hat{n}$, and a rotation angle, θ, about this axis.

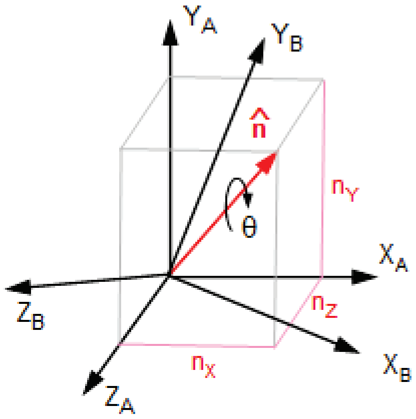


Figure 20. Quaternion as rotation axis and angle

A quaternion can then be defined that represents a rotation from Frame A to Frame B. That is, a quaternion can be defined that represents the orientation of Frame B with respect to Frame A. The components of this quaternion are related to the rotation axis and angle in Figure 20 according to the following formulas:

$$\begin{aligned} q_0 &= \cos(\theta / 2) \\ q_1 &= n_x \sin(\theta / 2) \\ q_2 &= n_y \sin(\theta / 2) \\ q_3 &= n_z \sin(\theta / 2) \end{aligned} \tag{1}$$

where,

q_0 = quaternion real component
q_1, q_2, q_3 = quaternion imaginary components
θ = rotation angle
n_X, n_Y, n_Z = rotation axis components

The scalar component of the quaternion gives the rotation angle, and the three imaginary components, when normalized, give the direction of the rotation axis.

The rotation angle is then readily computed from the arc cosine of the quaternion scalar component as

$$\theta = 2 \operatorname{a\cos}(q_0) \tag{2}$$

The quaternion has its own special set of mathematical operations so that it can be used very effectively for simulating and predicting rotational motion.

Mathematical Formulas

Quaternion Product

The quaternion product is perhaps the most important quaternion operation since it is used extensively in vector transformation and rotational kinematics. It is also used to compute the quaternion derivative. Multiplying two quaternions together is a fairly complex process compared to vector multiplication. Separating each quaternion into scalar and vector components, the formula for multiplying two quaternions can be expressed as

$$q_1 \otimes q_2 = \left(s_1 s_2 - V_1 \bullet V_2,\ s_1 V_2 + s_2 V_1 + V_1 \times V_2\right) \quad (3)$$

where,

$q_1 = (s_1, V_1)$
$q_2 = (s_2, V_2)$
s_i = scalar (real) component for quaternion i
V_i = vector (imaginary) components for quaternion i
$\otimes$ = quaternion product
$\bullet$ = vector dot product
$\times$ = vector cross product

This formula shows that the quaternion product requires five individual products including a vector dot product and a vector cross product.

Quaternion Conjugate

The quaternion conjugate is similar to the conjugate for standard complex numbers except that the quaternion has three imaginary numbers instead of only one. So the quaternion conjugate is computed by reversing the sign of the three imaginary components.

$$q* = (s,-V_i) \quad (4)$$

where,

$q*$ = quaternion conjugate
$q = (s,V_i)$ **=** quaternion
s = real component
V_i = imaginary components (i = 1,2,3)

Quaternion Magnitude

The quaternion magnitude is a scalar value that is equal to the four dimensional Euclidean norm which is simply the square root of the sum of the squares of the four components.

$$\overline{q} = \sqrt{q \otimes q*} = \sqrt{s^2 + V_1^2 + V_2^2 + V_3^2} \quad (5)$$

where,

q = quaternion, (s,V_i)
$\overline{q}$ **=** quaternion magnitude
$q*$ **=** quaternion conjugate
s = real component
V_i = imaginary components (i = 1,2,3)
$\otimes$ = quaternion product

Quaternion Inverse

The quaternion inverse is computed by dividing its conjugate by its magnitude squared as shown in the following formula:

$$q^{-1} = \frac{q*}{\overline{q}^2} = \frac{q*}{q \otimes q*} \quad (6)$$

where,

q = quaternion
q^{-1} = quaternion inverse
$\overline{q}$ **=** quaternion magnitude
$\otimes$ = quaternion product

Normalized Quaternion

A quaternion must be normalized to represent a valid 3D orientation or to compute valid vector rotations. Normalized quaternions, or versors, have the special property that their conjugate is equal to their inverse. When quaternions are used in numerical simulations to represent orientations, the quaternion must be renormalized at frequent intervals to account for the accumulation of numerical round off error. A normalized quaternion (or unit quaternion) is computed by simply dividing the quaternion by its magnitude.

$$Q_N = \frac{q}{\overline{q}}, \quad Q_N^{-1} = Q_N^{*} \tag{7}$$

where,

Q_N = normalized quaternion
q = unnormalized quaternion
$\overline{q}$ **=** quaternion magnitude

Pure Quaternion

A pure quaternion is defined as a quaternion with a zero for the scalar value (q_0=0). A standard 3D vector can be readily stored in a pure quaternion. Pure quaternions can then be used to rotate vectors or transform the vector coordinates between different rotated reference frames.

Vector Transformation

The components a vector can be transformed between any two different reference frame orientations using the following formula:

$$V_B = Q \otimes V_A \otimes Q^{*} \tag{8}$$

where,

V_A = vector (pure quaternion) in Frame A coordinates
V_B = vector (pure quaternion) in Frame B coordinates
Q = normalized quaternion for orientation of Frame B relative to Frame A.
Q^* = conjugate (or inverse) of Q
$\otimes$ = quaternion product

Quaternion Time Derivative

If the quaternion, q, represents the orientation of a rotating Frame B relative to some other reference Frame A, then the quaternion rate or time derivative of this quaternion can be expressed in terms of the corresponding angular velocity vector, ω, that defines the rate at which Frame B rotates relative to Frame A. This angular velocity vector can be expressed in terms of Frame A or Frame B coordinates as ω_A or ω_B respectively. The formula for the quaternion derivative is given by

$$\dot{q} = 0.5\left(\omega_A \otimes q\right) \quad \text{or} \quad \dot{q} = 0.5\left(q \otimes \omega_B\right) \tag{9}$$

where,

$\dot{q}$ = quaternion rate of Frame B relative to Frame A

q = quaternion for Frame B orientation relative to Frame A

ω_A = angular velocity of Frame B in Frame A coordinates

ω_B = angular velocity of Frame B in Frame B coordinates

$\otimes$ = quaternion product

Verification Tests

Test Problem 1

Rotation About Z Axis

Verification Test 1 is the most simple type of 3D rotational transformation with a single rotation about the Z axis. This test has two cases: Case 1 computes the components of a fixed vector in a rotated reference frame. Case 2 computes the components of a rotated vector given the unrotated vector components in the same reference frame. Exact solutions for this problem are done manually and then compared against the *QuatView* results. The manual solutions are done using two methods: the rotation matrix and the rotation quaternion. The *QuatView* results agree precisely with both manual solutions which verifies that the *QuatView* vector rotation calculations are correct.

Given:

1. Two 3D right-handed Cartesian reference frames, Frame A and Frame B, with Frame B oriented at a counterclockwise angle of 30° about the Z axis of Frame A (see Figure C1-1).
2. A fixed 3D vector with coordinates given in Frame A as (0.6, 0.8, 0.0).

Find:

1. The vector coordinates with respect to Frame B.
2. The vector coordinates with respect to Frame A if the vector is not fixed and instead rotates counterclockwise with an angle of 30°.

Solution:

Case 1:

Case 1 considers a fixed 3D vector that is defined with respect to Frame A as

$V^A = (0.6, 0.8, 0.0)$.

Another frame, designated as Frame B, is rotated by a counterclockwise angle, θ = 30° about the Z axis with respect to Frame A. The objective of this case is to determine the components of the vector V with respect to Frame B.

The input vector, V, has a magnitude of unity and an angle β of 53.13° with respect to the X_A axis. Figure C1 shows V, Frame A, and Frame B, along with the angles θ and β.

The solution to this problem is computed manually using two methods: the rotation matrix and the rotation quaternion. The results from these two methods are then compared against the output from *QuatView*.

A) DCM Solution:

The basic formula for doing a rotational transformation on a vector is

$$V^B = [\mathbf{R}]_A^B V^A \qquad \text{(C1-1)}$$

where V^A is the vector with respect to Frame A, V^B is the vector with respect to Frame B, and $[\mathbf{R}]_A^B$ is the DCM that defines the orientation of Frame B with respect to Frame A. Each element of the DCM is given by $\cos(\alpha_{ij})$, where α_{ij} is the angle between the i'th axis of Frame B and the j'th axis of Frame A. The matrix row number is i and the column number is j. So, looking at Figure C1-1, the first element of the first row is the cosine of the angle between axes X_B and X_A which gives $\cos(\theta)$. Similarly the second element of the first row is the cosine of the angle between axes X_B and Y_A which gives $\cos(90-\theta)$ which is equal to $\sin(\theta)$. Finally, the third element of the first row is the cosine of the angle between axes X_B and Y_A which gives cos(90)=0. Using this approach for the remaining rows, all of the elements of the DCM can be computed as

$$[\mathbf{R}]_A^B = \begin{bmatrix} c\theta & s\theta & 0 \\ -s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(C1-2)}$$

Eqn. C1-2 is the general DCM for expressing the coordinates of any vector in a rotated frame where the rotation is a counterclockwise angle, θ, about the Z axis of the original (unrotated) frame. Note that this equation is only valid if both frames have right-handed coordinate systems.

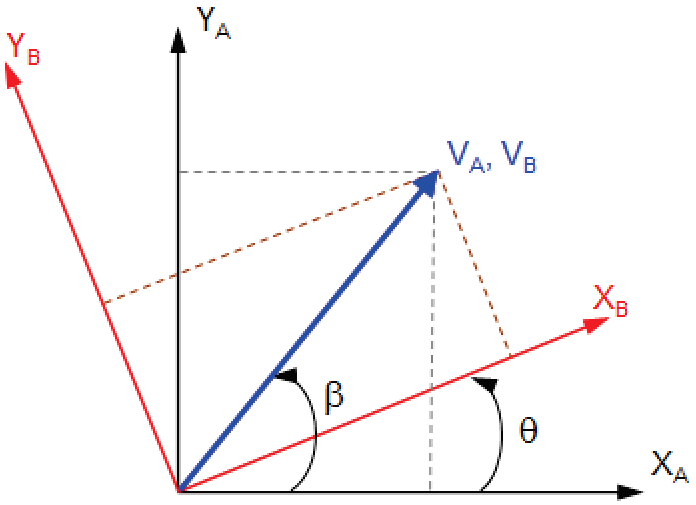


Figure C1-1. Verification Test 1, Case 1 - V Is Fixed To Frame A

Plugging the given value for θ=30° into Eqn. C1-2 gives the DCM as

$$[\mathbf{R}]_A^B = \begin{bmatrix} 0.500 & 0.866 & 0 \\ -0.866 & 0.500 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Using this DCM in Eqn. C1-1 and performing the matrix multiplication then gives the following values for the vector components in Frame B:

$$V_x^B = V_x^A \cos(\theta) + V_y^A \sin(\theta) = (0.6)(0.5) + (0.8)(0.866) = 0.9196$$

$$V_y^B = -V_y^A \sin(\theta) + V_y^A \cos(\theta) = (-0.6)(0.866) + (0.8)(0.5) = 0.3928$$

Therefore, the requested vector components with respect to Frame B are

V^B = (0.9196, 0.3928, 0.0000) **<-------ANSWER**

B) Quaternion Solution:

The same solution can be computed with quaternions using the quaternion rotation formula

$$\left(V^B\right) = \left(Q_A^B\right) \otimes \left(V^A\right) \otimes \left(Q_A^B\right)^* \qquad \text{(C1-3)}$$

where (V^A) and (V^B) are pure quaternions containing the vector components V^A and V^B, Q_A^B is the rotation quaternion defining the orientation of Frame B with respect to Frame A, ⊗ is the quaternion multiplication operator, and * is the quaternion conjugate operator.

The rotation quaternion that represents the orientation of Frame B with respect to Frame A is given by the standard formula

$$Q_A^B = (q_0, q_1, q_2, q_3) = \left(\cos\left(\frac{\theta}{2}\right), \hat{x} \sin\left(\frac{\theta}{2}\right), \hat{y} \sin\left(\frac{\theta}{2}\right), \hat{z} \sin\left(\frac{\theta}{2}\right) \right) \qquad \text{(C1-4)}$$

where θ is the counterclockwise angle about the rotation axis from Frame A to Frame B and $(\hat{x}, \hat{y}, \hat{z})$ are the components of the rotation axis. Since the rotation axis for this problem is the Z axis of Frame A, the rotation axis vector is simply (0,0,1). This means that the quaternion components q_1 and q_2 are zero, and component q_3 is given by

$q_3 = \sin(\theta/2) = \sin(15) = 0.2588$

The scalar component, q_0, is then

$q_0 = \cos(\theta/2) = \cos(15) = 0.9659$

The quaternion that defines the orientation of Frame B with respect to Frame A is then

Q_A^B = (0.9659, 0.0, 0.0, 0.2588)

The pure quaternion for (V^A) is simply

(V^A) = (0.00, 0.60, 0.80, 0.00)

Plugging these values for (V^A) and Q_A^B into Eqn. C1-3 gives

$\left(V^B\right)$ = (0.9659, 0.0, 0.0, 0.2588) ⊗ (0.00, 0.60, 0.80, 0.00) ⊗ (0.9659, 0.0, 0.0, -0.2588)

Performing the two quaternion multiplications in accordance with Eqn. 2 involves a fairly lengthy amount of algebra, all of which will not be repeated here. The end result of the quaternion multiplications is

$$\left(V^B\right) = (0.00, 0.9196, 0.3928, 0.00)$$

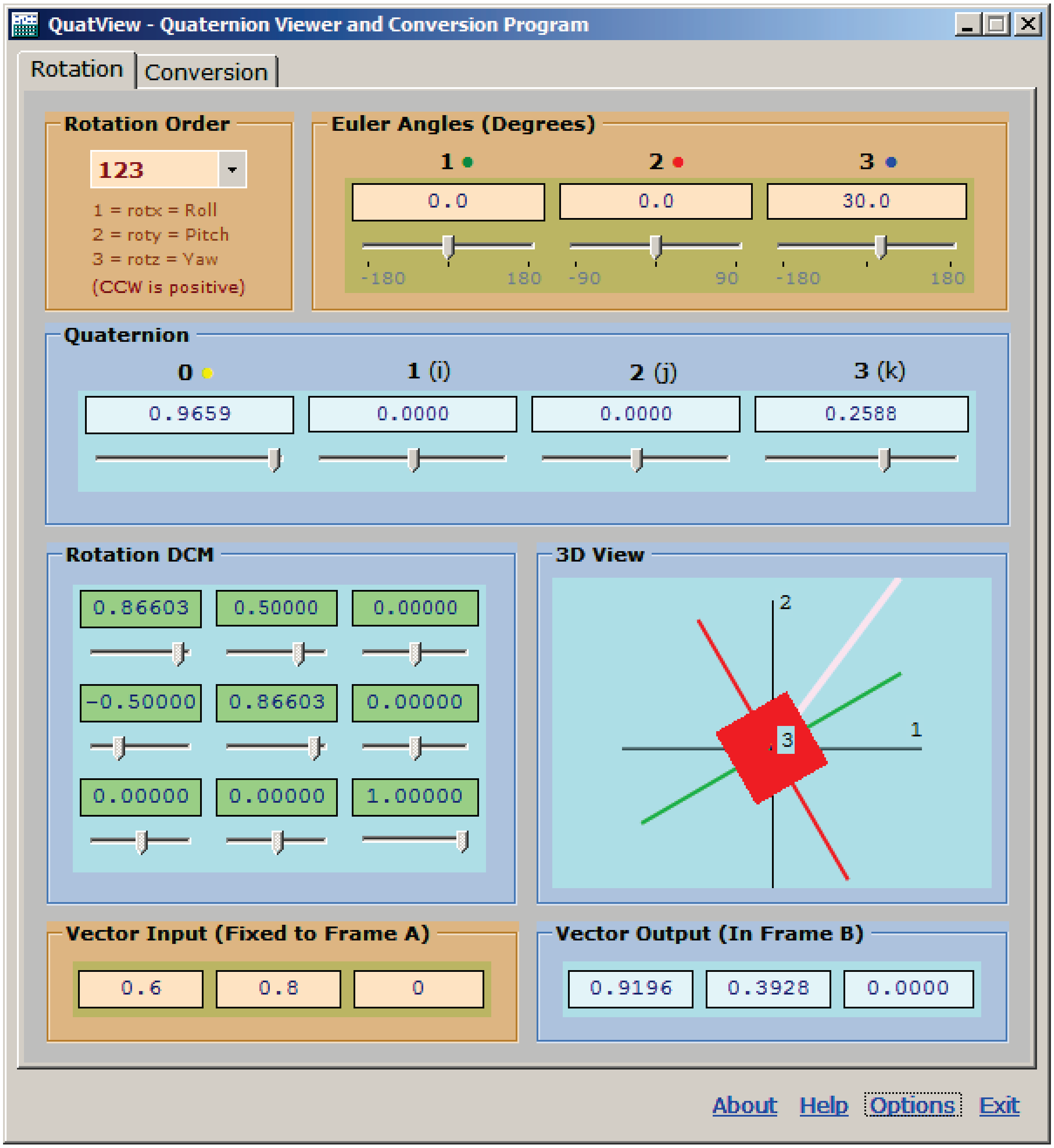
Separating out the vector from the pure quaternion then gives the following final answer for the vector components in Frame B:

$V^B = (0.9196, 0.3928, 0.0000)$ **<-------ANSWER**

This answer agrees precisely with the answer from the DCM solution, which is to be expected.

C) QuatView Solution:

Figure C1-2 shows the *QuatView* screen shot for this verification problem. The orientation of Frame B with respect to Frame A is defined by the three Euler angles entered as (0.0, 0.0, 30.0). The input vector, which is defined in Frame A as (0.6, 0.8, 0.0), must also be entered.



Rotation Order

123

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees)

1	2	3
0.0	0.0	30.0

Quaternion

0	1 (i)	2 (j)	3 (k)
0.9659	0.0000	0.0000	0.2588

Rotation DCM

0.86603	0.50000	0.00000
-0.50000	0.86603	0.00000
0.00000	0.00000	1.00000

3D View

Vector Input (Fixed to Frame A)

0.6	0.8	0

Vector Output (In Frame B)

0.9196	0.3928	0.0000

Figure C1-2. QuatView Results For Case 1 (Fixed Vector in Rotated Frame)

Vector plotting is also turned on using the Options dialog. The vector initial reference frame must also be set to the default setting of "Frame B (Rotating)" using the Options dialog.

Once the Frame B orientation and vector components are entered, then *QuatView* automatically displays the complete solution which includes the Frame B orientation quaternion and the fixed vector components in Frame B.

Taking the results directly from Figure C1-2 gives

Q_A^B = (0.9659, 0.0, 0.0, 0.2588) <-------- Quaternion For Frame B

V^B = (0.9196, 0.3928, 0.0000) **<-------ANSWER**

These answers agree precisely with the manual solutions given above. This validates the *QuatView* vector rotation calculations.

Case 2:

Case 2 considers the same input vector as Case 1, except that for Case 2 this vector is rotated counterclockwise by an angle of θ = 30° about the Z axis instead of being fixed. The objective of this case is to determine the components of the rotated vector with respect to Frame A.

Figure C1-3 shows the vector in its unrotated (V_A) and rotated (V_B) orientations with respect to Frame A. The angles θ and β are also shown, where β = 53.13° is the initial vector orientation angle prior to the rotation. The final orientation angle after the rotation is then (θ+β).

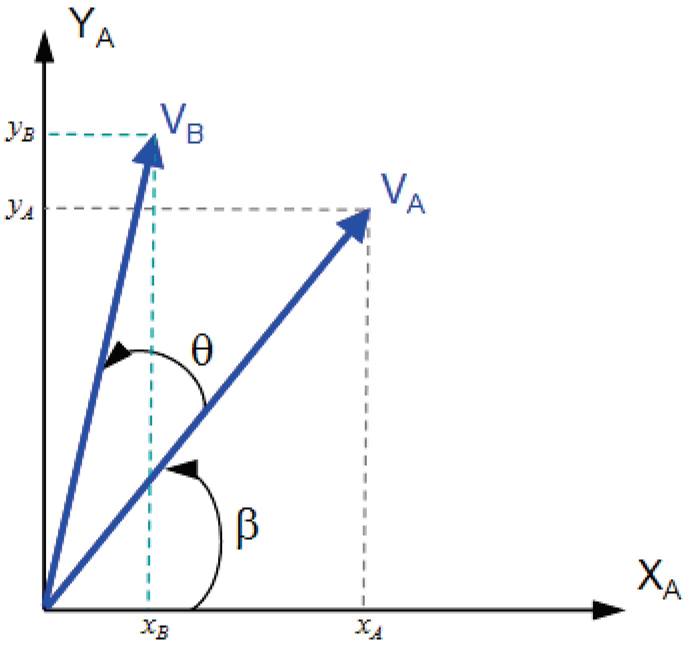


Figure C1-3. Verification Test 1, Case 3 - V Rotates With Frame B

The solution to this problem is computed manually using two methods: the DCM and the rotation quaternion. The results from these methods are then compared against the output from *QuatView*.

It turns out that Case 1 and Case 2 are closely related, with solution procedures that are very similar. In fact, the DCM solution for Case 2 is the same as Case 1 with the only difference being that the transpose of the Case 1 DCM is used for Case 2. Likewise, the only difference with the quaternion solution is that the rotation quaternion transpose for Case 2 appears on the LHS of equation C1-3 instead of on the RHS.

A) DCM Solution:

The basic formula for rotating a vector is

$$V^B = [\mathbf{R}]_A^B V^A \qquad \text{(C1-5)}$$

where $[\mathbf{R}]_A^B$ is the DCM that defines the vector rotation from A to B. The simplest way to derive this DCM is to relate the final and initial vector components in accordance with Figure C1-3. As this figure shows, with a vector magnitude of unity the initial vector components (x_A, y_A) are given by

$$x_A = \cos(\beta)$$

$$y_A = \sin(\beta) \qquad \text{(C1-6)}$$

Similarly, the final vector components are given by

$$x_B = \cos(\theta+\beta)$$

$$y_B = \sin(\theta+\beta) \qquad \text{(C1-7)}$$

Using the trigonometric formulas for the sine and cosine of the sum of two angles then yields

$$x_B = \cos(\theta+\beta) = \cos(\theta)\cos(\beta) - \sin(\theta)\sin(\beta)$$

$$y_B = \sin(\theta+\beta) = \sin(\theta)\cos(\beta) + \cos(\theta)\sin(\beta) \qquad \text{(C1-8)}$$

Substituting Eqns. C1-6 into C1-8 then gives

$$x_B = x_A \cos(\theta) - y_A \sin(\theta)$$

$$y_B = x_A \sin(\theta) + y_A \cos(\theta) \qquad \text{(C1-9)}$$

Putting Eqns. C1-9 into matrix format per Eqn. C1-5 then gives the DCM as

$$[\mathbf{R}]_A^B = \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{(C1-10)}$$

Plugging θ=30° into Eqn. C1-10 then gives the following DCM:

$$[\mathbf{R}]_A^B = \begin{bmatrix} 0.500 & -0.866 & 0 \\ 0.866 & 0.500 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that this DCM is simply the transpose of the DCM used for Case 1. This means that expressing a fixed vector in a rotated reference frame is the transpose of rotating the vector with respect to a fixed frame.

Performing the matrix multiplications in Eqn. C1-5 gives the rotated vector components as

$$V_x^B = V_x^A \cos(\theta) - V_y^A \sin(\theta) = (0.6)(0.5) - (0.8)(0.866) = 0.1196$$

$$V_y^B = -V_y^A \sin(\theta) + V_y^A \cos(\theta) = (0.6)(0.866) + (0.8)(0.5) = 0.9928$$

The requested vector components with respect to Frame B are then

V^B = (0.1196, 0.9928, 0.0000) **<-------ANSWER**

B) Quaternion Solution:

The quaternion transformation formula for a rotating vector in a fixed coordinate system is given by

$$\left(V^B\right) = \left(Q_A^B\right)^* \otimes \left(V^A\right) \otimes \left(Q_A^B\right) \qquad \text{(C1-11)}$$

where (V^A) and (V^B) are pure quaternions containing the vector components V^A and V^B, Q_A^B is the rotation quaternion defining the orientation of Frame B with respect to Frame A, ⊗ is the quaternion multiplication operator, and * is the quaternion conjugate operator. Note that this formula is similar to Eqn. C1-3 from Case 1 with the only difference being that the rotation quaternion transpose is on the LHS instead of the RHS.

Using Eqn. C1-4, the rotation quaternion for a counterclockwise rotation from A to B with $\theta = 30°$ is computed as

$$Q_A^B = (0.9659, 0.0, 0.0, 0.2588)$$

The pure quaternion for the input vector is the same as Case 1 which is given by

$$(V^A) = (0.00, 0.60, 0.80, 0.00)$$

Plugging these values for (V^A) and Q_A^B into Eqn. C1-11 gives

$$\left(V^B\right) = (0.9659, 0.0, 0.0, -0.2588) \otimes (0.00, 0.60, 0.80, 0.00) \otimes (0.9659, 0.0, 0.0, 0.2588)$$

Performing the two quaternion multiplications in accordance with Eqn. 2 gives

$$\left(V^B\right) = (0.00, 0.1196, 0.9928, 0.00)$$

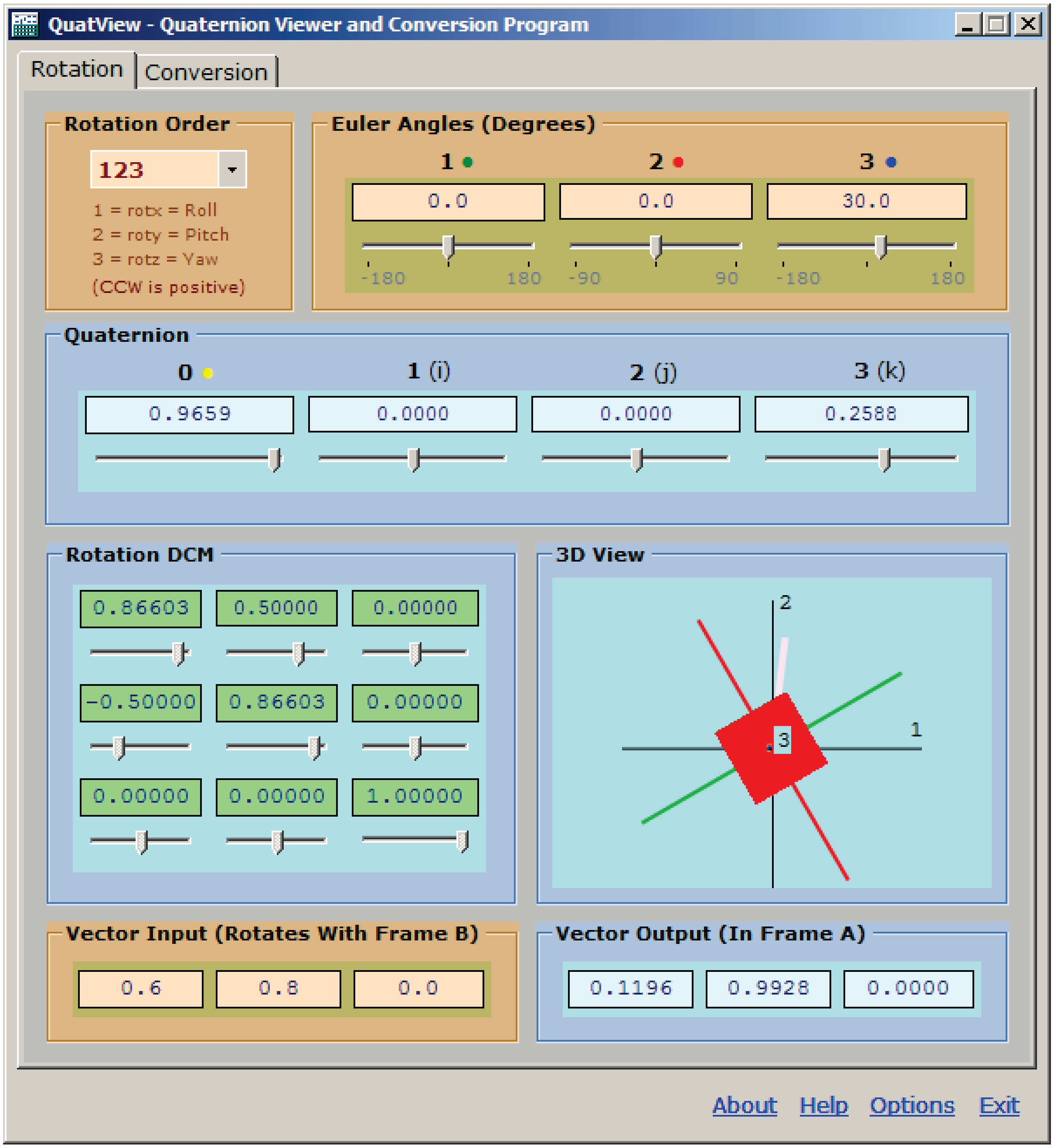
Separating out the vector from the pure quaternion then gives the following final answer for the vector components in Frame B:

V^B = (0.1196, 0.9928, 0.0000) **<-------ANSWER**

This answer agrees precisely with the answer from the DCM solution, which is to be expected.

C) QuatView Solution:

Figure C1-4 shows the *QuatView* results for Case 2. The orientation of Frame B with respect to Frame A is defined by the three Euler angles entered as (0.0, 0.0, 30.0). The vector is entered in Frame A as (0.6, 0.8, 0.0). The vector initial reference frame must be set to "Frame A (Fixed)" using the Options dialog. Vector plotting is also turned on using the Options dialog.



Rotation Order

123

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees)

1	2	3
0.0	0.0	30.0

Quaternion

0	1 (i)	2 (j)	3 (k)
0.9659	0.0000	0.0000	0.2588

Rotation DCM

0.86603	0.50000	0.00000
-0.50000	0.86603	0.00000
0.00000	0.00000	1.00000

Vector Input (Rotates With Frame B)

0.6	0.8	0.0

Vector Output (In Frame A)

0.1196	0.9928	0.0000

Figure C1-4. QuatView Results For Case 2 (Rotated Vector in Fixed Frame)

Once the rotation angles and vector components are entered, then *QuatView* automatically displays the complete solution which includes the Frame B rotation quaternion and the rotated vector components with respect to Frame A.

Taking the results directly from Figure C1-4 gives

Q_A^B = (0.9659, 0.0, 0.0, 0.2588) <-------- Quaternion For Rotated Vector

V^B = (0.1196, 0.9928, 0.0000) **<-------ANSWER**

These results agree precisely with the manual solutions given above using the DCM and quaternion methods. This validates the *QuatView* vector rotation calculations.

Test Problem 2

Conversion From Euler Angles To Quaternion

Verification Test 2 is a test of the conversion calculations between the Euler angles and the quaternion. These calculations are done by the rotation GUI (Figure 1) and the conversion GUI (Figure 5). The test objectives are to verify the correctness of the conversion calculations from Euler angles to quaternion as well as from quaternion to Euler angles.

The verification is done by comparing *QuatView* results against the results from a Matlab quaternion script that have already been verified via the Matlab aerospace toolbox.

Given:

3. A set of Euler angles (33,44,55) with rotation sequence 123.

Find:

3. The quaternion that corresponds to the given Euler angles.
4. Convert the computed quaternion back to Euler angles.
5. Repeat 1 and 2 with all of the other possible eleven rotation sequences: 123,231,312, 213,132, 121,323,212,313,131,232. Put all the results into a single table for comparison against similar results from the Matlab script.

Solution:

Table C2-1 lists the *QuatView* results for rotation order 123 as well as the other eleven possible rotation orders. These results were generated using the Rotation tab to compute the quaternion and then using the Conversion tab along with the copy/paste capability to convert the quaternion back to Euler angles. In every case, the quaternion conversion produced the exact same Euler angles that were used to start with. This verifies that the Euler angle to quaternion and the quaternion to Euler angle calculations are the inverse of each other.

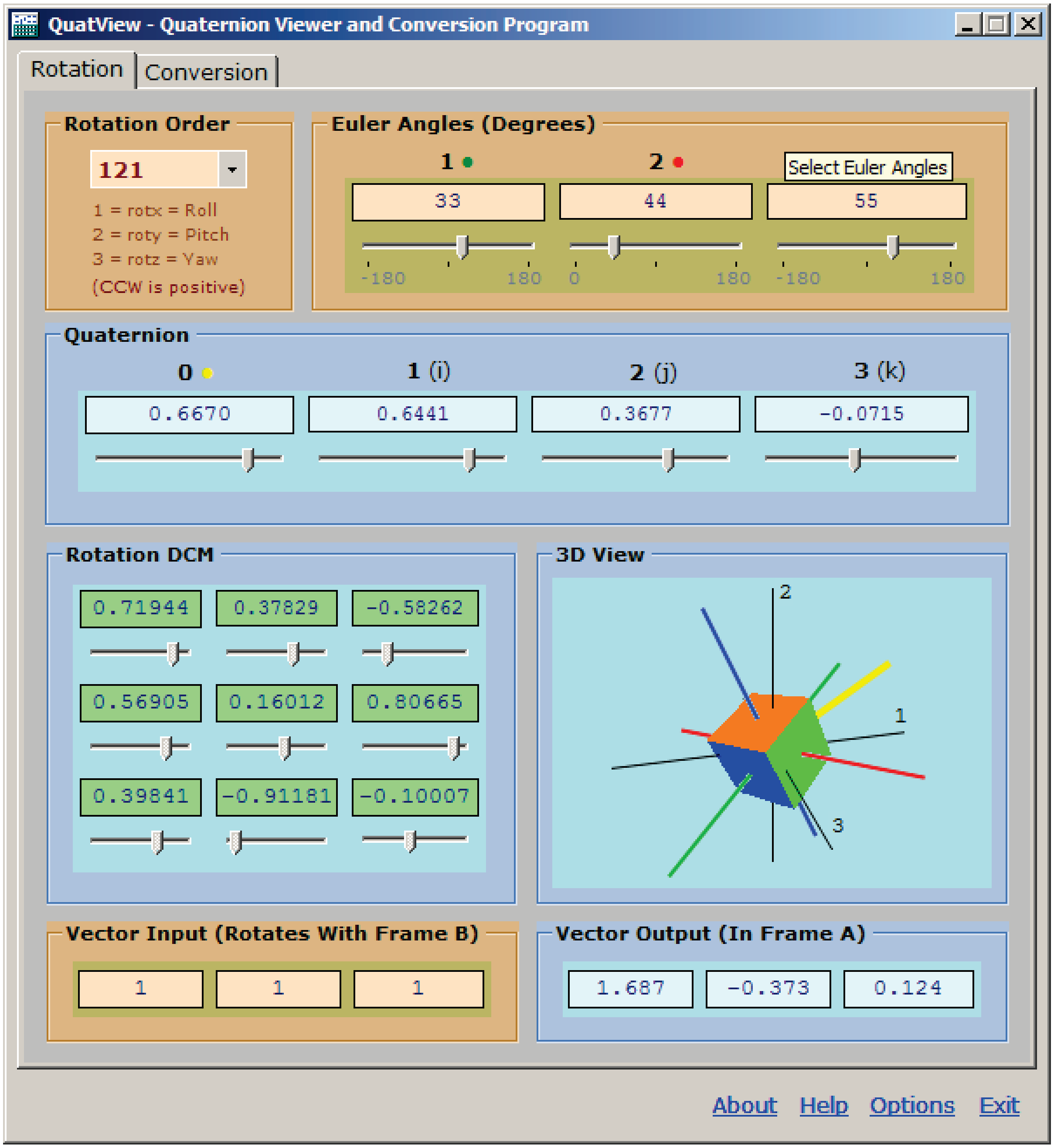
Once the *QuatView* results were generated, then the validated Matlab script was run 12 times to produce the results listed in Table C2-2. Comparison of the *QuatView* and Matlab results shows them to be the same. This verifies that the Euler angle to quaternion and the quaternion to Euler angle calculations in *QuatView* are correct. Figure C2-1 and C2-2 show the *QuatView* results for the 121 rotation order.

Table C2-1. Verification Test 2, QuatView Results

Rotation Order	q0	q1	q2	q3
123	0.7394	0.3994	0.1970	0.5049
231	0.7394	0.3994	0.4402	0.3161
312	0.7394	0.0677	0.4402	0.5049
321	0.8377	0.0677	0.4402	0.3161
213	0.8377	0.3994	0.1970	0.3161
132	0.8377	0.0677	0.1970	0.5049
121	0.6670	0.6441	0.3677	-0.0715
131	0.6670	0.6441	0.0715	0.3677
212	0.6670	0.3677	0.6441	0.0715
232	0.6670	-0.0715	0.6441	0.3677
313	0.6670	0.3677	-0.0715	0.6441
323	0.6670	0.0715	0.3677	0.6441

Table C2-2. Verification Test 2, Matlab Results

Rotation Order	q0	q1	q2	q3	QuatView Match
123	0.7394	0.3994	0.1970	0.5049	**Y**
231	0.7394	0.3994	0.4402	0.3161	**Y**
312	0.7394	0.0677	0.4402	0.5049	**Y**
321	0.8377	0.0677	0.4402	0.3161	**Y**
213	0.8377	0.3994	0.1970	0.3161	**Y**
132	0.8377	0.0677	0.1970	0.5049	**Y**
121	0.6670	0.6441	0.3677	-0.0715	**Y**
131	0.6670	0.6441	0.0715	0.3677	**Y**
212	0.6670	0.3677	0.6441	0.0715	**Y**
232	0.6670	-0.0715	0.6441	0.3677	**Y**
313	0.6670	0.3677	-0.0715	0.6441	**Y**
323	0.6670	0.0715	0.3677	0.6441	**Y**


QuatView - Quaternion Viewer and Conversion Program

Rotation Order

121

1 = rotx = Roll
2 = roty = Pitch
3 = rotz = Yaw
(CCW is positive)

Euler Angles (Degrees)

1	2	
33	44	55

Quaternion

0	1 (i)	2 (j)	3 (k)
0.6670	0.6441	0.3677	-0.0715

Rotation DCM

0.71944	0.37829	-0.58262
0.56905	0.16012	0.80665
0.39841	-0.91181	-0.10007

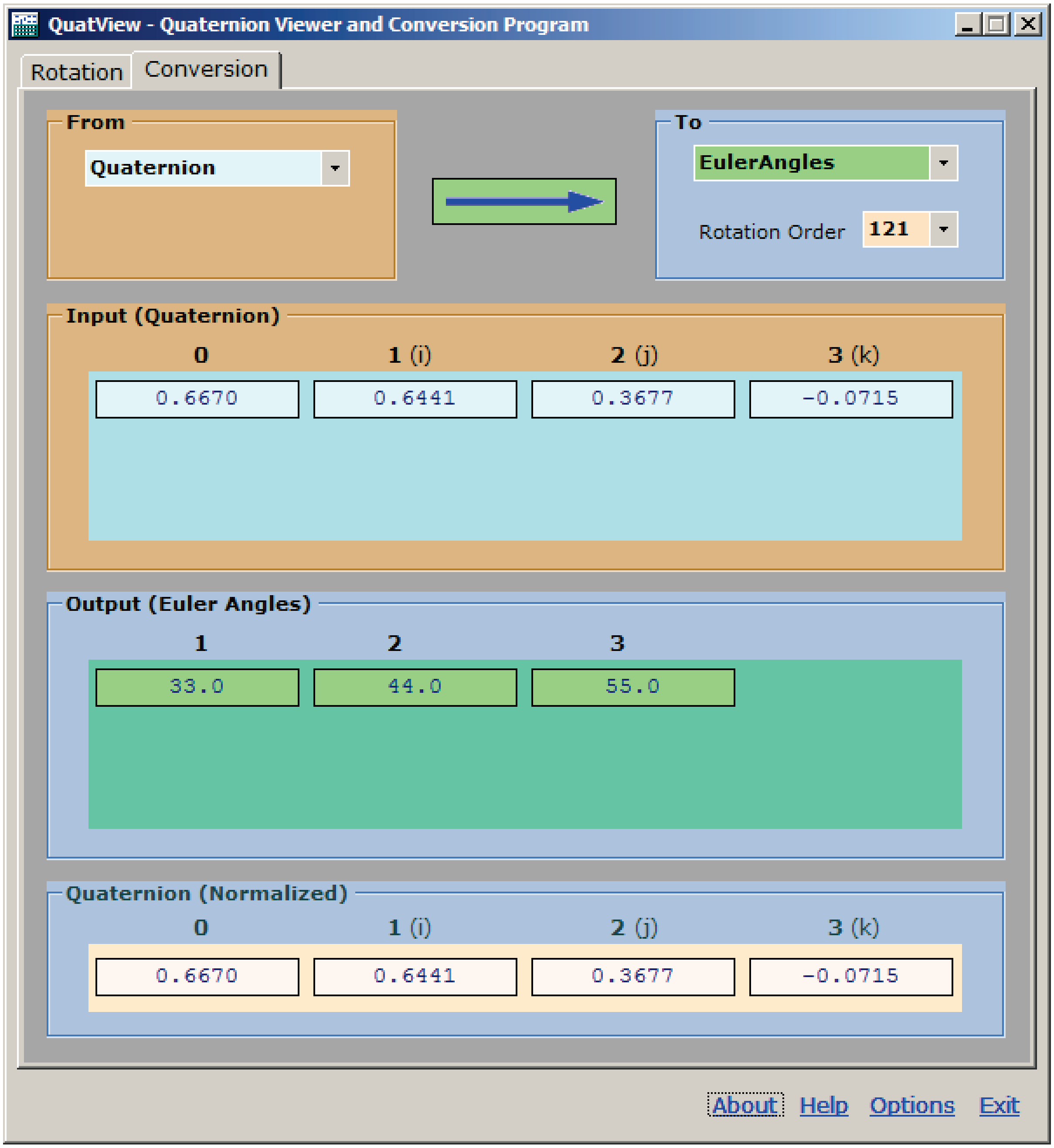
Vector Input (Rotates With Frame B)

1	1	1

Vector Output (In Frame A)

1.687	-0.373	0.124

Figure C2-1. QuatView Results For Rotation Order 121 (EA to Q)



From
Quaternion

To
EulerAngles
Rotation Order 121

Input (Quaternion)

0	1 (i)	2 (j)	3 (k)
0.6670	0.6441	0.3677	-0.0715

Output (Euler Angles)

1	2	3
33.0	44.0	55.0

Quaternion (Normalized)

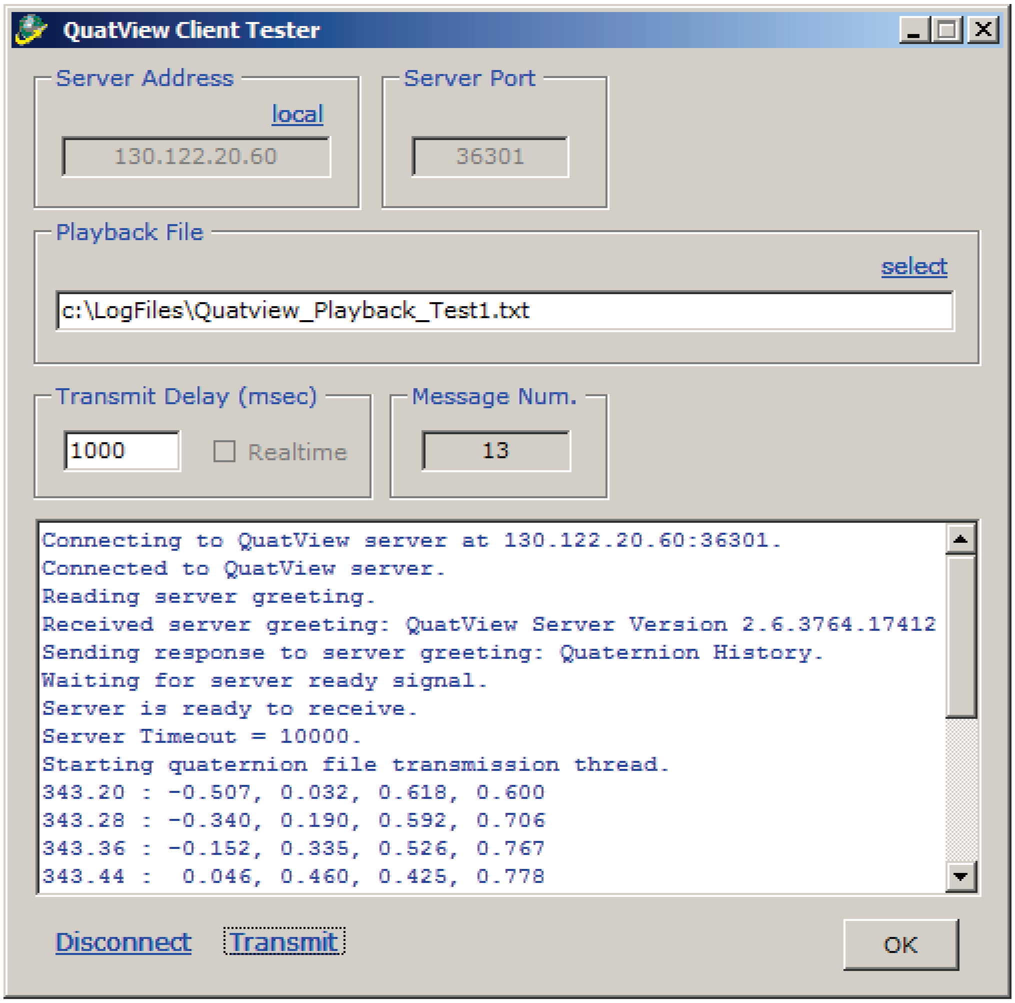
0	1 (i)	2 (j)	3 (k)
0.6670	0.6441	0.3677	-0.0715

Figure C2-2. QuatView Results For Rotation Order 121 (Q to EA)

Client Tester

QuatView includes a test program that demonstrates how to use the QuatViewClient class that is installed with *QuatView*. The client tester program is a small GUI application that simply reads a quaternion history file and transmits it to the *QuatView* server using the methods and properties provided by the QuatViewClient class as described here.

Figure 20 shows the *QuatView* Client Tester GUI. The GUI has four input fields to enter the server IP address, port number, playback file path, and the transmit delay time. It also has two output fields that display the transmitted message count and any status messages produced during the transmission. The bottom of the GUI has two link buttons: one for connecting/disconnecting to/from the server and the other for starting/stopping the quaternion transmissions.



Server Address: 130.122.20.60

Server Port: 36301

Playback File: c:\LogFiles\Quatview_Playback_Test1.txt

Transmit Delay (msec): 1000 ☐ Realtime

Message Num.: 13

```
Connecting to QuatView server at 130.122.20.60:36301.
Connected to QuatView server.
Reading server greeting.
Received server greeting: QuatView Server Version 2.6.3764.17412
Sending response to server greeting: Quaternion History.
Waiting for server ready signal.
Server is ready to receive.
Server Timeout = 10000.
Starting quaternion file transmission thread.
343.20 : -0.507, 0.032, 0.618, 0.600
343.28 : -0.340, 0.190, 0.592, 0.706
343.36 : -0.152, 0.335, 0.526, 0.767
343.44 :  0.046, 0.460, 0.425, 0.778
```

Figure 20. QuatView Client Tester GUI

Once a connection has been made to the *QuatView* server, the user can begin sending the quaternion data file by clicking on the Transmit link button. The program will then read and transmit the playback file with the specified delay time between each quaternion. Alternatively, the user can request a variable or "realtime" delay between each quaternion. This realtime delay is computed from the time values in column 1 of the playback file.

Once a connection has been made or a transmission has begun, the user can stop the transmission or disconnect at any time by clicking on the respective link button. This will transmit a special stop signal or disconnect signal sent to the *QuatView* server. A stop signal will automatically be sent once the end of the playback file is reached. If any network error occurs during the playback file transmission, then an error signal will be sent so that the *QuatView* server can respond accordingly.

Acronyms and Abbreviations

2D	Two Dimensional
3D	Three Dimensional
DCM	Direction Cosine Matrix
DLL	Dynamic Link Library
DOF	Degrees of Freedom
Eqn.	Equation
GUI	Graphical User Interface
IO	Input/Output
LHS	Left Hand Side
RHS	Right Hand Side

References

Books

1. J. B. Kuipers, *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*, Princeton University Press, 2002. [ISBN-10: 0691102988]
2. Andrew J. Hanson, *Visualizing Quaternions*, Morgan Kaufmann, 2006. [ISBN-10: 0120884003]
3. David M. Bourg, *Physics for Game Developers*, O'Reilly, 2002. [ISBN-10: 0596000065]

Internet Articles

1. *Wikipedia Article - Quaternions*
2. Wikipedia Article - Quaternions and spatial rotation
3. Wikipedia Article - Rotation Representation
4. Wikipedia Article - Euler Angles
5. Wikipedia Article - Rotation Operator
6. Wikipedia Article - Transformation Matrix

Open Source Dependencies

QuatView uses source code derived from the open-source software vendors listed in the table below.

Item	Description	Distribution	Licensor	License Type
SharpGL	C# wrapper class for OpenGL	QuatView.exe (static link)	David Kerr	GNU GPL v3

Revision History

3.0. 2016-03-04

1. Added compatibility with Windows 10.
2. Updated OpenGL drawing for compatibility with Windows 10
3. Updated build for Dot Net 4.0.

2.9. 2011-09-01

1. Help file update.

2.8. 2011-05-12

1. Added main window context menu.

2.7. 2010-11-20

1. Added X-Axis rotation constraint for 3D view to keep angle between +/-80°.

2.6. 2010-04-18

1. Added larger scalable 3D orientation viewer window.
2. Fixed stop and disconnection problems with client tester program.
3. Major update to the help file.

2.5. 2010-03-13

1. Added TCP server to display quaternion data stream from remote clients.
2. Added TCP client class and DLL so that remote applications can easily communicate with QuatView.
3. Added client test application that can read a quaternion history file and transmit the file contents to QuatView running on a remote computer.
4. Added log file capability for server mode.
5. Added mouse wheel control for 3D zoom.

2.4. 2010-02-18

1. Changed playback progress bar to trackbar with interactive user control of playback state.
2. Changed playback file reader to store the entire playback history in memory instead of reading the file one record at a time during playback.
3. Updated playback mode display and added playback mode dialog.
4. Added persistence for option dialog settings via configuration file.
5. Updated help file.

2.3. 2009-09-28

1. Fixed error with quaternion component ordering in playback mode with scalar last.
2. Changed process to high priority during playback for increased real-time performance.
3. Included slerp and additional operator math methods to quaternion class.

2.2. 2009-09-27

1. Made all playback settings persistent between program sessions via user configuration file..
2. Added continuous animation style to options dialog.
3. Added user input of DCM normalization tolerance to options dialog.

2.1. 2009-09-23

1. Disabled all GUI link buttons during playback.
2. Fixed problem with real time playback after stop.
3. Added check for playback being too far behind real time.
4. Set allowable playback time error to 50 msec. If playback gets behind by more than this allowable value, then the playback time display turns red.
5. Changed cursor to hourglass during program version check.
6. Updated help file.

2.0. 2009-09-20

1. Added ASCII file playback capability with selective playback speed from 1 Hz to 60 Hz.
2. Added real-time playback capability with time read from playback file and maximum frame rate of 60 Hz.
3. Fixed problem with quaternion sliders not working via keyboard input.
4. Improved GUI update routines and increased animation performance.
5. Updated help file.

1.7. 2009-08-26

1. Fixed bug with animation through Euler angle singularity.
2. Added context menu option to orthonormalize the DCM if it fails the orthonormality check.
3. Added context menu option to display the DCM orthonormality error.
4. Added web update link and version check to About box.

1.6. 2009-07-28

1. Make cube the default shape.

1.5. 2009-07-16

1. Added Earth globe texture to sphere.
2. Added "Stop Animation" button to GUI.
3. Updated help file.

1.4. 2009-07-11

1. Added capability to plot cone, cylinder, and sphere
2. Added capability to select major axis for cone and cylinder.
3. Added capability to enter L/D for cone and cylinder.

1.3. 2009-06-11

1. Added vector color selection.

2. Changed animation mode to up/down instead of recycle.
3. Updated help file.

1.2. 2009-06-02

1. Added vector plotting and line antialiasing.
2. Updated help file.

1.1. 2009-05-26

1. Added animation capability.
2. Corrected error with vector rotation calculation.
3. Added option for fixed or rotating vector reference (Frame A or Frame B).
4. Updated help file

1.0. 2009-04-24

Initial release.

Software License Agreement

QuatView License Agreement and Warranty Disclaimer

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS SOFTWARE LICENSE ("LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HEREIN, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE AUTHOR GRANTS YOU THE RIGHTS CONTAINED HEREIN IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO ACCEPT AND BE BOUND BY THE TERMS OF THIS LICENSE, YOU CANNOT MAKE ANY USE OF THE WORK.

1. **Definitions.**
    a. **"Articles"** means, collectively, all articles written by Author which describes how the Source Code and Executable Files for the Work may be used by a user.
    b. **"Author"** means the individual or entity that offers the Work under the terms of this License.
    c. **"Derivative Work"** means a work based upon the Work or upon the Work and other pre-existing works.
    d. **"Executable Files"** refer to the executables, binary files, configuration and any required data files included in the Work.
    e. "**Publisher**" means the provider of the website, magazine, CD-ROM, DVD or other medium from or by which the Work is obtained by You.
    f. **"Source Code"** refers to the collection of source code and configuration files used to create the Executable Files.
    g. **"Standard Version"** refers to such a Work if it has not been modified, or has been modified in accordance with the consent of the Author, such consent being in the full discretion of the Author.
    h. **"Work"** refers to the collection of files distributed by the Publisher, including the Source Code, Executable Files, binaries, data files, documentation, whitepapers and the Articles.
    i. **"You"** is you, an individual or entity wishing to use the Work and exercise your rights under this License.
2. **Fair Use/Fair Use Rights.** Nothing in this License is intended to reduce, limit, or restrict any rights arising from fair use, fair dealing, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.
3. **License Grant.** Subject to the terms and conditions of this License, the Author hereby grants You a worldwide, royalty-free, non-exclusive, perpetual

(for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. You may use the standard version of the Source Code or Executable Files in Your own applications.
   b. You may apply bug fixes, portability fixes and other modifications obtained from the Public Domain or from the Author. A Work modified in such a way shall still be considered the standard version and will be subject to this License.
   c. You may otherwise modify Your copy of this Work (excluding the Articles) in any way to create a Derivative Work, provided that You insert a prominent notice in each changed file stating how, when and where You changed that file.
   d. You may distribute the standard version of the Executable Files and Source Code or Derivative Work in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution.
   e. The Articles discussing the Work published in any form by the author may not be distributed or republished without the Author's consent. The author retains copyright to any such Articles. You may use the Executable Files and Source Code pursuant to this License but you may not repost or republish or otherwise distribute or make available the Articles, without the prior written consent of the Author.

   Any subroutines or modules supplied by You and linked into the Source Code or Executable Files this Work shall not be considered part of this Work and will not be subject to the terms of this License.

4. **Patent License.** Subject to the terms and conditions of this License, each Author hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, import, and otherwise transfer the Work.
5. **Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
   a. You agree not to remove any of the original copyright, patent, trademark, and attribution notices and associated disclaimers that may appear in the Source Code or Executable Files.
   b. You agree not to advertise or in any way imply that this Work is a product of Your own.
   c. The name of the Author may not be used to endorse or promote products derived from the Work without the prior written consent of the Author.
   d. You agree not to sell, lease, or rent any part of the Work. This does not restrict you from including the Work or any part of the Work inside a larger software distribution that itself is being sold. The Work by itself, though, cannot be sold, leased or rented.
   e. You may distribute the Executable Files and Source Code only under the terms of this License, and You must include a copy of, or the

Uniform Resource Identifier for, this License with every copy of the Executable Files or Source Code You distribute and ensure that anyone receiving such Executable Files and Source Code agrees that the terms of this License apply to such Executable Files and/or Source Code. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute the Executable Files or Source Code with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License.

   f. You agree not to use the Work for illegal, immoral or improper purposes, or on pages containing illegal, immoral or improper material. The Work is subject to applicable export laws. You agree to comply with all such laws and regulations that may apply to the Work after Your receipt of the Work.

6. **Representations, Warranties and Disclaimer.** THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABLE QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

7. **Indemnity.** You agree to defend, indemnify and hold harmless the Author and the Publisher from and against any claims, suits, losses, damages, liabilities, costs, and expenses (including reasonable legal or attorneys' fees) resulting from or relating to any use of the Work by You.

8. **Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL THE AUTHOR OR THE PUBLISHER BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK OR OTHERWISE, EVEN IF THE AUTHOR OR THE PUBLISHER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

9. **Termination.**

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of any term of this License. Individuals or entities who have received Derivative Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 6, 7, 8, 9, 10 and 11 will survive any termination of this License.

    b. If You bring a copyright, trademark, patent or any other infringement claim against any contributor over infringements You claim are made by the Work, your License from such contributor to the Work ends automatically.
    c. Subject to the above terms and conditions, this License is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, the Author reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

10. **Publisher**. The parties hereby confirm that the Publisher shall not, under any circumstances, be responsible for and shall not have any liability in respect of the subject matter of this License. The Publisher makes no warranty whatsoever in connection with the Work and shall not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. The Publisher reserves the right to cease making the Work available to You at any time without notice
11. **Miscellaneous**
    a. This License shall be governed by the laws of the location of the head office of the Author or if the Author is an individual, the laws of location of the principal place of residence of the Author.
    b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this License, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
    c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

This License constitutes the entire agreement between the parties with respect to the Work licensed herein. There are no understandings, agreements or representations with respect to the Work not specified herein. The Author shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Author and You.