Crafter

spacecraft design tool

User Manual

Contents

1 Overview 1

2 Installation 2
2.1 Installing Geomview 2
2.2 Installing Crafter 2

3 Interaction 3
3.1 Starting Crafter 3
3.2 Command line options 4
3.3 Overview of Geomview 5
3.4 Basic Interaction: The Model Editor 7
Hierarchy list 7
Selection 8
Navigating in the hierarchy list 8
File name 8
Menus 9
3.5 Creating and attaching objects 9
3.6 Editing an object's properties 10
General properties 10
Long name 11
Group properties vs. individual properties . . . 11
Transformations 12
Coordinate axes 13
Appearance 14
Color 16
Buttons on the General Properties panel 17
Class-specific properties 17
Model properties 18
Options window 19
The Null primitive 20
3.7 Editing the hierarchical structure 20
3.8 Reading and writing files 21
3.9 Feeding commands to Crafter 22
Typing in commands directly 23
Reading commands from a pipe 23
3.10 Quitting Crafter 23
3.11 Keyboard Shortcuts 23

4 More on Transformations 25

5 Command Language 28
5.1 Brief introduction to Tcl 28
Command syntax 28
Variables and substitution 29
Quotes and braces 29
Control flow 30
Comments 31

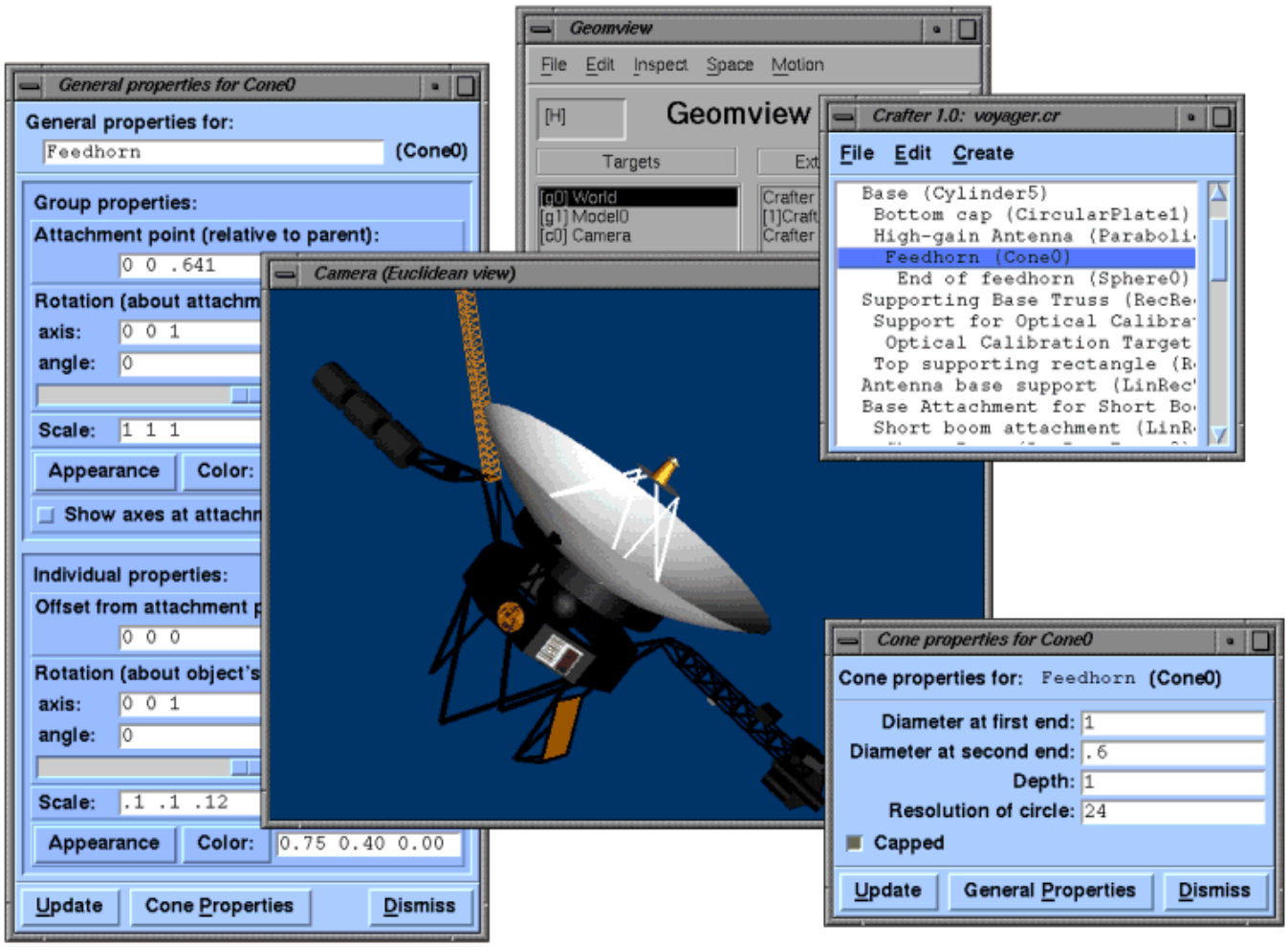
5.2 Crafter-specific commands 31
Conventions 31
Manipulating objects 31
File manipulation 32
Examples 32

6 Appendix 33
6.1 Menu items 33
6.2 Primitives and properties 35

1. Overview

Crafter is an interactive program for creating geometric models of spacecraft. Given a palette of spacecraft pieces (truss, parabolic dish, cylinder, etc.), the user builds a hierarchical, geometric structure. Each part is attached to the structure with a specified transformation, consisting of a scale, a rotation and a translation.

As an external module for Geomview, Crafter uses Geomview to display the spacecraft model at all times, allowing the user to view it from any angle. Changes made to the model take effect in the Geomview window immediately.



Crafter's learning curve is orders of magnitude smaller than full blown CAD packages. Crafter is also scriptable in Tcl, making it easy to integrate with other programs.

2. Installation

Before installing Crafter, version 1.6 or better of Geomview must be installed.

2.1. Installing Geomview

The compressed tar file for the Geomview distribution is named something like `geomview-1.6.1-sgi.tar.gz`. Unpacking the Geomview distribution creates an directory called `Geomview` in which the main Geomview tree resides. Installing Geomview puts relevant executable files in a common directory, such as `/usr/local/bin`. To unpack and install Geomview, move the distribution to the directory that you want to put the Geomview tree, `cd` to that directory and type:

```
gunzip < geomview-1.6.1-sgi.tar.gz | tar xvopf -
cd Geomview
./installbin
```

substituting the actual name of the compressed tar file for `geomview-1.6.1-sgi.tar.gz`. *See the Geomview manual for complete directions on the installation procedure.*

2.2. Installing Crafter

There are three things you should have in mind before installing Crafter.

1. where you want the Crafter source tree installed. This can be anywhere that is readable to those who will be using it.

2. the path to the Geomview executable file. **Geomview must already be installed.** (The Crafter installation will look to see if `geomview` is in the search path.)

3. the path to install the Crafter executable and other executable files associated with Crafter. This should be in the search path of the users of Crafter (typically `/usr/local/bin`).

Installing Crafter is similar to installing Geomview. The tar file for the Crafter distribution is named something like `crafter-1.0-sgi.tar.Z`. To unpack and then install Crafter, go the directory you want to install the Crafter source tree, move the distribution to that directory, and then type commands like the following:

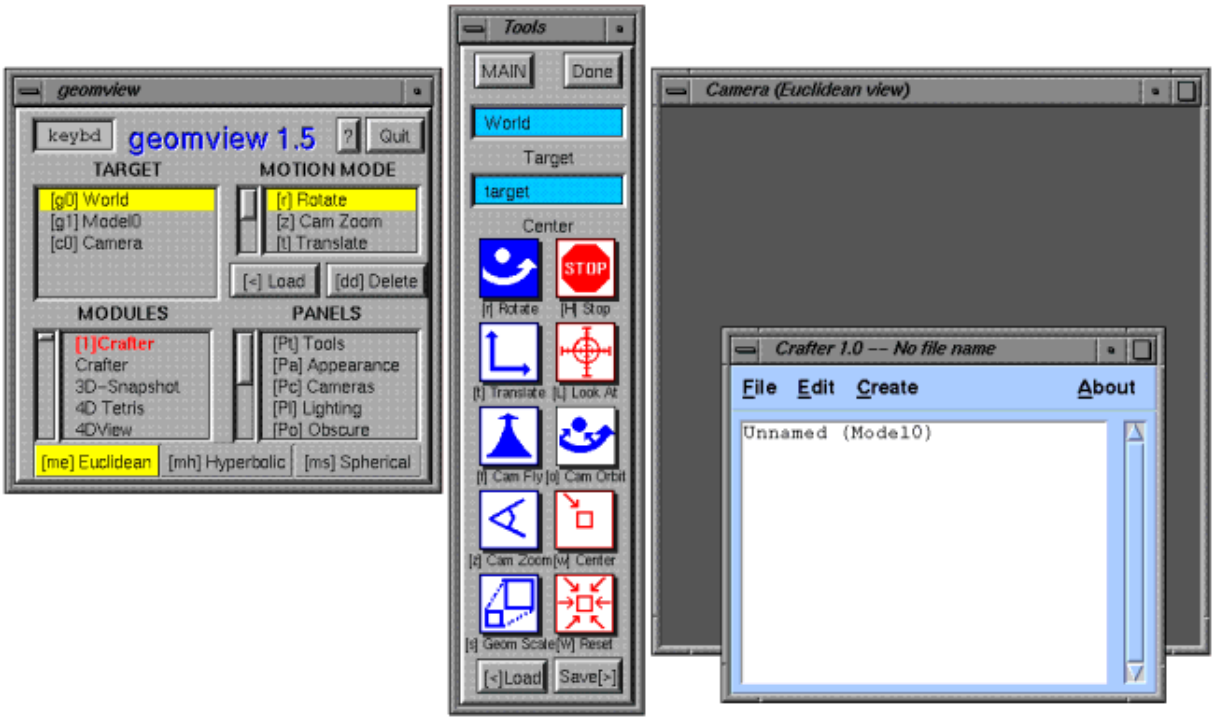
```
zcat crafter-1.0-sgi.tar.Z | tar xvopf -
cd Crafter
./install-crafter
```

substituting the actual name of the tar file for `crafter-1.0-sgi.tar.Z`.

3. Interaction

3.1. Starting Crafter

The simplest way to start Crafter is to type `crafter` from a shell window. First Geomview will start up with its three windows. Then Crafter's main window will appear (the lower-right of the windows below).



Geomview is used only to display the model and let you look at it from different angles. Crafter windows handle all editing of the model.

3.2. Command line options

When starting Crafter, you can specify a file name on the command line. For example, typing:

```
crafter voyager.cr
```

will load the file `voyager.cr` if it exists. Otherwise, `voyager.cr` will be the name of the current file once you save it. If more than one file name is given, then the first file is loaded the normal way, and subsequent files are merged with the first one.

Other command line options available are:

`-c` *`command`*
: read Tcl commands from the command line. For example, to start Crafter with highlighting on, type

```
crafter -c "SetProps Model0 {highlightsel 1}"
```

Multiple commands may be specified by separating them with semicolons (;).

`-f` *`filename`*
: read Tcl commands from a file. Example:

```
crafter -f startup.tcl
```

In a file, commands may be separated by newlines or semicolons (;).

`-p` *`pipename`*
: read Tcl commands from a named pipe. If no path is given, the pipe is put in `/tmp/geomview`. Example:

```
crafter -p mypipe
```

Commands are read from the pipe `/tmp/geomview/mypipe`.

`-noui`
: do not display the user-interface. This doesn't make much sense without the -c, -f or -p option.

`-nogv`
: do not display the model in geomview

3.3. Overview of Geomview

Crafter is an external module to Geomview. That means that Geomview starts Crafter as a separate process and communicates with Crafter through standard input and standard output. Theoretically, the only of Geomview's windows you need to be concerned with is the camera window. However, knowing a little more about Geomview may help you understand why things behave the way they do. It will also give you a better understanding of how to customize Crafter.

Geomview windows

When Geomview starts up, it has three panels. The panel on the left is Geomview's *main* control panel. The skinny panel in the middle is the *tools* panel and is for selecting different kinds of motions. The window on the right is the *camera* window, which is used to display the Crafter model.

Main panel

The main panel consists of a menu bar and two important lists: the list of *Targets* and the list of *External Modules*.

The list of targets contains all the objects that Geomview currently knows about. When you are using Geomview with Crafter, the list should look like this:

```
[g0] World
[g1] Model0
[c0] Camera
```

`Model0` is the entry for the Crafter model. Geomview treats the entire Crafter model as one object. `World` corresponds to all the currently loaded objects, treated as if they were one object. `Camera` refers to the camera window on the right. While using Crafter, it is safe just to keep `World` selected as the target object.

The other list is the list of *External Modules*. When using Crafter, this list should contain at most three lines. It should look something like the following:

```
Crafter
[1]Crafter "voyager.cr"
Crafter "voyager.cr"
```

The red line starting with a number in brackets represents the Crafter process running. Clicking on this entry kills the Crafter process. The other two lines allow you to restart Crafter. Be careful.

Tools panel

The *Tools* panel allows you to change the type of motion you get when you drag the mouse in the *Camera* window. It also has other operation that allow you to change the camera angle without changing the mode of motion. The most common motion modes you might want to use with Crafter are:

- Rotate
  - dragging left mouse rotates object in direction of drag.
  - dragging middle mouse rotates object around axis perpendicular to the screen.
- Translate
  - dragging left mouse translates object in direction of drag.
  - dragging middle mouse translates object in direction perpendicular to the screen.
- Cam Fly
  - dragging left mouse rotates camera in direction of drag.
  - dragging middle mouse translates camera perpendicular to the screen.
- Cam Zoom
  - dragging left mouse changes camera's field of view.

Other common operations that do not change the motion mode are:

- Stop
  Halts all motion (from inertia).
- Look At
  Moves the camera to a position such that it is looking at the target object.
- Reset
  Stops all motion and causes all objects to move mack to their home positions.

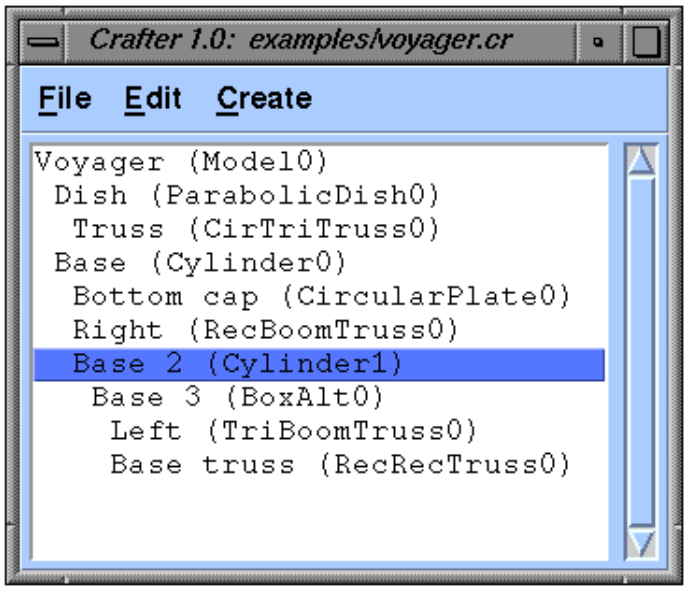
Camera window

The *camera window* is the most important Geomview window if you are using Crafter. The commands in the Tools panel can be invoked by typing their keyboard shortcuts in the camera window. The left and middle mouse buttons have different effects depending on the current motion mode. But no matter what mode, the right mouse button is used to select objects in the camera window. A double right click on an object makes that object the target object. Since when using Crafter the only object is `Model0`, double clicking is not very useful. However, a single right click over a part of the model tells Crafter to select that part.

See the Geomview manual for more information about its features.

3.4. Basic Interaction: The Model Editor

Crafter's main window, the Model Editor, displays the overall structure of the model and allows you to edit it.



3.4.1. Hierarchy list

The main part of the Model Editor is a scrolling list showing the model's hierarchy. The topmost entry in the list contains the name of the model. If it is a new model, this will be the only entry in the list. Otherwise, each entry contains the name of a piece of the structure followed by the type of piece it is. The indentation represents its place in the hierarchy. The best way to understand this is with an example. In the previous figure, `Voyager` is the name of the model. `Dish` (a parabolic dish) and `Base` (a cylinder) are at the top-most level of the model. `Truss` is attached to `Dish`. `Bottom cap`, `Right` and `Base 2` are attached to `Base`. And so on.

As you probably noticed, what's in parentheses after the name of the piece is not just the type, or *class*, of the piece, but also a number. Actually the thing in parentheses is the *address* of the piece. An address is a unique identifier for an object, assigned to the piece by Crafter, and may be different each time you run Crafter. It consists of the class name followed by a number with no spaces in between. The name of the piece, also called its *long name*, can be any string you want. It may contain spaces and does not need to be unique. The long name can be edited in the General Properties Panel described later.

3.4.1.1. Selection

Crafter allows you to *select* objects from this list for doing editing operations. To select a single object, click on its entry in the hierarchy list. To select multiple objects, drag the cursor over the region of the list you want to select. If the items you want to select are not contiguous in the list, press the *control* key along with the left mouse button to add to the selection without first clearing the existing selection.

Another and perhaps easier way to select an object is to click the right mouse button over the object in the Geomview window. The piece of the model that is directly under the cursor will be selected in the hierarchy list.

Note that when the cursor is over a properties panel (general or class-specific), the current selection automatically changes to the object represented by the panel.

3.4.2. Navigating in the hierarchy list

Navigating the hierarchy list is made easy with the arrow keys. The up and down arrow keys move the selection up and down the list. The left arrow moves the selection up in the list to the parent of the currently selected object. If more than one object is selected, then the topmost selected item is considered to be the currently selected item while using the arrow keys.

In the previous example, `Base 2 (Cylinder1)` is selected. Pressing the left arrow key moves the selection to `Base (Cylinder0)`

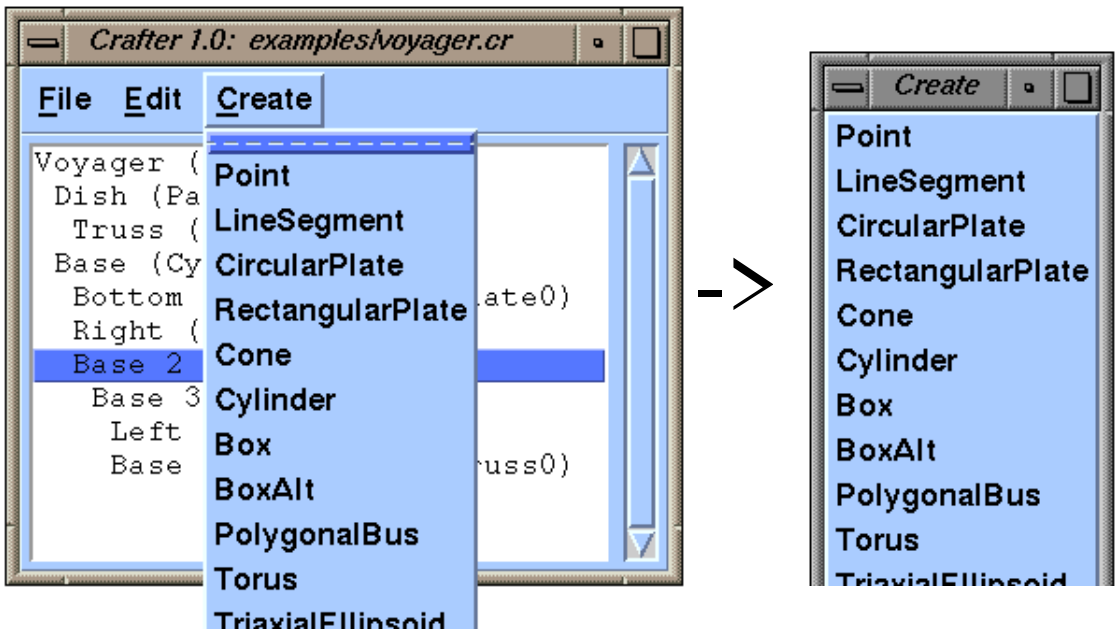
Pressing `<ENTER>` in the Model Editor window is like double-clicking on the current selection; it brings up the General Properties Window (described later) for the currently selected object(s).

3.4.3. File name

In the title bar of the Model Editor is the name of the current file being edited. If you save the model without specifying a new name, this is the name that will be used.

3.4.4. Menus

Across the top of the window is a menu bar with menus for file operations, edit operations and creating and attaching objects. All menus can be *torn off* and posted as separate windows. In a posted menu, the items are invoked simply by clicking on them. This is particularly useful for the **Create** menu, which allows you to create and add objects to the model. To tear off a menu, post the menu as you would normally, but instead of selecting a menu item, select the dashed line at the top of the menu. This creates a separate window of menu items that is available at all times.



3.5. Creating and attaching objects

The **Create** menu contains a palette of spacecraft pieces, or *primitives*, that may be attached to the model. It may be useful to tear off this menu if you are using it a lot.

To create and attach a new object, first you need to decide which existing object you want to attach it to, or if you want to make it a top-level object in the model. Select the parent object's entry from the hierarchy list in the Model Editor, or select the `(Model0)` entry if you want it to be at the top level of the model. Then from the Create menu, select the class of object you want to attach. A new object will appear in the model displayed in Geomview, a new entry will appear in the hierarchy list, and a window called the General Properties Panel will pop up. This panel controls properties such as scale, rotation, transformation and appearance.

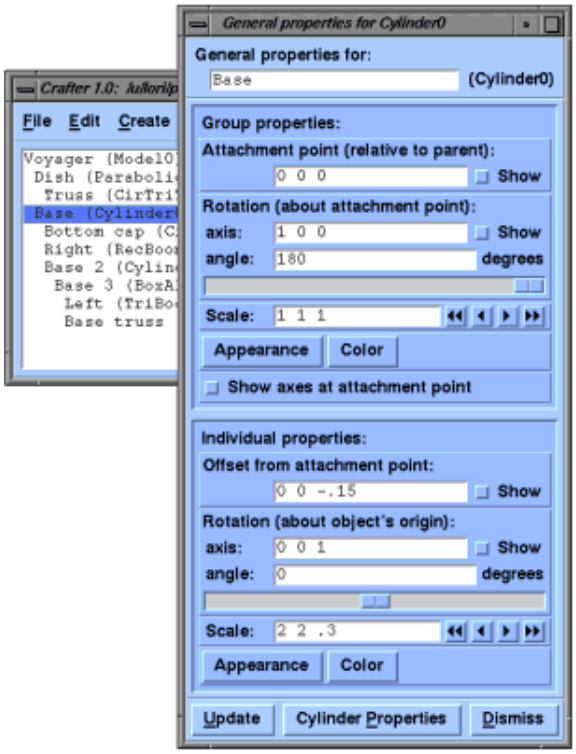
3.6. Editing an object's properties

Each object has several properties associated with it. Some of them are general properties and some are specific to the class. Separate panels are used for editing each type of property.

Note that when the cursor is over either of these panels (general or class-specific), the current selection in the hierarchy list automatically changes to the object represented by the panel.

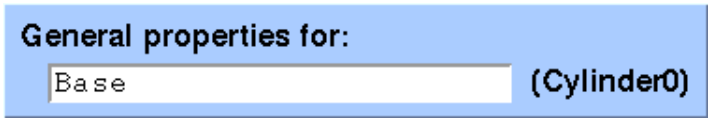
3.6.1. General properties

An object's general properties can be edited using the General Properties Panel, which you can get by either double-clicking on the object in the hierarchy list, or by selecting the object in the list and then selecting the **General Properties...** item from the **Edit** menu.



General properties include a name for object, appearance information, and transformations which specify the way and object is attached to its parent. at its parent's origin, with no rotation or scaling. By default, an object is attached to its parent with its own origin at its parent's origin, with no rotation or scaling.

3.6.1.1. Long name



Every piece of the model is assigned a unique name or address. This address may change each time you load the file, though. To make your pieces easier to identify, you may supply a *long name* to each piece. It can be any string you want. It may contain spaces and does not need to be unique. The long name can be edited at the top of the General Properties Panel.

3.6.1.2. Group properties vs. individual properties

You may specify transformation and appearance properties in two ways: (1) properties that apply to the group as a whole -- that is the object and all objects hierarchically below it -- or (2) properties that apply only to that object.

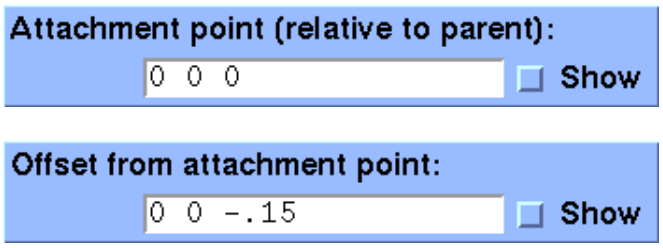
Appearance properties include attributes such as color, line width and shading. The relationship between group appearance and individual appearance is fairly straight forward. When an individual appearance property is given, it overrides any group definitions of that property for that object. If an appearance property is not defined at the individual level, then the group definition nearest up in the hierarchy is used. Overriding appearance properties is determined on a per-property basis. For example, changing the line width of an individual object will not reset all other properties at the individual level.

The difference between group transformations and individual transformations is a bit more complicated. Properties defined at the individual level do not cancel-out properties defined higher in the hierarchy. They are applied in addition to transformations at all other levels. When a transformation property is applied to a group, the whole group is treated as one piece; scaling, rotating and translating act on the group as a whole. Individual transformation properties are applied to the object as well as to the *attachment points* of the rest of the group. In other words, changing the scale or rotation of an individual object does not change the scale or rotation of the rest of the group, but may change the positions of the objects in the group.

3.6.1.3. Transformations

There are three types of transformations you can specify -- translation, scale and rotation. Coordinate vectors are used in specifying of each of them. The format for a vector is "`<x> <y> <z>`" -- three numbers separated by spaces, and no punctuation.

Translation

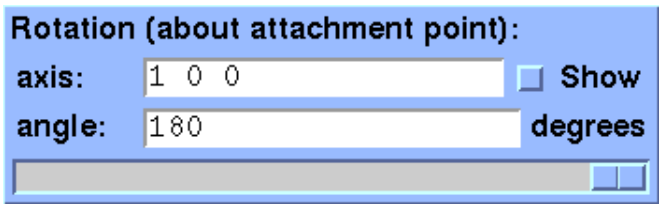


There are two fields that affect translation. The attachment point relative to the parent, and the offset from this attachment point.

The attachment point is in the coordinate system of the parent. The offset point is in the coordinate system of the child. By default, both of these are "`0 0 0`".

To display either of these vectors in the model, click in the checkbox marked **Show** to the right of the text field.

Rotation



There are two parts to specifying a rotation, the axis of rotation and the angle of rotation. The angle of rotation is specified in degrees and may be typed into the **angle** text field by hand or may be changed using the slider below it. Using the slider immediately updates the Geomview display. The axis is commonly either the X, Y or Z axis, but may be any arbitrary vector. The vector must be typed in by hand to the **axis** text field.

To display either of the rotation vectors in the model, click in the checkbox marked **Show** to the right of the text field.

Scale

Scales are specified by vectors as well. The x, y and z portions of the vector give the scale factors for the X, Y and Z dimensions, respectively. For uniform scaling, all three numbers should be the same.

A short cut for changing the scale uniformly is provided with step buttons to the right of the text field. Clicking on these buttons causes the scale to increment or decrement proportionally, by either a little step or a big step. The four buttons from left to right are: decrement big step, decrement little step, increment little step, and increment big step.

Notes about applying transformations:

- When applying the transformations, first the individual transformations are applied and then the group transformations, then the parent's group transformations, then the grandparents' group transformation, and so on.
- Within a set of transformations (e.g. the individual transformations), first the scale is applied, then the rotation, then the translation.
- For any group transformation, though, before the group translation (i.e. determining the attachment point) is applied, the parent's individual transformation is applied to the attachment point.

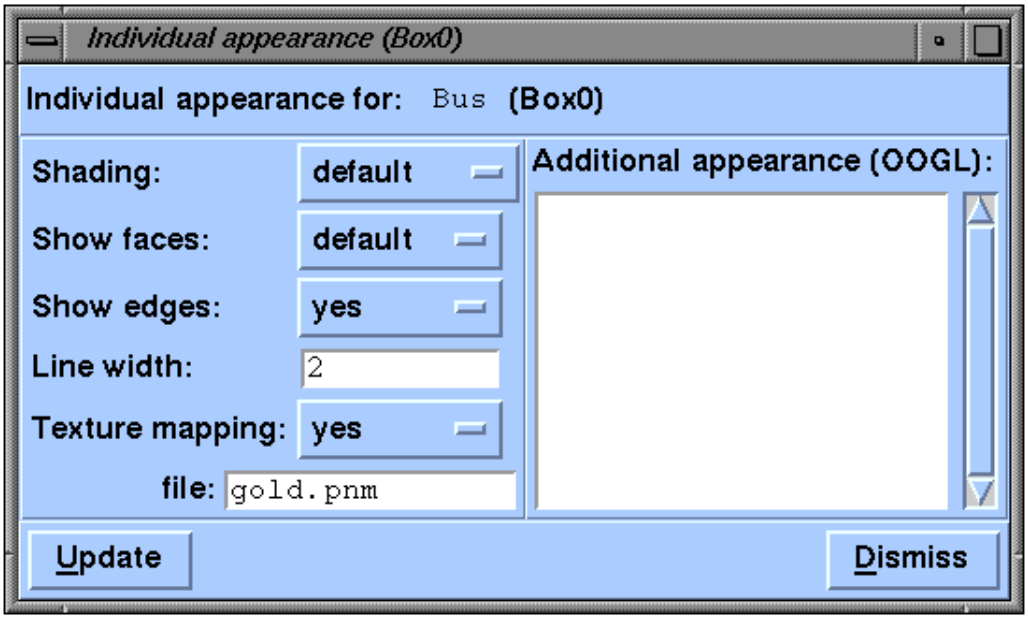
See the section on transformation properties for some examples.

3.6.1.4. Coordinate axes

Because Geomview allows you to change the orientation of the model in the camera window, sometimes its difficult to remember which direction is which in the display. To aid with this, you can display coordinate axes of unit length at the attachment point in the display. The axes show the coordinate system of the group. To display the coordinate axes, click the **Show axes at attachment point** checkbox.

3.6.1.5. Appearance

Crafter allows you to modify the appearance of objects in the model. Appearance properties include: shading, showing faces, showing edges, line width, texture mapping, face color and edge color. Other Geomview appearance attributes such as material properties may be set as well. To change the face color or edge color, click on the **Color** button or edit the face color directly in the text field to the right of the color button. To change any other appearance property, click on the **Appearance** button. Make sure you choose which type of properties you want to change and click on the appropriate button in either the **Group properties** section or the **Individual properties** section. Separate windows are used for color and appearance properties.



In the appearance editor shading and showing faces and edges are controlled by toggle menus. Line width is controlled by a text field. Texture mapping is controlled by two properties. First a toggle menu for whether or not to apply texture mapping, and second a text field for the name of the image file to use. The image format type must be indicated by the last few characters of the file name. Supported image types are `.ppm (or .ppm.Z or .ppm.gz)`, `.pgm (or .pgm.Z or .pgm.gz)`, `.tiff`, `.gif` and `.sgi`. For this feature to work, some programs must be available in geomview's search path:

- `zcat` for `.Z` files
- `gzip` for `.gz` files
- `tifftopnm` for `.tiff` files
- `giftoppm` for `.gif` files

Other arbitrary appearance attributes may be specified by writing the appropriate OOGL (Geomview's object language) in the larger text window on the right. For instance, if you want make an object to be shinier, you might type the following in the **Additional appearance** text window:

```
*material {
  *shininess 100
  *ks 0.8
}
```

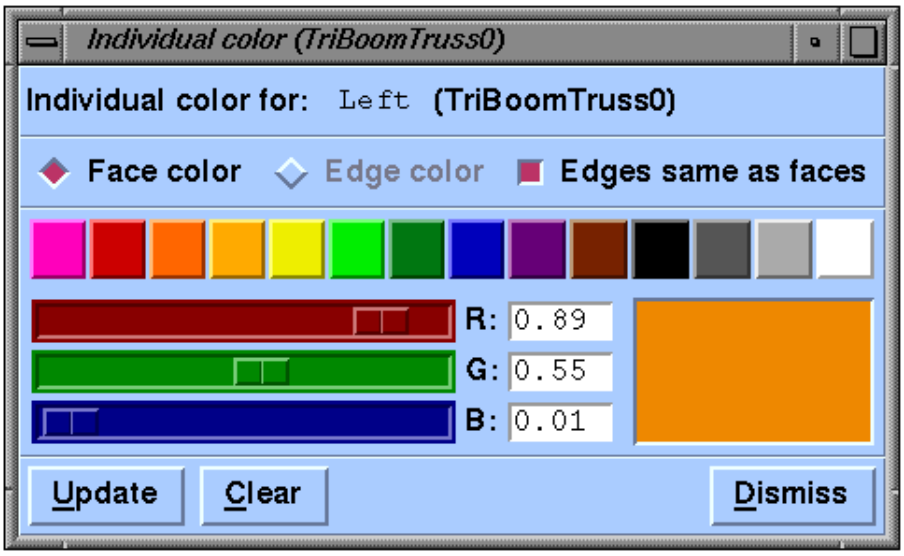
See the Geomview manual for more information about specifying appearances.

When you are done editing the appearance, you must click the **Update** button to see the changes in Geomview. (Since carriage returns are allowed in the text area, they do not cause the display to be updated as they do in single line text fields.)

Warning: The syntax of the appearance is not checked before sending it to Geomview. If there is an error, Geomview will send information about it to standard error (stderr) and the pipe between Crafter Geomview will be broken. You may save your work in Crafter, but need to restart Crafter to send updates to Geomview again.

By default, the toggle menus are set to **default** (no surprise) and the line width field is empty. This means that nothing is said about these properties for this object. In this case the object will inherit the properties though the hierarchy. If a property such as shading is defined both in the toggle menu on the left and the **Additional appearance** text window on the right, then the definition in the text window overrides the setting of the toggle window.

3.6.1.5.1. Color



Since color is such a common appearance property to change, it is given its own window. You can also edit it directly in the General Properties panel. The simplest way to choose a color is to select one of the existing colors in the palette -- click on one of the colorful buttons across the middle of the window. The color change takes effect immediately in the Geomview window. If none of these colors is what you want, pick one that is close and then edit it with the red, green and blue sliders at the middle-left. These control the red, green and blue components of the color. Alternatively you can type in these three values directly in the text fields immediately to the right of the respective sliders. A sample of the color you are making is shown in the rectangular trough at the right.

By default, faces and edges are colored the same. If you want them colored separately, click in the checkbox labeled **Edges same as faces** to deselect this option. Now you may use the radio buttons at the top of the window to alternate between **Face color** and **Edge color**.

3.6.1.6. Buttons on the General Properties panel

There are three buttons at the bottom of the General Properties Panel:

Update
: Pressing this button updates the Geomview display for all properties in the panel. Pressing <ENTER> in a text field updates Geomview just for that field.

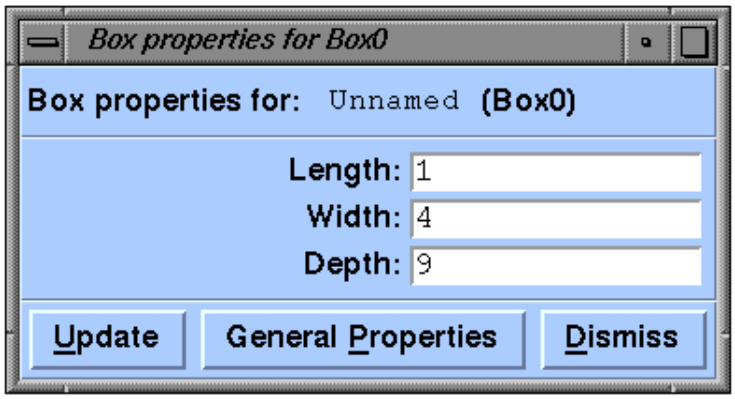
Class-specific Properties
: This button's label depends on the class. For example, if the object is a Box, then the button will be labeled "Box Properties." Pressing this button pops up a window for editing the object's class-specific properties.

Dismiss
: The dismiss button closes the window keeping any changes you made to the properties.

3.6.2. Class-specific properties

Each type of object has its own properties. For example, a box has length, width, and depth, and a parabolic dish has diameter and ratio (of focal-length to diameter). These properties are called *class-specific properties*. To edit these properties, select the object in the hierarchy list in the Model Editor, and then select **Class-specific Properties...** from the **Edit** menu, or if the object's General Properties Panel is already up, then you may click on the **Class-specific Properties** button at the bottom of that panel.

Here is an example of a Box's Class-specific Properties Panel (or Box Properties Panel):



At the top of the panel is the name and class of the object. Depending on the class, the panel may have text entry fields, checkboxes, and other fields. Pressing <ENTER> in any text field updates all class-specific properties. Clicking in a checkbox, or moving a slider, also updates all class-specific properties for the object.

There are three buttons at the bottom of the Class-specific Properties Panel:

Update
Pressing the update button updates the Geomview display for all class-specific properties in the panel. Pressing <ENTER> in a text field also updates all properties.

General Properties
Pressing this button pops up the object's General Properties Panel.

Dismiss
The dismiss button closes the window keeping any changes you made to the properties.

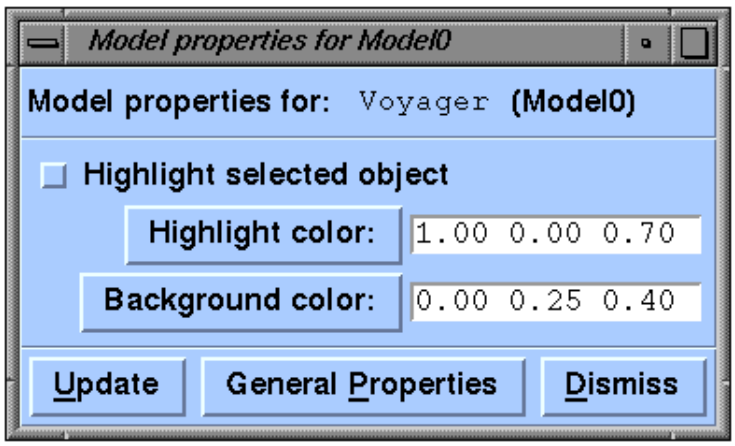
3.6.3. Model properties

Not just primitives have properties associated with them, the model does, too. In fact, a model is just like a primitive, but doesn't have its own geometry associated with it and it's always at the top level of the hierarchy.

You can change the overall scale, rotation and translation of everything in the model by changing the group properties of the model. You can transform just the attachment points of the top level objects by changing the individual properties of the model. Changing the model's group appearance is like setting a default appearance for all objects. Setting the model's individual appearance has no effect.

3.6.4. Options window

The Options Window allows you to specify options that apply to the whole model. These options are just the class-specific properties of a model (i.e. model properties). To invoke this window, select **Options...** from the **Edit** menu. If the model's General Properties Panel is already up, then you may click on the **Model Properties** button at the bottom of that panel to get the Options Window.



The options controlled in this window are:

Highlight selected object
: highlight the currently selected items in the model hierarchy.

Highlight color
: the color used to highlight the selected object(s). Changing this color is like changing a color in the General Properties Panel. Type in three real numbers between 0.0 and 1.0 representing the red, green and blue components of the color, or use the color editor by clicking on the **Highlight color** button. The red, green and blue components may be adjusted by moving the sliders. The color is automatically updated in the text field and in the Geomview display.

Background color
: the color in the background of the Geomview camera window. This is the only option on this panel that is saved with the file. Editing the background color is just like editing the highlight color; you may click on the button to get the color editor or enter the color vector directly.

Note that the background color is the only option saved with the file.

3.6.5. The Null primitive

The null primitive is a primitive that has no geometry of its own. It is useful for creating groups of objects that don't have a master object that they are attached to. Editing its properties are like editing the model's properties. Group transformations act on the whole group. Individual transformations act only on the attachment points. Group appearance effects the whole group. And individual appearance means nothing since the object has no geometry of its own.

3.7. Editing the hierarchical structure

Groups of objects may be cut and pasted using the items in the **Edit** menu. First select an object or multiple objects in the hierarchy list in the Model Editor. Then invoke the appropriate operation from the **Edit** menu. Multiple objects may be selected for cutting, copying, duplicating or deleting, but only one object may be selected for pasting into. When cutting, copying, duplicating and deleting, the selected object and all objects below it in the hierarchy are also affected. For example, deleting an object also deletes everything that is attached to it hierarchically. A summary of the edit operations follows.

Cut
: Remove the currently selected group(s) from the model and keep it in a clipboard for pasting later.

Copy
: Copy the currently selected group(s) into the clipboard without removing it from the model.

Paste
: Attach a copy of what's in the clipboard to the selected object.

Duplicate
: Copy the currently selected group and paste it in the parent of the currently selected object, without changing the contents of the clipboard.

Delete
: Remove the currently selected group from the model without changing the contents of the clipboard.

Paste from file...
: Paste the contents of a Crafter file into the currently selected object. A window is popped up allowing you to specify the file name of a crafter file. The top-level objects in the file are attached to the currently selected object. The model properties are ignored. This operation has no effect on the clipboard.

3.8. Reading and writing files

Basic commands for reading and writing crafter models are provided in the **File** menu along with some short cuts for common file operations. If a menu item has a "..." at the end of its label (e.g. "Open"), then a file chooser like the following is popped for you to specify a filename.

By default, only files ending with a `.cr` suffix and directories are shown in the scrolling list. To change this filter, enter a name with wildcards (e.g. `*.jpl` or `*`) and click the **Filter** button.

To specify a file name, double-click on one in the list or enter the name directly into the text field at the bottom. A single-click on a file in the list puts the file name in the text field, where you may edit it if you like and press `<ENTER>` or press the **Open** or **Save** button when you're done.

The **Delete** and **Make Dir** buttons affect the files on the Unix file system. **Delete** removes the specified file and **Make Dir** makes a directory with the specified name. The toggle menus at the top let you go back to recent directories you've been browsing.

New
: Delete everything in the current model; create an empty, unnamed model.

Open...
: Replace the current model with the model in a specified file.

Save
: Save the model in the current file.

Save As...
: Save the model in a specified file.

Save Geometry...
: Save the geometry of the model (in Geomview's OOGL format) to a specified file.

Revert
Replace the current model with the most recently saved version of the model.

Print...
Save the current camera view as a PostScript file.

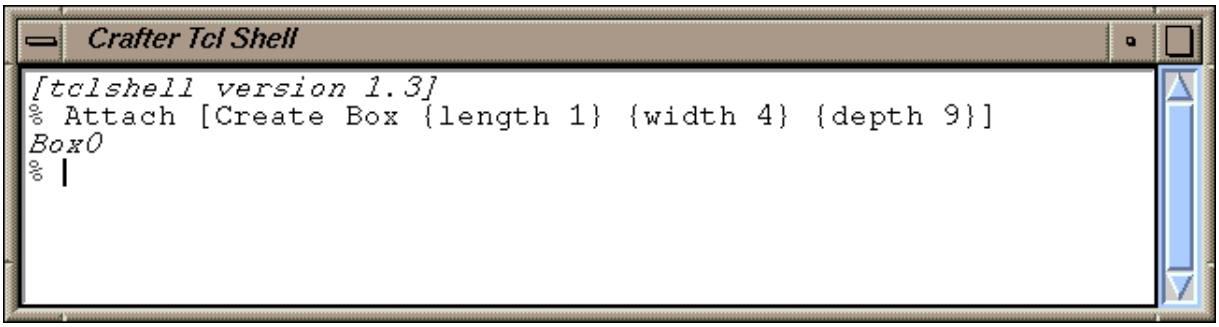
3.9. Feeding commands to Crafter

Crafter is written in Tcl/Tk, which is an interpreted command language for controlling applications and developing graphical user interfaces. Since it is interpreted, any operation that can be done through the user interface, can also be done with a Tcl command.

There are two ways in which you can feed Tcl commands to Crafter: (1) by typing commands into a Tcl shell window, and (2) by sending commands through a pipe.

Each of these operations can be invoked from the **File** menu.

Tcl Shell...
Open a window for typing in Tcl commands. The window allows you to type commands to modify the model. Type a command at the prompt and the result of the command will be displayed in italics in the window.



Open Pipe...
Create a named pipe for receiving Tcl commands. By default the pipe is put in `/tmp/geomview`. Commands may then be sent to Crafter by sending them to the named pipe. The file browser is popped up for you to enter a pipe name.

3.10. Quitting Crafter

To quit Crafter and Geomview, select the **Quit** item from Crafter's **File** menu. Alternatively, you can quit both programs from Geomview by clicking the **Quit** button in the upper-right corner of Geomview's main window or by pressing the <ESC> button in Geomview's camera window.

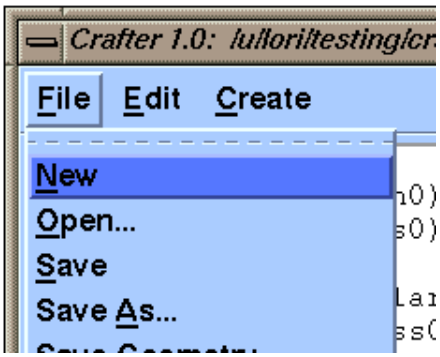
If you don't want to quit Geomview, then kill Crafter by going to the list of external modules in Geomview's main window and clicking on Crafter's red instance entry. This entry begins with a number in brackets. For example, the middle of the three lines below is the instance entry:

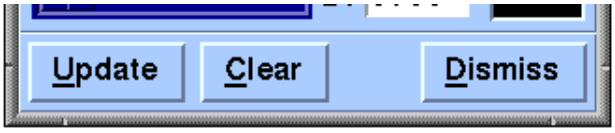
```
Crafter
[1]Crafter "voyager.cr"
Crafter "voyager.cr"
```

You may restart Crafter by clicking on one of the entries in the list of modules. One or two entries are provided -- one with no arguments and with the arguments you specified on the command line if any.

3.11. Keyboard Shortcuts

Menu items and buttons can be invoked not only with the mouse, but also with keyboard shortcuts. Each menu in the Model Editor and many of the buttons in other panels have one letter in the label underlined. This is the called a *mnemonic*.





Menu items

To invoke a menu item with the keyboard, first post the menu by typing the **<ALT>** key plus the mnemonic character while the mouse is the Model Editor. Once a menu is posted, pressing the mnemonic of a menu item invokes that particular item. Alternatively, the up and down arrow keys may be used in a posted menu to select a menu item. Then press the **<ENTER>** key to invoke it.

For example, to post the **File** menu, type **<ALT> + <f>**. Then, to choose the **Save** menu item, either press the **<s>** key or press the **<DOWN>** arrow two times and then press **<ENTER>**.

Buttons

Invoking a button using the keyboard is much more straight forward than invoking menu items. With the cursor in the window containing the button you want to invoke, simply type the **<ALT>** key plus the mnemonic character. For example, to invoke the **Dismiss** button on any of the panels, type **<ALT> + <d>**.

Text fields

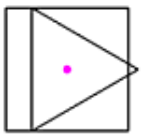
When typing into a text field, the following operations are allowed using special characters:

Control-a	Moves insertion cursor to beginning of entry.
Control-e	Moves insertion cursor to end of entry.
Control-/	Selects all the text in the entry.
Control-\	Clears any selection in the entry.
Control-d	Deletes the character to the right of the insertion cursor.
Meta-d	Deletes the word to the right of the insertion cursor.
Control-w	Deletes the word to the left of the insertion cursor.
Control-k	Deletes all characters to the right of the insertion cursor.
Control-t	Reverses the order of the two characters to the right of the insertion cursor.
Control-r	Resets the entry to its default value.

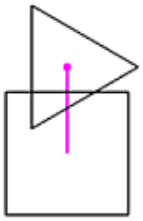
4. More on Transformations

Examples of changing transformations

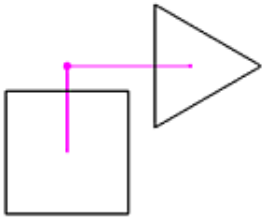
Start out with a triangle attached to a square. By default the center of the triangle is attached to the center of the square.



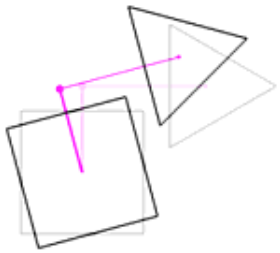
Changing the attachment point to (0, 0.7, 0) moves the triangle up.



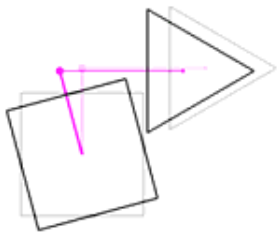
Changing the triangle's offset from the attachment point to (1, 0, 0) moves the triangle to the right.



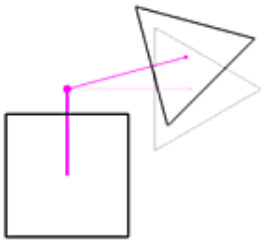
Now we can see what happens when we alter the other transformation properties. If we change the group rotation of the square, we see that everything moves as one piece.

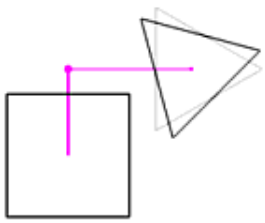


If instead we change just the individual rotation of the square, we see that the triangle's attachment point rotates along with the square, but the triangle itself only translates with the attachment point; it does not rotate.



Changing the triangle's group rotation *(shown below on the left)* acts about the attachment point. Changing its individual rotation *(shown below on the right)* acts about its own origin.





Changing the scale is similar to changing the rotation. Changing the box's group scale *(shown below on the left)* scales everything as one piece. Changing the box's individual scale *(shown below on the right)* scales the box and the attachment points, but does not change the scale of the triangle.

Changing the triangle's group scale *(shown below on the left)* is centered at the attachment point. Changing its individual scale *(shown below on the right)* is centered at its own origin.

5. Command Language

Crafter Spacecraft Design Program

Crafter is implemented in the Tcl/Tk environment. Tcl is a simple scripting language for controlling applications. Tcl ("tool command language") has generic programming constructs such as variables, loops and procedures, and may also be extended to include more commands. The Tk toolkit is an extension to Tcl that provides easy development of graphical user interfaces under the X Window System.

Crafter is written entirely in Tcl/Tk and uses the windowing shell called `wish`. Manipulating Crafter's user interface, such as clicking on buttons or selecting menu items, invokes Tcl commands. For instance, selecting a primitive the **Create** menu, causes Crafter's `Create` and `Attach` commands to be invoked.

Commands can also be invoked by typing them in directly to a Tcl shell window or sending them to Crafter through a pipe. To write commands for Crafter, you should understand the syntax of Tcl and some of the basic Tcl commands, then learn the Crafter-specific commands. You do not need to know any of the Tk commands.

5.1. Brief introduction to Tcl

5.1.1. Command syntax

Tcl is an interpreted command language that is very similar in style to a shell command language but has some characteristics of *C* and *LISP* languages as well. Like a shell command, a Tcl *command* consists of one or more *words* separated by spaces or tabs. For example, the command

```
expr 10 + 1
```

consists of four words (the command name, "`expr`", plus three arguments "`10`", "`+`", and "`1`").

A Tcl *script* consists of one or more commands separated by semicolons (`;`) or newlines. Here is an example of a script.

```
set a Hello; set b world
expr 10 + 1
expr 100 * cos(1.2)
lindex {a b {c d} e} 2
```

Expression syntax in Tcl is modeled after that of *ANSI C*. Tcl also provides list processing commands similar to those in *LISP*. In Tcl, lists and sublists are delimited by braces ("`{`" and "`}`").

5.1.2. Variables and substitution

Tcl allows storage of information in variables similar to the way that shell does. A variable name can be any string, but usually is something starting with a letter and sometimes contains numbers. The value of a variable is also a string. The way a value is treated (as an integer, floating point, character, etc.) depends on the command that uses it.

To set the value of a variable, use the `set` command. Its syntax is "`set` *`varName value`*". To retrieve the value of a variable, you can use the dollar (`$`) operator before the variable name. For example, in the following script,

```
set a 10
expr $a + 1
```

the value of `a` is set to `10` in the first command. In the second command, the value of the variable `a` is substituted before calling the `expr` command. This is called *variable substitution*. The `expr` command gets the arguments "`10 + 1`" instead of "`$a + 1`". The return value of the command is `11`.

Of course, commands would not be very useful unless we could use the return value somehow. That's what *command substitution* is for. Enclosing a command or script in brackets ("`[`" and "`]`") causes the command to be executed as a separate script and the result of the script is substituted in place of the bracketed command. In the following example,

```
set a 10
set b [expr $a + 1]
```

after variable and command substitution, the second command would be:

```
set b 11
```

This is similar to command substitution in the shell using backquotes (e.g. `` `ls` ``).

5.1.3. Quotes and braces

Words enclosed in quotes or braces are treated as one word. The difference between quotes and braces is that substitution is not performed on words in braces. In the following example,

```
set a 10
set msg1 "$a squared is [expr $a*$a]."
set msg2 {$a squared is [expr $a*$a].}
```

msg1 is set to the string '`10 squared is 100.`' and msg2 is set to the string '`$a squared is [expr $a*$a].`'

5.1.4. Control flow

Tcl supports common control flow operations, such as *if* and *while*. These constructs do not require any special syntax of the Tcl language, but are simply commands like any other that take advantage of the fact that arguments enclosed in braces are not evaluated. The command can then determine when or whether to evaluate the arguments. For example, the syntax of the while command is `while` *`condition script`*. Consider the following example,

```
set i 0
while {$i < 10} {
    puts $i
    set i [expr $i + 1]
}
```

The while command first evaluates the condition (the first argument) as an expression. If the result is non-zero, then the script is evaluated. When the script is done, the expression is evaluated again, and so on.

Because the control flow operations are just commands like any other, at least the opening brace of the script must be on the line with the rest of the command. In other words, the following is incorrect syntax:

```
while {$i < 10}
{
    puts $i
    set i [expr $i + 1]
}
```

More examples of control flow commands follow.

```
if {$i > 0} {
    puts "$i is greater than 0."
} else {
    puts "$i is not greater than 0."
}

for {set i 0} {$i < 10} {incr i} {
    puts $i
}

foreach color {red orange yellow green blue indigo violet} {
    puts "My favorite color is $color."
}
```

5.1.5. Comments

If the first non-blank character of a command is a pound sign (`#`), then the pound sign and all characters following it until the end of the line are treated as a comment and discarded. If you want to put a comment on the same line as a command, you must separate the command from the comment with a semicolon, otherwise the comment will be considered part of the command.

```
# This is a comment.
set a Hello   ;# This is also a comment.
set b world   # This is sending extra arguments to the set command.
```

For more information about Tcl, see *Tcl and the Tk Toolkit* by John Ousterhout or see the Tcl/Tk online manual pages (man pages).

5.2. Crafter-specific commands

Knowing the general syntax of Tcl commands will help you write scripts that use the Crafter-specific commands for creating, attaching and modifying primitives.

5.2.1. Conventions

addresses
: Existing objects are referenced by an *address*, which is a unique identifier for the object. It is the value returned by the `Create` command when the object is first created.

properties
: When specifying properties and values to a command, name-value pairs are used. For example, `{color {.5 0 0}} {length 3}` specifies that the `color` should be "`.5 0 0`" and the `length` should be "`3`".

5.2.2. Manipulating objects

The main commands for manipulating objects are:

`Create` *`class prop1`* `[`*`prop2`*`...]`
: creates a new object of type *class* and returns its address.

`Delete` *`addr`* deletes the object specified by the address.

`Attach [-p` *`parent`*`]` *`addr1`* `[`*`addr2`*`...]`
: attaches the object(s) specified by address to the *parent* if specified, otherwise to the top-level model.

`Detach` *`addr1`* `[`*`addr2`*`...]`
: detaches the object(s) specified by address from their parents.

`SetProps` *`addr prop1`* `[`*`prop2`*`...]`
: sets the properties of the specified object and updates the Geomview display.

5.2.3. File manipulation

Simple commands for reading and writing files are provided.

`Open` *`filename`*
: loads the model in *filename*, discarding the existing model if one exists.

`Save` *`filename`*
: saves the current model in *filename*

`Quit`
: closes the Crafter program.

5.2.4. Examples

So, to create and attach a new Box, for example, you could type the following:

```
set b [Create Box {length 1} {width 4} {height 9}]
Attach $b
```

To attach a small sphere to the box, you could then type

```
Attach -p $b [Create Sphere {diam .5} {pos {0 .5 4.5}}]
```

The `pos` field controls the sphere's attachment point to the box.

If you now want to change the width of the box, you can do so using the `SetProps` command:

```
SetProps $b {width 3}
```

6. Appendix

6.1. Menu items

File menu

New
Delete everything in the current model; create an empty, unnamed model.

Open...
Replace the current model with the model in a specified file.

Save
Save the model in the current file.

Save As...
Save the model in a specified file.

Save Geometry...
Save the geometry of the model (in Geomview's OOGL format) to a specified file.

Revert
Replace the current model with the most recently saved version of the model.

Print...
Save the current camera view as a PostScript file.

Tcl Shell...
open a window for typing in Tcl commands. The window allows you to type commands to modify the model. Type a command at the prompt and the result of the command will be displayed in italics in the window.

Open Pipe...
create a named pipe for receiving Tcl commands. By default the pipe is put in `/tmp/geomview`. Commands may then be sent to Crafter by sending them to the named pipe.

About
Display version and copyright information.

Quit
Close Crafter without quitting Geomview.

Edit menu

Cut
Remove the currently selected group from the model, and keep it in a clipboard for pasting later.

Copy
Copy the currently selected group into the clipboard without removing it from the model.

Paste
Attach a copy of what's in the clipboard to the selected object.

Duplicate
Copy the currently selected group and paste it in the parent of the currently selected object, without changing the contents of the clipboard.

Delete
Remove the currently selected group from the model without changing the contents of the clipboard.

Paste from file...
Paste the contents of a Crafter file into the currently selected object. A window is popped up allowing you to specify the file name of a crafter file. The toplevel objects in the file are attached to the currently selected object. The model properties are ignored. This operation has no effect on the clipboard.

General Properties...
Open the General Properties Panel for the selected object.

Class-specific Properties...
Open the Class-specific Properties Panel for the selected object.

Options...
Open the Options panel (Equivalent to model properties)

Create menu

[see list of primitives]

6.2. Primitives and properties

General properties

These properties apply to all primitives. The names of the properties along with their default values and a short description are shown below. *Note that figures do not necessarily represent the default values of the properties.*

`longname`	`["Unnamed"]`	-- friendly name for the object
`iappear`	`[""]`	-- individual appearance
`iscale`	`["1 1 1"]`	-- individual scale (vector)
`irotv`	`["0 0 1"]`	-- individual rotation axis (vector)
`irota`	`[0]`	-- individual rotation angle (degrees)
`ipos`	`["0 0 0"]`	-- individual position relative to attachment point (vector)
`appear`	`[""]`	-- appearance of this and attachments
`scale`	`["1 1 1"]`	-- scale of this and attachments (vector)
`rotv`	`["0 0 0"]`	-- rotation axis of this and attachments (vector)
`rota`	`[0]`	-- rotation angle of this and attachments (degrees)
`pos`	`["0 0 0"]`	-- position of attachment point (vector)
`icolor`	`[""]`	-- individual color.
`iecolor`	`[""]`	-- individual appearance -- edge color.
`iscolor`	`[1]`	-- individual appearance -- same color?
`ishade`	`["default"]`	-- individual appearance -- shading.
`iedges`	`["default"]`	-- individual appearance -- draw edges?
`ifaces`	`["default"]`	-- individual appearance -- draw faces?
`iewidth`	`[""]`	-- individual appearance -- edge width.
`color`	`[""]`	-- color of this and attachments.
`ecolor`	`[""]`	-- group appearance -- edge color.
`scolor`	`[1]`	-- group appearance -- same color?
`shade`	`["default"]`	-- group appearance -- shading.
`edges`	`["default"]`	-- group appearance -- draw edges?
`faces`	`["default"]`	-- group appearance -- draw faces?
`ewidth`	`[""]`	-- group appearance -- edge width.

Common properties

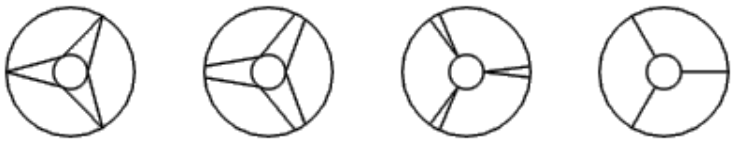
These properties appear in many primitives.

`angle` **`[60]`** **-- arc in XY plane between struts**

This property is used in trusses where there is a circle at one end and the struts of the truss may be separated at that end. The angle is *half* the measure of the arc (in degrees) that the ends of the struts make in the plane. Possible values for this property range from 0 to 60 degrees.

Below are examples of a CirCirTruss viewed from above with edges colored black, shown with varying values for the angle.

at 60 degrees at 55 degrees at 5 degrees at 0 degrees

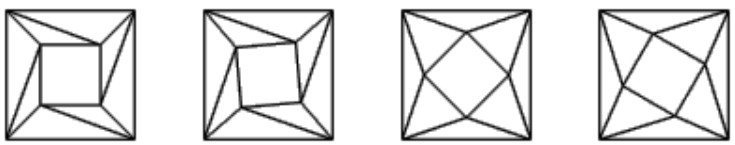


`angle` **`[0]`** **-- angle of twist between ends**

This property is used in trusses where both ends are the same shape and one is rotated in relation to the other. For rectangular ends, values of this angle range from 0 to 90 degrees. For triangular ends, values range from 0 to 120 degrees.

Below are examples of a RecRecTruss viewed from above with edges colored black, shown with varying values for the angle.

at 0 degrees at 5 degrees at 45 degrees at 60 degrees

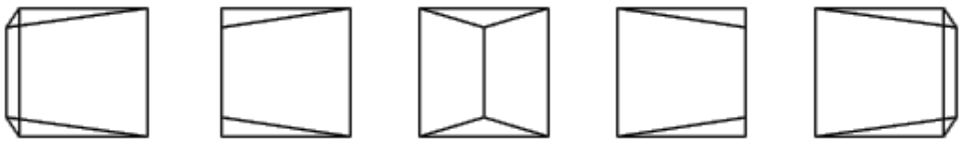


`fraction [.5 or .3]` -- location of the line relative to the width of the other end

This property is used in trusses where one end is a line. It determines where to place the line in relation to the other end. It represents a fraction of the bottom width. There are not limits on what the value of the fraction may be. A value of "0" means directly over the left side; a value of "1" means directly over the right side.

The following examples show varying values of the fraction for a LinRecTruss viewed from above with edges colored black.

fraction = -0.1 fraction = 0.0 fraction = 0.5 fraction = 1.0 fraction = 1.1

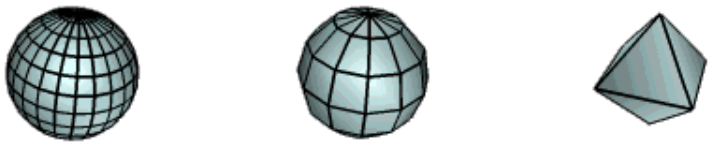


`res [24]` -- resolution of circle

This property is used on objects that have some curvature that is smoothly shaded. Cylinder, Sphere and Torus are some examples where this property is used. By default a circle is drawn with 24 segments. Changing this value changes how many segments make up a 360 degree circle. This is useful for saving drawing time if higher resolution is not needed.

The following examples show what a Sphere looks at different resolutions.

res = 24 res = 12 res = 4

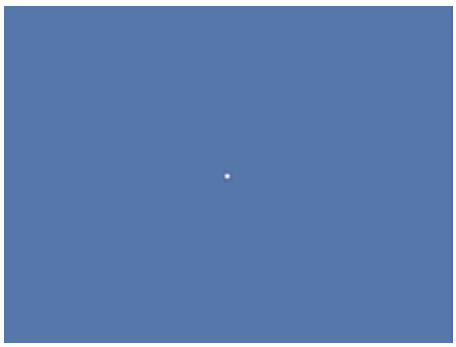


List of primitives

Null primitive

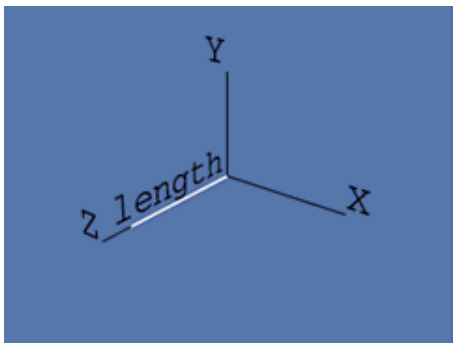
(no class-specific properties)

Point



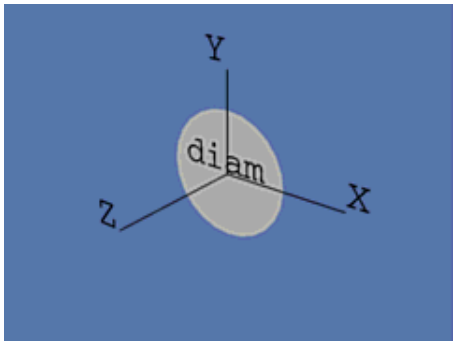
(no class-specific properties)

Line Segment



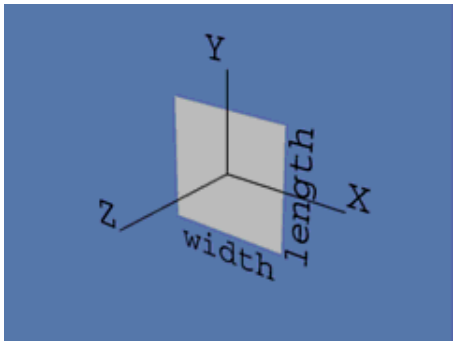
`length` `[1]` -- length of line segment in Z direction

Circular Plate



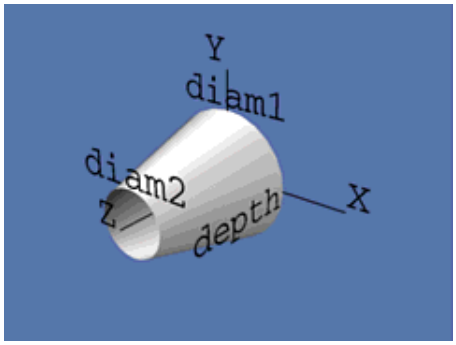
`diam` `[1]` -- diameter of circle

Rectangular Plate



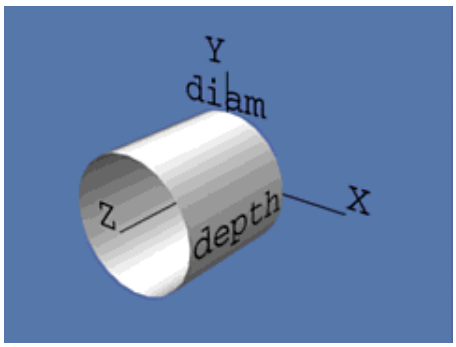
`length` `[1]` -- length of rectangle in Y direction
`width` `[1]` -- width of rectangle in X direction

Cone



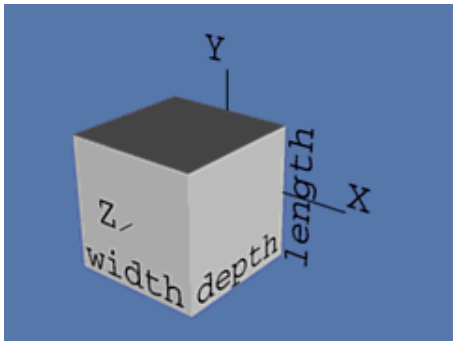
`diam1`	`[1]`	-- diameter at first end (at z = 0)
`diam2`	`[0]`	-- diameter at second end (at z = depth)
`depth`	`[1]`	-- depth of cone in Z direction
`capped`	`[0]`	-- whether the ends of the cone have caps or not ("1" or "0")
`res`	`[24]`	-- resolution of circle (see common properties)

Cylinder



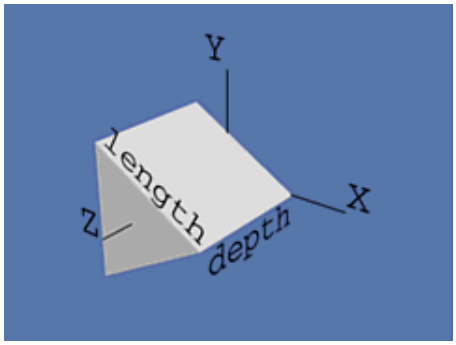
`diam`	`[1]`	-- diameter of cylinder
`depth`	`[1]`	-- depth of cylinder in Z direction
`capped`	`[0]`	-- whether the ends of the cylinder have caps or not ("1" or "0")
`res`	`[24]`	-- resolution of circle (see common properties)

Box



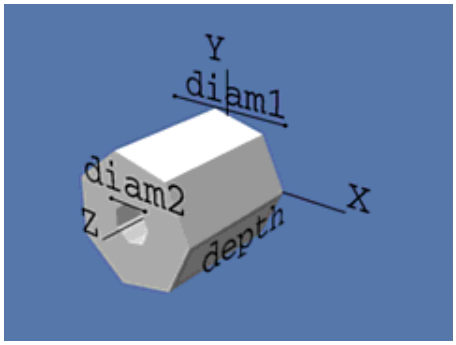

```
length      [1]     -- length of box in Y direction
width       [1]     -- width of box in X direction
depth       [1]     -- depth of box in Z direction
```

Box Alt



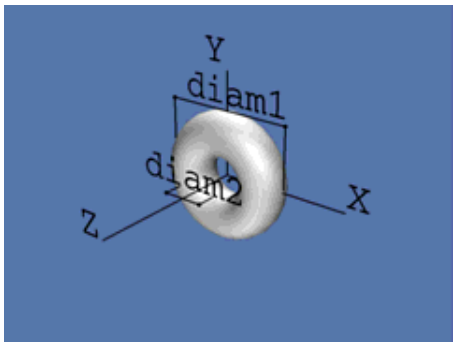

```
number      [3]     -- number of sides around the box
length      [1]     -- length of each side
depth       [1]     -- depth of box in Z direction
```

Polygonal Bus



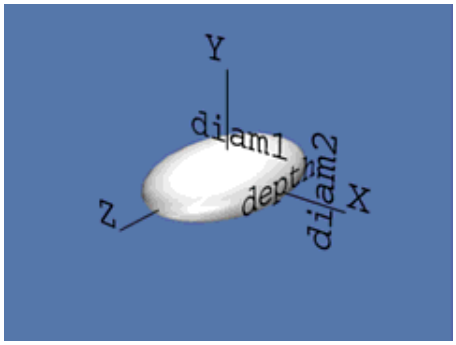
number	[3]	-- number of sides around the bus
diam1	[1]	-- outer diameter (twice distance from center axis to vertex)
diam2	[.3]	-- inner diameter (twice distance from center axis to vertex)
depth	[1]	-- depth of polygonal bus in Z direction

Torus



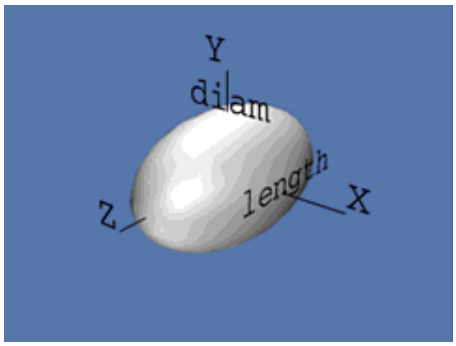
diam1	[1]	-- outer diameter
diam2	[.3]	-- inner diameter
res	[24]	-- resolution of circle (see common properties)

Triaxial Ellipsoid



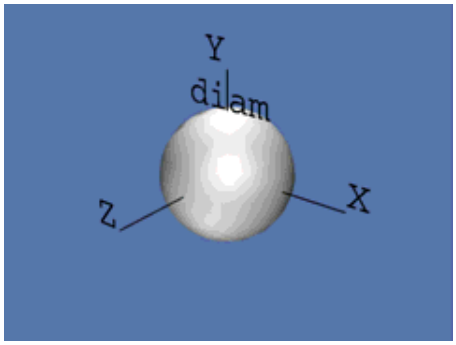
`diam1`	`[1]`	-- diameter in X direction
`diam2`	`[1]`	-- diameter in Y direction
`depth`	`[1]`	-- depth of ellipsoid in Z direction
`res`	`[24]`	-- resolution of circle (see common properties)

Ellipsoid



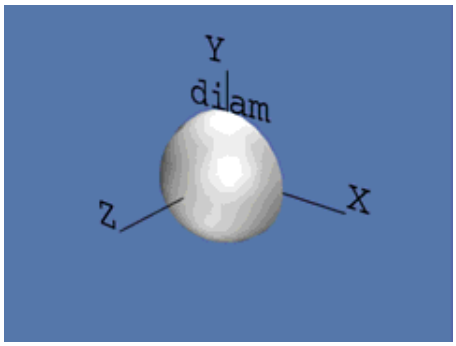
`diam`	`[1]`	-- diameter in XY plane
`depth`	`[1]`	-- depth of ellipsoid in Z direction
`res`	`[24]`	-- resolution of circle (see common properties)

Sphere



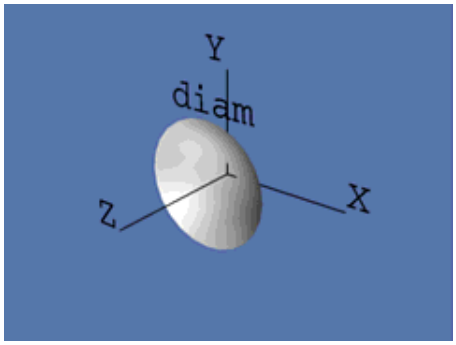
`diam`	`[1]`	-- diameter of sphere
`res`	`[24]`	-- resolution of circle (see common properties)

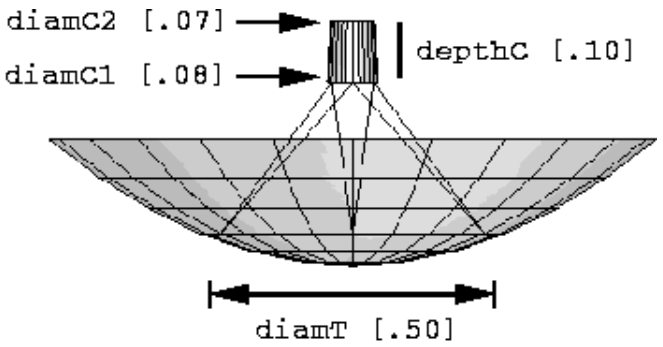
Hemisphere



`diam`	`[1]`	-- diameter of hemisphere
`capped`	`[0]`	-- whether the hemisphere is capped or not ("1" or "0")
`res`	`[24]`	-- resolution of circle (see common properties)

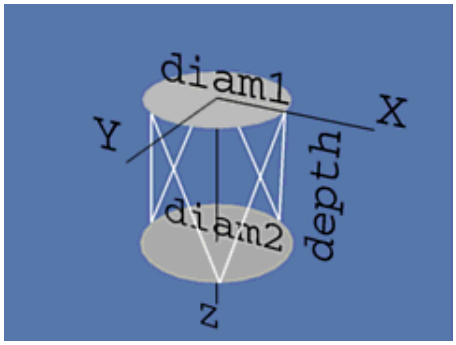
Parabolic Dish

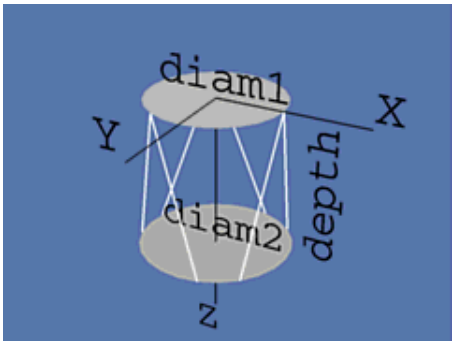




`diam`	`[1]`	-- diameter of paraboloid
`ratio`	`[.30]`	-- ratio of focal length to diameter
`capped`	`[0]`	-- whether the parabolic dish is capped or not ("1" or "0")
`res`	`[24]`	-- resolution of circle (see common properties)
`attach`	`[0]`	-- whether a truss and feedhorn should be attached ("1" or "0")
`diamT`	`[.50]`	-- diameter at base of truss
`diamC1`	`[.08]`	-- diameter at base of feedhorn
`diamC2`	`[.07]`	-- diameter at top of feedhorn
`depthC`	`[.10]`	-- depth of feedhorn in Z direction
`cappedC`	`[1]`	-- whether the feedhorn is capped or not ("1" or "0")

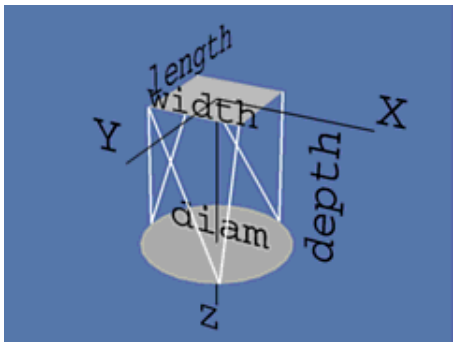
Cir Cir Truss





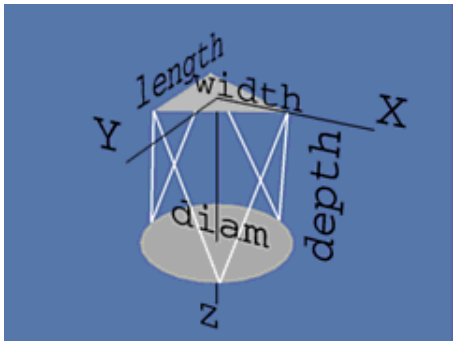
`diam1`	`[1]`	-- diameter of circle at first end (at z = 0)
`diam2`	`[1]`	-- diameter of circle at second end (at z = depth)
`depth`	`[1]`	-- depth of truss in Z direction
`angle`	`[60]`	-- arc in XY plane between struts (see common properties)

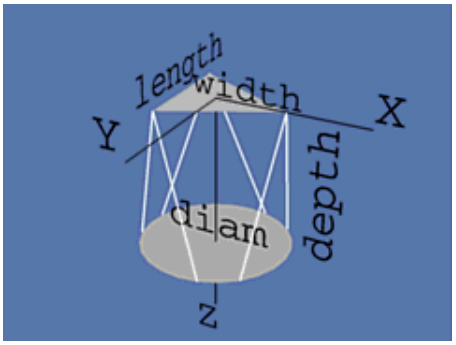
Cir Rec Truss



`length`	`[1]`	-- length of rectangle at first end (at z = 0) in Y direction
`width`	`[1]`	-- width of rectangle at first end (at z = 0) in X direction
`diam`	`[1]`	-- diameter of circle at second end (at z = depth)
`depth`	`[1]`	-- depth of truss in Z direction

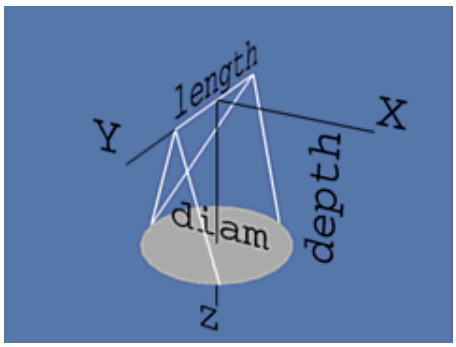
Cir Tri Truss

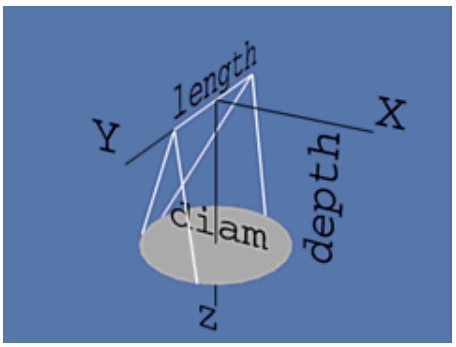




`length`	`[1]`	-- length of triangle at first end (at z = 0) in Y direction
`width`	`[.866]`	-- width of triangle at first end (at z = 0) in X direction
`diam`	`[1]`	-- diameter of circle at second end (at z = depth)
`depth`	`[1]`	-- depth of truss in Z direction
`angle`	`[60]`	-- arc in XY plane between struts (see common properties)

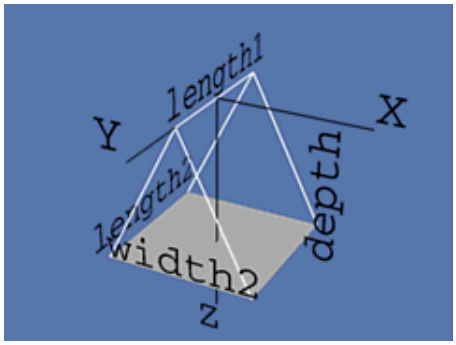
Lin Cir Truss

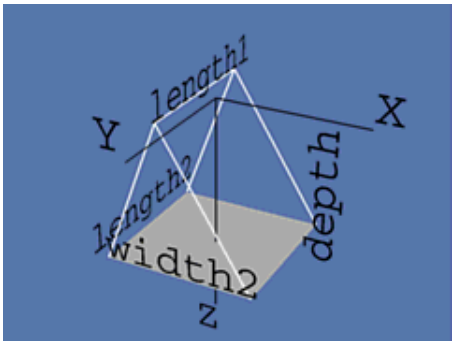




`length`	`[1]`	-- length of line at first end (at z = 0) in Y direction
`diam`	`[1]`	-- diameter of circle at second end (at z = depth)
`depth`	`[1]`	-- depth of truss in Z direction
`angle`	`[60]`	-- arc in XY plane between struts (see common properties)

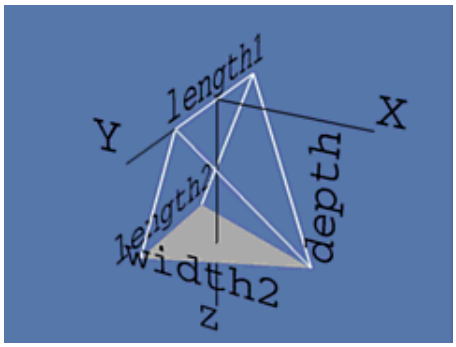
Lin Rec Truss

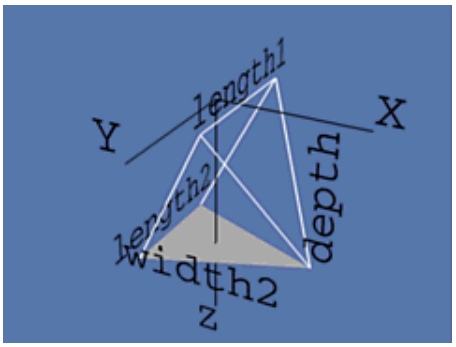




`length1`	`[1]`	-- length of line at first end (at z = 0) in Y direction
`length2`	`[1]`	-- length of rectangle at second end (at z = depth) in Y direction
`width2`	`[1]`	-- width of rectangle at second end (at z = depth) in X direction
`depth`	`[1]`	-- depth of truss in Z direction
`fraction`	`[.5]`	-- location of the line relative to the width of the square (see common properties)

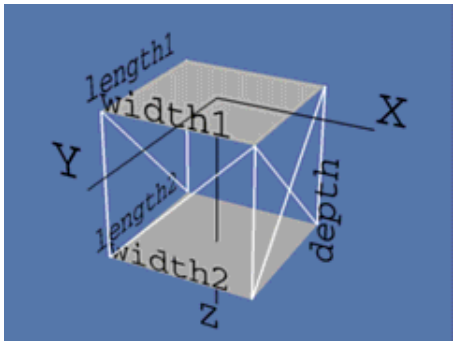
Lin Tri Truss

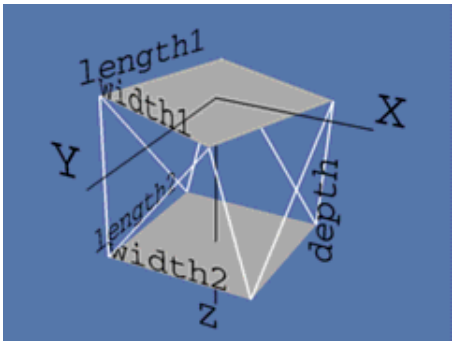




`length1`	`[1]`	-- length of line at first end (at z = 0) in Y direction
`length2`	`[1]`	-- length of triangle at second end (at z = depth) in Y direction
`width2`	`[.866]`	-- width of triangle at second end (at z = depth) in X direction
`depth`	`[1]`	-- depth of truss in Z direction
`fraction`	`[.333]`	-- location of the line relative to the width of the triangle (see common properties)

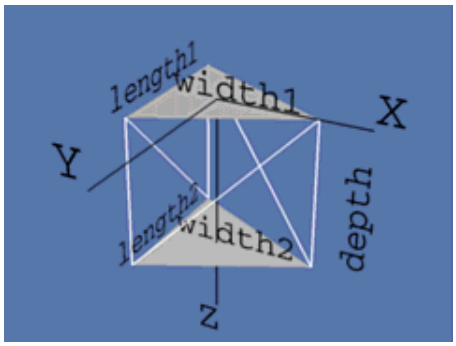
Rec Rec Truss

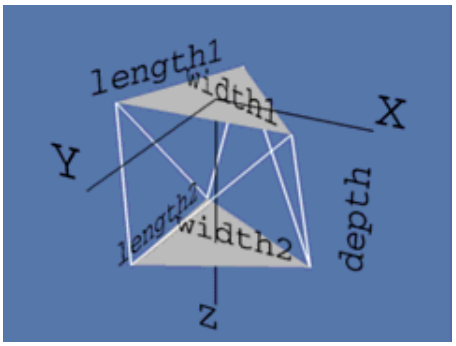




length1	[1]	-- length of rectangle at first end (at z = 0) in Y direction
width1	[1]	-- width of rectangle at first end (at z = 0) in X direction
length2	[1]	-- length of rectangle at second end (at z = depth) in Y direction
width2	[1]	-- width of rectangle at second end (at z = depth) in X direction
depth	[1]	-- depth of truss in Z direction
angle	[0]	-- angle of twist between ends (see common properties)

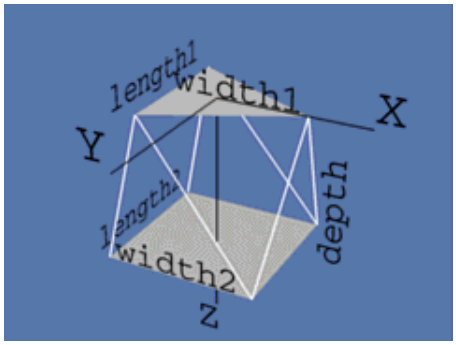
Tri Tri Truss

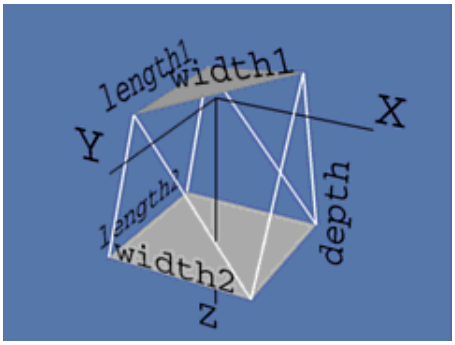




length1	[1]	-- length of triangle at first end (at z = 0) in Y direction
width1	[.866]	-- width of triangle at first end (at z = 0) in X direction
length2	[1]	-- length of triangle at second end (at z = depth) in Y direction
width2	[.866]	-- width of triangle at second end (at z = depth) in X direction
depth	[1]	-- depth of truss in Z direction
angle	[0]	-- angle of twist between ends (see common properties)

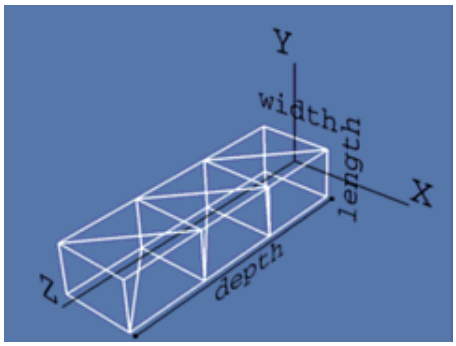
Tri Rec Truss





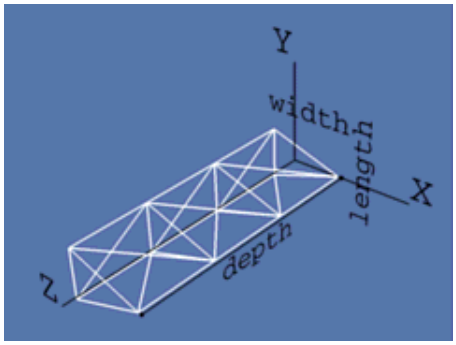
`length1`	`[1]`	-- length of triangle at first end (at z = 0) in Y direction
`width1`	`[.866]`	-- width of triangle at first end (at z = 0) in X direction
`length2`	`[1]`	-- length of rectangle at second end (at z = depth) in Y direction
`width2`	`[1]`	-- width of rectangle at second end (at z = depth) in X direction
`depth`	`[1]`	-- depth of truss in Z direction
`angle`	`[0]`	-- angle of tilt between triangle's plane and XY plane

Rec Boom Truss



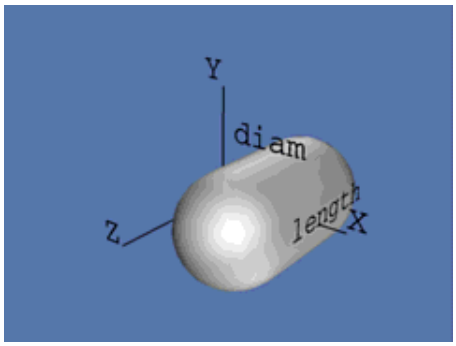
`length`	`[1]`	-- length of rectangle in Y direction
`width`	`[1]`	-- width of rectangle in X direction
`depth`	`[1]`	-- total depth of truss in Z direction
`number`	`[1]`	-- total number of sections in boom

Tri Boom Truss



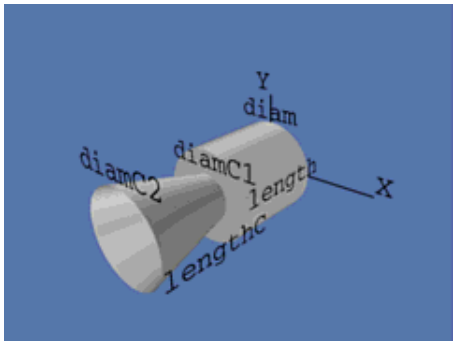
`length`	`[1]`	-- length of triangle in Y direction
`width`	`[.866]`	-- width of triangle in X direction
`depth`	`[1]`	-- total depth of truss in Z direction
`number`	`[1]`	-- total number of sections in boom

Tank Shape



`diam`	`[1]`	-- diameter of tank
`length`	`[1]`	-- length of cylindrical portion of tank in Z direction
`side`	`[1]`	-- whether the tank is attached at the side or the end ("1" or "0")
`res`	`[24]`	-- resolution of circle (see common properties)

Thruster Shape



`length`	`[1]`	-- length of cylindrical portion of thruster in Z direction
`diam`	`[1]`	-- diameter of cylindrical portion of thruster
`lengthC`	`[1]`	-- length of cone portion of thruster in Z direction
`diamC1`	`[1]`	-- diameter at first end of cone portion (at z = length)
`diamC2`	`[.5]`	-- diameter at second end of cone portion (at z = length + lengthC)
`res`	`[24]`	-- resolution of circle (see common properties)

OOGLPrimitive

`filename`	`[""]`	-- name of the OOGL file containing the geometry of the object