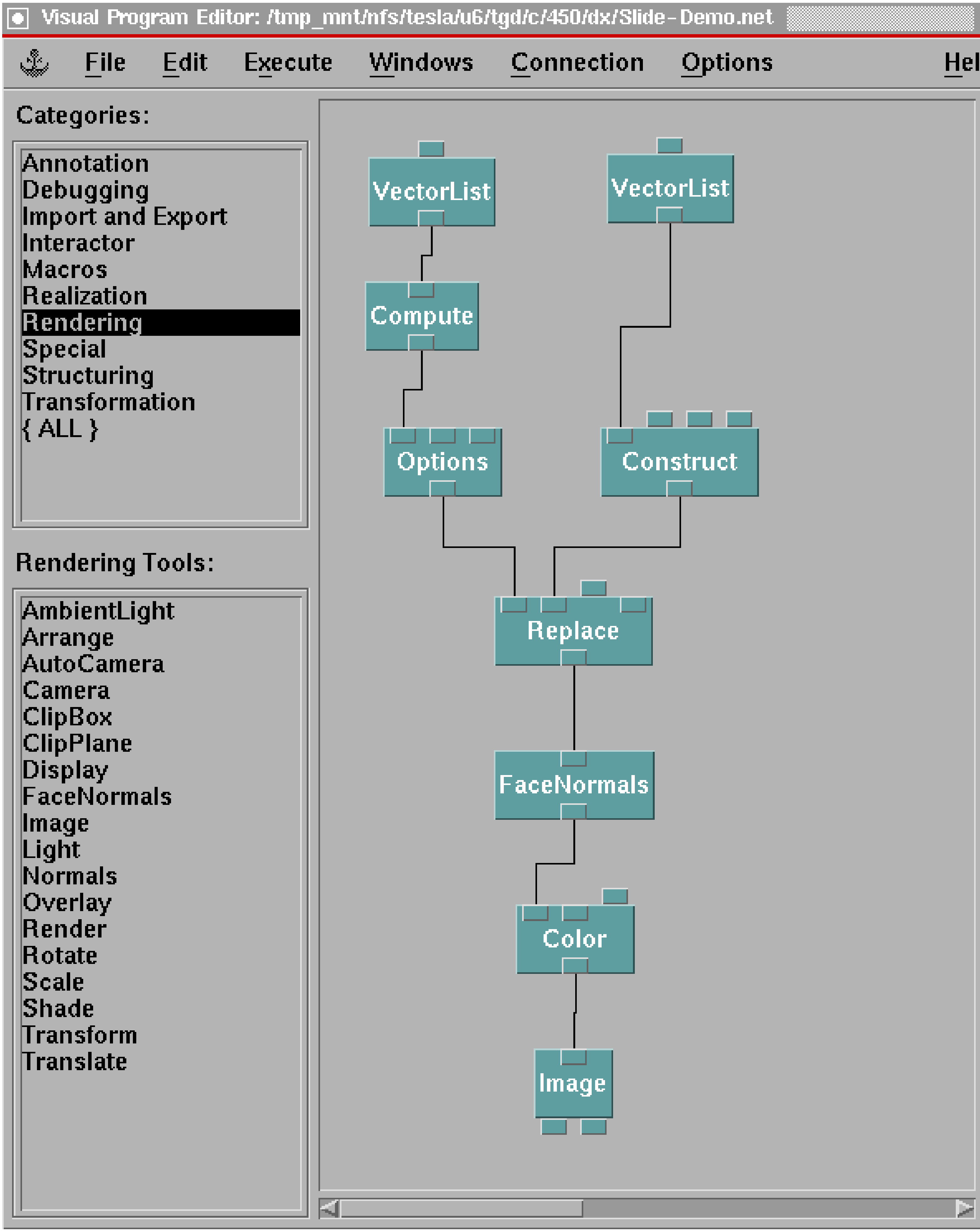
IBM Visualization Data Explorer (DX)

- **Visual Programming Language and Rendering System**
- **Intended for Visualizing Scientific Data**. But we will use it for class projects.
- **Installed on all College of Engineering HP's**
  (Maximum of 10 simultaneous users.)
- **Web Pages**
  `http://www.cs.orst.edu/~tgd/public_html/classes/450/450.html`
  Based on pages from Cornell University CS418 class.


Visual Program Editor: /tmp_mnt/nfs/tesla/u6/tgd/c/450/dx/Slide-Demo.net
File
Edit
Execute
Windows
Connection
Options
Hel
Categories:
Annotation
Debugging
Import and Export
Interactor
Macros
Realization
Rendering
Special
Structuring
Transformation
{ ALL }
Rendering Tools:
AmbientLight
Arrange
AutoCamera
Camera
ClipBox
ClipPlane
Display
FaceNormals
Image
Light
Normals
Overlay
Render
Rotate
Scale
Shade
Transform
Translate
VectorList
VectorList
Compute
Options
Construct
Replace
FaceNormals
Color
Image

VPE Structure

Four Regions:

- **Pull-Down Menu Bar**
- **Categories Pallet.** Divides the modules into various categories.
- **Tools Pallet.** Lists the tools available within this category.
- **Canvas.** The visual program is constructed here.

Each module has **input tabs** and **output tabs**. Only one "wire" can be connected to an input tab, but an output tab can go multiple places.

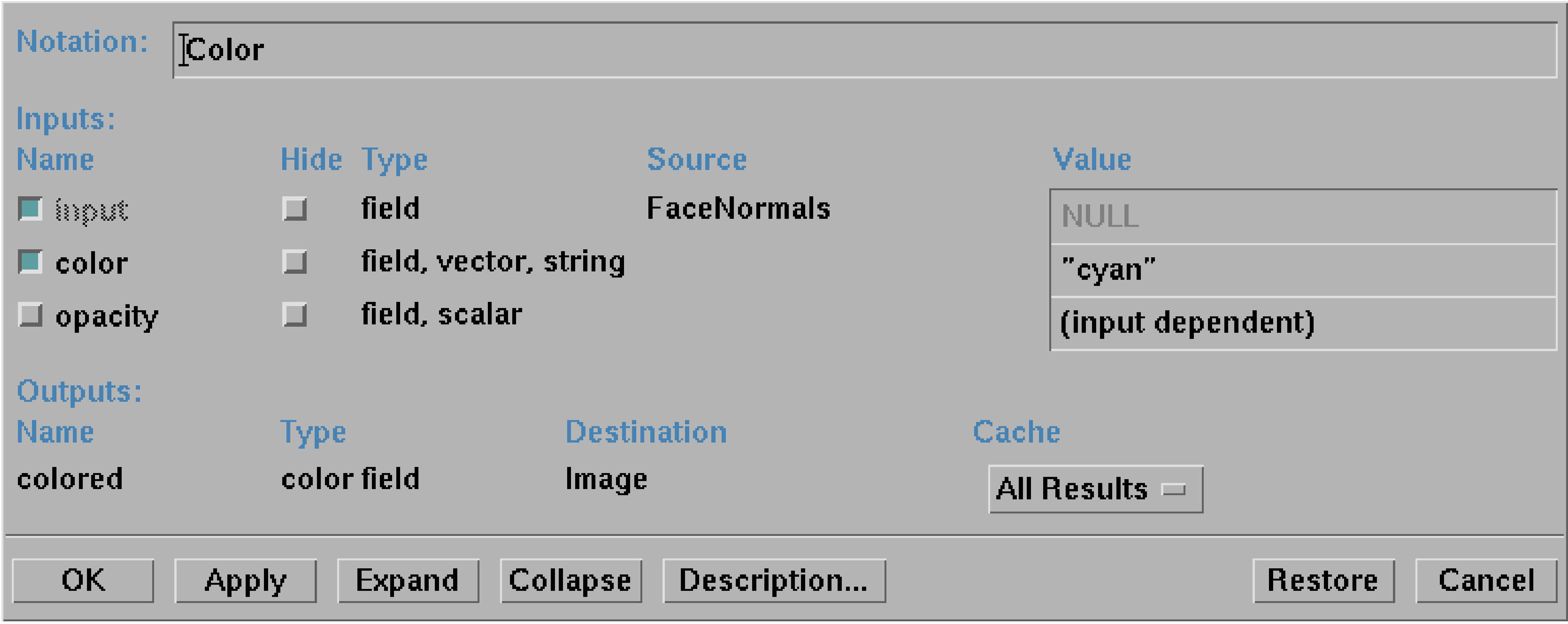
Left mouse button can select or move module icons.
Double-clicking "opens" the icon.
Middle mouse button brings up a context-specific help window.

Example Visual Program

- The first six modules (2 **VectorList**s, a **Compute**, an **Options**, a **Construct**, and a **Replace**) build a 3-D polyhedron.
- The **FaceNormals** module computes the unit normal vector for each face.
- The **Color** module assigns an intrinsic color to each face.
- The **Image** module renders the image.

Inside the Color Module

If we double-click on **Color**, we see the following:



Module Layout

- **Notation**. You can edit this field to add helpful comments.
- **Inputs:**. Each input is listed. If pushbutton under "Name" is depressed, then that input is required, and it must either be supplied by a wire or else provided with a value in the "Value" column. In this case, the first input ("input") is supplied by a wire from the **FaceNormals** module. The second input ("color") is supplied by a string "cyan". The third input ("opacity") is not required.

  It is also possible to specify the color as an RGB vector: `[0 1 1]`.

  It is possible to hide non-required inputs. We could do this by depressing the pushbutton in the "Hide" column and then pressing "Collapse". If we did this with the "Opacity" input, the third input tab would disappear from the module icon. The "Expand" button will reveal all of the inputs. There are actually 5 potential inputs to this module.
- **Output**. The output destination is shown.
- **Cache**. By default, all modules cache their computations. This makes animations run quickly after the first iteration. It also consumes lots of memory. You may need to quit and restart from time-to-time to reclaim memory.

Executing the Program

If we pull down **Execute** and select "Execute Once", the following image is generated:

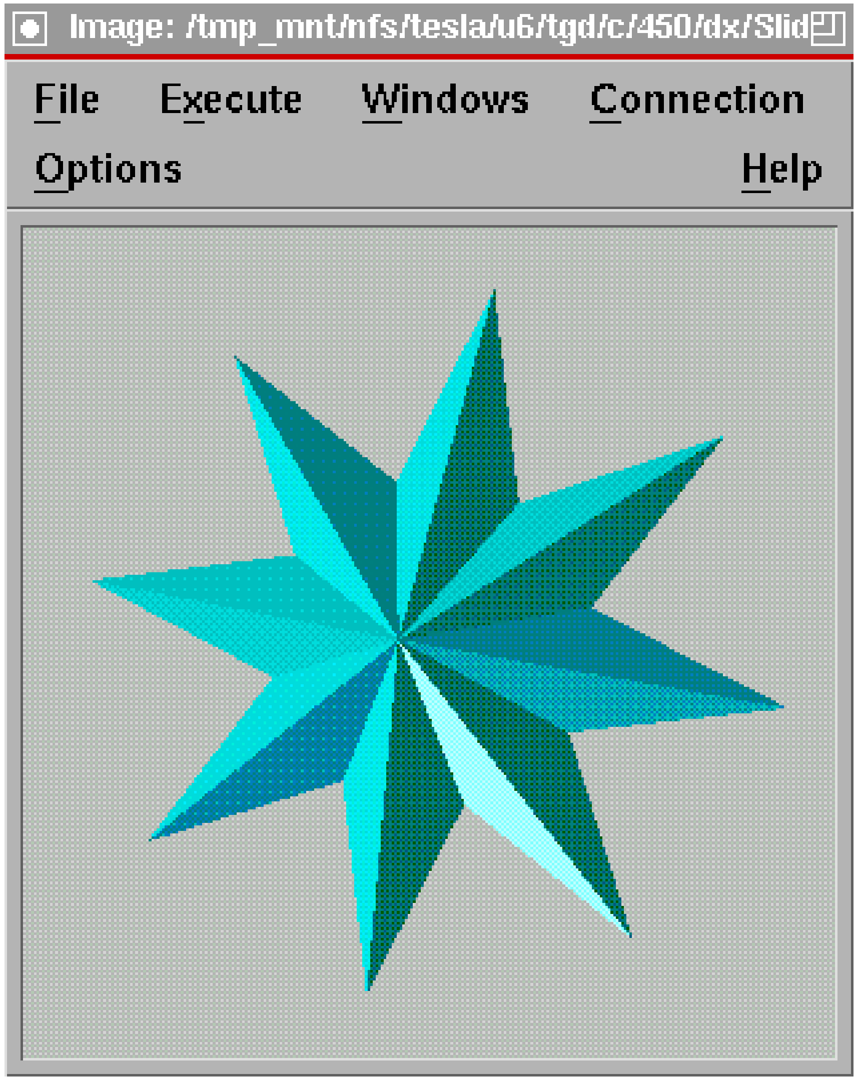


Image Options

Pulling down the **Options** menu on the Image window gives several important choices:

- **View Control**. This brings up another window that gives you various mouse-controls, especially zoom and rotate.
- **Autoaxes. . . .** This enables the display of labeled axes, which can be helpful in debugging.
- **Display Rotation Globe**. This brings up a small globe in the lower left-hand corner, which can help orient you when doing rotations.

View Control

- **Rotate** Mode.
  This brings up a small x/y/z axis indicator in the image. Left and middle mouse now act like a "track ball". The right mouse button will rotate the image about its center.
- **Zoom** Mode.
  Pressing left mouse in the window allows you to position a rectangle. When you release the mouse, the image will zoom in so that the part inside the rectangle will fill the entire window.

  Pressing right mouse will also position a window. When you release the mouse, the image will zoom out so that the current contents of the window will shrink to the indicated rectangle.

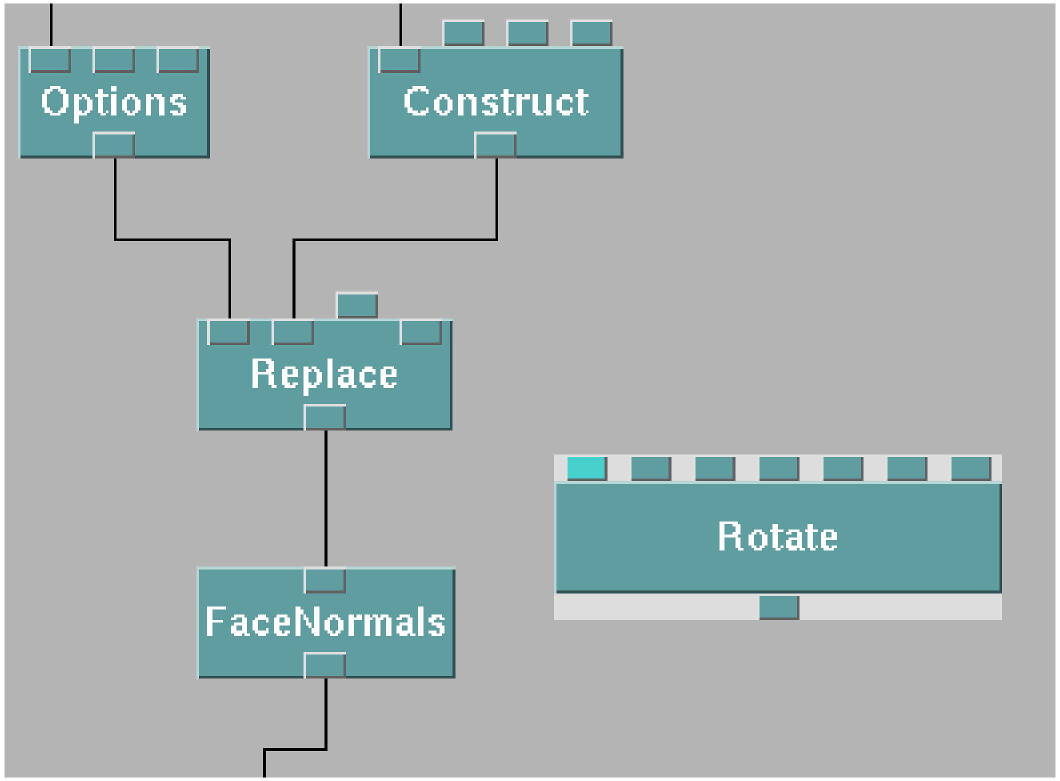
Making Our 8Star Spin

DX makes it easy to animate displays. To do this, we will modify our visual program to include two new modules:

- **A Sequencer**. This is an interactive module that generates a sequence of numbers. We will configure it to generate numbers between 0 and 44 degrees in 15-degree steps.
- **A Rotate** module. This will rotate the object by the number of input degrees.

To select the **Rotate** module, we use the left mouse button on the lower left pallet. We then move into the image window and place the module by pressing the left mouse button.

Adding Rotate



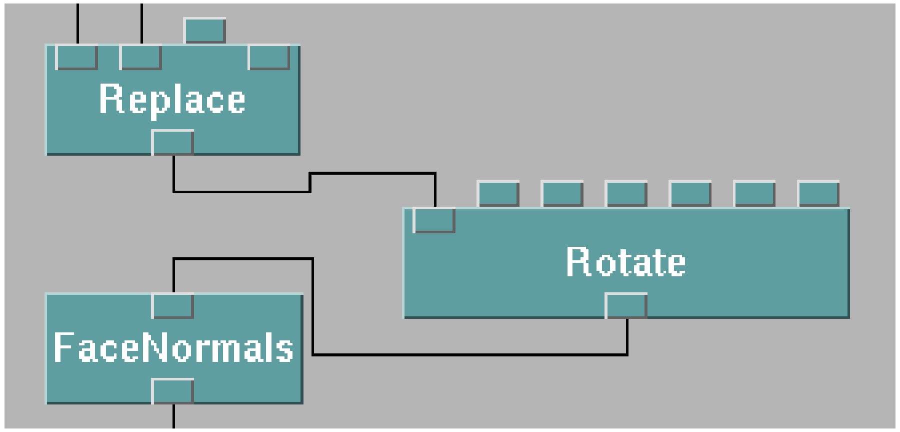
The light blue input tab indicates that it is a required input.

Connecting Rotate

We must first break the connection between **FaceNormals** and **Replace**. We do this by pressing the left mouse button on the **FaceNormals** input tab, moving away, and releasing.

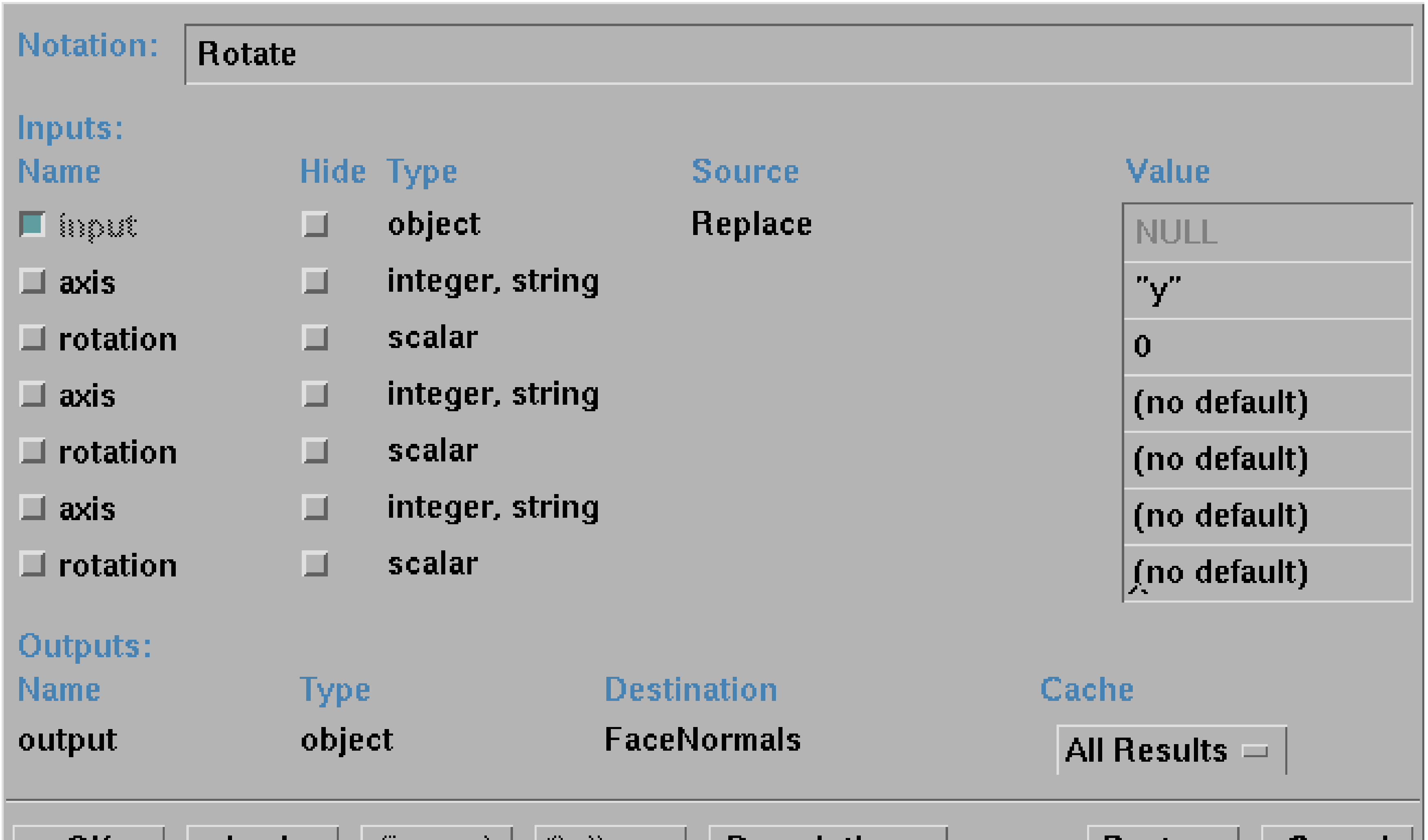
Then we must connect **Replace** to **Rotate**. We press the left button on the output tab of **Replace** and drag it to the light input tab of **Rotate**.

We then hook the output of **Rotate** to the input of **FaceNormals**



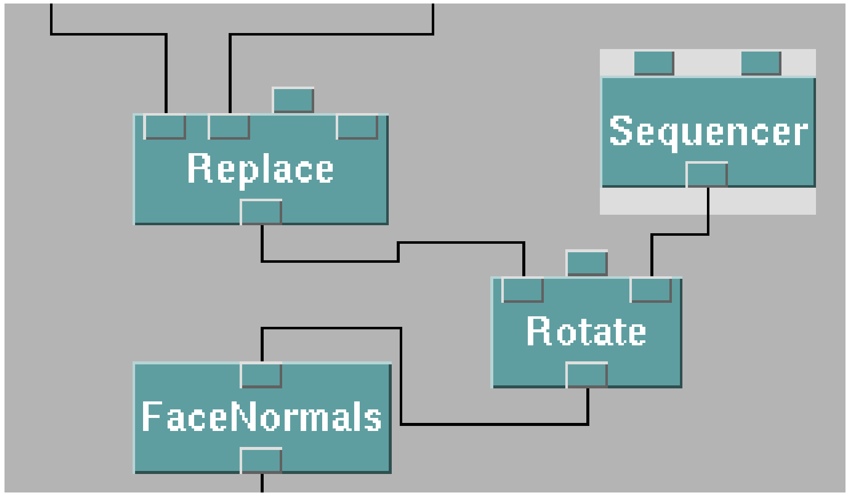
Eliminating Redundant Input Tabs

The **Rotate** module has 7 input tabs. What do they do? They permit rotating around up to 3 axes at once. We will only rotate around "y", which is the default. We can "hide" all of the others and "Collapse" to remove them.



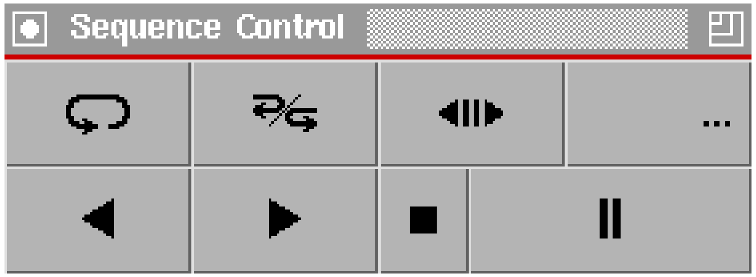
Hooking up the Sequencer

The **Sequencer** module is located in the **Special** category. We position it in the program and connect it to the "Rotation" input of the **Rotate** module:



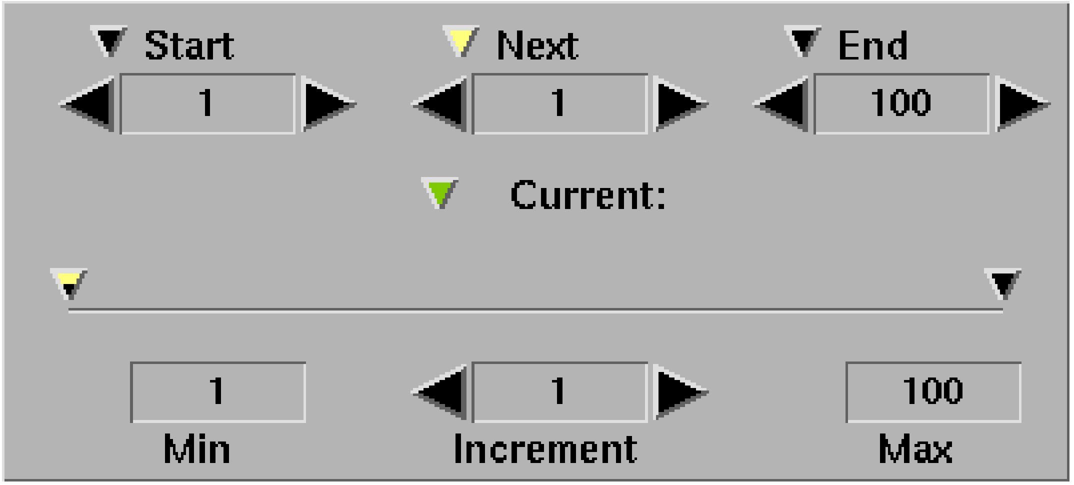
Configuring the Sequencer

When we double-click in the sequencer, we get a window that looks like a set of VCR controls.

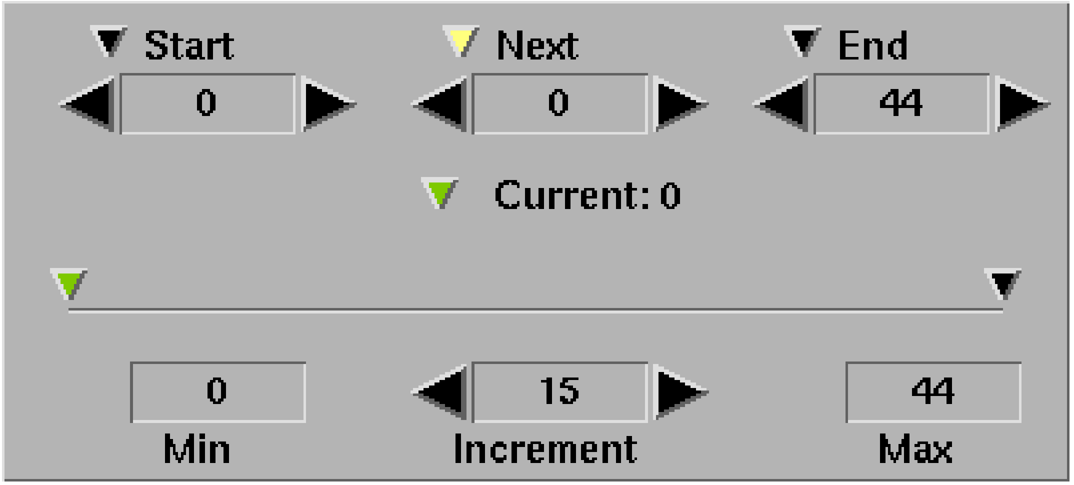


In order, these are **Continuous Loop**, **Palindrome**, **Single-Step**, **Configuration**, **Reverse**, **Forward**, **Stop**, **Pause**.

Configuring the Sequencer (2)



We want our sequencer to start at 0 degrees and step up to 44 degrees in steps of 15 degrees each. If we left-button on a field, we can type a new value (terminated by RETURN) from the keyboard. Here is the desired configuration:



Executing the Animation

If we press the **Forward** button, the program will begin executing under control of the sequencer, and we will see the 8star rotate. It stops when it has rotated 30 degrees. By turning on the **Continuous Loop** mode, it appears to spin around. We can get smoother animations by using 5 degree steps.

DX Data Model

A complex and flexible data structure is passed along the “wires” of the visual program. This data structure is hierarchical with the following structure:

Group
- Field
  - Component
    - Attribute
      - Value

Any sub-tree of this hierarchy can also travel along a wire.

Each level of the hierarchy can have names.

DX Polyhedron

```
Field.  3 components.
Component number 0, name 'positions':
 Generic Array.  18 items, float, real, 3-vector
 data values:
        // the vertexes of the polyhedron
 Attribute.  Name 'dep':
  String.  "positions"
Component number 1, name 'box':
 Generic Array.  8 items, float, real, 3-vector
 data values:
       // vertexes of a 3D bounding box
  Attribute.  Name 'der':
  String.  "positions"
Component number 2, name 'connections':
 Generic Array.  31 items, integer, real, 3-vector
 data values:
      // Each face: indexes into 'positions'
 Attribute.  Name 'element type':
  String.  "triangles"
 Attribute.  Name 'ref':
  String.  "positions"
```

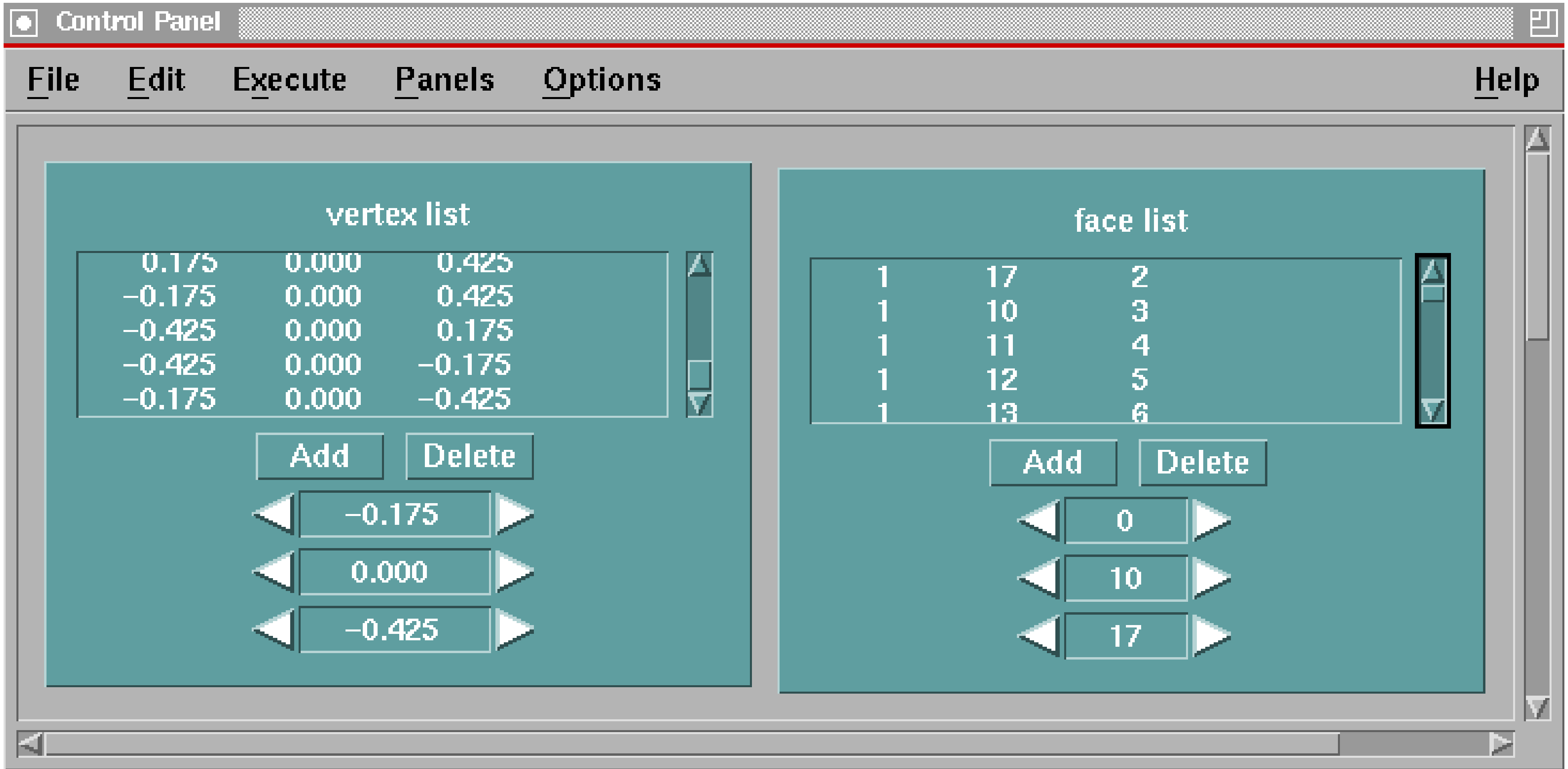
Positions and Connections

```
Component number 0, name 'positions':
data values:
    0         0         0
    0         0.5       0
    0         0        -1
    0.700     0        -0.700
    1         0         0
    0.700     0         0.700
    0         0         1
   -0.700     0         0.700
   -1         0         0
   -0.700     0        -0.700
    0.175     0        -0.425
    0.425     0        -0.175
    0.425     0         0.175
    0.175     0         0.425
   -0.175     0         0.425
   -0.425     0         0.175
   -0.425     0        -0.175
   -0.175     0        -0.425
```

```
Component number 2, name 'connections':
data values:
    1    17     2          4    11    12
    1    10     3          5    12    13
    1    11     4          6    13    14
    1    12     5          7    14    15
    1    13     6          8    15    16
    1    14     7          9    16    17
    1    15     8          0    11    10
    1    16     9          0    12    11
    1     2    10          0    13    12
    1     3    11          0    14    13
    1     4    12          0    15    14
    1     5    13          0    16    15
    1     6    14          0    17    16
    1     7    15          0    10    17
    1     8    16
    2    17    10
    3    10    11
```

Control Panels

All of this information was entered interactively using a control panel. As with the **Sequencer**, when you position certain "Interactor" modules, such as **VectorList**, they are controlled by a **Control Panel** rather than by a pop-up configuration box. I entered all of the vertexes and faces using this control panel.



What Image Expects

The **Image** module expects a "Field" or a "Group" of "Fields". Each field should have the following components:

- **Positions**. The vertexes.
- **Connections**. The faces. In our case, the faces are triangles, which is indicated by the "element type" attribute with value "triangles".
- **Normals**. The normals at each face.
- **Colors**. One color for each vertex. These are the intrinsic colors of the object. Specified as RGB values from 0 to 1.

The data structure can also have transformation matrices pre-pended to it. Our **Rotate** module does this, rather than modifying all of the position information directly. The actual work is done by **Image**.

Constructing the Polyhedron

The output of each **VectorList** module is just a simple data object (an array of 3-element vectors). We must combine these into a "Field" with "Positions" and "Connections" components having the proper attributes.

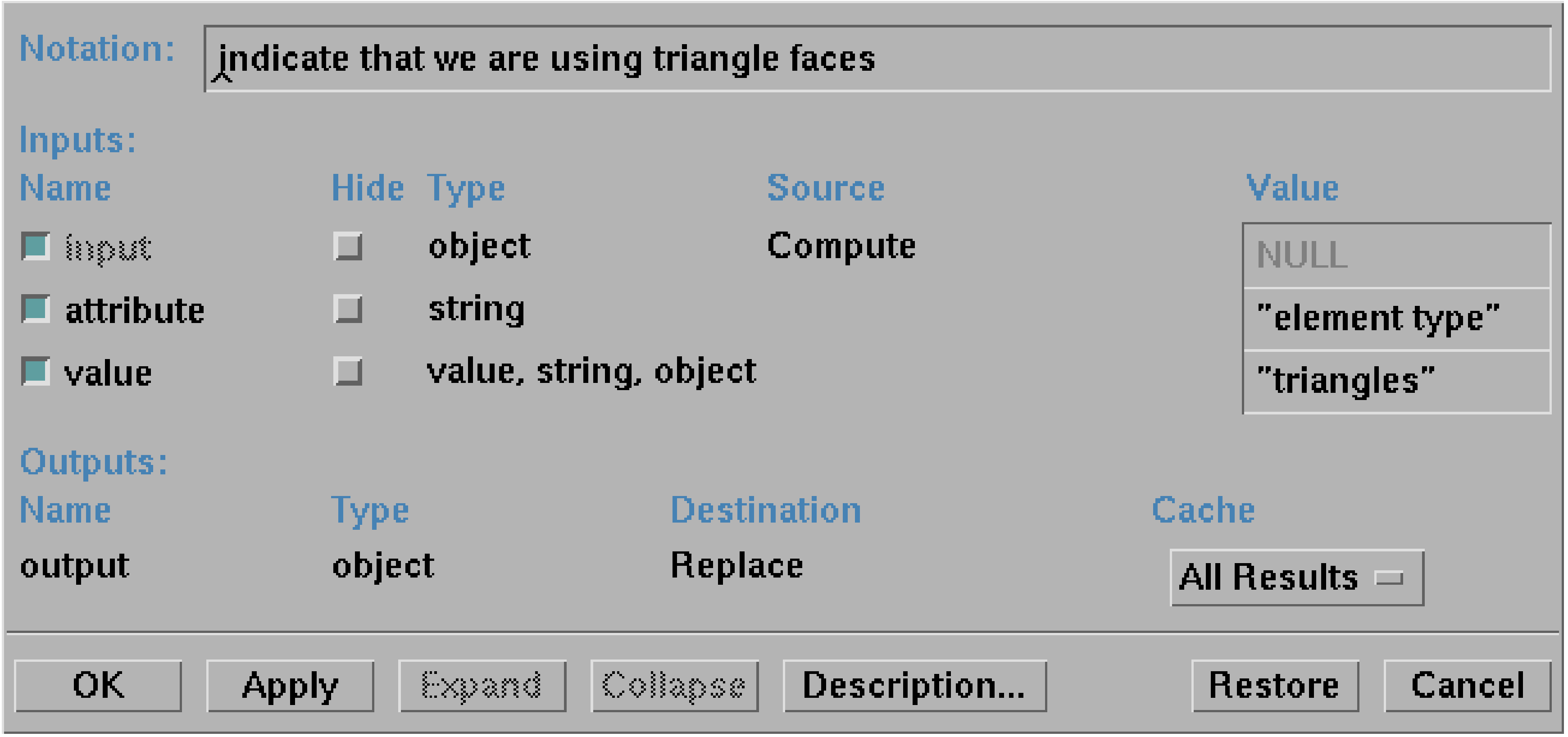
The **Compute** module is used to force the values from the first **VectorList** to be integers (see below).

The **Options** module is used to add the attribute "element type" with value "triangles" to the left array.

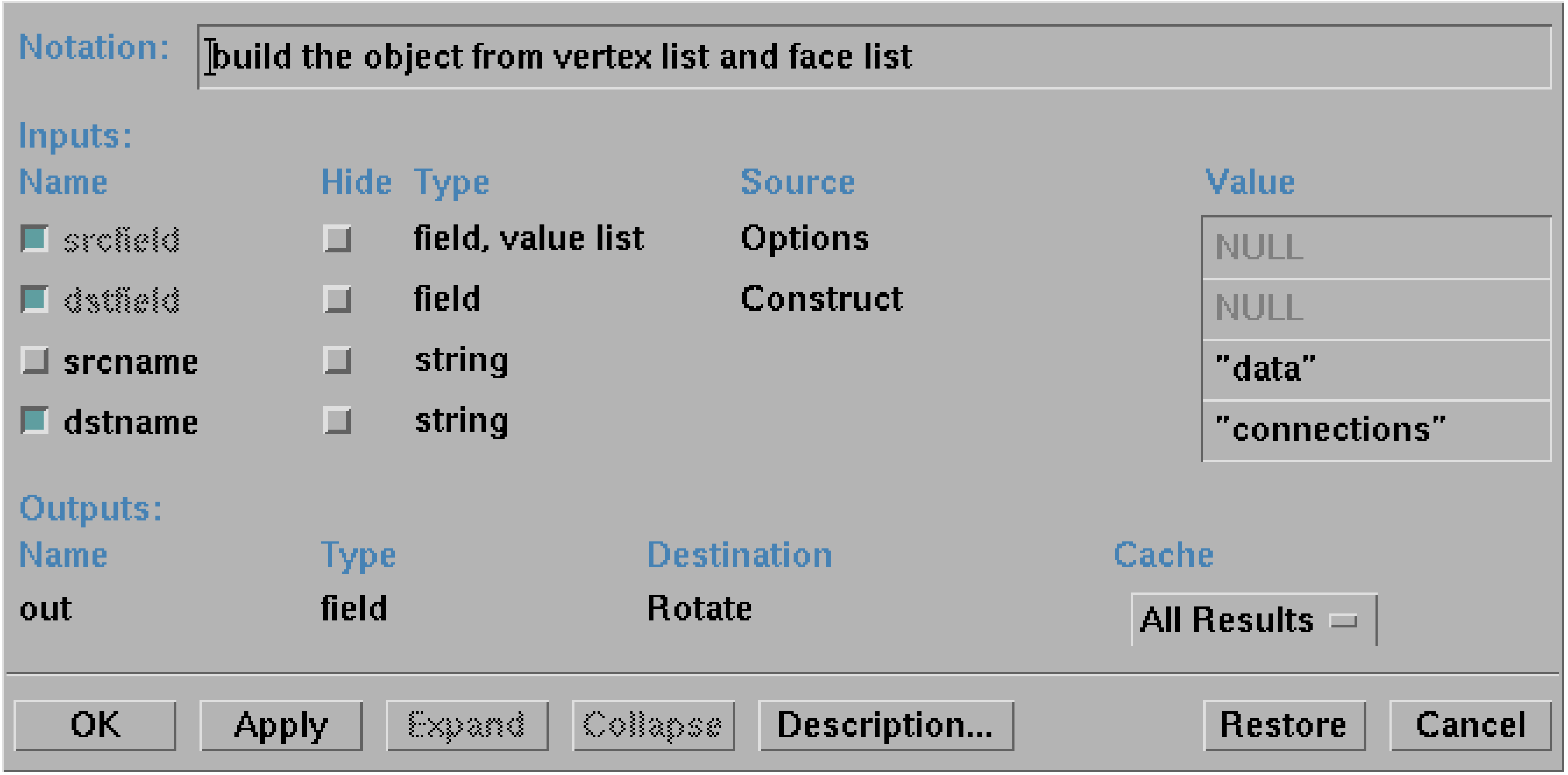
The **Construct** module builds a "Field" and places its input into the "Positions" component of the field. It also creates a (useless) "Connections" component that connects all of the positions into a single sequence.

The **Replace** module uses its input to replace the "Connections" component from **Construct** with the face lists coming from **Options**.

Options Configuration

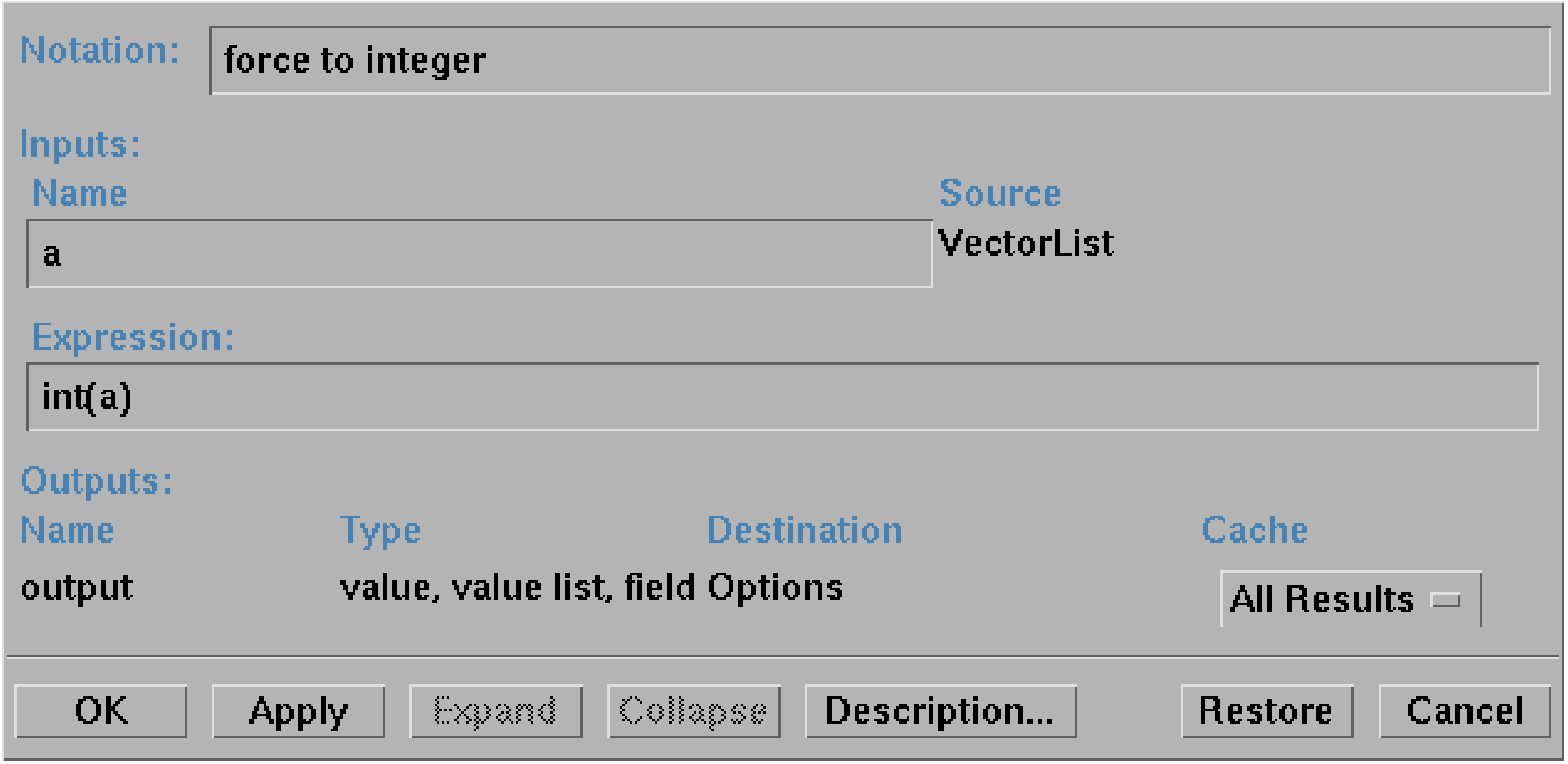


Replace Configuration



The Compute Module

The **Compute** module allows you to perform arbitrary computations on each input value. Each input is given a variable name. You then enter an expression in C-like syntax that will be the output value.



Object Syntax

- **Numbers**. Same as C.
- **Vectors**. Use Square Brackets: `[1 2 4]`. Commas may be used optionally to separate fields.
- **Lists**. Use Braces: `{1, 2, 3}`. Commas are required.

Miscellaneous Topics

- **Transmitters and Receivers**.
- **Debugging Aids**. The **Print** module will display its input onto a Message Window. It works best if you double-click and set the "options" input to the string `"dr"`.