Chapter
II-16

3D Graphics

Overview

Igor can create various kinds of 3D graphics including:

- **Surface Plots**
- **3D Scatter Plots**
- **3D Bar Plots**
- **Path Plots**
- **Ribbon Plots**
- **Isosurface Plots**
- **Voxelgram Plots**

Image Plots, **Contour Plots** and **Waterfall Plots** are considered 2D graphics and are discussed in other sections of the help.

Igor's 3D graphics tool is called "Gizmo". Most 3D graphics that you produce with Gizmo will be based on data stored in waves. It's important to understand what type of wave data is required for what type of 3D graphic, as explained in the following sections explain.

1D Waves

1D waves can not be used for 3D plots.

If you have three 1D waves that represent X, Y and Z coordinates which you want to display as a 3D plot, you must convert them into a triplet wave. For example:

```
Concatenate {xWave,yWave,zWave}, tripletWave
```

Now you can plot the triplet wave using one of the methods described below.

The conversion of three 1D waves into a triplet wave is appropriate when the data are not sampled on a rectangular grid. If you know that your data are sampled on a rectangular grid you should convert the wave that contains your Z data into a 2D wave using the Redimension operation and then proceed to plot the surface using the 2D wave. You can perform this conversion, for example, using the commands:

```
Duplicate/O zWave, zMatrixWave
Redimension/N=(numRows,numColumns) zMatrixWave
```

2D Waves

A 2D wave, sometimes called a "matrix of Z values", is an M-row by N-column wave where each element represents a scalar Z value. You can apply wave scaling (see **Waveform Model of Data** on page II-57) to associate an X value with each row and a Y value with each column.

2D waves can be displayed as 3D graphics in **Surface Plots** and **3D Bar Plots**.

(2D waves can also be displayed as 2D graphics in **Image Plots**, **Contour Plots**, and **Waterfall Plots**.)

Triplet Waves

A triplet wave is an M-row by 3-column wave containing an XYZ triplet in each row. The X value appears in the first column, the Y value in the second and the Z value in the third. A triplet wave is a 2D wave interpreted as containing X, Y and Z coordinates.

Triplet waves can be displayed as 3D graphics in **3D Scatter Plots**, **Surface Plots**, **Path Plots** and **Ribbon Plots**. In a surface plot the triplet wave defines triangles on a surface.

(Triplet waves can also be displayed as 2D graphics in **Contour Plots**.)

3D Waves

A 3D wave, sometimes called a "volume", is an M-row by N-column by L-layer wave where each element represents a scalar Z value. To be used in 3D graphics, 3D waves must contain at least two elements in each dimension.

3D waves can be displayed as 3D graphics in **Surface Plots**, **Isosurface Plots**, and **Voxelgram Plots**.

(3D waves can also be displayed as 2D graphics in **Image Plots** where a single layer of a 3D wave is displayed as an image.)

Gizmo Overview

Igor's 3D plotting tool is called "Gizmo".

Gizmo is based on OpenGL, an industry standard system for 3D graphics. Gizmo converts Igor data and commands into OpenGL data and instructions and displays the result in an Igor window called a "Gizmo window".

You create a Gizmo window by choosing Windows→New→3D Plot and then appending graphic objects using the Gizmo menu. You can do this without any knowledge of OpenGL. At this level you can create surface plots, scatter plots, path plots, voxelgrams, isosurface plots, 3D bar plots and 3D pie charts. Such objects are called "wave-based objects" or "data objects" to differentiate them from drawing objects, discussed next. After creating the basic plot you can modify various properties using Gizmo's "info" window.

Advanced users can also construct graphics using a set of 3D primitives including lines, triangles, quadrangles, cubes, spheres, cylinders, disks, tetrahedra and pie wedges. Such objects are called "drawing objects" to differentiate them from wave-based objects. If you apply proper scaling you can combine drawing objects and wave-based objects in the same Gizmo window.

Gizmo supports advanced OpenGL features such as lighting effects, shininess and textures. Advanced users who want to create sophisticated 3D graphics will benefit from some familiarity with 3D graphics in general and OpenGL in particular.

System Requirements

Much of Gizmo's operation depends on your computer's graphics hardware and its graphics driver software. We suggest running Gizmo on hardware that includes a dedicated graphics card with at least 512MB of VRAM. Gizmo should also work on computers with onboard graphics and shared memory but you will experience slower performance and may get errors when exporting graphics.

Hardware Compatibility

Gizmo graphics may be affected by the version of OpenGL that you are running. This depends on your graphics hardware, graphics driver version and graphics acceleration settings.

Compatibility with Previous Versions

Prior to Igor Pro 7.00 Gizmo was implemented as an XOP (plug-in). It is now built into Igor.

Some changes were made to Gizmo in Igor7, mostly because of changes to OpenGL. See **Gizmo Compatibility** on page II-382 for details.

Gizmo Guided Tour

The tutorials in the following sections will give you a sense of Gizmo's basic capabilities, how you create basic Gizmo plots, and the type of data that you need for a given type of plot.

At various points in the tour you are instructed to execute Igor commands. The easiest way to do this is to open the “3D Graphics” help file using the Help→Help Windows submenu and do the tour from the help

file rather than from this manual. Then you can execute the commands by selecting them in the help file and pressing Ctrl-Enter.

Gizmo 3D Scatter Plot Tour

In this tour we will create a triplet wave containing XYZ data which we will plot as a 3D scatter plot.

1. **Start a new experiment by choosing File→New Experiment.**
2. **To create a triplet wave containing XYX scatter data, execute:**

   ```
   Make/O/N=(20,3) data = gnoise(5)
   data[][2] = 2*data[p][0] - 3*data[p][1] + data[p][0]^2 + gnoise(0.05)
   ```

   This scatter data represents random locations in the XY plane with Z values that are approximately equal to a polynomial function in X and Y.

   As we see next, the quickest way to display this data in Gizmo is by right-click selecting "Gizmo Plot" in the Data Browser.
3. **Choose Data→Data Browser.**

   The Data Browser window appears.
4. **Right-click the data wave icon and choose New Gizmo Plot.**

   Igor created a Gizmo 3D scatter plot from the data wave in a new window named Gizmo0.

   It also created the Gizmo info window entitled "Gizmo0 Info".

   (If you don't see the "Gizmo0 Info" window, choose Gizmo→Show Info.)

   You can rotate the Gizmo scatter plot by dragging the contents of the Gizmo0 window and using the arrow keys. Feel free to play.

   ...

   That was entirely too easy and not very instructive so we will redo it without using the Data Browser shortcut.
5. **Start a new experiment by choosing File→New Experiment.**
6. **To create a triplet wave containing XYX scatter data, execute:**

   ```
   Make/O/N=(20,3) data = gnoise(5)
   data[][2] = 2*data[p][0] - 3*data[p][1] + data[p][0]^2 + gnoise(0.05)
   ```
7. **Choose Windows→New→3D Plot.**

   Notice from the history area of the command window that Igor has executed:

   ```
   NewGizmo
   ```

   Igor also created an empty Gizmo0 window and the Gizmo0 Info window.

   (If you don't see the "Gizmo0 Info" window, choose Gizmo→Show Info.)
8. **Click the + icon at the bottom of the object list in the Gizmo0 Info window and choose Scatter.**

   Igor displays the Scatter Properties dialog.
9. **Choose data from the Scatter Wave menu.**

   This menu displays only triplet waves. If you want to create a 3D scatter plot, you must have a triplet wave.

   There are several additional options but we will leave them in their default states for now.
10. **Click Do It.**

    Igor created a 3D scatter object named scatter0 and added it to the object list in the info window. It is not yet visible in the Gizmo0 window because we have not yet added it to the display list.
11. **Drag the scatter0 object from the object list to the display list.**

    Spheres representing the XYZ data appear in the Gizmo0 window. Although the plot is by no means complete, you can click and drag the body of the Gizmo0 window to rotate the display.

12. **In the Gizmo Info window, click the "+" icon at the bottom of the object list and choose Axes.**

    The Axes Properties dialog appears. Click the Axis tab if it is not already selected.

13. **Click the Axis tab if it is not already selected.**

    The Axis Type pop-up menu should be set to Box and all of the axis checkboxes (X0, X1...Z2, Z3) should be checked.

14. **Click Do It.**

    Igor created an axis object named axes0 and added it to the object list in the info window. It is not yet visible in the Gizmo0 window because we have not yet added it to the display list.

15. **Drag the axes0 object from the object list to the display list.**

    You now have box axes around the scatter spheres.

16. **Double-click the axes0 object in either the object list or the display list.**

    The Axes Properties dialog reopens.

17. **Click the Ticks and Labels tab.**

18. **Select X0 from the Axis pop-up menu and check the Show Tick Marks and Show Numerical Labels checkboxes.**

19. **Select Y0 from the Axis pop-up menu and check the Show Tick Marks and Show Numerical Labels checkboxes.**

20. **Select Z0 from the Axis pop-up menu and check the Show Tick Marks and Show Numerical Labels checkboxes.**

21. **Click Do It.**

    You now have labeled tick marks for the X0, Y0 and Z0 axes.

    But which axis is which?

22. **Right-click the body of the Gizmo0 window and select Show Axis Cue.**

    Igor adds an axis cue that shows you which dimension is which.

    Rotate the display a bit to get a sense of the axis cue.

23. **Click the close box of the Gizmo0 Info window.**

    The Gizmo0 Info window is hidden. It is usually of interest while you are constructing or tweaking a 3D plot and can be hidden when you just want to view the plot. You can make it visible at any time by choosing Gizmo→Show Info or by right-clicking the Gizmo0 window and choosing Show Info Window.

24. **Click the close box of the Gizmo0 window.**

    Igor displays the Close Window dialog asking if you want to save the window as a window recreation macro. This works the same as a graph recreation macro.

25. **Click the Save button to save the window recreation macro.**

    The recreation macro is saved in the main procedure window.

26. **Choose Windows→Procedure Windows→Procedure Window.**

    This displays the main procedure window containing the Gizmo0 recreation macro. You may need to scroll up to see the beginning of it which starts with:

    ```
    Window Gizmo0() : GizmoPlot
    ```

27. **Close the procedure window by clicking its close box.**

28. **Choose Windows→Other Macros→Gizmo0.**

    Igor executes the Gizmo0 recreation macro which recreates the Gizmo0 window.

29. **Choose File→Save Experiment and save the experiment as "Gizmo 3D Scatter Plot Tour.pxp".**

    This is just in case you want to revisit the tour later and is not strictly necessary.

At this point, you have completed the construction of a 3D scatter plot. As you may have noticed, there are many scatter plot and axis options that we did not explore. You can do that now, by double-clicking the scatter0 and axes0 icons in the Gizmo Info window, or you can leave that for later and continue with the next section of the tutorial.

Gizmo Surface Plot Tour

In this tour we will create a 2D wave containing Z values which we will plot as a surface plot.

1. **Start a new experiment by choosing File→New Experiment.**
2. **To create a 2D matrix of Z values, execute:**

   ```
   Make/O/N=(100,100) data2D = Gauss(x,50,10,y,50,15)
   ```

   This matrix data represents Z values on a regular grid.

   As you saw in the first tour, the quickest way to display this data in Gizmo is by right-clicking an appropriate wave in the Data Browser and selecting Gizmo Plot. The same shortcut works with a matrix of Z values. But we will do it the hard way as this will give you a better understanding of Gizmo.
3. **Choose Windows→New→3D Plot.**

   Notice from the history area of the command window that Igor has executed:

   ```
   NewGizmo
   ```

   Igor also created an empty Gizmo0 window and the Gizmo0 Info window.
4. **Click the + icon at the bottom of the object list in the Gizmo0 Info window and choose Surface.**

   Igor displays the Surface Properties dialog.
5. **Choose Matrix from the Source Wave Type pop-up menu and data2D from the Surface Wave pop-up menu.**

   There are several additional options but we will leave them in their default states for now.
6. **Click Do It.**

   Igor created a surface object named surface0 and added it to the object list in the info window. It is not yet visible in the Gizmo0 window because we have not yet added it to the display list.
7. **Drag the surface0 object from the object list to the display list.**

   The surface appears in the Gizmo0 window. You are now looking at it from the top.
8. **Using the mouse, rotate the surface in the Gizmo0 window to reorient it so you can see the side view of the Gaussian peak.**
9. **In the Gizmo info window, click the "+" icon at the bottom of the object list and choose Axes.**

   The Axes Properties dialog appears. Click the Axis tab if it is not already selected.
10. **Click the Axis tab if it is not already selected.**

    The Axis Type pop-up menu should be set to Box and all of the axis checkboxes (X0, X1...Z2, Z3) should be checked.
11. **1Click Do It.**

    Igor created an axis object named axes0 and added it to the object list in the info window. It is not yet visible in the Gizmo0 window because we have not yet added it to the display list.
12. **Drag the axes0 object from the object list to the display list.**

    You now have box axes around the surface plot.
13. **Double-click the axes0 object in the display list. Using the resulting Axes Properties dialog, turn on tick marks and tick mark labels for the X0, Y0 and Z0 axes.**

    This is the same as what we did in the preceding tour.

    Now we will add a colorscale annotation.

14. **Choose Gizmo→Add Annotation.**

    The Add Annotation dialog appears.

15. **From the Annotation pop-up menu in the top/left corner of the dialog, choose ColorScale.**

16. **Click the Position tab and set the Anchor pop-up menu to Right Center.**

17. **Click the ColorScale Main tab and set the following controls as indicated:**

    Color Table: Rainbow

    Axis Range/Top: 100

    Axis Range/Bottom: 0

18. **Click Do It.**

    Igor creates the colorscale annotation.

    Note that its tick mark labels don't agree with the Z axis tick mark labels in the surface plot. We need to set the colorscale range to match the range of the data. We will do this by re-executing the command that created the colorscale.

19. **Click on the last command in the history area of the command window and press Enter to copy it to the command line.**

    The command line should now show this:

    ```
    ColorScale/C/N=text0/M/A=RC ctab={0,100,Rainbow,0}
    ```

20. **Edit the command as shown next and press Enter to execute it:**

    ```
    ColorScale/C/N=text0/M/A=RC ctab={WaveMin(data2D),WaveMax(data2D),Rainbow,0}
    ```

    Now the colorscale tick mark labels agree with the Z axis tick mark labels in the surface plot.

    Next we will add an image plot beneath the surface plot.

21. **In the Gizmo info window, click the "+" icon at the bottom of the object list and choose Image.**

    The Gizmo Image Properties dialog appears.

22. **Set the Source Type to 2D Matrix of Z Values, and select data2D from the Source Wave pop-up menu.**

    We will use the same wave as the source for both the surface plot and the image.

23. **From the Intitial Orientation pop-up menu, choose XY Plane Z=0.**

24. **Uncheck all checkboxes except Translate and set the X, Y, and Z components of the translation to 0, 0, and -1 respectively.**

    The translation in the Z direction moves the image from the center of the display volume to the bottom of the display volume, placing it on the "floor" of the surface plot.

25. **Click Do It.**

    Igor created an image object named image0 and added it to the object list in the info window. It is not yet visible in the Gizmo0 window because we have not yet added it to the display list.

26. **Drag the image0 object from the object list to the display list.**

    You now have an image plot below the surface plot. You may need to rotate the display to see it.

27. **Double-click the image0 object in the object list, set the Z translation to -1.5, then click Do It.**

28. **This separates the image plot from the surface plot, making it easier to see.**

    The translation parameters are in +/-1 display volume units, not in the units of the axes. Display volume units are used in many instances, especially when dealing with drawing objects such as spheres and boxes.

29. **Choose File→Save Experiment and save the experiment as "Gizmo Surface Plot Tour.pxp".**

    This is just in case you want to revisit the tour later and is not strictly necessary.

Gizmo 3D Scatter Plot and Fitted Surface Tour

In this tour we will create a 3D scatter plot from a triplet wave, perform a curve fit, and append a surface showing how the curve fit output relates to the original scatter data.

1. **Start a new experiment by choosing File→New Experiment.**
2. **To create a triplet wave containing XYX scatter data, execute:**

   ```
   Make/O/N=(20,3) data = gnoise(5)
   data[][2] = 2*data[p][0] - 3*data[p][1] + data[p][0]^2 + gnoise(0.05)
   ```

   This scatter data represents random locations in the XY plane with Z values that are approximately equal to a polynomial function in X and Y.
3. **Choose Data→Data Browser.**

   The quickest way to display this data in Gizmo is by right-click selecting "Gizmo Plot" in the Data Browser.
4. **Right-click the data wave icon and choose New Gizmo Plot.**

   Igor created a Gizmo 3D scatter plot from the data wave in a new window named Gizmo0.
5. **In the command line, execute:**

   ```
   CurveFit/Q poly2d 2, data[][2]/X=data[][0,1] /D
   ```

   Igor performs a 2D polynomial curve fit and produces output waves and variables. The main output is in the wave fit_data.
6. **Close the Data Browser window.**

   This is just to reduce clutter on your screen.

   Next we will now add a surface to the Gizmo plot.
7. **Click the + icon at the bottom of the object list in the Gizmo0 Info window and choose Surface.**

   Igor displays the Surface Properties dialog.
8. **Choose Matrix from the Source Wave Type pop-up menu and fit_data from the Surface Wave pop-up menu.**

   There are several additional options but we will leave them in their default states for now.
9. **Click Do It.**

   Igor created a surface object named surface0 and added it to the object list in the info window. It is not yet visible in the Gizmo0 window because we have not yet added it to the display list.
10. **Drag the surface0 object from the object list to the display list.**

    The surface appears in the Gizmo0 window and appears to fit the scatter objects pretty well.
11. **Using the mouse, rotate the contents of the Gizmo plot to inspect the fit from various angles.**
12. **Choose File→Save Experiment and save the experiment as "Gizmo 3D Scatter Plot and Fitted Surface Tour.pxp".**

    This is just in case you want to revisit the tour later and is not strictly necessary.

Gizmo Surface Using Voronoi Interpolation Tour

We have already seen how to create a surface plot from a 2D matrix of Z values. In this tour we illustrate the how to plot a 3D surface representation of XYZ scatter data. The process involves triangulation of the XYZ data using Voronoi interpolation.

1. **Start a new experiment by choosing File→New Experiment.**
2. **To create a triplet wave containing XYZ scatter data, execute:**

   ```
   Make/O/N=(20,3) data = enoise(5)
   data[][2] = 2*data[p][0] - 3*data[p][1] + data[p][0]^2 + gnoise(0.05)
   ```
3. **Choose Data→Data Browser.**

4. **Right-click the data wave icon and choose New Gizmo Plot.**

   Igor created a Gizmo 3D scatter plot from the data wave in a new window named Gizmo0.

   Next we will triangulate the scatter data using Voronoi interpolation.

5. **Execute this in the command line:**

   ```
   ImageInterpolate/S={-5,0.1,5,-5,0.1,5}/CMSH Voronoi data
   ```

   The Voronoi interpolation created two waves: M_ScatterMesh and M_InterpolatedImage. M_ScatterMesh consists of a series of XYZ coordinates that define polygons in 3D space which fit the scatter data. We will use M_ScatterMesh to append a surface to the 3D plot.

6. **Click the + icon at the bottom of the object list in the Gizmo0 Info window and choose Surface.**

   Igor displays the Surface Properties dialog.

7. **Choose Triangles from the Source Wave Type pop-up menu and M_ScatterMesh from the Surface Wave pop-up menu.**

   There are several additional options but we will leave them in their default states for now.

8. **Click Do It.**

   Igor created a surface object named surface0 and added it to the object list in the info window. It is not yet visible in the Gizmo0 window because we have not yet added it to the display list.

9. **Drag the surface0 object from the object list to the display list.**

   The surface appears in the Gizmo0 window.

10. **Using the mouse, rotate the contents of the Gizmo plot to inspect the fit from various angles.**

    The surface fits the scatter objects pretty well.

11. **Double-click the surface0 object in the display list, click the Grid Lines and Points tab, check the Draw Grid Lines checkbox, and click Do It.**

    This shows the polygons created by Voronoi interpolation and represented by the M_ScatterMesh wave.

12. **Clean up by executing:**

    ```
    KillWaves M_InterpolatedImage
    Rename M_ScatterMesh, VoronoiMesh
    ```

    It's a good idea to rename waves that Igor creates with default wave names so that, if you later execute another command that uses the same default wave name, you will not inadvertently overwrite data. Also we don't need the M_InterpolatedImage wave.

13. **Choose File→Save Experiment and save the experiment as "Gizmo Surface Using Voronoi Interpolation Tour.pxp".**

    This is just in case you want to revisit the tour later and is not strictly necessary.

That concludes the Gizmo guided tour. There are more examples below. Also choose File→Example Experiments→Visualization for sample experiments.

Gizmo Windows

For each 3D plot, Gizmo creates a display window and its associated info window. The display window presents a rotatable representation of your 3D objects. You use the info window to control which objects are displayed, the order in which they are drawn, and their properties. You can hide both windows to reduce clutter when you do not need them. You can also kill and recreate Gizmo windows like you kill and recreate graphs.

You can create any number of Gizmo display windows. Keeping multiple Gizmo display windows open has some drawbacks. Even inactive and hidden Gizmo display windows consume graphics resources that could otherwise be used for the active Gizmo display window. Also, in some laptop computers you may be

able to reduce power consumption by closing Gizmo display windows. Depending on your hardware, saving unused Gizmo display windows as recreation macros may be beneficial.

For brevity, we sometimes use the term "Gizmo window" to refer to the Gizmo display window. We use "Gizmo info window" or "info window" to refer to the Gizmo info window.

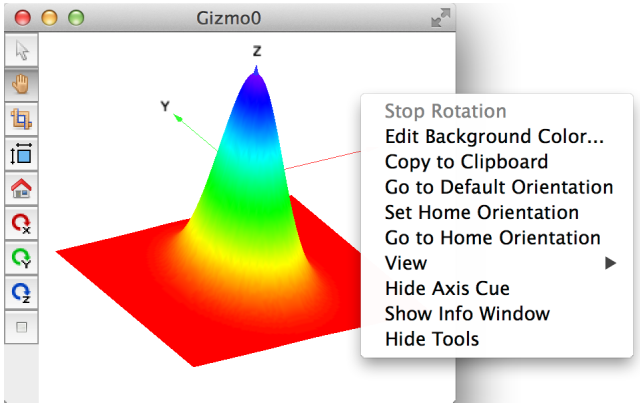
The Gizmo Display Window

The Gizmo display window presents a rotatable 3D display of objects representing waves and 3D drawing primitives as specified by the display list in the associated Gizmo info window.

You can display a palette of tools by choosing Gizmo→Show Tools.

You can access a pop-up menu to modify the appearance of the Gizmo display window by right-clicking (Windows) or Ctrl-clicking (Macintosh) in the display window.

Here is what the display window looks like with the tool palette and pop-up menu showing:



You can rotate the scene in the window by clicking and dragging the mouse as if you were rotating a virtual trackball positioned at the center of the window. You can also use the tool palette, mouse wheel, cursor keys and the x, y, z keys on your keyboard to rotate the scene.

Gizmo Display Window Tool Palette

To display the Gizmo tool palette, choose Gizmo→Show Tools or right-click and choose Show Tools. The tool palette contains the following icons, from top to bottom:

Arrow Tool

When the arrow tool is selected, dragging the body of the display window rotates the 3D scene. The arrow tool and the hand tool are mutually exclusive.

Hand Tool

When the hand tool is selected, dragging the body of the display window pans the 3D scene. The arrow tool and the hand tool are mutually exclusive.

Axis Tool

Clicking the axis tool displays the Axis Range dialog. This is equivalent to choosing Gizmo→Axis Range.

Aspect Ratio Tool

Toggles "aspect ratio" mode.

When aspect ratio mode is off, the length of each axis is the same.

When aspect ratio mode is on, the length of each axis is proportional to the range of data displayed against that axis.

Home Tool

Clicking the home tool sets the X, Y and Z rotation angles to 0 or to some other orientation that you designated as "home".

Rotate About X

Clicking the Rotate About X tool starts the 3D scene rotating about the X axis. To stop it, click the Stop tool or click once in the body of the display window.

Rotate About Y

Clicking the Rotate About Y tool starts the 3D scene rotating about the Y axis. To stop it, click the Stop tool or click once in the body of the display window.

Rotate About Z

Clicking the Rotate About Z tool starts the 3D scene rotating about the Z axis. To stop it, click the Stop tool or click once in the body of the display window.

Stop Tool

Clicking the stop tool stops all rotation.

Gizmo Display Window Contextual Menu

The Gizmo Display contextual menu provides shortcuts for common tasks. It contains the following items, from top to bottom:

Stop Rotation

Stops the rotation of the 3D scene, if any.

Edit Background Color

Sets the background color for the Gizmo window.

Copy to Clipboard

Copies the Gizmo plot to the clipboard using the format set in the Export Graphics dialog (Edit menu).

Go to Default Orientation

Sets the orientation of the 3D space to the orientation in effect when a new Gizmo window is first created.

Set Home Orientation

Stores the current orientation as the orientation to be used when Go to Home Orientation is selected or the Home icon in the tool palette is clicked.

Go to Home Orientation

Rotates the 3D scene to the home orientation.

View

Rotates the 3D scene to one of several preset orientations.

Show Axis Cue

Displays arrows showing the X, Y and Z directions.

Show Info Window

Shows the Gizmo info window associated with the active Gizmo display window.

Show Tools

Shows the Gizmo tool palette.

Match Window Size

Sets another Gizmo display window to the same size as the active Gizmo display window.

This item appears only if you have multiple Gizmo display windows.

Rotate to Match

Rotates another Gizmo display window to the same orientation as the active Gizmo display window.

This item appears only if you have multiple Gizmo display windows.

Sync to Gizmo

Locks the rotation of the current Gizmo plot to the same orientation as the another Gizmo plot. If you rotate the other plot, both rotate to the same orientation.

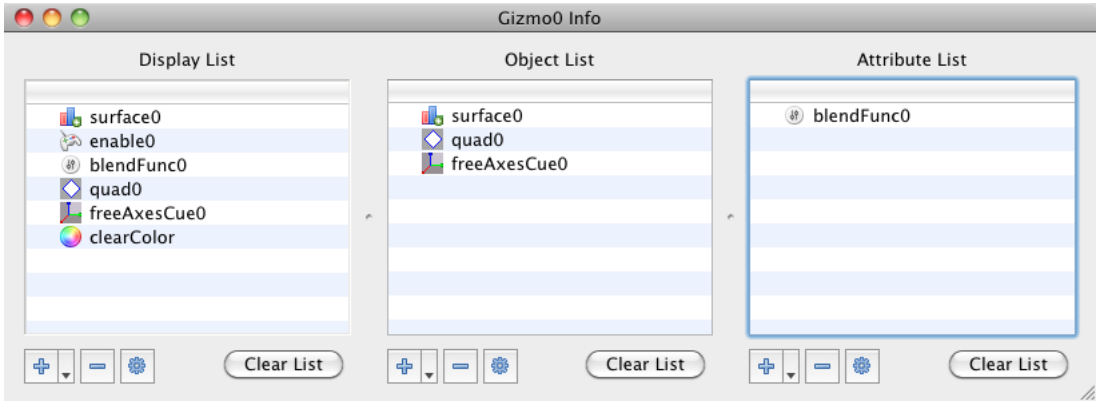
To have mutual syncing you must set each window to be synced to the other.

This item appears only if you have multiple Gizmo display windows.

The Gizmo Info Window

The Gizmo info window is the main user interface for controlling the display of objects in the Gizmo display window. Each info window has an associated Gizmo display window that shows the resulting graphical plot.

The info window contains three lists: the display list, the object list, and the attribute list. You use these lists to add objects and to modify their appearance in the Gizmo display window. Only items that appear in the display list are actually drawn in the Gizmo display window.



You can create objects using the + icon at the bottom of the object list. Clicking the + icon displays a menu from which you choose the type of object to add. The + icon below the attribute list adds attributes while the + icon below the display list adds operations.

The object list contains only objects and the attribute list contains only attributes. The display list can contain objects dragged in from the object list, attributes dragged in from the attribute list, and operations.

You can edit an item's properties by double-clicking its icon or by selecting it and clicking the gear icon below the list. You can remove an item from a list by selecting it and pressing the delete key or by clicking the - icon.

You can drag an item from a list in one Gizmo info window to a list in another Gizmo info window so long as the item is appropriate for the destination list.

The Gizmo Object List

The middle list in the info window is the object list. It lists all of the objects that you have created which are then available for use in the display list.

Gizmo supports many types of objects including wave-based objects such as surface plots and drawing primitives such as spheres. If you click the + icon under that object list you see a menu of the available object types. See **Gizmo Objects** on page II-331 for details.

For an object to appear in the Gizmo plot, you must drag it to the display list.

The Gizmo Display List

The display list controls what actually appears in the Gizmo display window. Gizmo processes the items in the display list in the order in which they appear.

In addition to objects that you drag in from the object list and attributes that you drag in from the attribute list, you can add the following operations to the display list:

ClearColor, ColorMaterial, Translate, Rotate, Scale, Main Transformation, Enable, Disable and Ortho.

Using the ColorMaterial, Enable and Disable operations requires some familiarity with OpenGL.

The Main Transformation item is used in conjunction with lighting. This is described under **Gizmo Positional Lights** on page II-347.

The Ortho operation controls the projection of the 3D space onto the 2D screen. This is described under **Gizmo Projections** on page II-335.

If you are familiar with OpenGL, you should note that Gizmo automatically generates a small number of OpenGL instructions e.g., viewing transformation, default lighting, etc., that are not visible on the list. If you provide your own alternatives the various defaults are simply not executed. For example, by default Gizmo provides a neutral ambient light to illuminate the scene. However, if you add one or more lights to the display list, the default ambient light is omitted.

Item Ordering in the Gizmo Display List

You can reorder items in the display list by dragging and dropping them in the desired locations. The order of items in the display list is important because it determines the order of execution of OpenGL drawing instructions which govern the appearance of the plot. This becomes obvious when you use operations such as translation, rotation, or scaling.

There are a few items for which the exact position in the list does not make any difference, but in the majority of cases a change in the order of items produces a visible change in the display. For example, if you switch the order of rotation and translation operations you will get a completely different result; see **Gizmo Object Rotation** on page II-338 for an example.

The Gizmo Attribute List

The attribute list appears on the right side of the info window.You create an attribute by clicking the + icon and selecting the type of attribute you want. You then drag that attribute into the display list as a global attribute or on top of an item in the object list as an embedded attribute.

The order of items in the attribute list is unimportant.

Attributes are discussed in detail under **Gizmo Attributes** on page II-332.

Gizmo Objects

There are five main categories of Gizmo objects: wave-based objects, axis objects, drawing primitive objects, lights, and miscellaneous objects.

Wave-based Gizmo objects, also called "data objects", get their data from waves and include the following types:

- **Path Plots**
- **Ribbon Plots**
- **Surface Plots**
- **Isosurface Plots**
- **Voxelgram Plots**
- **3D Scatter Plots**
- **3D Bar Plots**
- **Gizmo Image Plots**

Axis object types include:

- **Axis Objects**
- **Axis Cue Objects**

Drawing primitive object types include:

- **Line Objects**
- **Triangle Objects**
- **Quad Objects**
- **Box Objects**
- **Sphere Objects**
- **Cylinder Objects**
- **Disk Objects**
- **Tetrahedron Objects**
- **Pie Wedge Objects**

There is only one light object type:

- Light Objects (see **Gizmo Colors, Material and Lights** on page II-341)

Miscellaneous object types include:

- **Group Objects**
- **Texture Objects**
- **Matrix4x4 Objects**

You create an object by clicking the + icon below the object list in the info window for a given Gizmo display window.

An object of a given type has a set of internal properties that you can edit when you first create the object. You can edit them later by double-clicking the object in the object or display lists.

Creating a Gizmo object puts it in the object list. It is not displayed until you drag it to the display list. You can drag a given object to the display list multiple times. This creates an new display object each time.

Gizmo Attributes

A Gizmo attribute encapsulates a setting which you can then apply to the Gizmo display list as a global attribute or to a specific Gizmo object as an embedded attribute.

Gizmo supports the following types of attributes:

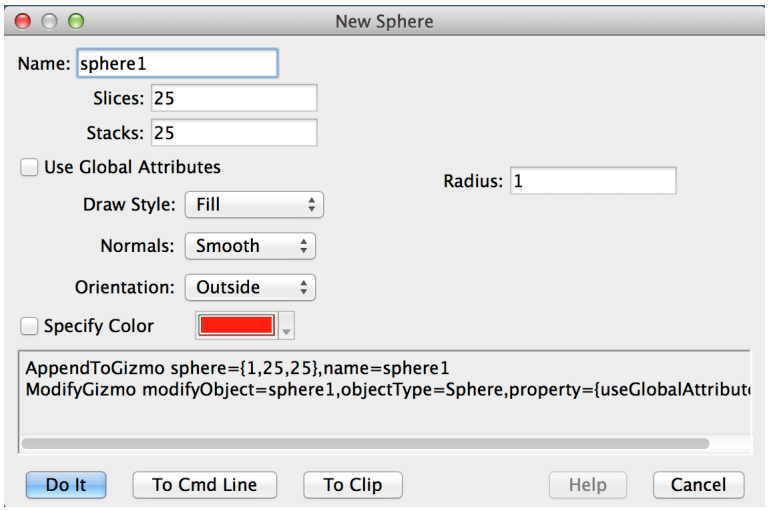
- Color
- Ambient
- Diffuse
- Specular
- Shininess
- Emission
- Blending
- Point size
- Line width
- Alpha test function

You create an attribute using the attribute list in the info window. You can then drag the attribute to the display list as a global attribute or into an object in the object list as an embedded attribute.

In addition to global and embedded attributes, Igor7 added internal attributes, described in the next section.

Internal Attributes

Internal attributes are built into objects. For example, the New Sphere dialog looks like this:



The draw style, normals, orientation and color settings are internal attributes of the sphere object.

The Use Global Attributes checkbox disables the controls under it and enables the use of the respective global attributes for the object in question. It does not affect the use of other global attributes.

The Specify Color checkbox does the same for color. If unchecked, the object has no intrinsic color. In this case you must add a color material operation and a color attribute to the display list before the object. If Specify Color is checked, Gizmo creates a default color material for the object and uses the specified internal color attribute.

Prior to Igor7, Gizmo supported no internal attributes so you had to use global or embedded attributes. As of Igor7, we recommend that you use internal attributes if they are available in preference to global or embedded attributes.

Global Attributes

When you drag an attribute to the display list, it acts as a global attribute that affects all objects later in the display list.

If you place an attribute such as color in the display list, OpenGL draws all subsequent objects that do not have an internal color specification using this global color. You also need a color material operation in order to see the applied color.

Internal attributes and embedded attributes override global attributes.

Embedded Attributes

Embedded attributes are deprecated and are supported mainly for backward compatibility. We don't recommend their use in new projects because primitive obects now have their own internal attributes which should be used instead.

When you drag an attribute on top of an object in the object list, it becomes embedded in that object. You can embed a given attribute in any number of objects and you can embed any number of attributes in a given object.

For example, if you create a sphere object and you want it to appear in blue, you can create a blue color attribute in the attribute list and drop it on top of the sphere object in the object list. The advantage of doing so, as opposed to directly setting the internal color attribute of the sphere object, is in allowing you to reuse the same color attribute with multiple objects. With this approach, by changing a single attribute you can change the color of all associated objects.

Internal attributes override embedded attributes and global attributes.

When an object with embedded attributes is drawn, Gizmo first stores the state of the drawing environment. It then executes the embedded attributes immediately before the object is drawn and finally it restores the state of the drawing environment. As a result, embedded attributes affect only the object in which they are embedded.

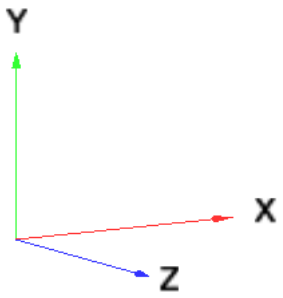
If you apply conflicting attributes to a given object, only the last attribute in the embedded list affects the object appearance. For example, if you have a sphere object with the following embedded attributes: red color, blue color and green color, the sphere is drawn in green.

Gizmo Display Environment

Gizmo constitutes an environment for displaying 3D graphics. This section discusses the main properties of that environment.

Gizmo Coordinate System

Gizmo uses a right-handed, 3D coordinate system. If the positive X axis points to the right and the positive Y axis points up then, by the right-hand rule, the positive Z axis points out of the screen towards you.



Gizmo Dimensions

The default Gizmo viewing volume is a space that is 4 units wide in all three dimensions. The actual display volume is two units in each dimension, centered in the middle of the viewing volume. Each dimension of

the display volume extends from -1 to +1 about the origin. The display volume is smaller than the viewing volume to avoid clipping at the corners when the plot is rotated.

All drawing objects, such as spheres and cylinders, are sized in units of the +/-1 display volume. So, for example, if you create a box that is 2 units on a side, it completely fills the display volume. If you create a cylinder that is 3 units high, then the top of the cylinder is clipped because it extends outside the viewing volume boundary.

Superimposed on the display volume and precisely filling it is an axis coordinate system against which wave-based data objects such as scatter and surface plots are plotted. You can set the axis coordinate range for each dimension by choosing Gizmo→Axis Range. The axis coordinate system exists even though, by default, no axes are visible.

The axis coordinate system is autoscaled by default. Consequently, when you initially display a wave-based object, it fills the range of each axis. Since the axis coordinate system fills the display volume, the displayed wave-based object also fills the display volume.

When you display two or more wave-based objects at the same time while the axes are set to autoscale, Gizmo sets the range of each axis based on the minimum and maximum in the respective dimension of all data objects combined.

Once you turn autoscaling off, the axis range that you set determines the extent to which wave-based objects fill the display volume.

When you combine drawing objects and wave-based objects, the dimensions and positions of the drawing objects remain in +/-1 display volume units whereas the wave-based objects are displayed against the axis coordinate system.

Gizmo Clipping

When you set the range of any axis, you may use values that do not include the full range of the data. To display the results correctly in this case, Gizmo creates clipping planes on the relevant sides of the display volume. Once created, these clipping planes affect both wave-based data objects and drawing objects. The clipping planes are not created unless a data object extends beyond the range of the axes.

If you want to do your own clipping, this automatic Gizmo clipping may intefere. To disable automatic clipping, for example for a surface object named surface0, you can execute:

```
ModifyGizmo modifyObject=surface0, objectType=surface, property={Clipped,0}
```

If you are working in advanced mode (see **Advanced Gizmo Techniques** on page II-377), you can create custom clipping planes to create special effects such as gaps in a surface plot. To use clipping planes, make sure that you are not using an axis range that is smaller than the span of the data in any dimension. Current graphics hardware support 6 to 8 clipping planes and axis-range clipping planes have a priority. For an example, open the Clipping Demo experiment.

Gizmo Projections

A number of different projections can be used to control the display of 3D objects in a 2D Gizmo window.

By default Gizmo uses a built-in default orthographic projection which does not appear in the display list. The default orthographic projection is ideal for most applications.

Gizmo calculates the parameters to use with the default orthographic projection by scanning all objects in the display list and computing their largest extent. The default orthographic projection will be 2 units in all directions which allows full rotation of wave-based objects without clipping.

The automatic scanning of objects on the display list does not take into account optional translation, rotation, or scaling operations. If you use any of these in the display list you should also provide a separate ortho operation with the appropriate definition of the projected space.

To add a projection operation, click the + icon under the display list. By default the only projection offered is Ortho and this is sufficient for nearly all purposes. In order to choose another type of projection you must check the Display Advanced Options Menus checkbox in the Gizmo section of the Miscellaneous Settings dialog which you can access via the Misc menu.

You can have any number of projection operations on the display list. When there is more than one projection Gizmo executes only the last one.

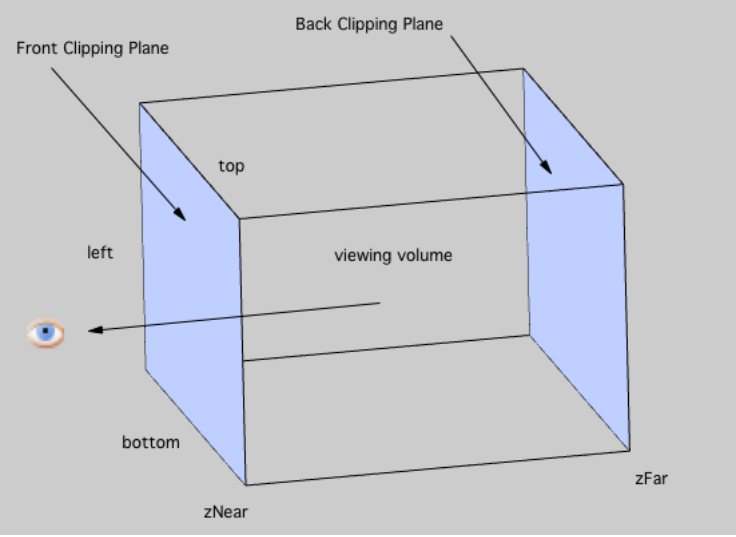
When you are using a projection other than Ortho you must also specify the viewing angle. As a result, the standard mouse rotation and mouse wheel zoom do not apply.

Orthographic

The orthographic projection maintains the geometric orientations and scalings of 3D objects. This is the default projection for the Gizmo display window and is recommended for most purposes.

Unlike perspective projection, discussed in the next section, orthographic projection preserves object parallelism and there is no object foreshortening. Orthographic projection is analogous to an arrangement where dimensions of objects in the displayed scene are small compared to the viewing distance. Another way to think of it is that the image plane is perpendicular to one of the coordinate axes. Only objects contained within the viewing volume are visible. This projection is also faster than perspective.

As shown in this diagram, the ortho projection depends on 6 parameters: left, right, bottom, top, zNear and zFar. The parameters are measured from the center of the display volume and are expressed in +/-1 display volume units.

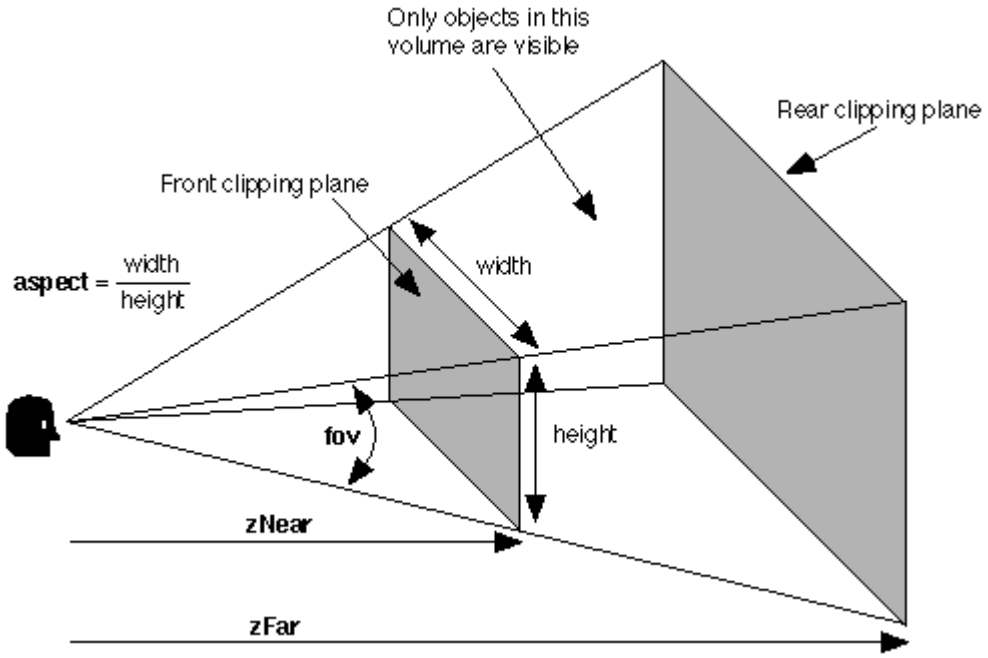


The zoom (using the mouse wheel) and pan tools are implemented by modifying the default ortho projection. If you add your own projection to the display list, the zoom tool is disabled.

Perspective

The perspective projection simulates the way your eye sees 3D objects. Although this is a realistic projection, it does not preserve the exact orientations or shapes of objects; for example, parallel lines may diverge or converge and there is foreshortening of objects. The viewing area is in the shape of a truncated pyramid in which the top has been cut off parallel to the base. Only objects within the viewing volume are visible. This is a symmetric perspective view of the viewing volume; a frustum, discussed in the next section, supports an asymmetric volume.

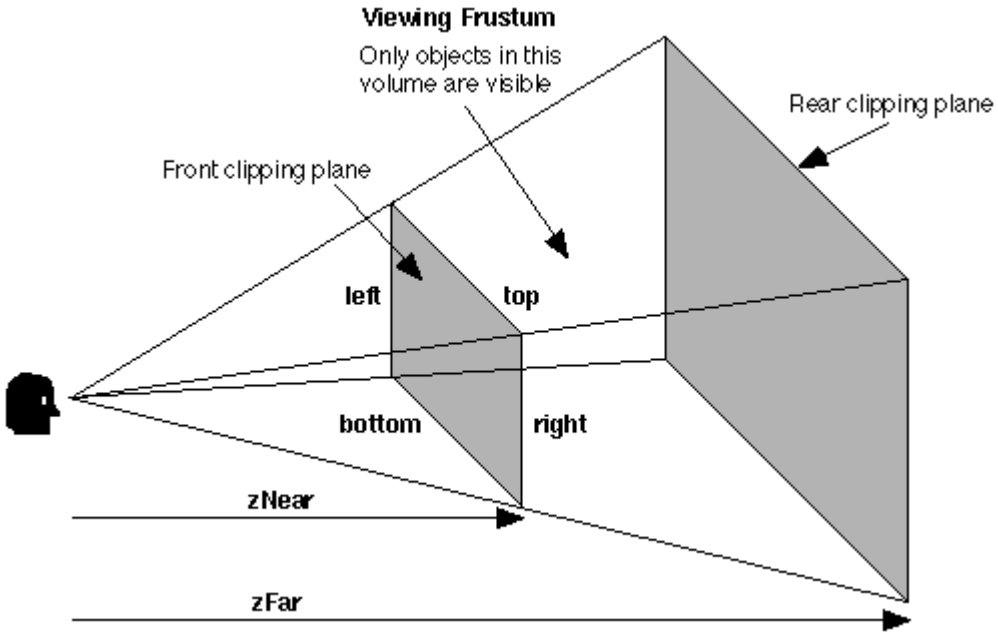
As shown in this diagram, the perspective projection depends on 4 parameters: fov, aspect, zNear and zFar.



Frustum

The frustum projection is essentially the same as the perspective projection, but in this case it has more flexible settings, which means that it does not have to be symmetrical or aligned with the Z axis.

As shown in this diagram, the frustrum projection depends on 6 parameters: left, right, bottom, top, zNear and zFar.



Viewport

A viewport is the rectangular 2D region of the window where the projected scene is drawn. You can use this to scale and distort the scene in the Gizmo display window.

The viewport projection depends on 4 parameters: left, bottom, width and height.

LookAt

A LookAt projection maps the center point to the negative Z axis, the eye point to the center, and the up vector to the Y axis. It may be useful if you want to change the origin of the coordinate system and the basic orientation.

Gizmo Object Rotation

You can click in a Gizmo display and drag the mouse to rotate the plot. The default rotation is implemented as if there is a virtual trackball in the center of the Gizmo window. Rotation through a positive angle is in a counterclockwise direction when viewed along the ray from the origin.

When experimenting with rotation it is helpful to display the axis cue which shows the X, Y and Z directions. Right-click and choose Show Axis Cue. The axis cue shows the orientation of the main Gizmo axes.

You can rotate the plot using the keyboard when a Gizmo display window is active. Press the x, y, or z keys to rotate the display counterclockwise in 1-degree increments about the respective axis. Press the shift key along with x, y, or z to rotate clockwise.

The up arrow and down arrow keys rotate the plot about a horizontal line drawn through the middle of the display area. The left arrow and right arrow keys rotate the plot about a vertical line drawn through the middle of the display area.

By default the mouse scroll wheel zooms the plot. You can change it to rotate the plot using a miscellaneous setting. Choose Misc→Miscellaneous Settings and click Gizmo in the list on the left to see the Gizmo miscellaneous settings. Using the mouse scroll wheel for scrolling behaves the same as using the arrow keys.

You can start the 3D scene rotating continuously by clicking and gently flinging with the mouse. This requires releasing the mouse button before stopping mouse movement. Click the plot once to stop continuous rotation. If the display pans instead of rotates then click the arrow tool in the Gizmo tool palette to enable rotation.

You can also start continuous rotation using the tool palette. Choose Gizmo→Show Tools and then click one of three rotation icons to start rotation about the X, Y or Z axis. Click the plot once or click the stop icon in the tool palette to stop rotation.

The Home icon in the tool palette rotates the scene to the home orientation. By default the home orientation is X=0, Y=0, Z=0. In this case, the positive X axis points to the right, the Y axis points up and the Z axis points out of the plane of the window toward you. You can set the home orientation by right-clicking and choosing Set Home Orientation.

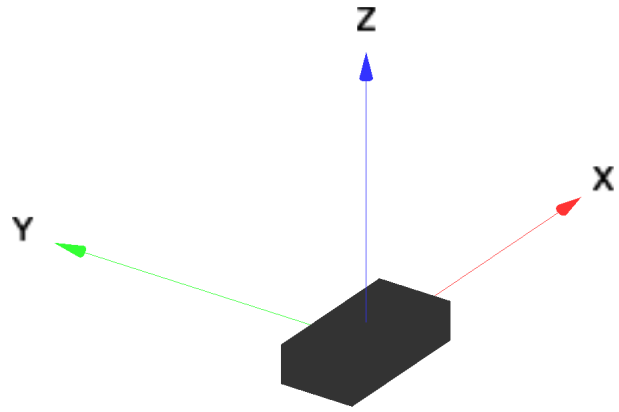
During continuous rotation you can use the x, y and z keys to tweak the orientation.

Gizmo Object Rotation and Translation

In the preceding section we discussed rotation of the entire Gizmo plot about the main Gizmo axes. Each Gizmo object has its own set of axes which, by default, correspond to the main axes. Though this is less frequently needed, it is possible to rotate a Gizmo object's axes independent of the main Gizmo axes. You do this by adding a rotate operation to the display list.

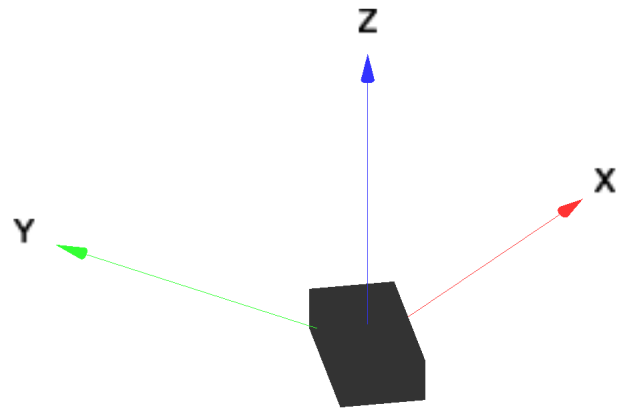
To see this we create a Gizmo with an axis cue and an unrotated box:

```
NewGizmo
ModifyGizmo showAxisCue=1
ModifyGizmo setQuaternion={0.435,-0.227,-0.404,0.777}
AppendToGizmo box={0.5,0.25,0.15}, name=box0
ModifyGizmo setDisplayList=0, object=box0
```



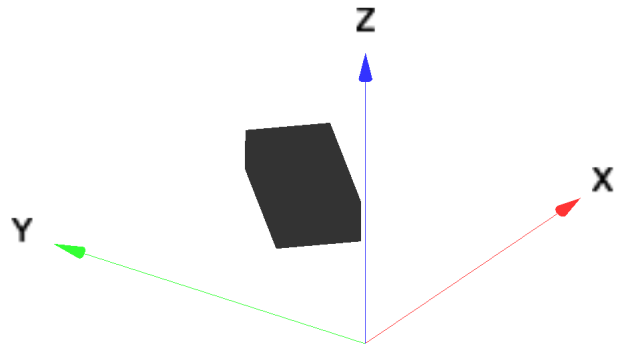
We then add a rotate operation to the display list, before the box object. This rotates the box object's axes by 45 degrees:

```
ModifyGizmo insertDisplayList=0, opName=rotate0, operation=rotate,
      data={45,0,0,1}
```



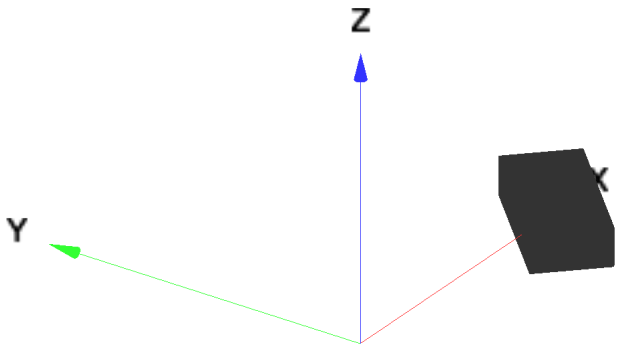
Now we insert a translation along the X axis:

```
ModifyGizmo insertDisplayList=1, opName=translate0, operation=translate,
      data={1,0,0}
```



The translation is along to the object's X axis, not the main X axis. Because of this, rotation followed by translation gives a different result than translation followed by rotation. To illustrate this we switch the order of the operations:

```
ModifyGizmo setDisplayList=0, opName=translate0, operation=translate,
      data={1,0,0}
ModifyGizmo setDisplayList=1, opName=rotate0, operation=rotate, data={45,0,0,1}
```



In this last case, at the time the translation was done, the object's axes were aligned with the main axes. The translation was along the object's X axis which pointed in the same direction as the main X axis.

Translate and rotate operations apply to all objects below them on the display list. They are cumulative. In other words, a translate or rotate operation on a given object starts from the current position or rotation of the object when the operation is applied.

Programming Rotations in Gizmo

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

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

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

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

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

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

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

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

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

To compute a rotation quaternion that represents two consecutive rotations, i.e., a rotation specified by quaternion q1 followed by a rotation specified by quaternion q2, we need to compute the product quater-

nion qr=q2*q1 using quaternion multiplication, which is not commutative. This can be computed using the following function:

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

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

Assume you want to set the Gizmo orientation to the rotation produced by the preceding example (X axis points to the right, Y axis points away from you, and the Z axis points up) followed by a 90-degree rotation about the Z axis, producing the orientation where the X axis points to toward you, the Y axis points to the right, and the Z axis points up. You could execute this:

```
Make/O/N=4 q1={sin(pi/4),0,0,cos(pi/4)}      // Z up, X right, Y away
Make/O/N=4 q2={0,0,sin(pi/4),cos(pi/4)}      // 90 degree rotation about Z
Make/O/N=4 qr// Resultant orientation
MultiplyQuaternions(q2,q1,qr)                // Compute resultant orientation
Print qr
 qr[0]= {0.5,0.5,0.5,0.5}
ModifyGizmo setQuaternion={0.5,0.5,0.5,0.5}
```

Another way to do this is to use the ModifyGizmo appendRotation command which does the quaternion multiplication for you:

```
ModifyGizmo setQuaternion={sin(pi/4),0,0,cos(pi/4)}   // Z up, X right, Y away
ModifyGizmo appendRotation={0,0,sin(pi/4),cos(pi/4)}
```

The last command rotates 90 degrees about the Z axis starting from the current orientation.

Gizmo Colors, Material and Lights

The rendered color of a Gizmo object depends on its internal color, color attributes, material as specified by a color material operation and lighting. The following sections discuss these topics.

Gizmo Color Specification

Colors of objects and lights are specified using four floating point numbers: RGBA.

The first three are the primary colors red, green, and blue which combine additively to give the final color. The intensity of each component is a number in the range [0.0 to 1.0].

The fourth component is alpha, which determines the opacity of an object. Values are between 1.0, a completely opaque color, and 0.0, a completely transparent or colorless object.

To conserve graphics resources, alpha blending for a Gizmo window is turned off by default. To create objects that have some degree of transparency, in addition to setting the alpha component of their color, you must enable transparency blending by choosing Gizmo Menu→Enable Transparency Blend.

Here is an example showing color specification with transparency:

```
// Create a new Gizmo with axis cue and set rotation
NewGizmo
ModifyGizmo showAxisCue=1
ModifyGizmo setQuaternion={0.435,-0.227,-0.404,0.777}

// Append a red, opaque sphere
AppendToGizmo/D sphere={0.25,25,25}, name=sphere0
ModifyGizmo modifyObject=sphere0, objectType=Sphere, property={colorType,1}
ModifyGizmo modifyObject=sphere0, objectType=Sphere,
      property={color,1.0,0.0,0.0,1.0}

// Append a blue, translucent box
AppendToGizmo/D box={0.5,0.5,0.5}, name=box0
ModifyGizmo modifyobject=box0, objectType=Box, property={colorType,1}
ModifyGizmo modifyobject=box0, objectType=Box,
      property={colorValue,0,0.0000,0.2444,1.0000,0.5000}
```

You now have a translucent blue box surrounding a red sphere but you can not see the red sphere because alpha blending is disabled by default. Now we enable it:

```
// Enable transparency blend
AppendToGizmo attribute blendFunc={770,771}, name=blendFunc0
ModifyGizmo insertDisplayList=0, opName=enableBlend, operation=enable,
      data=3042
ModifyGizmo insertDisplayList=0, attribute=blendFunc0
```

The last set of commands is what is executed when you choose Gizmo Menu→Enable Transparency Blend.

Converting Igor Colors to Gizmo Colors

For historical reasons, Igor represents color components as integer values from 0 to 65535. OpenGL, and consequently Gizmo, represent color components as floating point values from 0.0 to 1.0. To convert from an Igor color component value, such as you might receive from the ColorTab2Wave operation, to a Gizmo color component value for use in a Gizmo command, you need to divide the Igor color component value by 65535.

Here is an example of such a conversion:

```
ColorTab2Wave Rainbow                              // Creates M_colors
MatrixOP/O gizmoRainbowColors = M_colors/65535  // Convert to SP and scale
Redimension/N=(-1,4) gizmoRainbowColors         // Add a column for alpha
gizmoRainbowColors[][3] = 1                        // Set the alpha to 1 (opaque)
```

Colors, Materials and Lighting

The perceived color of an object depends on the combination of the color, the material and the lighting.

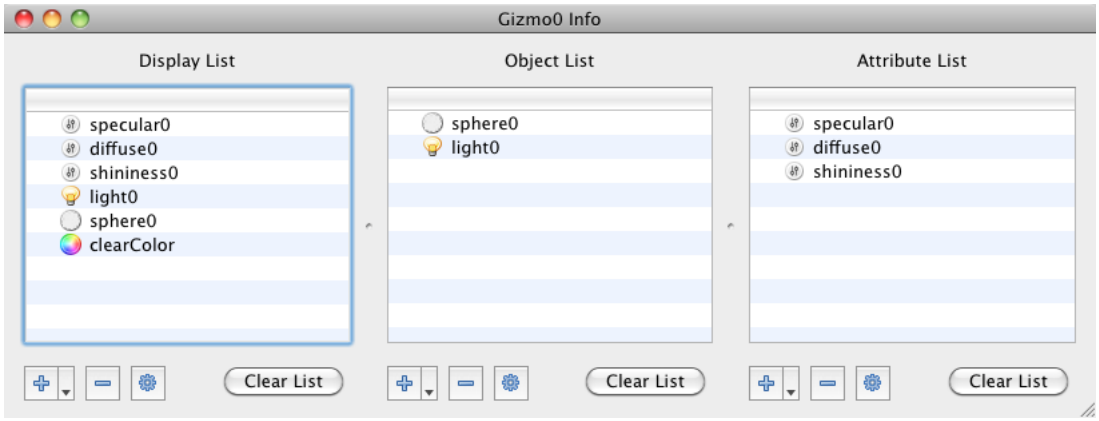
A color material specifies which faces of an object are to be colored by OpenGL and the way the object emits or responds to light. The "face" property can be set to GL_FRONT, GL_BACK or GL_FRONT_AND_BACK. The "mode" property can be set to GL_EMISSION, GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR or GL_AMBIENT_AND_DIFFUSE.

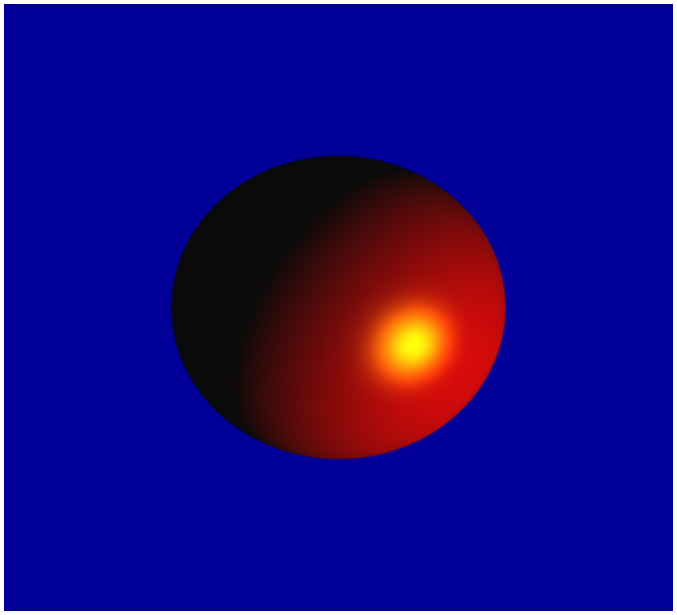
When you create a Gizmo object you have the option to specify a color or to leave it unspecified. If you specify a color, Gizmo creates a default color material for the object. The default color material has the GL_FRONT_AND_BACK and GL_AMBIENT_AND_DIFFUSE settings. If you don't specify a color then

Gizmo does not create a default color material and you must create a color material yourself. This color material affects all objects that appear later in the display list if they have no default color material.

Whatever color material is in effect for a given object, you can modify it by adding ambient, diffuse, specular, shininess or emission attributes above it in the display list.

To create a shiny object (e.g., sphere), start with a sphere object, add to it shininess and specular attributes and then add the sphere to the display list following a light object that has matching diffuse and specular components. In this case the info window and display window like this:





For an example, open this demo experiment: Material Attributes.

Color Waves

Wave-based objects can be drawn in fixed colors or using the built-in Igor color tables. You can also use your own color waves to specify the colors of the objects in the Gizmo display window. The format of a color wave is similar to that of the corresponding data wave except that each data node (vertex) has red, green, blue and alpha color components associated with it.

One situation where a color wave is useful is when you want to display a set of scalar values (e.g., temperature measurements) corresponding to points on a 3D surface. In this case you have one wave that describes the shape of the surface and another wave containing the scalar measurements. The application of a color wave gives you complete freedom to represent the scalar data distributed on the surface. In most cases you

can create an appropriate color wave using the **ModifyGizmo** makeColorWave and makeTripletColorWave keywords to create a color wave for the data based on one of the built-in tables and then specifying an appropriate alpha in the color wave.

The required color wave format depends on the format of the data wave that describes the object to which the color is to be applied. This table shows some of the various data wave formats with their corresponding color wave formats. Dimensionality is indicated in parenthesis:

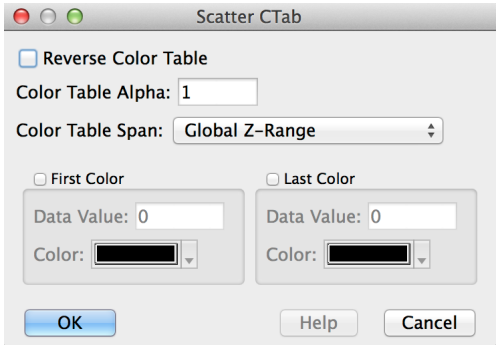
Data Wave	Color Wave
Triplet (Mx3) Used in path, ribbon and scatter plots	(Mx4) with RGBA in successive columns
Matrix of Z values (MxN) Used in surface and 3D bar chart plots	(MxNx4) with RGBA in successive layers
Sequential Quads (Mx4x3) Used in parametric surface plots	(Mx4x4) with RGBA in successive layers
Disjoint Quads (Mx12) Used in parametric surface plots	(Mx4x4) with RGBA in successive layers
Triangles (Mx3) Used in parametric surface plots	(Mx4) with RGBA in successive columns
Parametric (MxNx3) Used in parametric surface plots	(MxNx4) with RGBA in successive layers

Color Tables in Gizmo

You can use Igor's built-in color tables to specify the colors of surface, scatter, path and ribbon objects. Isosurface, voxelgram, 3D bar chart, and image objects do not support the use of color tables.

For surface, scatter, path and ribbon objects, you can select the color table in the properties dialog for a given type of object. To see a list of available color tables, see **CTabList** and for more information, see **Color Table Details** on page II-308.

When you choose a color table you can also set related options. In the properties dialog you set these options using the Details button which displays a subdialog that looks like this:



The Color Table Alpha setting applies to all colors in the color table.

The Color Table Span setting determines the numeric quantity used to select a color from the color table. For a scatter plot the most common choice is Global Z Range. By default this means that the lowest Z value displayed in the plot is mapped to the first color and the highest Z value is mapped to the last color. The color for a specific scatter element is chosen based on that element's Z value.

This mapping can be tweaked using the First Color and Last Color settings. If you enable First Color and enter a corresponding data value then that data value is mapped to the first color in the color table and any scatter element whose data value is less than the entered value is displayed using the color selected from

the color pop-up menu below. If you enable Last Color and enter a corresponding data value then that data value is mapped to the last color in the color table and any scatter element whose data value is greater than the entered value is displayed using the color selected from the color pop-up menu below.

Reflection

Gizmo supports four types of surface object interactions with lights: ambient, diffuse, specular, and shininess.

Lights have ambient, diffuse and specular components.

Materials have ambient, diffuse, specular and shininess attributes.

Ambient

Ambient reflectance determines the overall color of the object. Ambient reflectance is most noticeable in object shadows. The total ambient reflectance is determined by the global ambient light and ambient light from individual light sources. It is unaffected by the viewpoint position.

Diffuse

A diffuse surface reflection scatters light evenly in all directions. This is the most important factor determining the color of an object. It is affected by the incident diffuse light color and by the angle of the incident light relative to the normal direction. It is most intense where the incident light falls perpendicular to the surface. It is unaffected by the viewpoint position.

Specular

Specular reflection governs the appearance of highlights on an object. The amount of specular reflection depends on the location of the viewpoint, being brightest along the direct angle of reflection.

Shininess

Shininess controls the size and brightness of a specular highlight. The shinier the object, the smaller and brighter (more focused) the highlight.

Normals, Lighting and Shading

When you display a scene with lighting effects, make sure to enable the calculation of normals for all objects in the display list. You can do this in the properties dialog for each object. Depending on the type of object, check the Calculate Normals checkbox or choose from the Normals pop-up menu. These settings are off by default to conserve graphics processing resources.

Normals are required because the shading of every pixel depends on the angle between the normal to the surface and the direction of the light source. In the special case of quadric objects (sphere, cylinder and disk) there are internal settings that let you choose between flat, smooth and no normals. All objects draw much slower when normals are calculated.

Transparency and Translucency

Proper implementation of transparency in OpenGL requires that objects be drawn from the back to the front of the scene, starting with the object that is farthest from the viewer and ending with the nearest object. For any fixed viewing transformation it is possible to sort the displayed objects as long as they consist of primitive non-intersecting elements. If you are drawing compound objects such as quadrics you have no control over the order of their constituent segments. Most wave-based objects are transformed into triangle arrays which can be distance-sorted as long as there are no intersecting triangles.

Distance sorting is computationally expensive so most applications avoid it using various tricks. The "poor man's" solution is to use alpha blending. This type of translucency can provide the desired effect for a restricted range of viewing angles and may require re-ordering the objects on the display list.

To use alpha blending, assign colors to two distinct objects on the display list. The translucent object should have an alpha value that corresponds to its opacity. An opaque object has alpha=1, whereas a transparent object has alpha=0. The final steps required for translucency are the addition of the blending function attribute and the enable operation. Select Gizmo→Enable Transparency Blend menu to create the blending function and the enable operation and add both to the display list.

The isosurface is a special case because by construction it consists of non-intersecting triangles. In most applications it is sufficient to sort the triangles in the order of the distance of the viewing point from the centroid of the triangle. You can obtain the triangles corresponding to an isosurface object using **ModifyGizmo** with the saveToWave keyword and establish a sample viewing point. The standard orthographic projection implies infinite distance to viewing point. An example of this type of sorting can be found in the Depth Sorting demo experiment.

Gizmo Lights

Gizmo supports both directional and positional light sources. The type and color of the lights that you add to the display affect the appearance of objects in the display window.

You create lights like any other object by selecting Light from the object list pop-up menu. Using the Light Properties dialog, shown below, you can specify the light type and various light parameters. For the light to have any effect, you must add it to the display list above any object that you want to illuminate.

Lighting effects are defined in terms of their ambient, diffuse, and specular components. The distribution of light intensity is described by the location of the light source, direction, cone angle, and attenuation. The final appearance of an object depends on the combination of the properties of the light and the properties of the object material.

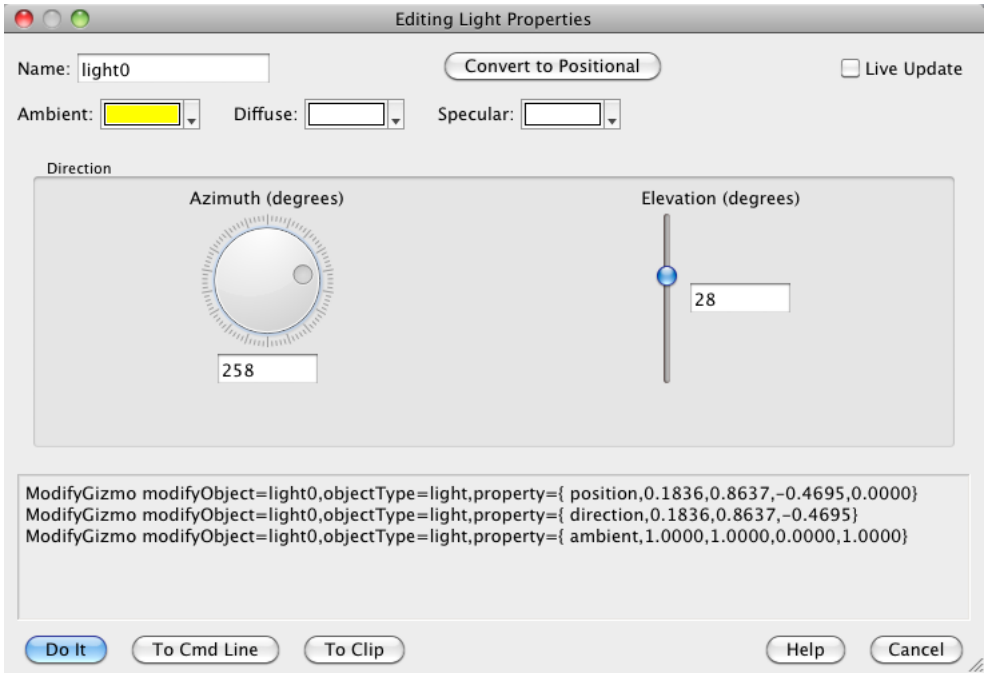
Lighting effects are computed in hardware on a per pixel basis. Therefore, when you want smooth shading, you must describe the object using a sufficiently large number of vertices. For simple objects, such as a single quad (4 vertices), you will likely not see much variation in lighting across the quad. Shading is computed using the dot product between the normal to the surface at each vertex and the direction of the light source. There is no accounting for objects obscuring other objects from the light source or for multiple reflections of light.

If you add no lights to the display list, Gizmo uses default, color-neutral ambient light. If you add a light to the display list, Gizmo removes the default lighting.

Gizmo Directional Lights

You can think of a directional light as a light positioned very far away from the scene so that its rays are essentially parallel within the display volume. The sun is a good example of a directional light. New light objects are directional by default.

The Light Properties dialog contains the controls you need to specify the light's position, color properties and distribution. When editing a light object that is already in the display list, you can click the Live Update checkbox to see how your changes affect the Gizmo Display.



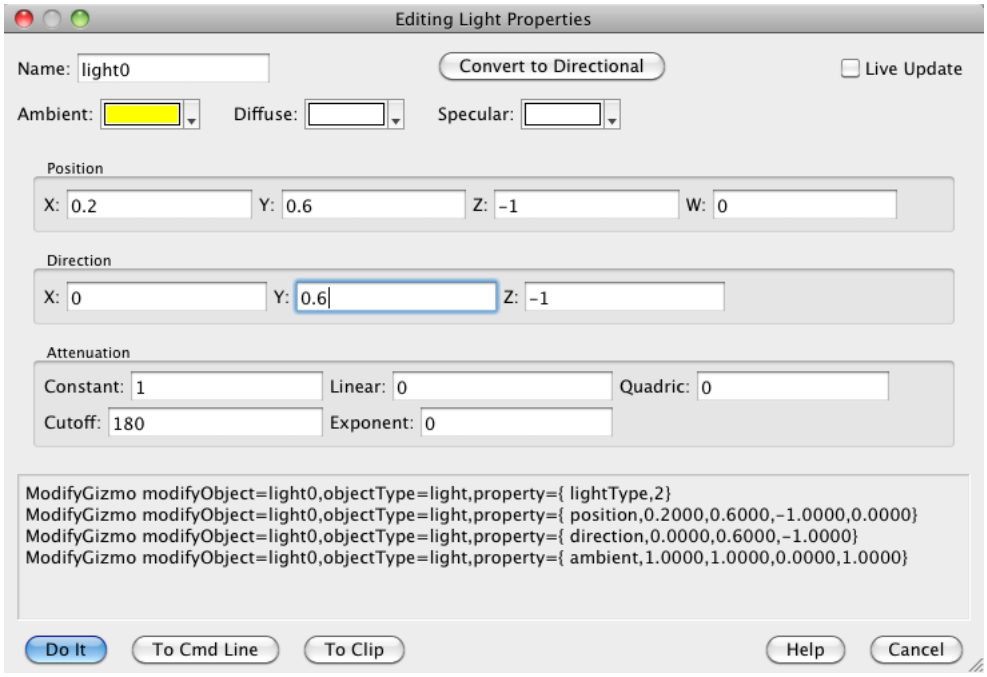
The position of the light is specified via two angles: azimuth and elevation. Elevation is also called "altitude", especially in astronomy. The meaning of these angles is described at http://en.wikipedia.org/wiki/Azimuth. When the elevation is +90 or -90 degrees, the azimuth is undefined.

Ambient, diffuse and specular lighting are described at http://en.wikipedia.org/wiki/Phong_reflection_model. Ambient light illuminates all parts of all objects equally regardless of their orientation and of the position of the light. Diffuse light is reflected off a surface in all directions, as when light hits a rough surface. Specular light is reflected in a specific direction, as when light hits a shiny surface. The illumination created by diffuse and specular light at a given point on an object depends on the angle of the light ray relative to the normal to the surface of that point.

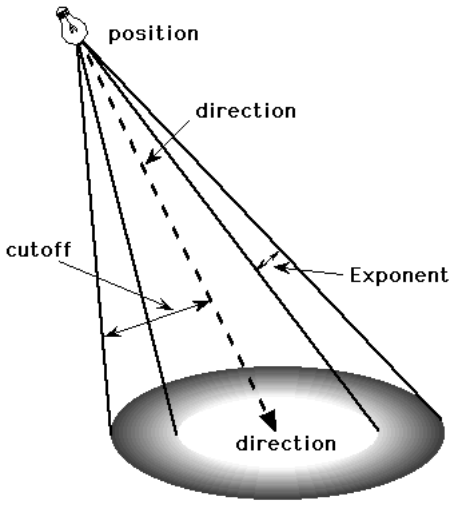
Gizmo Positional Lights

A positional light is a light source that originates at a finite distance from the illuminated scene. A desk lamp is a typical example of a positional light. It can be placed somewhere above the desk and it produces non-uniform illumination as its intensity falls off as a function of distance from the center of the illumination spot.

When you create a light object, it is initialized as a directional light. You can click click the Convert to Positional button in the dialog to get the corresponding positional light settings which you can then adjust as desired.



The parameters for specifying positional lights are illustrated here:



Positional Light Geometry

The top three controls in the directional light dialog specify the RGBA values of each of the ambient, diffuse and specular light components. You should provide some ambient component in at least one of the lights in the display list. This requirement holds even if you are trying to create a predominantly diffuse or specular effect.

When you create a Gizmo object you have the option to specify a color or to leave it unspecified. If you specify a color, Gizmo creates a default color material for the object. The default color material has the GL_FRONT_AND_BACK and GL_AMBIENT_AND_DIFFUSE settings. If you are interested in specular effects you must add specular and shininess attributes (see **Gizmo Colors, Material and Lights** on page II-341).

The Position and Direction controls in the dialog describe the position and direction of the light source. The position is expressed in homogeneous coordinates where the last element (w) is used to normalize the X, Y

and Z components. Therefore, if you set w=1, then the X, Y, and Z components specify the absolute position of the light in space. If you set w=0, your light source is infinitely far away and you effectively created a directional light source.

We suggest that you set w=1, and set the direction using the next group of controls in the dialog, which describes the direction of the light as the three components of a vector pointing from the position of the light source to the point that you want the center of the specular spot to illuminate. The typical error here is entering the position of the illuminated spot instead of the direction vector.

The specified position of the light source is subject to the same transformations that apply to any other objects in the display list. In particular, as you rotate display objects, the lights will likewise rotate. If you want to keep the lights stationary so they illuminate different part of the rotating display, you must add a main transformation operation to the display list immediately after the last light object and before all the rotating objects. This will keep all objects listed above the main transformation stationary, and apply the rotation only to all subsequent display list items.

Using the main transformation operation allows you to have some things fixed and other things rotatable. The main transformation operation sets the point in the display list after which coordinate transformations are applied. Coordinate transformations affect translation and scaling in addition to rotation. No transformations are applied to everything above the main transformation in the display list and therefore those items are drawn in their default view, unless you insert explicit translate, rotate or scale operations.

By default, the spot cutoff angle (the light cone half-angle) is 180 degrees which means that the light provides uniform illumination in all directions. If you take the trouble to specify position and direction it will be useful to reduce the cutoff angle to something more realistic - less than 90-degrees.

The constant, linear and quadratic attenuations combine to attenuate positional lights (i.e., for which w is non-zero). The exponent value, in the range 0 to128, determines the intensity falloff from the center of the spot by multiplying the center intensity by the cosine of the angle between the direction of the light and the vertex in question, raised to the power of the exponent value. This can be used to make the light highly specular.

The Directional Light Demo experiment contains a control panel that you can use to explore the interplay between the various positional light parameters and how they affect the lighting on an object.

Gizmo Drawing Objects

Gizmo provides access to a number of drawing primitives. These include **Line Objects**, **Triangle Objects**, **Quad Objects**, **Box Objects**, **Sphere Objects**, **Cylinder Objects**, **Disk Objects**, **Tetrahedron Objects** and **Pie Wedge Objects**. You can use the dialogs associated with the different objects to set the various object properties.

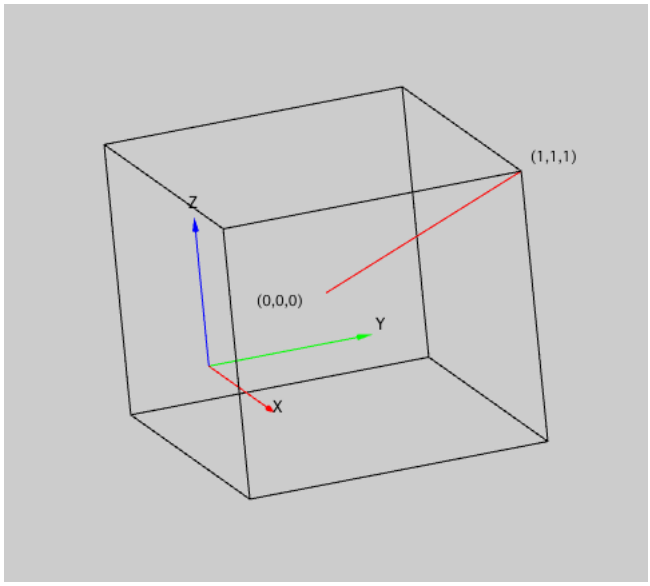
As explained under **Gizmo Dimensions** on page II-334, the size of a drawing object is expressed in display volume units. The display volume extends from the origin to +/- 1 in each dimension. A box of size 1 centered at the origin extends halfway from the origin the edge of the display volume in each dimension.

Unlike drawing objects, wave-based objects such as scatter plots and surface plots are displayed against a separate coordinate system. If you combine drawing objects with wave-based objects, remember that drawing object positions and sizes are always expressed in terms of the +/-1 display volume. If you draw a 2D wave as a surface and you would like to draw a box around it, just add a box whose length, width and height are two units.

Line Objects

A line object is a straight line that connects the two endpoint coordinates. You can add an arrowhead at the start or end of the line or at the mid point. This example shows a line from (1,1,1) to (0,0,0) created by the command

```
AppendToGizmo/D line={1,1,1,0,0,0}
```



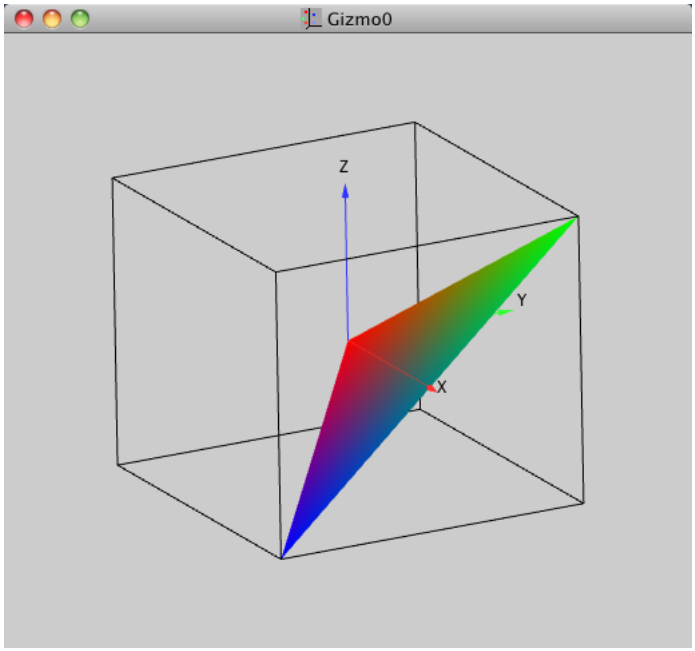
Triangle Objects

A triangle is a planar object bounded by a simple polygon connecting its three vertices. A triangle object is always drawn filled. The fill color is determined by the internal color attribute, an embedded color attribute, or a global color attribute, in that order of precedence. All lighting attributes also apply.

In some situations, it may be more straightforward to create the triangle in a simple orientation and then use translate and rotate operations to position the triangle in its final orientation.

You can set the color of the triangle on a vertex by vertex basis and OpenGL will interpolate the colors between the vertices. Here is an example:

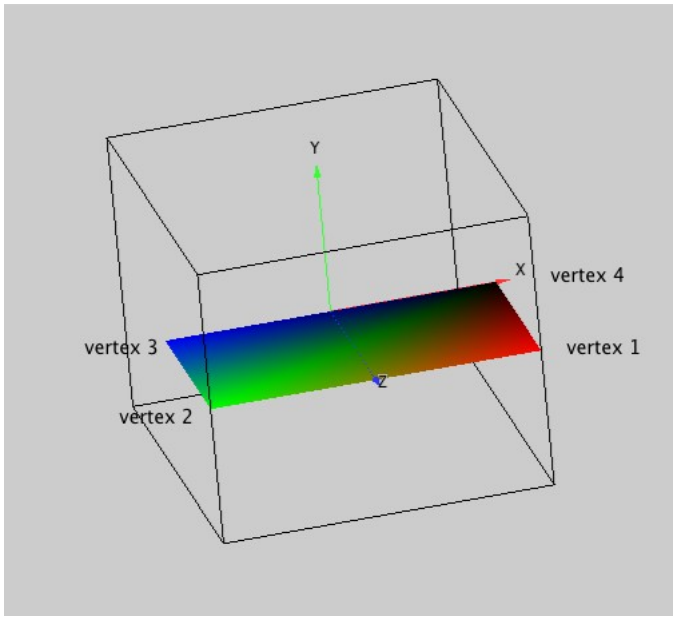
```
AppendToGizmo/D triangle={0,0,0,1,1,1,1,-1,-1}
ModifyGizmo modifyobject=triangle0, objectType=triangle,
      property={colorType,2}
ModifyGizmo modifyobject=triangle0, objectType=triangle,
      property={colorValue,0,1,0,0,1}
ModifyGizmo modifyobject=triangle0, objectType=triangle,
      property={colorValue,1,0,1,0,1}
ModifyGizmo modifyobject=triangle0, objectType=triangle,
      property={colorValue,2,0,0,1,1}
```



Quad Objects

A quad object fills the sheet connecting the four vertices that are positioned sequentially in any direction starting from the first vertex. A quad object is always drawn filled. The fill color is determined by the internal color attribute, an embedded color attribute, or a global color attribute, in that order of precedence. Normals to the quad are calculated on a per-vertex basis. This gives rise to gradual shading when lighting calculations are enabled. This example shows a quad created by the commands:

```
AppendToGizmo/D quad={1,0,1,-1,0,1,-1,0,0,1,0,0}
ModifyGizmo modifyObject=quad0, objectType=quad, property={colorType,2}
ModifyGizmo modifyObject=quad0, objectType=quad,
      property={colorValue,0,1,0,0,1}
ModifyGizmo modifyObject=quad0, objectType=quad,
      property={colorValue,1,1.5259e-05,0.6,0.30425,1}
ModifyGizmo modifyObject=quad0, objectType=quad,
      property={colorValue,2,1.5259e-05,0.244434,1,1}
ModifyGizmo modifyObject=quad0, objectType=quad,
      property={colorValue,3,0,0,0,1}
```

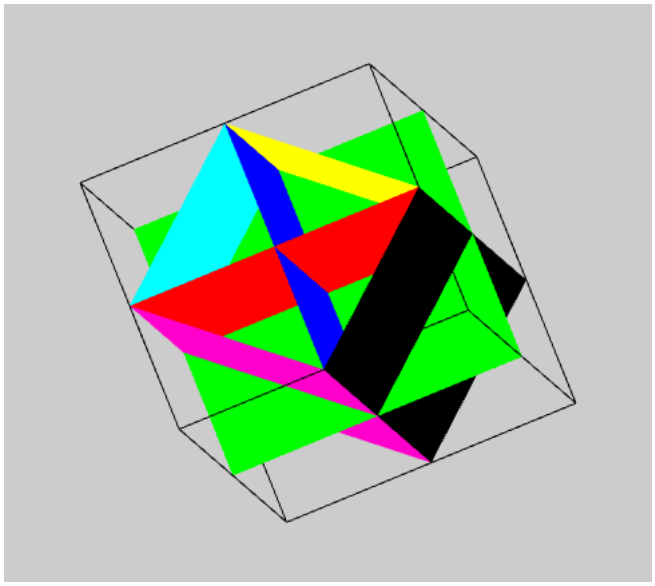


When creating a quad, you may find it easier to create the quad using simple coordinates in either the X, Y, or Z planes and then use translate and rotate operations to position the quad in the desired final orientation.

This table lists some basic quad examples with unit dimensions.

Quad Orientation	Command
XZ Plane	quad={1,0,1,-1,0,1,-1,0,-1,1,0,-1}
XY Plane	quad={1,1,0,-1,1,0,-1,-1,0,1,-1,0}
YZ Plane	quad={0,1,1,0,1,-1,0,-1,-1,0,-1,1}
Oblique Z Plane, +X, +Y	quad={1,0,1,1,0,-1,0,1,-1,0,1,1}
Oblique Z Plane, -X, +Y	quad={-1,0,1,-1,0,-1,0,1,-1,0,1,1}
Oblique Z Plane, -X, -Y	quad={-1,0,1,-1,0,-1,0,-1,-1,0,-1,1}
Oblique Z Plane, +X, -Y	quad={1,0,1,1,0,-1,0,-1,-1,0,-1,1}

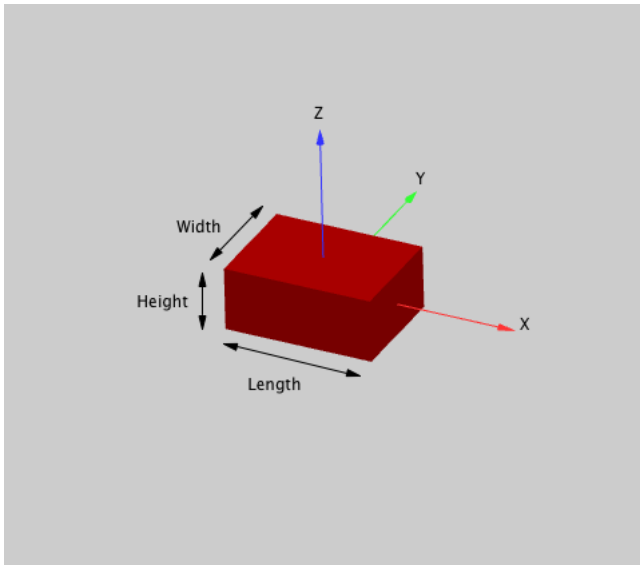
Here is what the seven quads described in the table look like after applying different colors:



Box Objects

A box object is defined by length along the X direction, width along the Y direction, and height along the Z direction. Boxes are centered on the origin and are always drawn filled. The fill color is determined by the internal color attribute, an embedded color attribute, or a global color attribute, in that order of precedence.

This illustration shows a box with length = 1, width = 0.75, height = 0.5:



To position a box in orientations where the sides are not parallel to the axes or not centered on the origin, add rotate and/or translate operations to the display list before the box item.

Sphere Objects

A sphere is a quadric object, meaning that it can be generated by a quadratic polynomial. Internally a quadric object is composed of a list of triangles and quads that approximate the curved surface.

A newly-created sphere is centered on the origin and both poles are on the Z axis. The numbers of slices (subdivisions around the Z axis; divisions of longitude) and stacks (subdivisions along the Z axis; divisions of latitude) determine the smoothness of the sphere. The greater the numbers of subdivisions, the smoother the sphere's appearance. Small values produce other geometric objects (see illustrations below). The more

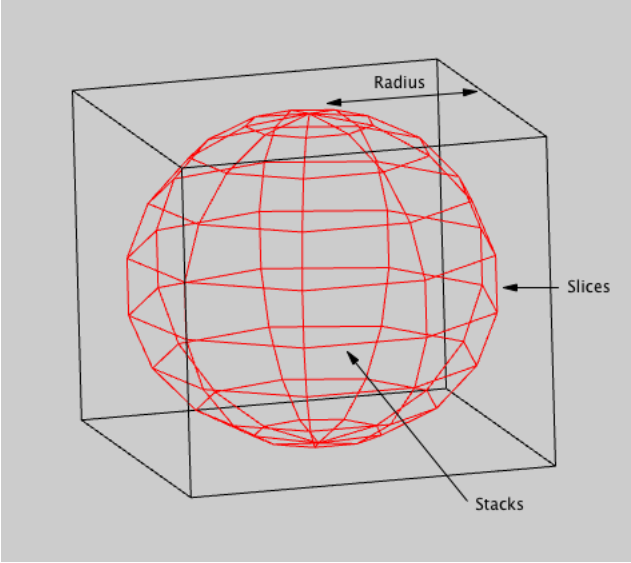
subdivisions you use, the more time it takes to draw the sphere, but in most applications this is not important unless the sphere is replicated many times, such as when used as a marker in a scatter plot.

By default, spheres are initially drawn filled. The fill color is determined by the internal color attribute, an embedded color attribute, or a global color attribute, in that order of precedence.

Use translate and rotate operations to position the sphere in other locations and orientations.

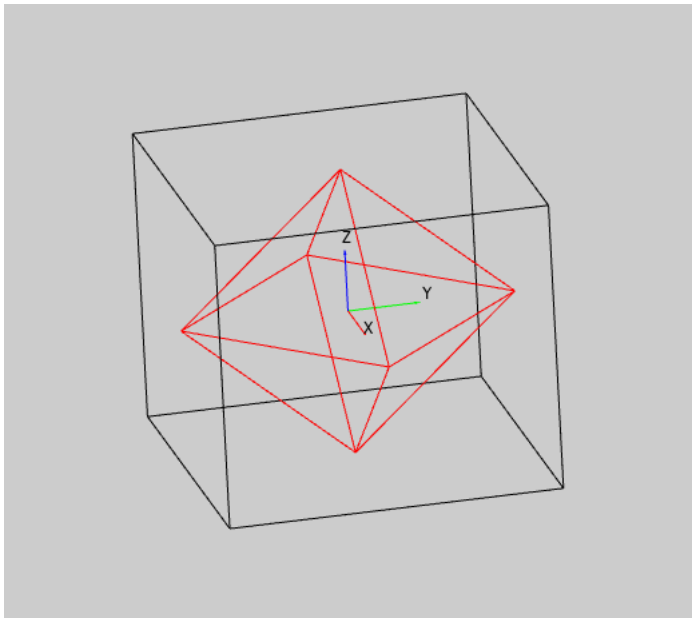
These commands generate a sphere with radius=1, stacks=10, and slices=10 and set its drawing style to lines:

```
AppendToGizmo/D sphere={1,10,10}
ModifyGizmo modifyObject=sphere0, objectType=Sphere,
      property={useGlobalAttributes,0}
ModifyGizmo modifyObject=sphere0, objectType=Sphere,
      property={drawStyle,100011}
```



Create other geometric shapes by using small values for the number of stacks and keeping slices = 2. These commands generate an octahedron with radius=1, stacks=4 and slices=2 and set its drawing style to lines:

```
AppendToGizmo/D sphere={1,4,2}
ModifyGizmo modifyObject=sphere0, objectType=Sphere,
      property={useGlobalAttributes,0}
ModifyGizmo modifyObject=sphere0, objectType=Sphere,
      property={drawStyle,100011}
```



You can create other geometric shapes by increasing the number of stacks and keeping slices=2.

Cylinder Objects

A cylinder is a quadric object, meaning that it can be generated by a quadratic polynomial. Cylinders are constructed from slices and rings. Specifying more slices creates a smoother cylinder. Initially, the cylinder axis is centered on the Z axis, height is in the positive Z direction, and the cylinder base is in the XY-plane.

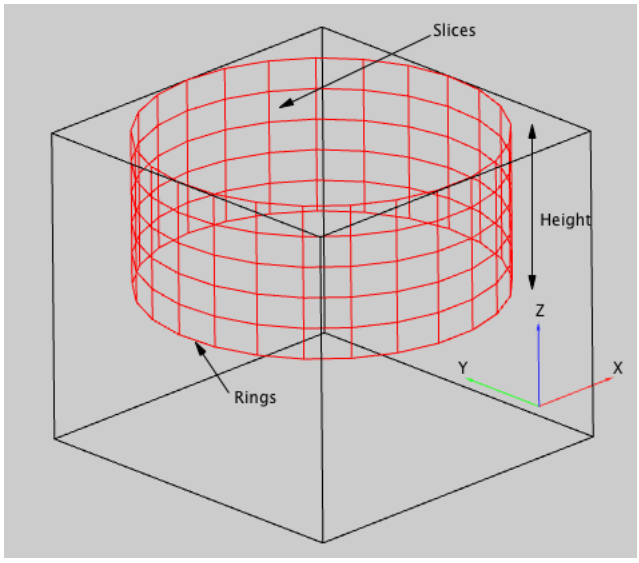
Create conical objects (see illustration below) by specifying different values for the base radius and top radius parameters.

By default, cylinders are initially drawn filled. The fill color is determined by the internal color attribute, an embedded color attribute, or a global color attribute, in that order of precedence.

Use translate and rotate operations to position the cylinder in other locations and orientations.

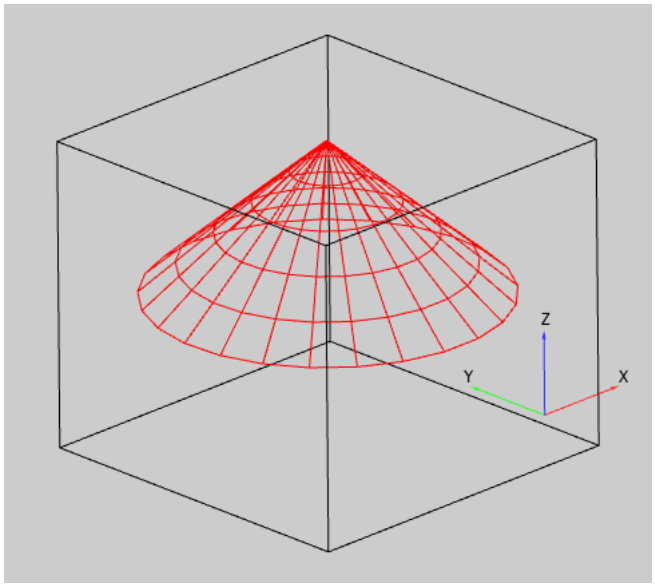
These commands generate a cylinder with baseRadius=1, topRadius=1, height=1, slices=25, and rings=5 and set its drawing style to lines:

```
AppendToGizmo/D cylinder = {1,1,1,25,5}
ModifyGizmo modifyObject=cylinder0, objectType=Cylinder,
      property={useGlobalAttributes,0}
ModifyGizmo modifyObject=cylinder0, objectType=Cylinder,
      property={drawStyle,100011}
```



A cone created using the same command but with topRadius=0:

```
AppendToGizmo/D cylinder = {1,0,1,25,5}, name=cone0
ModifyGizmo modifyObject=cone0, objectType=Cylinder,
      property={useGlobalAttributes,0}
ModifyGizmo modifyObject=cone0, objectType=Cylinder,
      property={drawStyle,100011}
```



You can create other types of cylindrical object shapes by specifying a small number for slices. Create a triangular cylinder with slices = 3; create a square cylinder or open-ended box with slices = 4. Use different values for baseRadius or topRadius to create pyramid shapes.

Disk Objects

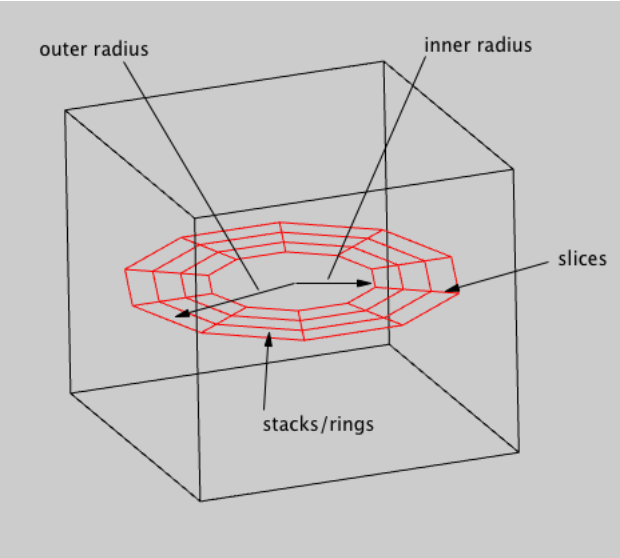
A disk is a quadric object, meaning that it can be generated by a quadratic polynomial. The disk is initially located in the XY plane centered on the origin. Create a smoother disk by specifying a greater number for slices.

By default, disks are initially drawn filled. The fill color is determined by the internal color attribute, an embedded color attribute, or a global color attribute, in that order of precedence.

Use translate and rotate operations to position the disk in other locations and orientations.

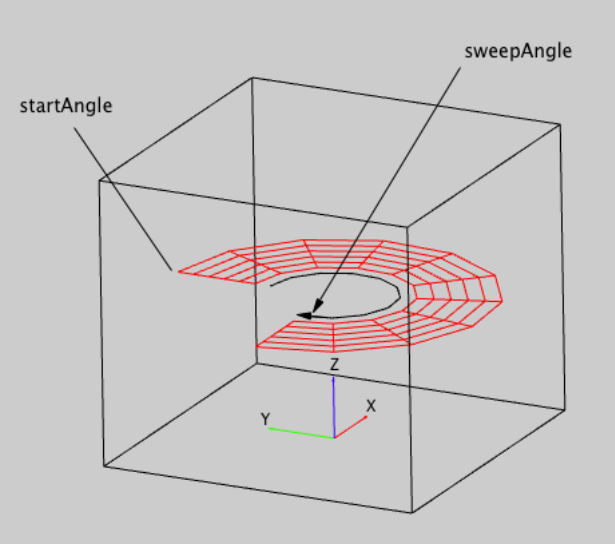
These commands generate a disk with innerRadius=0.5, outerRadius=1, slices=10, stacks=5, startAngle=0, sweepAngle=360 and set its drawing style to lines:

```
AppendToGizmo/D disk={0.5,1,10,5,0,360}
ModifyGizmo modifyObject=disk0, objectType=Disk,
      property={useGlobalAttributes,0}
ModifyGizmo modifyObject=disk0, objectType=Disk, property={drawStyle,100011}
```



This command changes the disk to a partial disk with sweepAngle set to 270 instead of 360. SweepAngle is specified in degrees clockwise from startAngle:

```
ModifyGizmo modifyObject=disk0, objectType=Disk, property={sweepAngle,270}
```



String Objects

String objects were used in Igor Pro 6 and before to create 3D text graphics. As of Igor Pro 7 you can use standard Igor annotations as you do in graphs.

Annotations are 2D text graphics that lie flat in a plane in front of all 3D graphics. They are well suited for general labeling purposes. To create an annotation choose Gizmo→Add Annotation and choose TextBox from the Annotation pop-up menu. For backward compatibility, Gizmo still supports string objects and they are useful if you want 3D graphics. Annotations are preferrable for general labeling.

For further discussion of annotations versus string objects, see **Changes to Gizmo Text** on page II-383. The rest of this section describes the original Gizmo string object feature.

String objects can be used for short text labels or annotations. You can create a string object using a command such as:

```
AppendToGizmo string="Hello", strFont="Geneva", name=string0
```

Like the primitive objects described above, string objects are 3D objects that are drawn in +/-1 display volume coordinates. You can use translate and rotate operations in order to place the string at the appropriate part of the graph and in the desired orientation.

The key to orienting strings correctly is understanding that they originate at the origin, character advance is in the X direction and text height is in the Y direction.

Multiple lines are not supported so a given string object results in a single line of text only.

ColorScale Objects

Colorscale objects were used in Igor Pro 6 and before to create 3D color scale graphics. As of Igor Pro 7 you can use standard Igor annotations as you do in graphs.

Annotations are 2D graphics that lie flat in a plane in front of all 3D graphics. They are well suited for general labeling purposes. To create a color scale as an annotation choose Gizmo→Add Annotation and choose ColorScale from the Annotation pop-up menu. For backward compatibility, Gizmo still supports colorscale objects and they are useful if you want 3D graphics. Annotations are preferrable for general labeling.

The rest of this section describes the original Gizmo ColorScale object feature.

Color scale objects are designed to provide an association between a sequence of colors and a numeric scale. You can create a color scale object using a command such as:

```
AppendToGizmo colorScale=colorScale0
```

You would typically following this with ModifyGizmo commands.

Color scales are also drawn in +/-1 display volume coordinates and by default appear planar though you can assign to them a positive depth value to make the color scale into a full 3D object.

You can choose the sequence of colors to be based on a built-in color table or provide your own sequence using a color wave.

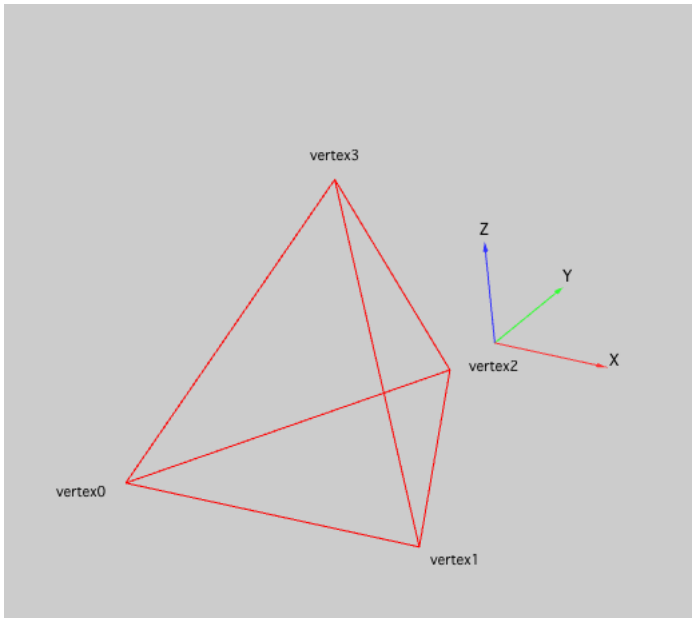
A color scale can be tied to a wave-based data object such as a surface plot but can also be independent on all objects in the plot. You can create multiple color scale objects in the same plot.

Tetrahedron Objects

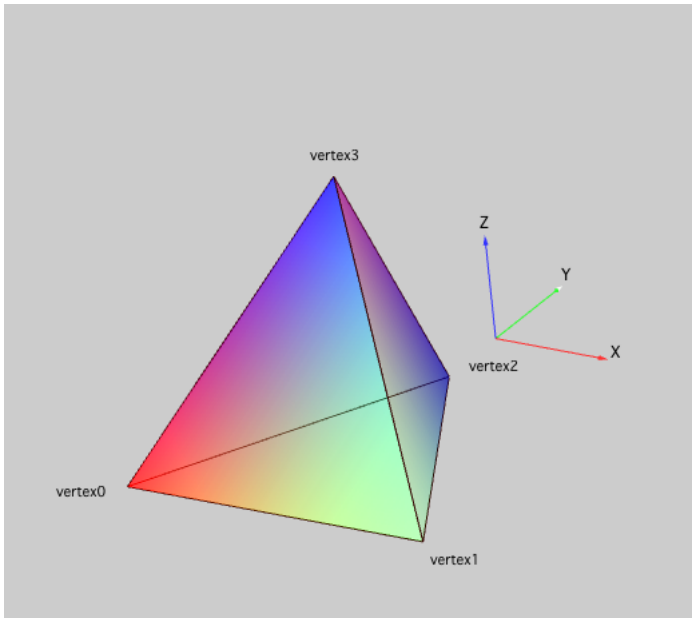
A tetrahedron is defined by 4 vertices. These commands generate a tetrahedron and set its drawing style to lines:

```
AppendToGizmo/D tetrahedron=tetrahedron0
ModifyGizmo ModifyObject=tetrahedron0, objectType=tetrahedron,
      property={vertex0,-1,-1,-1}
ModifyGizmo ModifyObject=tetrahedron0, objectType=tetrahedron,
      property={vertex1,1,-1,-1}
ModifyGizmo ModifyObject=tetrahedron0, objectType=tetrahedron,
```

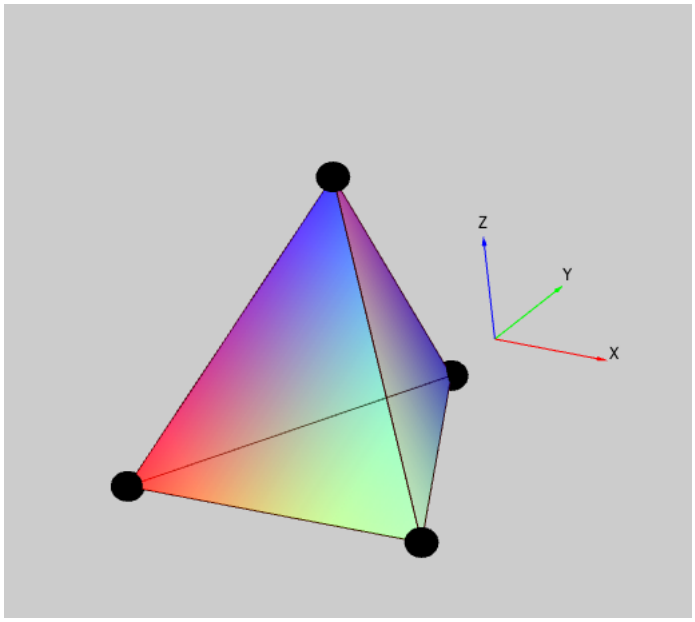
```
      property={vertex2,0,1,-1}
ModifyGizmo ModifyObject=tetrahedron0, objectType=tetrahedron,
      property={vertex3,0,0,1}
```



You can turn on filling and specify a color for each vertex to produce a tetrahedron like this:



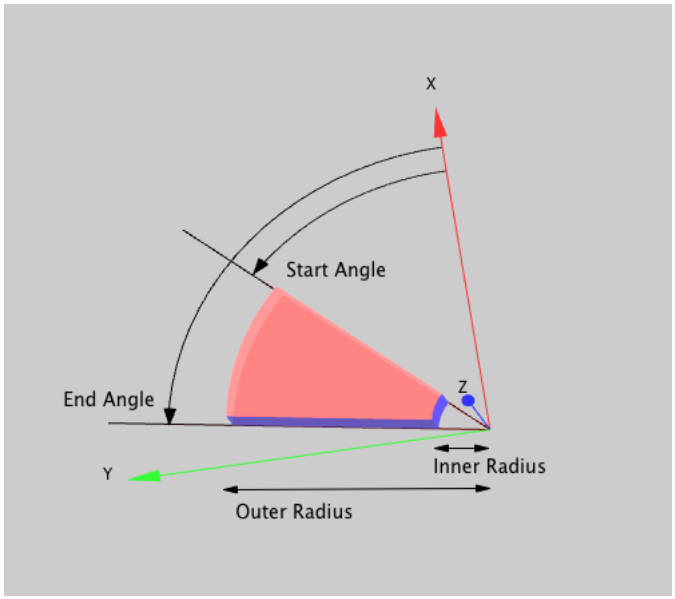
You can also specify that a sphere is to be drawn at each tetrahedron vertex:



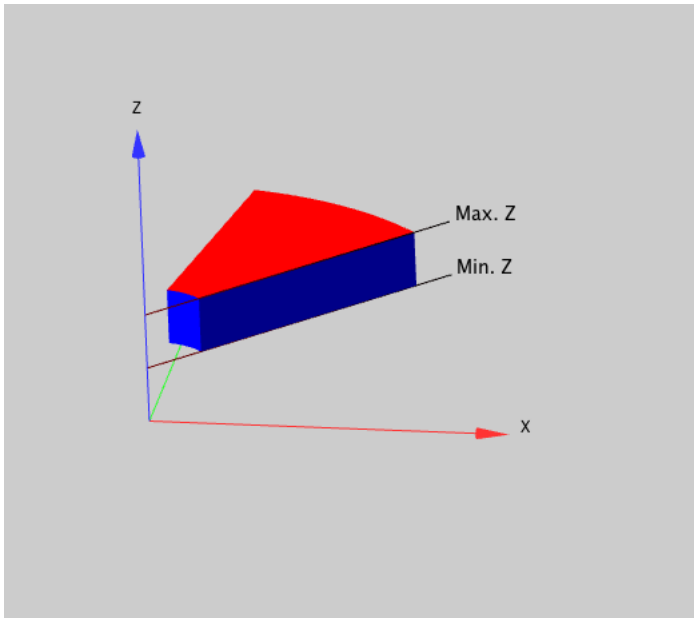
Pie Wedge Objects

A pie wedge object is defined by two angles, two radii and two Z values. You can use multiple pie wedge objects to create many types of 3D pie charts.

This illustration shows a pie wedge from the top:



This illustration shows a pie wedge from the side:



Gizmo Axis and Axis Cue Objects

Gizmo axis objects are used mostly with wave-based data objects such as surface plots and scatter plots. They allow you to indicate the range of data values and to set the scale of data objects.

Axis cue objects show the orientation of the X, Y and Z directions. Axis objects show both orientation and numeric range.

Axis Cue Objects

Gizmo supports two axis cue objects: the default axis cue and the free axis cue. These objects indicate the orientation of the X, Y and Z axes.

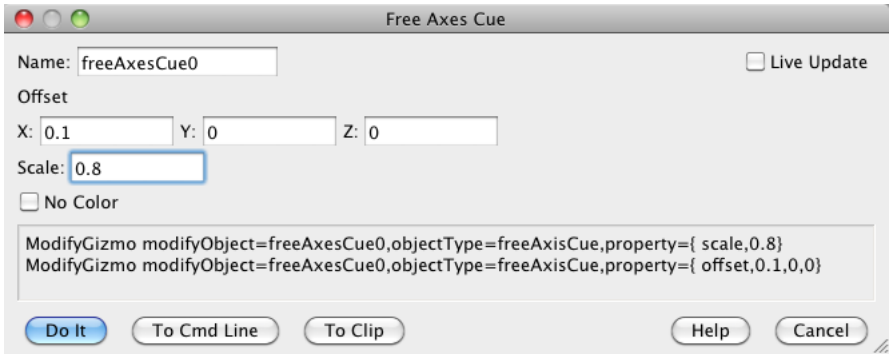
The default axis cue is a triplet axis object that is centered at the display volume origin. The X, Y and Z cue lines extend in the positive X, Y and Z directions of the display volume respectively.

There is just one default axis cue. It is used only to provide an indication of which direction is which. It does not support tick marks, tick mark labels or axis labels. To display the default axis cue, right-click the Gizmo display window and select Show Axis Cue.

The default axis cue object is drawn with the X axis in red, the Y axis in green and the Z axis in blue. The characters labeling the three axes are all drawn in black. Internally, the axes are drawn with emission color material which helps differentiate the axis cue from the background.

A free axis cue object can be drawn at any offset from the origin and at any scale. By default it has the same orientation as the default axis cue. Use a rotate operation if you want to rotate it.

You can create multiple free axis cues and even use them as markers in scatter plots. To add a free axis cue, click the + icon in the object list of the Gizmo info window and choose Free Axis Cue.



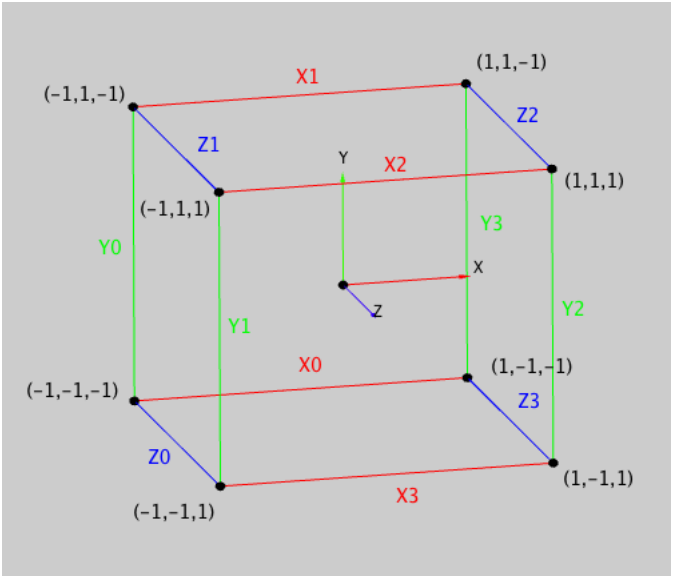
The free axis cue is colored the same as the default axis cue unless you disable its color by checking the No Color checkbox in the Free Axis Cue dialog. In this case it takes on the current color of the drawing environment as set by the previous color attribute in the display list. You also need a color material operation in the display list to apply a color to a free axis.

Axis Objects

As explained under **Gizmo Dimensions** on page II-334, a Gizmo display has an axis coordinate system that is superimposed on the +/-1 display volume. You set the range of this axis coordinate system using the Axis Range dialog, accessible via the Gizmo menu. Wave-based objects, such as scatter plots and surface plots, are displayed against this axis coordinate system. An axis object is a visual representation of it.

Gizmo supports three types of axis objects: box, triplet, and custom. Gizmo treats axis objects just like other objects that you add to the Gizmo display window with one exception: you must have at least one data object or drawing object on the display list for an axis object to be meaningful.

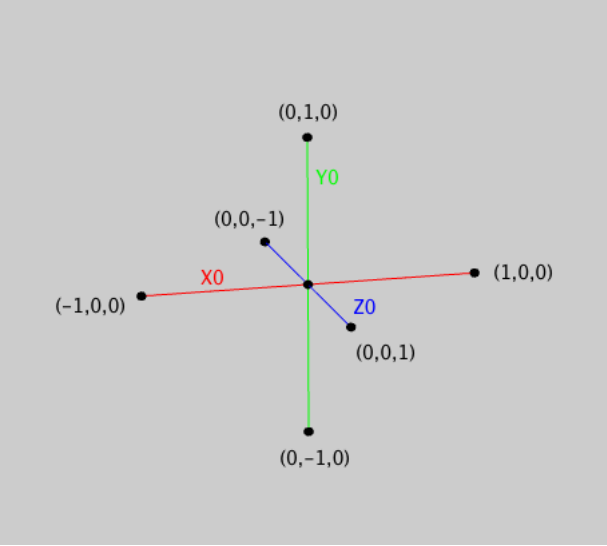
The corners of the box axis object correspond to the corners of the display volume. Thus the location of each corner in display volume space is -1 or +1 in each dimension. In this diagram the coordinates shown are the display volume coordinates of the corners of the box axes.



Box axes consist of 12 axes named X0, X1, X2, X3, Y0, Y1, Y2, Y3, Z0, Z1, Z2 and Z3. You can control each axis individually, assign it a color, range, tick marks, and tick mark labels. You can also add grid lines or paint any of the six sides of the box.

Triplet axes are a system of three orthogonal axes that intersect at the origin and span the +/-1 display volume. You can control each axis individually, assign its tick marks, tick mark labels and color. The origin of the axes is the center of the -/+1 display cube.

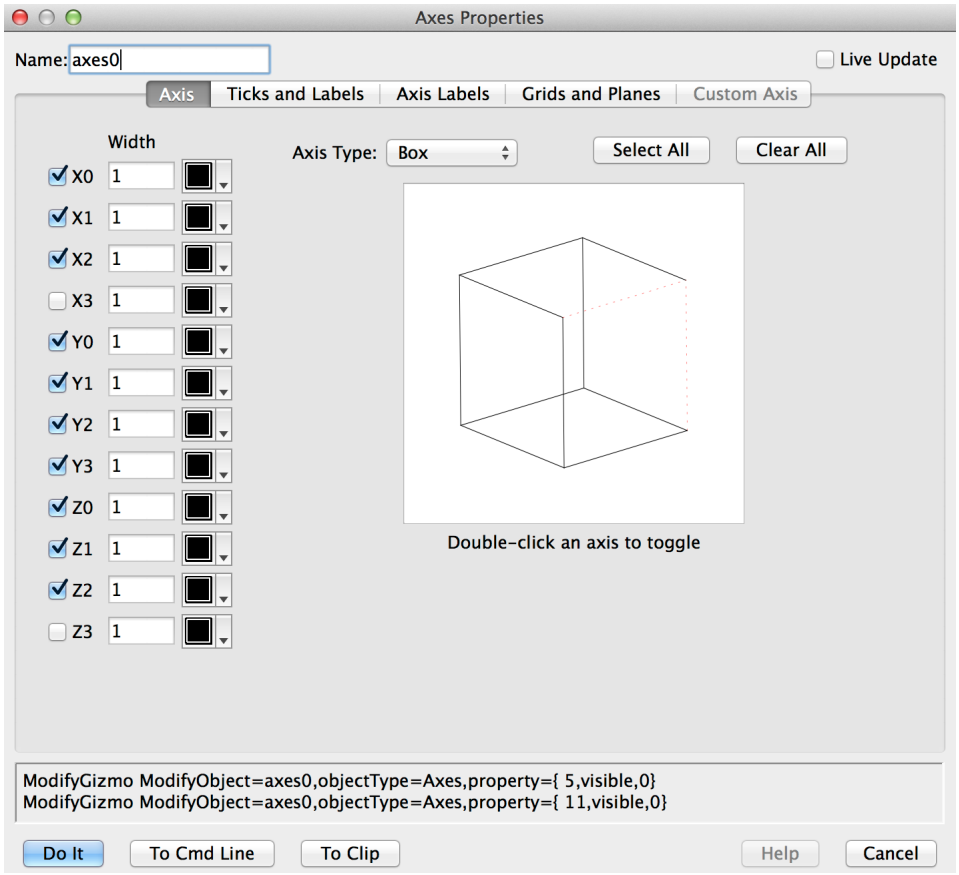
Each triplet axis is identified by name, X0, Y0, and Z0. This diagram shows display volume coordinates for triplet axes and the name of each axis:



A custom axis is a single line in space connecting your chosen start and end points as specified in -1/+1 display volume coordinates. Unlike box and triplet axes which reflect the global axis range as set by Gizmo→Axis Range, a custom axis has its own, independent axis range. You can add tick marks and tick mark labels to the custom axis as to any other axis.

By default, Gizmo attempts to find an appropriate position for tick mark labels. Using the manual position mode, you can take control over their position and orientation.

In most cases you will control the various axis settings using the Axes Properties dialog. When using the dialog it is important to keep track of the different axes that you can select with the checkboxes. To help you identify the various axes in your plot, the Axis tab of the dialog displays an outline of the axes in the same orientation as they appear in the Gizmo window. You can toggle the display of axes by double-clicking them.



See **AppendToGizmo** and **ModifyGizmo** for programming details.

Gizmo Wave Data Formats

Objects like surface plots, path plots, isosurfaces and voxelgrams are based on the data in a wave and are therefore called "wave-based data objects" or "data objects" for short. The wave supplying the data is called the "data wave" or the "source wave".

The following sections describe the data format requirements for the different data objects.

Scatter, Path, and Ribbon Data Formats

Scatter and path plots require a triplet wave, which is a 2D wave containing 3 columns for the X, Y, and Z coordinates of each vertex. A color wave for a scatter or path plot is a 2D wave in which each row specifies the color of the corresponding vertex in the data wave. The color wave has 4 columns which specify RGBA entries in the range of [0,1].

A ribbon plot also requires a triplet wave. A color wave for ribbon plot is the same as for scatter or path plots with one row of RGBA values per vertex. See also **ModifyGizmo** with the keyword pathToRibbon and **Ribbon Plots** on page II-373.

Surface Object Data Formats

The data wave format depends on the type of surface plot.

A simple surface plot is created from a 2D wave also known as a "matrix" or "matrix of Z values". The color wave for this type of surface plot is a 3D RGBA wave where the four layers in the wave contain the R, G, B and A components in the range of [0,1].

If you are plotting a parametric surface then the data wave is 3D wave containing three layers. At each row/column position, layer 0 contains the X coordinate, layer 1 the Y coordinate, and layer 2 the Z coordinate. The color wave for a parametric surface is a 3D RGBA wave that has the same number of elements in the X and Y dimensions as the data wave and where layer 0 contains the red component, layer 1 the green component, layer 2 the blue component and layer 3 the alpha component.

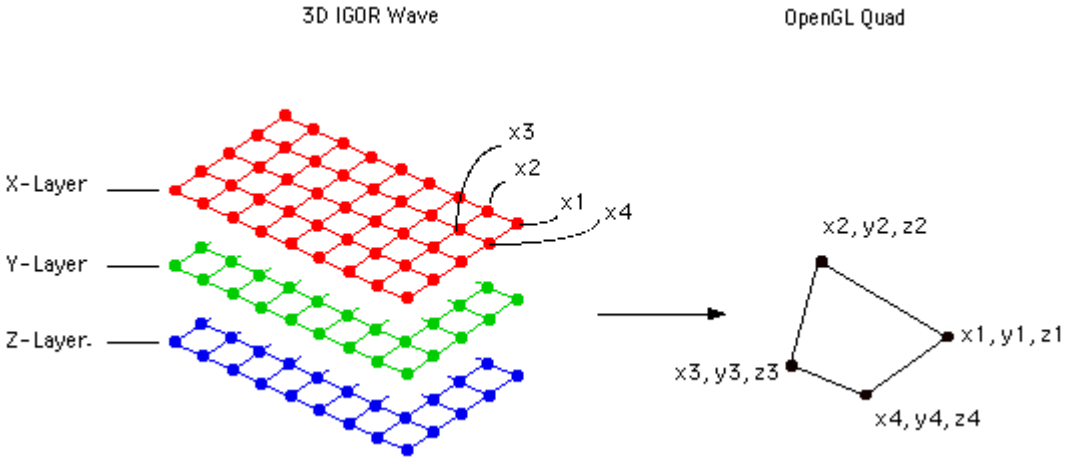
If you are plotting sequential quads (they could be the **ImageTransform** output of a Catmull-Clark B-Spline), the data wave is a 3D wave with four rows and three layers. Each row contains stores the four vertices of a quad and the three layers contain the X, Y, and Z coordinates.

If you are plotting disjoint quads the data wave is 2D with 12 columns corresponding to the X, Y, and Z values of the quad vertices taken in a counterclockwise direction.

The color wave for sequential quads and disjoint quads is a 4 column 2D wave in which the columns correspond to RGBA values in the range [0,1].

Parametric Surface Data Formats

A parametric surface is defined by equations in which the X, Y, and Z coordinates are calculated for, usually, a pair of parameters. To define a parametric surface for Gizmo you need to have a 3D wave containing X, Y, and Z layers. The only restriction on the dimensions of this wave is that it must not have 4 columns. Gizmo constructs the parametric surface from quads. Each quad is defined by four neighboring vertices as shown in the diagram below.



You can find examples of parametric surfaces in these demo experiments:

```
Igor Pro 7 Folder:Examples:Visualization:Mobius Demo
Igor Pro 7 Folder:Examples:Visualization:Spherical Harmonics Demo
Igor Pro 7 Folder:Examples:Visualization:Gizmo Sphere Demo
```

Image Object Data Format

The data wave for an image object can be in one of three formats:

- A 2D matrix of Z values
- A 3D RGB wave of type unsigned byte with 3 layers
- A 3D RGBA wave of type unsigned byte with 4 layers

In the 3D formats, the red component is stored in layer 0, the green component in layer 1, and the blue component in layer 2. The alpha component, if any, is stored in layer 3. Each component value ranges from 0 to 255.

Isosurface and Voxelgram Object Data Formats

The data for an isosurface or a voxelgram object is a 3D volumetric wave.

Isosurfaces and voxelgrams do not use color waves.

NaN Values in Waves

In general, you should avoid using NaNs in data waves plotted in Gizmo. NaNs appears as holes in surfaces or as discontinuities in path plots. A surface object whose data wave contains one or more NaN values is drawn in the usual way except that all constituent triangles for which at least one vertex is a NaN are not drawn. When possible, it is best to display missing data in Gizmo using transparency (see **Transparency and Translucency** on page II-345).

Gizmo Data Objects

Data objects, also called "wave-based objects", are display objects representing data in Igor waves. They include **3D Scatter Plots**, **Path Plots**, **Surface Plots**, **Ribbon Plots**, **Isosurface Plots**, **Voxelgram Plots** and **3D Bar Plots**.

Data Object Scaling

Data objects are displayed against a set of X, Y and Z data axes. These data axes exist whether or not you add an axis object to the plot. The data axes fill the +/-1 display volume.

By default, the data axes are autoscaled to the range of data in all data objects in the plot. You can change the range of the data axes using Gizmo→Set Axis Range.

The following commands illustrate these points. Execute them in a new experiment to follow along:

```
// Create a Gizmo plot with a surface object
NewGizmo
ModifyGizmo showAxisCue=1
Make/O/N=(100,100) data2D = Gauss(x,50,10,y,50,15)
AppendToGizmo/D surface=data2D, name=surface0
ModifyGizmo setQuaternion={0.206113,0.518613,0.772600,0.302713}
```

If you now choose Gizmo→Set Axis Range, you can see that all data axes, which are now invisible, are in autoscale mode (Manual checkboxes are unchecked) and that the X and Y axes range from 0 to 99. These values come from the fact that the default X and Y values of the data2D wave range from 0 to 99. The Z axis range is set based on the range of Z values in data2D. (If you opened the Axis Range dialog, click Cancel to dismiss it now.)

To help us visualize these data axes we add a box axis object with tick marks and tick mark labels:

```
// Add box axes with tick marks and tick mark labels
AppendToGizmo/D Axes=BoxAxes, name=axes0
ModifyGizmo ModifyObject=axes0, objectType=Axes, property={4,ticks,3}
ModifyGizmo ModifyObject=axes0, objectType=Axes, property={8,ticks,3}
ModifyGizmo ModifyObject=axes0, objectType=Axes, property={9,ticks,3}
```

If we change the values in the data, since the data axes are in autoscaling mode, the data still fills the axes (which always fill the display volume) after the change:

```
// Change the X and Y range of the data
SetScale x, 0, 10, "", data2D; SetScale y, 0, 10, "", data2D
```

Now we set the X and Y data axes to manual scaling mode but leave their ranges unchanged:

```
// Set the X and Y axes to manual scaling mode
ModifyGizmo setOuterBox={0,9.9,0,9.9,1.5286241e-11,0.001061033}
ModifyGizmo scalingOption=48
```

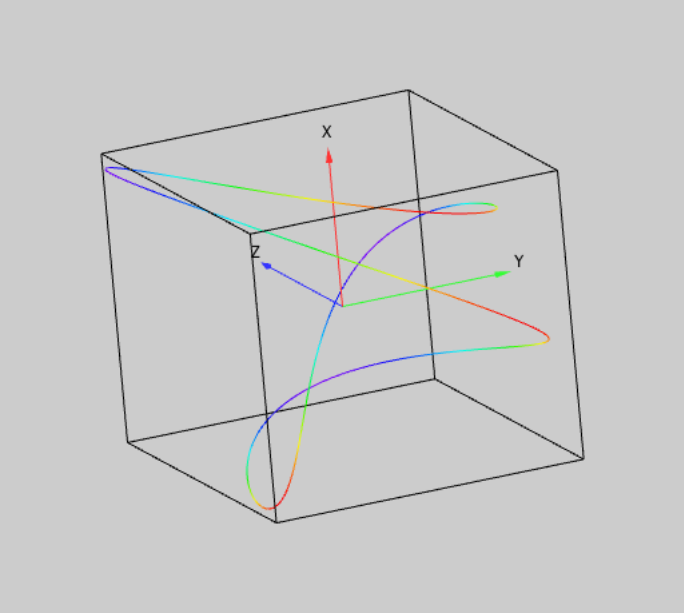
If we now change the range of the data, the data axes remain unchanged and the data no longer fills the axes:

```
// Change the X and Y range of the data
SetScale x, 0, 5, "", data2D; SetScale y, 0, 5, "", data2D
```

To recap, X, Y and Z data axes always exist, whether they are displayed or not. They always fill the +/-1 display volume. Data objects are displayed against the data axes which, by default, are autoscaled. Consequently, by default, data objects fill the display volume. If you change the data axes to manual scaling and change their range or change the range of the data, the data axes still fill the +/-1 display volume but the data objects no longer exactly fit.

Path Plots

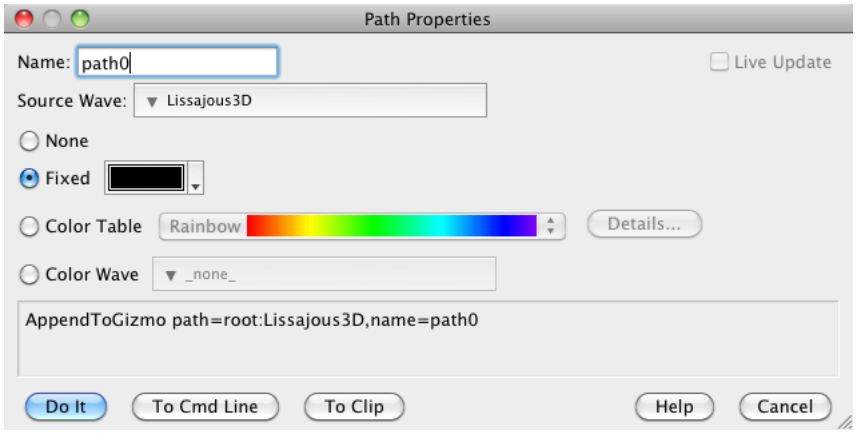
Each data point in a path plot is connected in sequential order by a straight line to each adjacent point.



To draw markers at the individual points, you need to append a scatter plot based on the same data wave.

A path plot requires a triplet wave, which is a 2D wave of 3 columns for X, Y, Z coordinates, respectively, of each vertex. A color wave for a path plot is a 2D wave in which each row specifies the color of the corresponding vertex in the data wave. The color wave has 4 columns which specify RGBA entries in the range of [0,1].

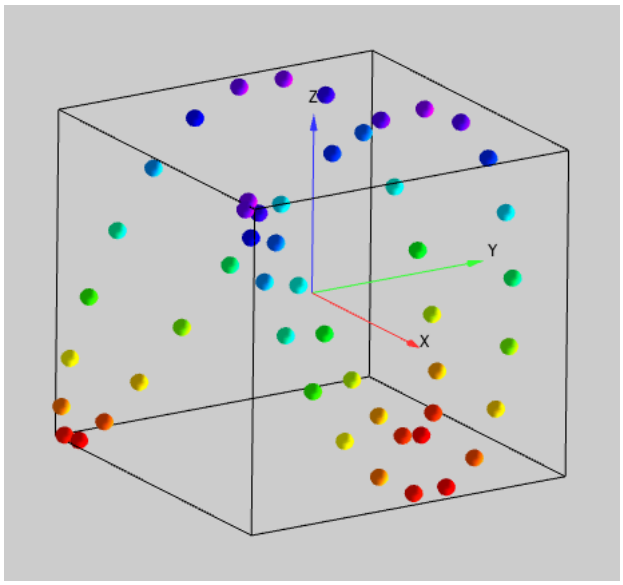
The Path Properties dialog looks like this:



The full list of available options is given under **ModifyGizmo**.

3D Scatter Plots

Each data point in a scatter plot is displayed as 3D marker.



If you want to connect the markers, you need to append a path plot based on the same data wave.

A scatter plot requires a triplet wave, which is a 2D wave of 3 columns for X, Y, Z coordinates, respectively, of each marker.

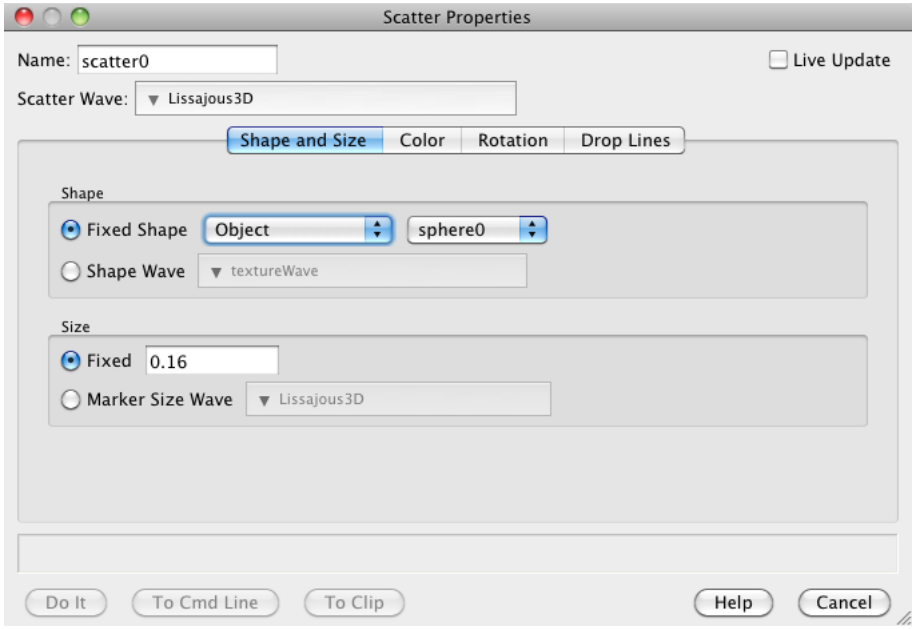
Each marker can be represented by a Gizmo object. You can select any one of the built-in drawing objects (e.g., box, sphere, cylinder or disk) or you can also select any object in the object list.

You can specify the size of the scatter objects to be a constant size, or provide a wave that contains a size specification for each scatter point.

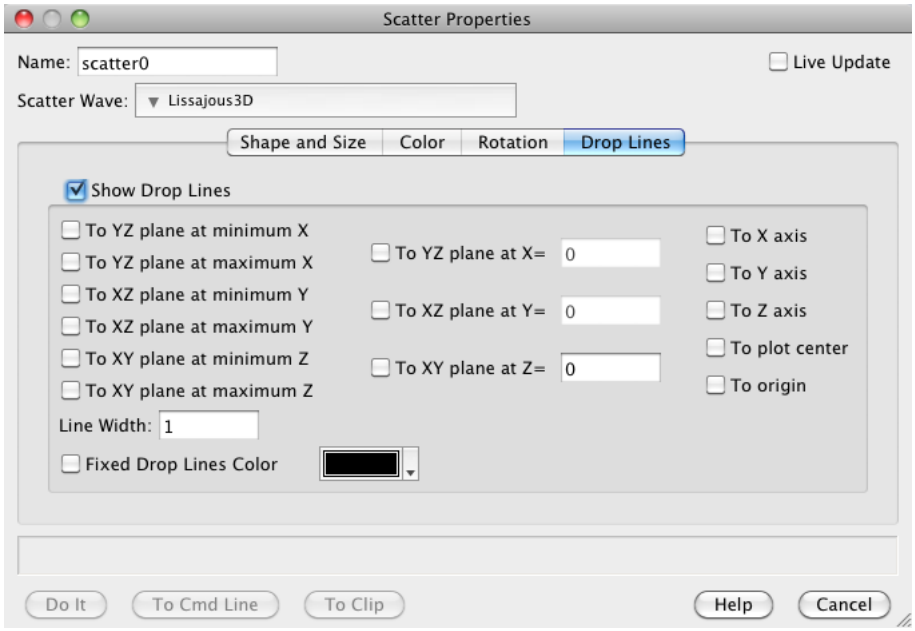
You can also specify the rotation of each scatter point. The rotation is specified by a 2D wave containing four columns. The first column is the rotation angle in degrees and the remaining three columns specify the normalized rotation axis vector, just as in the rotate operation.

A scatter plot can be colored using a constant color, a color taken from one of the built-in color tables, or a user-specified color wave. A color wave for a scatter plot is a 2D wave in which each row specifies the color of the corresponding element in the data wave. The color wave has 4 columns which specify RGBA entries in the range of [0,1].

The Scatter Properties dialog allows you to set all properties of the object. If you use the dialog when the object is already in the display list then checking the Live Update box lets you to see all changes as you make them.

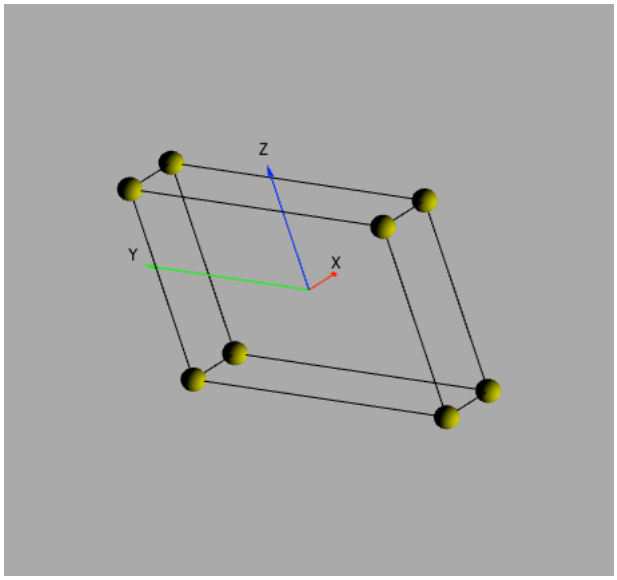


One useful feature of scatter plots is the ability to draw a "drop line". Drop lines start at the center of each marker and extend to one of 14 possible points, lines or planes. You can choose any combination of drop lines from the Drop Lines tab of the Scatter Properties dialog.



The full list of available options is given under **ModifyGizmo**.

You can display crystal structures in Gizmo using scatter plots. Here is an example:



Crystals are easy to display as scatter objects that are centered at coordinates provided in a triplet wave. For example, you can use a sphere as the object drawn at each center. You can specify additional waves to control the color and size of each sphere.

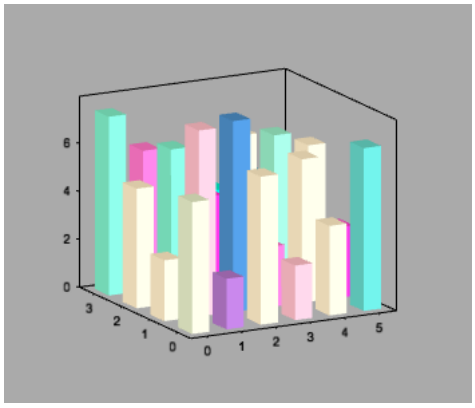
Igor provides two built-in transformations that convert triplet waves from crystallography coordinates to rectangular coordinates and vice versa. See the Crystal Demo experiment and the **WaveTransform** operation with the keywords crystalToRect and rectToCrystal.

3D Bar Plots

A 3D bar plot consists of a number of straight (parallel to all three axes) rectangular bars. The bars may be arranged on a regular grid or may be completely scattered within the volume.

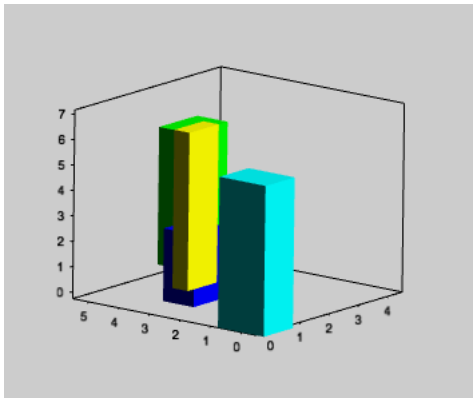
There are two modes for 3D bar plots: basic and refined.

In a basic 3D bar plot, the bars all start at zero and extend in the positive Z direction. Here is an example:



The basic 3D bar plot represents a 2D matrix of positive values. Each value is displayed as a zero-based bar according to its row/column in the matrix. All bars have the same width.

In a refined bar plot, bars can be drawn at arbitrary positions and all bar dimensions are under your control. Here is an example:



The refined 3D Bar plot displays requires a 6-column input wave where each row represents a single bar and the columns contain the following information:

Column 0:	The X center of the bar
Column 1:	The Y center of the bar
Column 2:	The lower Z value
Column 3:	The upper Z value
Column 4:	The width of the bar in the X direction
Column 5:	The width of the bar in the Y direction

Using the refined mode it is possible to create stacked and overlapping 3D bars plots as well as bars of varying sizes.

To find out more open the 3D Bar Plot Demo experiment.

Gizmo Image Plots

A Gizmo Image is a form of a quad object that is internally textured by image data.

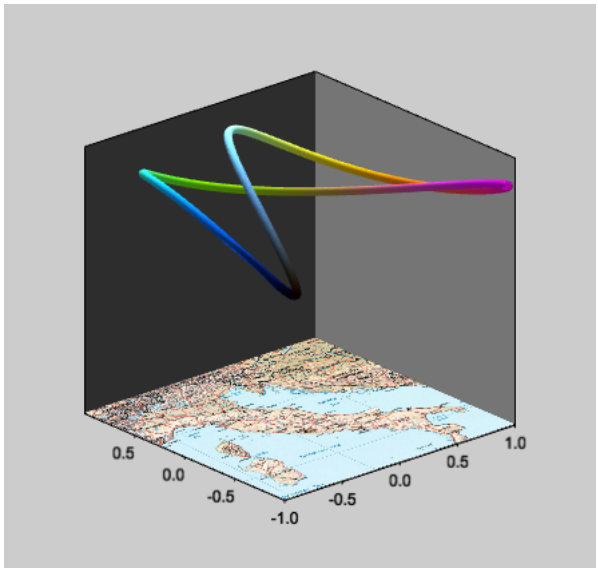
The image source wave can be in one of three formats:

- A 2D matrix of Z values
- A 3D RGB wave of type unsigned byte with 3 layers
- A 3D RGBA wave of type unsigned byte with 4 layers

The latter two formats are created by the **ImageLoad** operation.

By default, the image is displayed at the bottom of the display volume but using rotation, translation and scaling options you can place the image anywhere within it. If you need to register the image relative to the data you can modify the axis range of the Gizmo display or change the scaling of the image. The scaling is uniform about the center of the image.

Igor includes a flight path example that combines a dense scatter plot with a Gizmo image object. In this case the image consists of a map:



To find out more open the Flight Path Demo experiment.

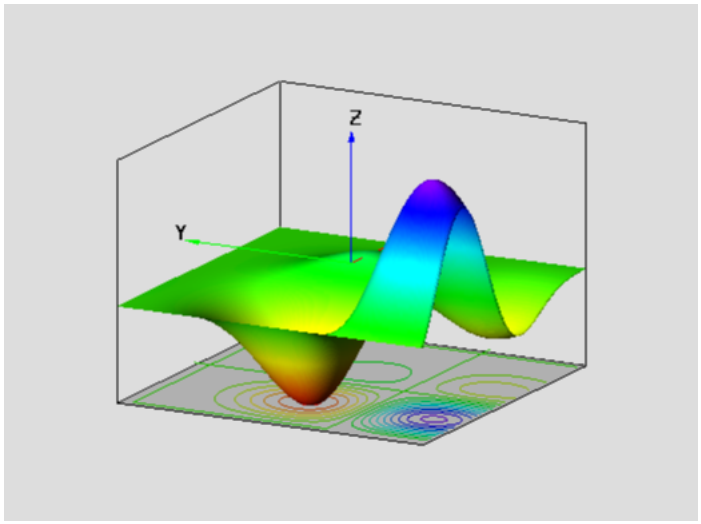
Surface Plots

A surface plot consists of a sheet connecting a grid of data values. You can specify surface colors from built-in color tables or custom color waves; see **Color Waves** on page II-343 for details. In addition to the surface, data values can also be displayed as points or as grid lines which can have their own color specification.

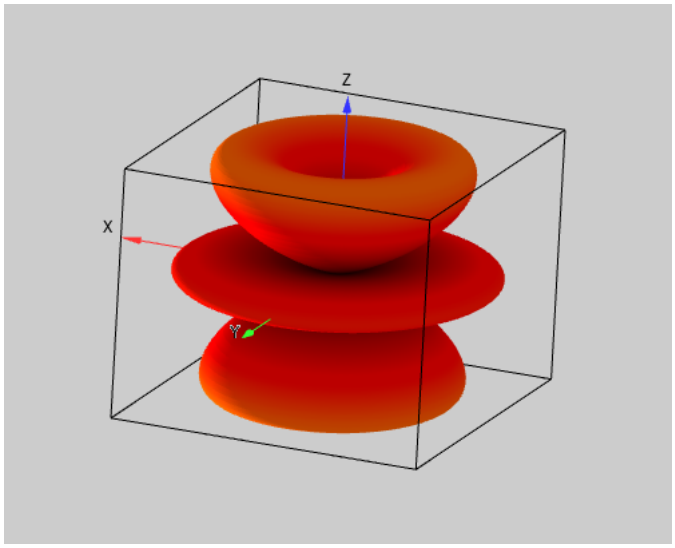
Typically data consist of an MxN 2D matrix of Z values which comprise the surface; see Surface Object Data Formats for details. Also supported are parametric surfaces which are 3D waves where each successive layer contains of the X, Y, and Z values in order; see **Parametric Surface Data Formats** on page II-365 for details.

The full list of available options is documented under **ModifyGizmo**.

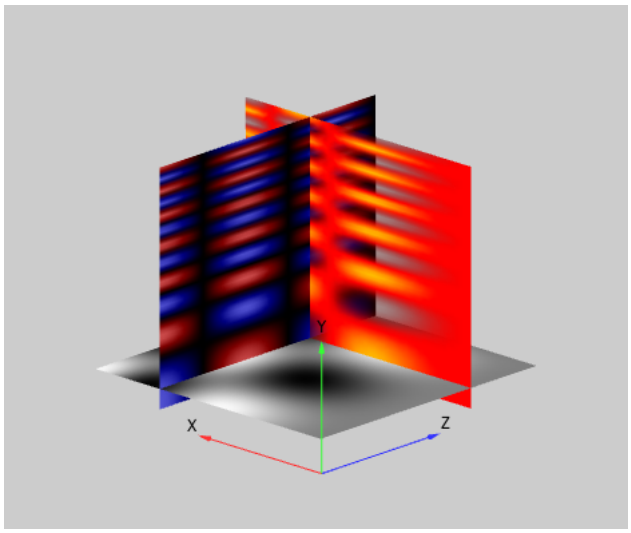
This example shows a surface plot with a contour map at the bottom:



This example shows a parametric surface plot of a spherical harmonic function:

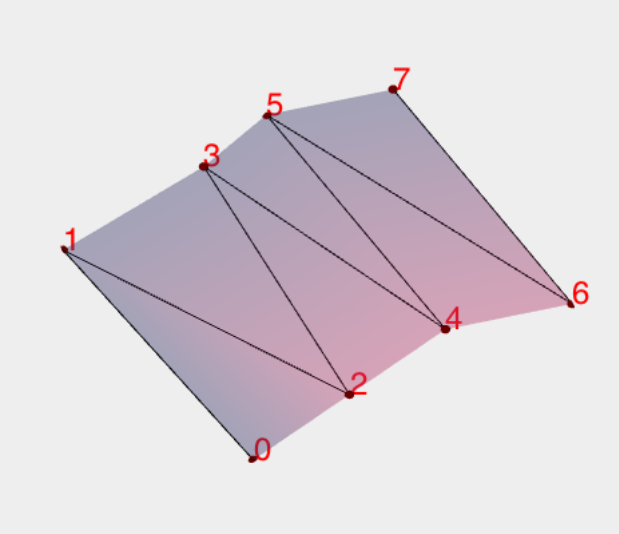


This example shows orthogonal slices of volumetric data:



Ribbon Plots

In a ribbon plot, data points are connected by a surface that defines the ribbon object. A ribbon is constructed from a list of triangles with alternating vertices as shown here:



To display the individual points or connections, you need to append a scatter plot or a path plot.

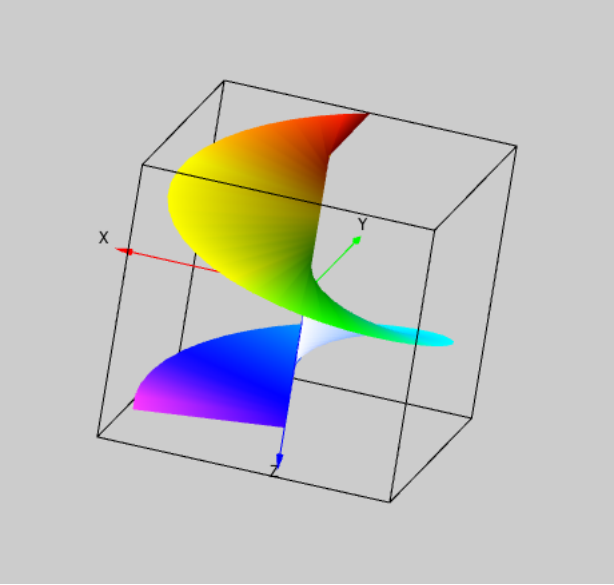
Data for a ribbon plot consist of a Nx3 matrix of values. Each row contains the X, Y, and Z values for the spatial coordinates of a point on the edge of the ribbon. The coordinates for each alternating edge of the ribbon follow in sequential order. The order of vertices for a ribbon is shown in the illustration above.

A ribbon must have at least four vertices and the total number of vertices must be even.

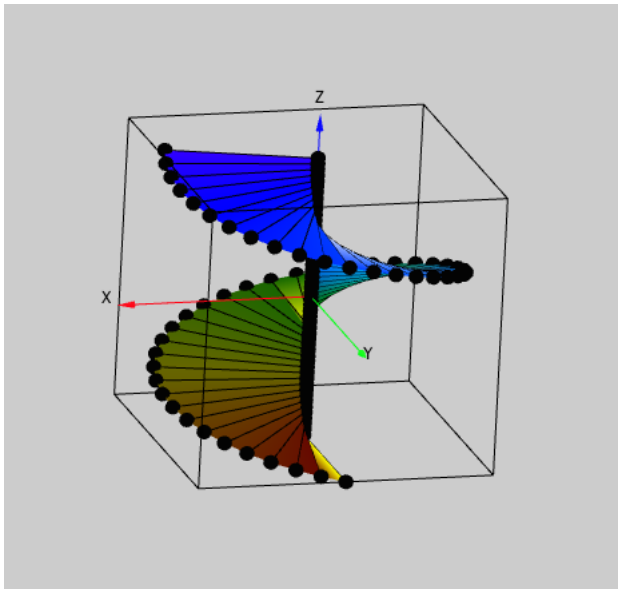
A ribbon plot can be colored using a constant color, a color taken from one of the built-in color tables, or a user-specified color wave. A color wave for a ribbon plot is a 2D wave in which each row specifies the color of the corresponding element in the data wave. The color wave has 4 columns which specify RGBA entries in the range of [0,1].

The full list of available options is given under **ModifyGizmo**.

This is an example of a simple ribbon plot:



This is an example of a ribbon plot overlayed with path and scatter plots of the same data in order to show the positioning of data points along the ribbon edges:



Voxelgram Plots

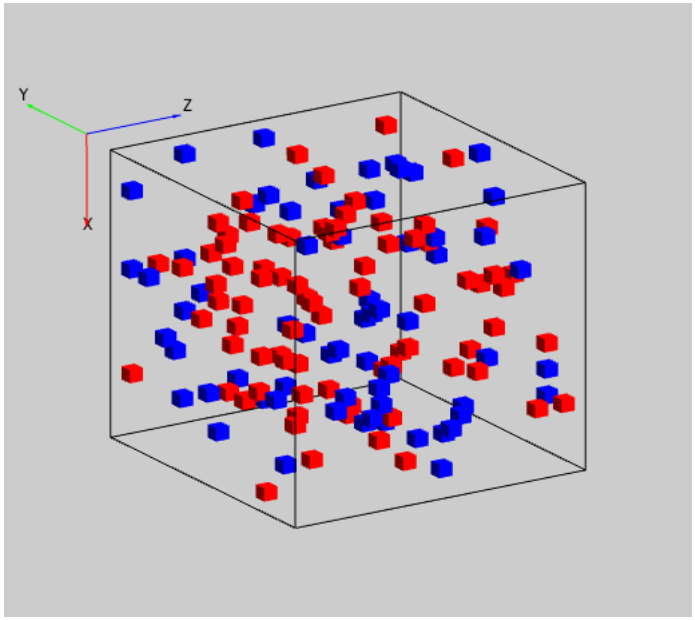
"Voxel" is short for "volume element".

A voxelgram is a representation of a 3D wave that uses color to indicate the wave elements containing certain values. You specify 1 to 5 values and, for each value, an associated RGBA color. If a given wave element's value matches one of the specified values within a specified tolerance, Gizmo displays that element using the associated RGBA color. If a wave element matches none of the specified levels then it is not displayed at all.

Voxels can be represented by cubes or points.

The full list of available options is given under **ModifyGizmo**.

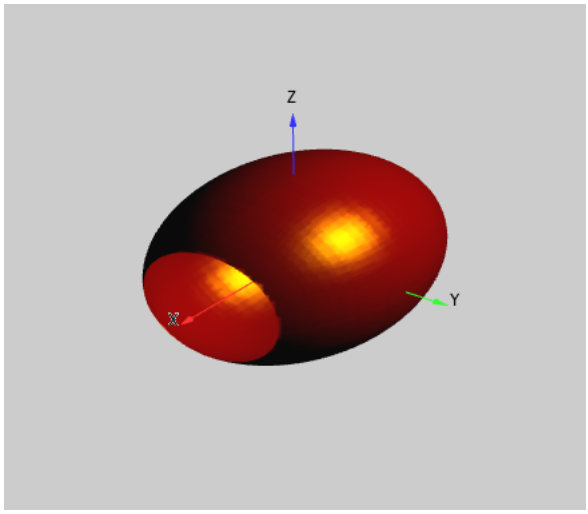
This example shows a voxelgram that uses two specified values:



Isosurface Plots

An isosurface is a 3D analog of 2D contour line. It is a surface drawn at the specified scalar value called the "isovalue". The surface may be represented by any combination of colors, grids, or points.

Here is an example of a translucent isosurface:



Isosurface generation is computationally intensive because it subdivides the volume into tetrahedra. If the isovalue is found in any tetrahedron, the intersection (one or two triangles) is calculated. The isosurface itself is composed of the collections of these triangles.

Isosurfaces are constructed from 3D volumetric data waves.

The full list of available options is given under **ModifyGizmo**.

Printing Gizmo Windows

You can print the image in the Gizmo display window by choosing File→Print when the Gizmo display window is the front window.

The window is printed at a multiple of screen resolution. The multiple is controlled by the Default Output Resolution Factor in the Gizmo section of the Miscellaneous Settings dialog which you can access via the Misc menu. The factory default value for this multiple is 2.

You can override the default printing resolution by executing:

```
ModifyGizmo outputResFactor = n
```

where n is a positive integer, typically 1, 2, 4 or 8. This applies to the active window only. It affects subsequent printing and overrides the Default Output Resolution Factor setting. This setting is not stored in the recreation macro for the Gizmo window and therefore does not persist. The maximum value of n that will work depends on the amount of video memory (VRAM) that you have in your graphics hardware.

You can also improve the output by anti-aliasing objects. To do this, add to the display list a blend function with GL_SRC_ALPHA and GL_ONE_MINUS_SRC_ALPHA and add an enable operation with GL_LINE_SMOOTH. You can also enable the GL_POINT_SMOOTH to smooth points in a scatter object.

If you are unable to print an image from Gizmo you may have run out of VRAM. This may produce a blank or distorted graphic. Some of the things that you should try are:

- Close any other Gizmo window that you might have open.
- Reduce the size of the Gizmo display window.

- Reduce the resolution as set by the Default Output Resolution Factor setting or via ModifyGizmo outputResFactor.
- If you are working on a system with more than one monitor move the display window to the one driven by a graphics card with the most VRAM.
- Run the experiment on hardware that has more VRAM.

Exporting Gizmo Windows

You can export a Gizmo plot using one of these techniques:

- Choose File→Save Graphics to export to a PNG, JPEG or TIFF file. This generates a **SavePICT** command.
- Choose Edit→Export Graphics to export to the clipboard as PNG, JPEG or TIFF.
- Choose Edit→Copy. This exports to the clipboard using the settings last set in the Export Graphics dialog.
- Right-click and choose Copy to Clipboard from the contextual pop-up menu. This is the same as choosing Edit→Copy.

The **ExportGizmo** operation is also available for backward compatibility only. It is obsolete and you should use SavePICT instead.

The Export Graphics dialog and the SavePICT operation give you control of the output resolution as a multiple of screen resolution. Exporting at high resolution requires sufficient video memory (VRAM). Most hardware supports 2x (two times screen resolution). You may be able to increase resolution further depending on the available VRAM.

If you are unable to export an image from Gizmo you may have run out of VRAM. This may produce a blank or distorted graphic. Some of the things that you should try are:

- Close any other Gizmo window that you might have open.
- Reduce the size of the Gizmo display window.
- Reduce the resolution as set in the Export Graphics or Save Graphics dialogs.
- If you are working on a system with more than one monitor move the display window to the one driven by a graphics card with the most VRAM.
- Run the experiment on hardware that has more VRAM.

Advanced Gizmo Techniques

There are many advanced Gizmo options that are, by default, hidden. To display them you need to check the Display Advanced Options Menus checkbox in the Gizmo section of the Miscellaneous Settings dialog which you can access via the Misc menu. Most Gizmo users will not need these options.

Group Objects

A Gizmo group object is an encapsulation of Gizmo objects and operations that are treated as a single object in the top-level Gizmo. To work with group objects you must enable the advanced Gizmo menus in Miscellaneous Settings.

The following example illustrates how you can use a group object to create a custom marker for a scatter plot.

Start a new Igor experiment and execute on the command line:

```
NewGizmo /I /JUNK=3
```

This creates a simple scatter plot with default red sphere markers.

Create a group object using the + icon below the object list in the info window. You can get the same result by executing the command:

```
AppendToGizmo/D group, name=group0
```

Double-click the group0 object in the object list. This opens a new info window for the group with display, object and attribute lists. Add a blue sphere object with a radius of 0.15 and a red cylinder object with top and bottom radii of 0.05 and a height of 0.3 to the group. Drag the sphere and cylinder objects to the group's display list. The following commands achieve the same thing:

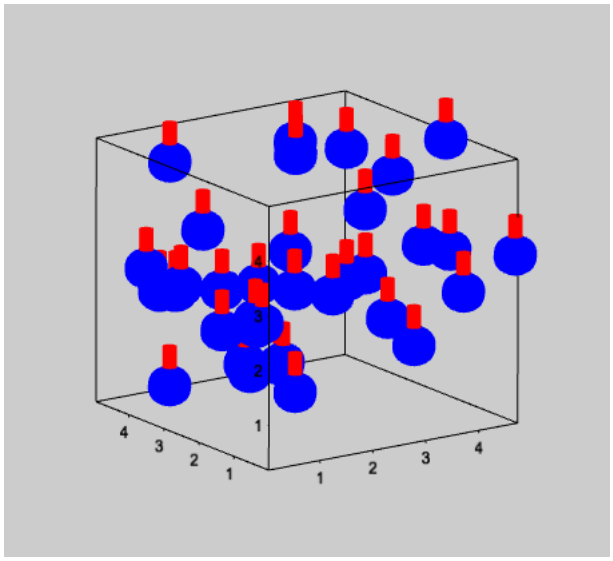
```
ModifyGizmo currentGroupObject="Gizmo0:group0"
AppendToGizmo/D sphere={0.15,10,15}, name=sphere0
ModifyGizmo modifyObject=sphere0, objectType=Sphere, property={colorType,1}
ModifyGizmo modifyObject=sphere0, objectType=Sphere, property={color,0,0,1,1}
ModifyGizmo modifyObject=sphere0, objectType=Sphere,
      property={useGlobalAttributes,0}
ModifyGizmo modifyObject=sphere0, objectType=Sphere, property={normals,100000}
AppendToGizmo/D cylinder={0.05,0.05,0.3,10,15}, name=cylinder0
ModifyGizmo modifyObject=cylinder0, objectType=Cylinder,
      property={colorType,1}
ModifyGizmo modifyObject=cylinder0, objectType=Cylinder,
      property={color,1,0,0,1}
ModifyGizmo modifyObject=cylinder0, objectType=Cylinder,
      property={useGlobalAttributes,0}
ModifyGizmo modifyObject=cylinder0, objectType=Cylinder,
      property={normals,100000}
ModifyGizmo setDisplayList=0, object=sphere0
ModifyGizmo setDisplayList=1, object=cylinder0
ModifyGizmo currentGroupObject="::"
```

Close the group0 info window.

In the Gizmo0 info window, double-click the scatter0 object to open the Scatter Properties dialog. In the Shape and Size tab of the dialog, select Object from the Fixed Shape pop-up menu and group0 from the pop-up menu just to the right of it. Click Do It. The following commands achieve the same thing:

```
ModifyGizmo ModifyObject=scatter0, objectType=scatter, property={Shape,7}
ModifyGizmo ModifyObject=scatter0, objectType=scatter,
      property={objectName,group0}
```

The resulting graph should look something like this:



Another example of group objects can be found in the Scatter Arrows demo experiment where the group object consists of connected cylinder and cone representing an arrowhead which is plotted at every scatter center with a user-specified rotation and scaling.

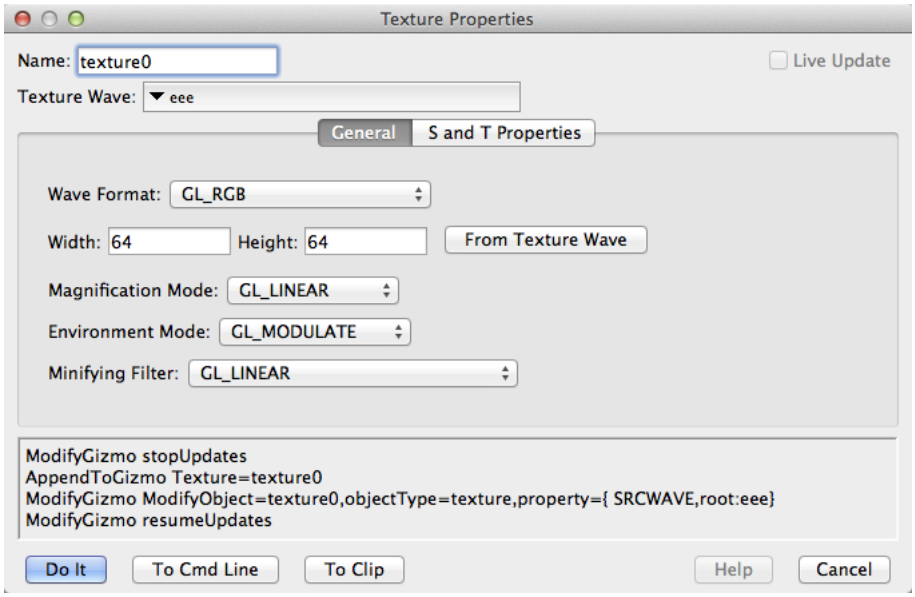
Texture Objects

Textures are used extensively in OpenGL so most graphics hardware are optimized to use them. An in-depth discussion of texture is beyond the scope of this document so we will focus on the use of textures in Gizmo. To work with texture objects you must enable the advanced Gizmo menus in Miscellaneous Settings.

A Gizmo texture object represents an OpenGL object that allows you to apply image information to surfaces of arbitrary shape. Textures in Gizmo are 1D or 2D. They can be applied to quad objects, quadric objects (spheres, cylinders and disks) and parametric surfaces. If you intend to use a texture on a simple quad you should use an image plot instead; see **Gizmo Image Plots** on page II-371.

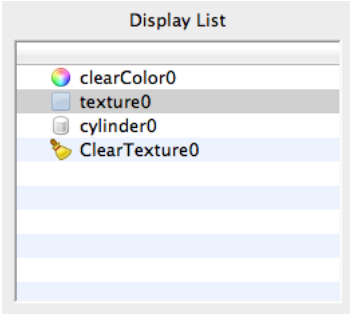
You usually create a texture from an image that was loaded using **ImageLoad** and is in the form of an unsigned byte 3D RGB wave where the color for each pixel is stored in sequential layers. Before you create a Gizmo texture you need to convert the standard image format into a 1D wave where the color entries for each pixel are stored sequentially as RGB or RGBA. This conversion can be accomplished by **ImageTransform** with the keyword imageToTexture.

The Texture Properties dialog determines how the texture wave is converted into a texture object and how the texture is applied in a drawing.



Before invoking the dialog it is useful to know the dimensions of the texture wave, its packing format and the original image size. ImageTransform stores this information in the wave note of the texture wave. The dialog loads the texture information from the wave note if you click the From Texture Wave button.

Once you create a texture object you can apply it to other objects. For example, to apply texture0 to cylinder0 your display list should contain the following sequence:



where cylinder0 is preceeded by texture0 and followed by clearTexture0. You must check the Uniform Texture Coordinates checkbox in the Cylinder Properties dialog. The texture object is placed just before the cylinder and the clear texture operation follows so that subsequent drawings are free of textures.

The Gizmo Earth demo experiment illustrates the use of textures. In this case a high-resolution rectangular texture is added to a parametric surface representing the sphere. The parametric surface consists of 61x61 vertices.

Matrix4x4 Objects

Matrix4x4 objects are used with Gizmo operations that affect the transformation matrix. The encapsulate a 2D wave with 4 rows and 4 columns.

Gizmo Subwindows

You can embed a Gizmo subwindow in a graph, panel or layout window. Here is a simple example:

```
NewPanel/N=MyPanel
NewGizmo/HOST=MyPanel/JUNK=2
```

You can direct any ModifyGizmo commands using full subwindow specification. For example, to change the colormap of the demo surface you can execute:

```
ModifyGizmo/N=MyPanel#GZ0 ModifyObject=surface0, objectType=surface,
      property={surfaceCTab,Blue}
```

For more information see **Subwindow Syntax** on page III-87.

Gizmo Troubleshooting

This section provides tips for resolving problems you may encounter while using Gizmo.

Troubleshooting Gizmo Lighting Problems

1. Make sure that the light object appears in the display list above the objects that you expect to be illuminated.
2. Check that your light specifies non-black colors for the ambient, diffuse and specular components.
3. Start with directional light which has fewer options than positional light with less to go wrong. Open the Light Properties dialog, check the Live Update checkbox, change the direction controls and check for changes in the scene. If you do not see changes in the scene then the direction of the light is more than likely not the problem.
4. Verify that you have checked the Compute Normals checkbox for each object that you expect to be illuminated.
5. Check that the normal orientation is what you expect. In the case of quadric objects, toggle the normals orientation between inside and outside. For other objects, consider the definitions of front and back surfaces. If you have doubt, add a front face operation to the display list right before an object. The operation allows you to define the direction representing front and back surfaces.

6. If you are drawing an object that looks fine at normal scale but appears to have the wrong lighting effects when drawn after a scale operation you should compute the object at the final size and avoid scaling. For example, a scatter plot draws markers by default with an isotropic scaling factor of 1 (no scaling). If you create a sphere object with radius 1 for use in a scatter plot, you would need to reduce its size somewhere. You can do so by setting the scale in the Scatter Properties dialog or by reducing the radius of the sphere.

Troubleshooting Gizmo Transparency Problems

1. Check the display list to make sure that there is a GL_BLEND enable operation above the transparent objects and that a blend function exists and has the correct parameters. See **Transparency and Translucency** on page II-345 for details.
2. Check the drawing order of objects. Transparent objects must be after opaque objects in the display list.
3. If you have transparent surfaces that have some overlapping facets, you have no choice but to decompose the surface into triangles and order them according to depth. You can find an example of this in the Depth Sorting demo experiment.

Troubleshooting Gizmo Clipping Problems

Try these tips if an object or part of an object is clippled or not visible.

1. Set the axis range to auto for all axes using Gizmo Menu→Axis Range.
2. Make sure that the display list does not contain any clipping planes.
3. Make sure that the object in question does not follow in the display list after translate, rotate or scale operations.
4. Check the zoom level and the main transformation. If you do not have any transformation operations on the display list insert a default ortho operation (+/-2 for all axes) at the top of the display list.

Gizmo Compatibility

Prior to Igor Pro 7.00 Gizmo was implemented as an XOP (plug-in). It is now built into Igor. It also uses a later version of OpenGL. Some syntactical changes were made.

OpenGL Changes

Before Igor7, Gizmo was implemented based on the original OpenGL 1.0 specification. Changes in OpenGL, especially since OpenGL 3.0, required that we modify various features of Gizmo. We have made an effort to allow Gizmo windows in old Igor experiments to open in Igor7 without error and to appear as close as possible to their original form. Notable exceptions include the orientation of string objects, axis labels and tick mark labels; see **Changes to Gizmo Text** on page II-383.

We have also marked in the Gizmo reference documentation some features as "obsolete" or "deprecated". "Obsolete" means that the feature is no longer supported. "Deprecated" means that it may be removed from a future version of Igor and you should use an alternative feature if possible.

Gizmo Commands in User-Defined Functions

As of Igor7, Gizmo commands can be used in user-defined functions. Supporting this required some changes to the command syntax. The main syntax change is the addition of the objectType keyword in ModifyGizmo. Prior to Igor7 you could use:

```
ModifyGizmo modifyObject=quad0, property={calcNormals,1}
```

The Igor7 syntax requires adding the objectType=<type> keyword as in:

```
ModifyGizmo modifyObject=quad0, objectType=Quad, property={calcNormals,1}
```

So that Igor7 Gizmo can interpret recreation macros created by Igor6, the objectType keyword is required only in user-defined functions, not from the command line or in macros.

For backward compatibility, the Gizmo XOP as of Igor Pro 6.30 accepts the objectType keyword.

Gizmo Recreation Macro Changes

Prior to Igor7 every Gizmo recreation macro contained the line:

```
ModifyGizmo startRecMacro
```

As of Igor7 the syntax includes a version number e.g.,

```
ModifyGizmo startRecMacro=700
```

For backward compatibility, the Gizmo XOP as of Igor Pro 6.30 accepts the new startRecMacro syntax.

As of Igor7 Gizmo uses counter-clockwise as the front face default except when executing Gizmo recreation macros from previous versions. This is unlikely to affect most you but if it does, you can override this default by specifying an explicit frontFace operation in the Gizmo display list.

Use of GL Constants in Gizmo Commands

Previous versions of Gizmo supported the use of OpenGL constants in commands. For example, you could write:

```
AppendToGizmo attribute blendFunc={GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA},
      name=blend0
```

In Igor7 the GL constants, such as GL_SRC_ALPHA, are not recognized and you must use the numeric equivalent instead. For example:

```
AppendToGizmo attribute blendFunc={770,771}, name=blend0
```

You can execute GetGizmo to determine what numeric value represents a given constant. For example:

```
GetGizmo constant="GL_SRC_ALPHA"
```

This prints "770" to the history. You can copy the printed number and past it into your procedure.

Exporting Gizmo Graphics

As of Igor7 you can export Gizmo graphics using File→Save Graphics which generates a **SavePICT** command. The **ExportGizmo** operation is only partially supported for some degree of backward compatibility. It can export to the clipboard or to an Igor wave and it can print but it can no longer export to a file. Use SavePICT instead.

Changes to Gizmo Text

Prior to Igor7 Gizmo displayed axis labels, tick mark labels, and string objects by composing a 3D filled polygon for each character. The polygon representation allowed Gizmo to handle each string as a true 3D object which could be drawn at any position in the display volume independent of the orientation of the axes. This approach had three disadvantages:

- Labels were not always legible and could disappear completely in some orientations
- The conversion into polygons made it impractical to use anti-aliasing to smooth the characters
- Font sizes were inconsistent across platforms

In Igor7 we moved Gizmo to a new text rendering technology that generates 2D smooth text. While this technology addresses the three issues mentioned, it does not produce text objects that match those produced by previous versions. Consequently you will see differences when loading pre-Igor7 experiments that use text, such as axis labels, with offsets and rotations which are no longer supported.

As of Igor7 you can use formatted text to construct complex axis labels. Double-click an axis item in the Display or object list to display the Axis Properties dialog and click the Axis Labels tab.

As of Igor7 you can add standard Igor annotations, including textboxes and colorscales, to a Gizmo window using Gizmo→Add Annotation. These annotations appear in an overlay in front of the 3D graphics and behave like annotations in a graph window.

Miscellaneous Gizmo Changes

As of Igor7 arguments to the shininess attribute have changed to front and back values.

As of Igor7, a Gizmo object optionally has an internal color attribute. When you create an object you have the option to specify a color or to leave it unspecified. If you specify a color, Gizmo creates a default color material for the object. The default color material has the GL_FRONT_AND_BACK and GL_AMBIENT_AND_DIFFUSE settings. If you don't specify a color then Gizmo does not create a default color material and you must create a color material yourself. This color material affects all objects that appear later in the display list if they have no default color material. This change was necessary in order to support creation of shiny surfaces.

Gizmo Hook Functions

This section is for advanced programmers only.

A hook function is a user-defined function called by Igor when certain events occur. It allows a programmer to react to events and possibly modify Igor's behavior. A window hook function is a hook function that is called for events in a particular window.

Igor's support for this feature is described under **Window Hook Functions** on page IV-276 and, as of Igor7, applies to Gizmo as well as other types of windows.

Because Gizmo was previously implemented as an XOP, it has its own hook function mechanism separate from the Igor mechanism. This section describes Gizmo's specific hook function support.

You can use either Igor hook functions or Gizmo hook functions or both for a Gizmo window. However using both may lead to confusion. If you install both an Igor hook function and a Gizmo hook function on a given Gizmo window, the Igor hook function is called first.

As in Igor itself, Gizmo originally had just one window hook function, installed by the ModifyGizmo hookFunction keyword. Later a named hook function, installed by ModifyGizmo namedHook, was added. Unnamed hooks are obsolete. We recommend that you use named hooks.

Gizmo Named Hook Functions

A named Gizmo window hook function takes one parameter - a WMGizmoHookStruct structure. This built-in structure provides your function with information about the status of various window events.

You install a named Gizmo hook function using **ModifyGizmo** with the namedHook or namedHookStr keywords. The hookEvents keyword is not relevant for named hook functions.

The hook function should usually return 0. In the case of mouse wheel hook events returning a non-zero value prevents Gizmo from rotating in response to the wheel.

The named window hook function has this format:

```
Function MyGizmoHook(s)
   STRUCT WMGizmoHookStruct &s

   strswitch(s.eventName)
      case "mouseDown":
         break
      case "mouseMoved":
```

```
         break
      case "rotation":
         break
      case "killed":
         break
      case "scaling":
         break
   endswitch

   return 0
End
```

See **WMGizmoHookStruct** for details on the structure.

As of this time the following event names are defined: mouseDown, mouseMoved, rotation, killed and scaling.

For an example using a named Gizmo window hook, open the Gizmo Window Hook demo experiment and look at the GizmoRotationNamedHook function in the GizmoRotation.ipf procedure file. This is a packed procedure file. It is in an independent module so you need to enable **Independent Modules** to see it.

Gizmo Unnamed Hook Functions

Unnamed hooks are obsolete though still supported for backward compatibility. Use named hooks instead. The following documentation is for historical reference only.

Each Gizmo window can have one and only one unnamed Gizmo window hook function. You designate a function as the unnamed window hook function using the **ModifyGizmo** operation with the hookFunction keyword.

The unnamed hook function is called when various window events take place. The reason for the hook function call is stored as an event code in the hook function's infoStr parameter. Certain events must be enabled using the **ModifyGizmo** operation with the hookEvents keyword.

The unnamed hook function has the following syntax:

```
Function functionName(infoStr)
   String infoStr

   String event = StringByKey("EVENT",infoStr)
   ...
   return 0          // Return value is ignored
End
```

infoStr is a string which containing a semicolon-separated list of keyword:value pairs. For documentation see “Gizmo Unnamed Hook Functions” in the “3D Graphics” help file.