Chapter 5

Visualizing Data

This chapter describes how to examine the values of variables and expressions in your program. This is referred to as *visualizing* data. In addition, it describes how to find out the type of a variable and change its values.

Section 5.1 is an overview of visualizing data. To learn:

- **How to choose the variable or expression whose values are to be visualized,** see Section 5.2.
- **How to work with graphical visualizers,** see Section 5.3.
- **How to visualize structures and pointers,** see Section 5.4.
- **How to print the type of a variable,** see Section 5.5.
- **How to change the values of a variable,** see Section 5.6.
- **How to change the radix of data,** see Section 5.7.

See Section 10.6 for a discussion of visualizing data in Node Prism.

5.1 Overview

You can visualize either variables (including arrays, structures, pointers, etc.) or expressions; see Section 2.9 for information on writing expressions in Prism. The data can reside on either the front end or the CM (for a CM-2 or CM-200), or on the partition manager or the nodes (for a CM-5). In addition, you can provide a *context*, so that Prism handles the values of data

Copyright © 1991–1994 Thinking Machines Corporation

elements differently, depending on whether they meet the condition you specify.

5.1.1 Printing and Displaying

Prism provides two general methods for visualizing data: *printing* and *displaying*.

- *Printing* data shows the value(s) of the data at a specified point during program execution.
- *Displaying* data causes its value(s) to be updated every time the program stops execution.

Printing or displaying to the history region of the command window prints out the numeric or character values of the data in standard fashion.

Printing or displaying to a graphical window creates a *visualizer*, which provides you with various options as to how to represent the data.

5.1.2 Methods

Prism provides these methods for choosing what to print or display:

- by choosing the **Print** or **Display** selection from the **Debug** menu in the menu bar (see Section 5.2.1)
- by selecting text within the source window (see Section 5.2.2)
- by adding events to the event table (see Section 5.2.3)
- by issuing commands from the command window (see Section 5.2.4)

In all cases, choosing **Display** adds an event to the event list, since displaying data requires an action to update the values each time the program is stopped. Note that, since **Display** updates automatically, the only way to keep an unwanted display window from reappearing is to delete the corresponding display event.

You create print events only via the event table and the **Events** menu.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

5.1.3 Notes

Note these points in visualizing data:

- You occasionally cannot print or display a CM-resident variable or array after interrupting execution; you receive an error message if you try. If this occurs, you must do a **Step** or **Next** after the interrupt before attempting to print.
- You cannot print or display any variables after a program finishes execution.
- Visualizers do not deal correctly with Fortran adjustable arrays. The size is determined when you create a visualizer for such an array. Subsequent updates to the visualizer will continue to use this same information, even though the size of the array may have changed since the last update. This will result in incorrect values in the visualizer. Printing or displaying values of an adjustable array in the command window or to a new window will work, however.
- Recall that you must define the type of a Paris parallel variable before you can print or display its values. (See Section 2.8.)
- On CM-5s with vector units, you can print or display a dereferenced pointer that points to vector-unit memory. By default, Prism prints the value in value in vector unit 0. To see the value in another vector unit, you must use the **`set`** command with the **`$dp`** variable to specify another vector unit. For example, the command

  ```
  set $dp = 3
  ```

  specifies that Prism is to print the value in vector unit 3.

Copyright © 1991–1994 Thinking Machines Corporation

5.2 Choosing the Data to Visualize

This section describes the methods Prism provides for printing and displaying data.

5.2.1 Printing and Displaying from the Debug Menu

To print a variable or expression at the current program location, choose **Print** from the **Debug** menu. It is also by default in the tear-off region.

To display a variable or expression every time execution stops, starting at the current program location, choose **Display** from the **Debug** menu.

When you choose **Print** or **Display**, a dialog box appears; Figure 17 shows an example of the **Print** dialog box.

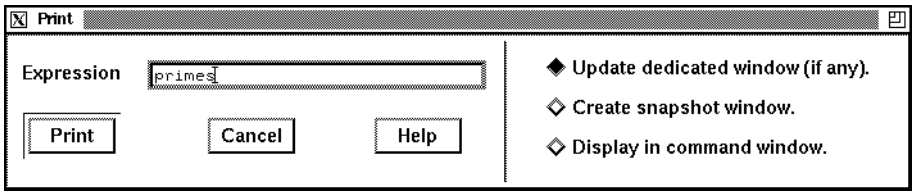

Figure 17. The Print dialog box from the Debug menu.

In the **Expression** box, enter the variable or expression whose value(s) you want printed. Text selected in the source window appears as the default; you can edit this text.

The dialog boxes also offer choices as to the window in which the values are to appear:

- You can specify that the values are to be printed or displayed in a standard window dedicated to the specified expression. The first time you print or display the data, Prism creates this window. If you print data, and subsequently print it again, this standard window is updated. This is the default choice for both **Print** and **Display**.
- You can create a separate *snapshot* window for printing or displaying values. This is useful if you want to compare values between windows.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

- You can print out the values in the command window.

Click on **Print** or **Display** to print the values of the specified expression at the current program location.

Click on **Cancel** or press the **Esc** key to close the window without printing or displaying.

5.2.2 Printing and Displaying from the Source Window

To print and display from the source window:

1. Select the variable or expression by dragging over it with the mouse or double-clicking on it.
2. Right-click the mouse to display a popup menu.
3. Click on **Print** in this menu to display a snapshot visualizer containing the value(s) of the selected variable or expression at that point in the program's execution. Click on **Display** to display a visualizer that is automatically updated whenever execution stops.

   To print without bothering to display the menu, press the **Shift** key while selecting the variable or expression.

NOTE: Prism prints the correct variable when you choose it in this way, even if the scope pointer sets a scope that contains another variable of the same name.

5.2.3 Printing and Displaying from the Event Table and the Events Menu

You can use the **Events** menu or the event table to define a print or display event that is to take place at a specified location in the program.

From the Events Menu

The **Print** dialog box (see Figure 18) prompts for the variable or expression whose value(s) are to be printed, the program location at which the printing

Copyright © 1991–1994 Thinking Machines Corporation

is to take place, and the name of the window in which the value(s) are to be displayed.

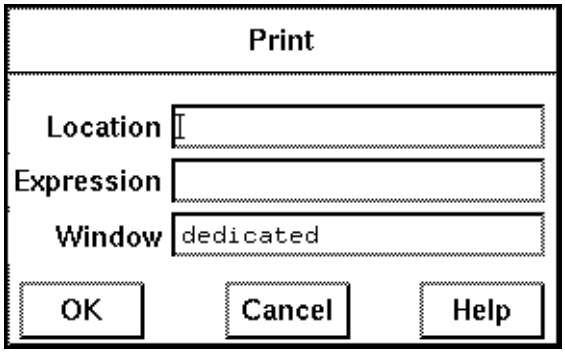

Figure 18. The Print dialog box from the Events menu.

Window names are **dedicated**, **snapshot**, and **command**; you can also make up your own name. The default is **dedicated**. See Section 2.7.3 for a discussion of these names.

When you have filled in the fields, click on **OK**; the event is added to the event table. When the location is reached in the program, the value(s) of the expression or variable are printed.

The **Display** dialog box is similar, but it does not prompt for a location; the display visualizer will update every time the program stops execution.

From the Event Table

Click on **Print** or **Display** in the **Common Events** buttons to create an event that will print or display data.

If you click on **Print**, the **Location** and **Action** fields are highlighted. Put a program location in the **Location** field. Complete the print event in the **Actions** field, specifying the variable or expression, and the window in which it is to be printed. For example,

```
print d2 on dedicated
```

If you click on **Display**, the **Location** field displays **stopped**, and the **Actions** field displays **print on dedicated**. Complete the description of the print

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

event, as described above. The variable or expression you specify is then displayed whenever the program stops execution.

5.2.4 Printing and Displaying from the Command Window

Use the **print** command to print the value(s) of a variable or expression from the command window. Use the **display** command to display the value(s). The **display** command prints the value(s) of the variable or expression immediately, and creates a display event so that the values are updated automatically whenever the program stops.

The commands have this format:

[**where (***expression***)**] *command variable*[**,** *variable* **...**]

The optional **where (***expression***)** syntax sets the context for printing the variable or expression; see below.

In the syntax, *command* is either **print** or **display**, and *variable* is the variable or expression to be displayed or printed.

Redirection of output to a window via the **on** *window* syntax works slightly differently for **display** and **print** from the way it works for other commands; see Section 2.7.3 for a discussion of redirection. Separate windows are created for each variable or expression that you print or display. Thus, the commands

```
display x on dedicated
display y/4 on dedicated
display [0:128:2]z on dedicated
```

create three windows, each of which is updated separately.

To print or display the contents of a register, precede the register's name with a dollar sign. For example,

```
print $pc
```

prints the program counter register. See Section 4.6.2 for a complete list of register names.

Copyright © 1991–1994 Thinking Machines Corporation

Setting the Context

You can precede the **print** or **display** command with a **where** statement that can make elements of a variable or array *inactive*. Inactive elements are not printed in the command window; Section 5.3.4 describes how they are treated in visualizers. Making elements inactive is referred to as *setting the context*.

To set the context, follow the **where** keyword with an expression in parentheses. The expression must evaluate to true or false for every element of the variable or array being printed. In CM Fortran, the expression can operate on a conformable array. In C*, it can operate on a parallel variable of the same shape as the variable being printed.

For example,

```
where (i .gt. 0) print i
```

prints (in the command window) only the values of **i** that are greater than 0.

In a C* program where **pvar1** and **pvar2** are of the same shape,

```
where (pvar1 > 0) display pvar2 on dedicated
```

displays as active only the elements of **pvar2** for which the value of the corresponding element of **pvar1** is greater than 0.

You can use certain Fortran intrinsics in the **where** statement. For example,

```
where (a .eq. maxval(a)) print a
```

prints the element of **a** that has the largest value. (This is equivalent to the **MAXLOC** intrinsic function.) See Section 2.9 for more information on writing expressions in Prism.

Note that setting the context affects only the printing or displaying of the variable. It does not affect the actual context of the program as it executes.

5.3 Working with Visualizers

The window that contains the data being printed or displayed is called a *visualizer*. Figure 19 shows a visualizer for a 3-dimensional array.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

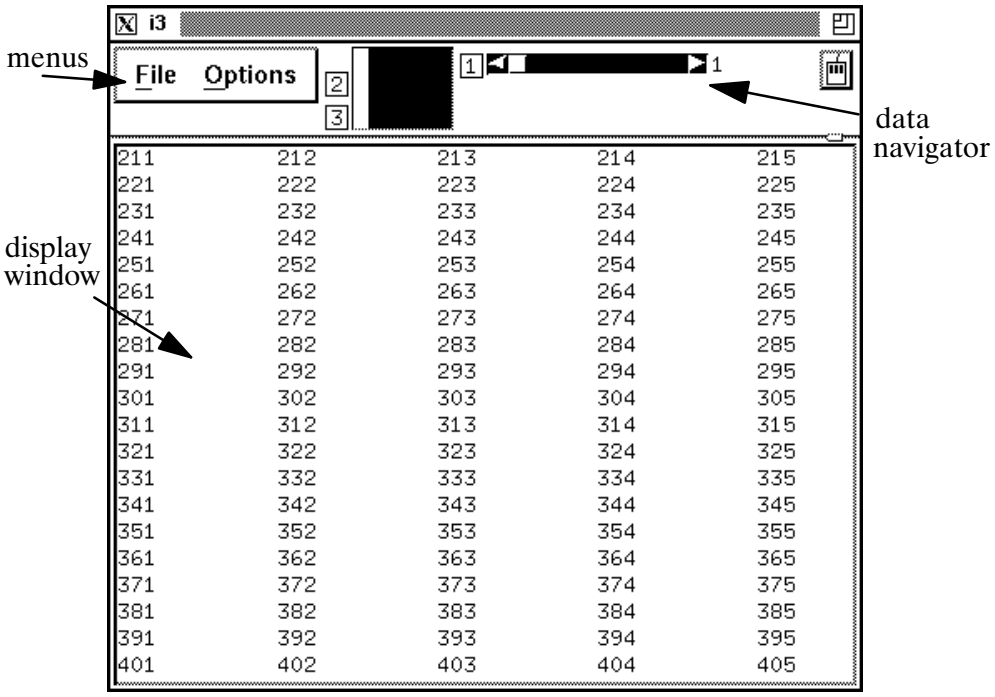

Figure 19. A visualizer for a 3-dimensional array.

The visualizer consists of two parts: the *data navigator* and the *display window*. There are also **File** and **Options** pulldown menus.

The *data navigator* shows which portion of the data is being displayed, and provides a quick method for moving through the data. The appearance of the data navigator depends on the number of dimensions in the data. It is described in more detail in Section 5.3.1.

The *display window* is the main part of the visualizer. It shows the data, using a representation that you can choose from the **Options** menu. The default is **text**: that is, the data is displayed as numbers or characters. Figure 19 is a text visualizer. The display window is described in more detail in Section 5.3.2.

The **File** menu lets you save, update, or cancel the visualizer; see Section 5.3.3 for more information. The **Options** menu, among other things, lets you change the way values are represented; see Section 5.3.4.

Copyright © 1991–1994 Thinking Machines Corporation

5.3.1 Using the Data Navigator in a Visualizer

The data navigator helps you move through the data being visualized. It has different appearances, depending on the number of dimensions in your data. If your data is a single scalar value, there is no data navigator.

For 1-dimensional arrays and parallel variables, the data navigator is the scroll bar to the right of the data. The number to the right of the buttons for the **File** and **Options** menus indicates the coordinate of the first element that is displayed. The elevator in the scroll bar indicates the position of the displayed data relative to the entire data set.

For 2-dimensional data, the data navigator is a rectangle in the shape of the data, with the axes numbered. The white box inside the rectangle indicates the position of the displayed data relative to the entire data set. You can either drag the box or click at a spot in the rectangle. The box moves to that spot, and the data displayed in the display window changes.

For 3-dimensional data, the data navigator consists of a rectangle and a slider, each of which you can operate independently. The value to the right of the slider indicates the coordinate of the third dimension. Changing the position of the bar along the slider changes which 2-dimensional plane is displayed out of the 3-dimensional data.

For data with more than three dimensions, the data navigator adds a slider for each additional dimension.

Changing the Axes

You can change the way the visualizer lays out your data by changing the numbers that label the axes. Click in the box surrounding the number; it is highlighted, and an I-beam appears. You can then type in the new number of the axis; you don't have to delete the old number. The other axis number automatically changes; for example, if you change axis 1 to 2, axis 2 automatically changes to become axis 1.

5.3.2 Using the Display Window in a Visualizer

The display window shows the data being visualized.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

In addition to using the data navigator to move through the data, you can drag the data itself relative to the display window by holding down the left mouse button; this provides finer control over the display of the data.

To find out the coordinates and value of a specific data element, click on it while pressing the **Shift** key. Its coordinates are displayed in parentheses, and its value is displayed beneath them. If you have set a context for the visualizer, you also see whether the element is active or inactive (see Section 5.3.4). Drag the mouse with the **Shift** key depressed, and you see the coordinates, value, and context of each data element over which the mouse pointer passes.

You can resize the visualizer to display more (or less) data either horizontally or vertically.

5.3.3 Using the File Menu

Click on **File** to pull down the **File** menu.

Choose **Update** from this menu to update the display window for this variable, using the value(s) at the current program location. See Section 5.3.5 for more information on updating a visualizer.

Choose **Snapshot** to create a copy of the visualizer, which you can use to compare with later updates.

Choose **Close** to cancel the visualizer.

5.3.4 Using the Options Menu

Click on **Options** to pull down the **Options** menu. See Figure 20.

Copyright © 1991–1994 Thinking Machines Corporation

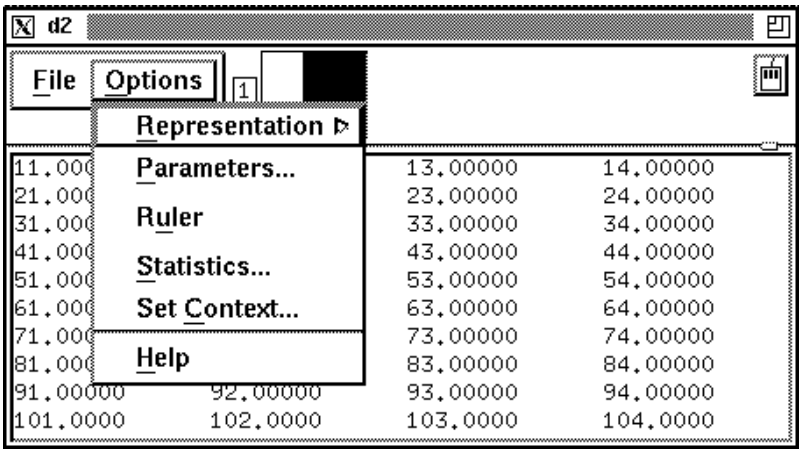

Figure 20. The Options menu in a visualizer.

Choosing the Representation

Choose **Representation** from the **Options** menu to display another menu that gives the choices for how the values are represented in the display window. The choices are described below. You can control aspects of the way these visualizers appear by changing their parameters, as described later in this section.

- Choose **Text** to display the values as numbers or letters. This is the default.

- Choose **Dither** to display the values as a shading from black to white. Groups of values in a low range are assigned more black pixels; groups of values in a high range are assigned more white pixels. This has the effect of displaying the data in various shades of gray. Figure 21 shows a 2-dimensional dither visualizer. The lighter area indicates values that are higher than values in the surrounding areas; the darker area indicates values that are lower than surrounding values.

  For complex numbers, Prism uses the modulus.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

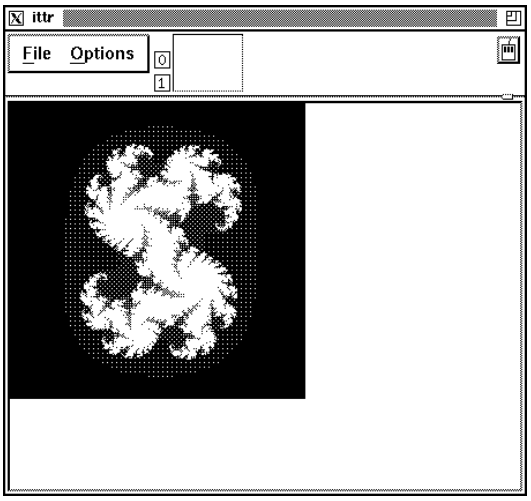

Figure 21. A dither visualizer.

- Choose **Threshold** to display the values as black or white. By default, Prism uses the mean of the values as the threshold; values less than or equal to the mean are black, and values greater than the mean are white. Figure 22 shows a threshold representation of a 3-dimensional array.

  For complex numbers, Prism uses the modulus.

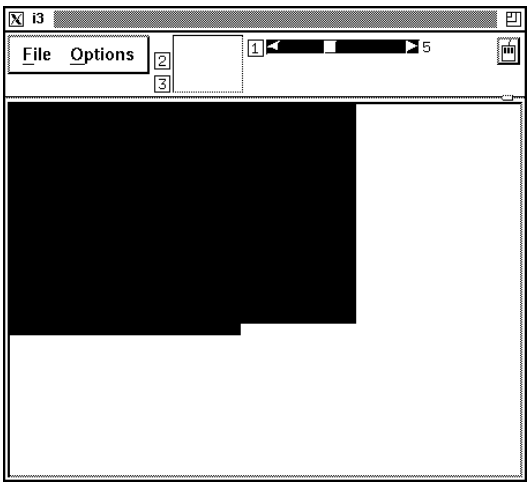

Figure 22. A threshold visualizer.

Copyright © 1991–1994 Thinking Machines Corporation

- Choose **Colormap** (if you are using a color workstation) to display the values as a range of colors. By default, Prism displays the values as a continuous spectrum from blue (for the minimum value) to red (for the maximum value). You can change the colors that Prism uses; see Section 9.3.2.

  For complex numbers, Prism uses the modulus.

- Choose **Graph** to display values as a graph, with the index of each array element plotted on the horizontal axis and its value on the vertical axis. A line connects the points plotted on the graph. This representation is particularly useful for 1-dimensional data, but can be used for higher-dimensional data as well; for example, in a 2-dimensional array, graphs are shown for each separate 1-dimensional slice of the 2-dimensional plane.

  Figure 23 shows a graph visualizer for a 1-dimensional array.

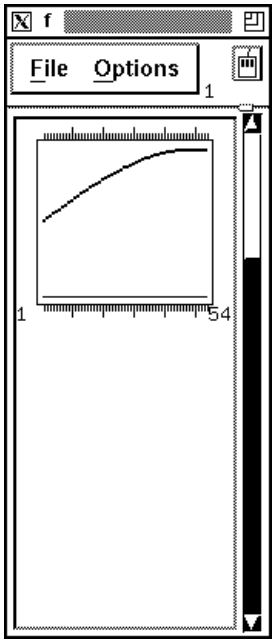

Figure 23. A 1-dimensional graph visualizer.

- Choose **Surface** (if your data has more than one dimension) to render the 3-dimensional contours of a 2-dimensional slice of data. In the representation, the 2-dimensional slice of data is tilted 45 degrees

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

away from the viewer, with the top edge further from the viewer than the bottom edge. The data values rise out of this slice. Figure 24 is an example.

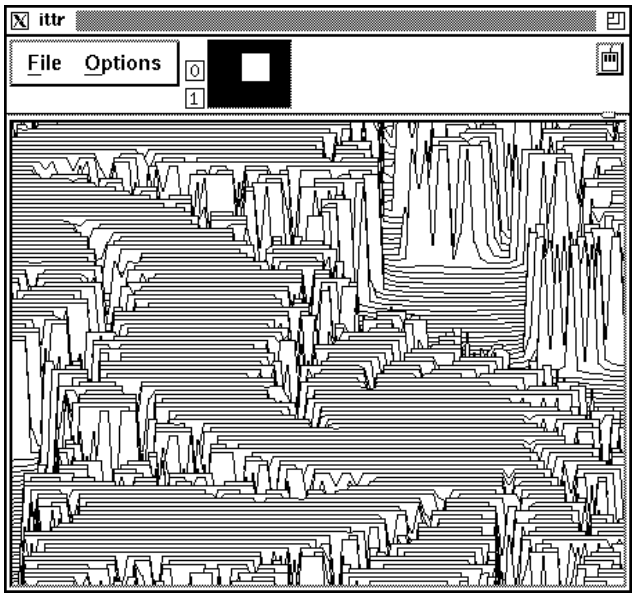

Figure 24. A surface visualizer.

NOTE: If there are large values in the top rows of the data, they may be drawn off the top of the screen. To see these values, flip the axes as described earlier in this section, so that the top row appears in the left column.

- Choose **Vector** to display data as vectors. The data must be a Fortran complex or double complex number, or a pair of variables to which the **CMPLX** intrinsic function has been applied (see Section 2.9.2). The complex number is drawn showing both magnitude and direction. The length of the vector increases with magnitude. There is a minimum vector length of five pixels, because direction is difficult to see for smaller vectors. By default, the lengths of all vectors scale linearly with magnitude, varying between the minimum and maximum vector lengths. Figure 25 shows a vector visualizer.

Copyright © 1991–1994 Thinking Machines Corporation

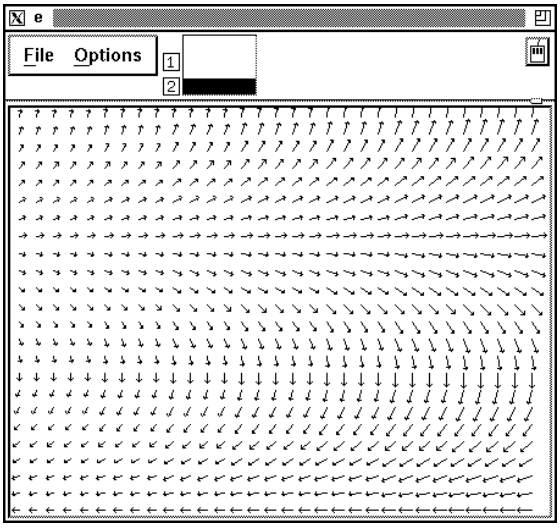

Figure 25. A vector visualizer.

Setting Parameters

Choose **Parameters** from the **Options** menu to display a dialog box in which you can change various defaults that Prism uses in setting up the display window; see Figure 26. If a parameters is grayed out or missing, it does not apply to the current representation.

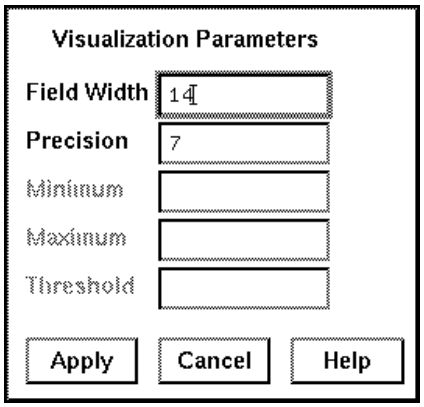

Figure 26. The Visualization Parameters dialog box.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

The parameters are:

- **Field Width** — Type a value in this box to change the width of the field that Prism allocates to every data element.

  For the text representation, the field width specifies the number of characters in each column. If a number is too large for the field width you specify, dots are printed instead of the number.

  For dither, threshold, colormap, and vector representations, the field width specifies how wide (in pixels) the representation of each data element is to be. By default, dither, threshold, and colormap visualizers are scaled to fit the display window. Note, however, that for dither visualizers, the gray shading may be more noticeable with a smaller field width.

  For the graph representation, the field width specifies the horizontal spacing between elements.

  For the surface representation, it specifies the spacing of elements along both directions of the plane.

- **Field Height** — For graph and surface representations, changing this value affects the maximum height (in pixels) to which Prism scales every data value.

- **Precision** — Type a value in this box to change the precision with which Prism displays real numbers in a text visualizer. The precision must be less than the field width. By default, Prism prints doubles with 16 significant digits, and floating-point values with 7 significant digits. You can change this default by issuing the **`set`** command with the **`$d_precision`** variable (for doubles) or **`$f_precision`** variable (for floating-point values). For example,

  ```
  set $d_precision = 11
  ```

  sets the default precision for doubles to 11 significant digits.

- **Minimum** and **Maximum** — For colormap representations, use these variables to specify the minimum and maximum values that Prism is to use in assigning color values to the data elements. Data elements that have values below the minimum and above the maximum are assigned default colors.

  For graph, surface, and vector representations, these parameters represent the bottom and top of the range that is to be represented.

Copyright © 1991–1994 Thinking Machines Corporation

Values below the minimum are shown as the minimum; values above the maximum are shown as the maximum.

By default Prism uses the entire range of values for all these representations.

- **Threshold** — For threshold representations, use this variable to specify the value at which Prism is to change the display from black to white. Data elements whose values are at or below the threshold are displayed as black; data elements whose values are above the threshold are displayed as white. By default, Prism uses the mean of the data as the threshold.

Displaying a Ruler

Choose **Ruler** from the **Options** menu to toggle the display of a ruler around the data in the display window. The ruler is helpful in showing which elements are being displayed. Figure 27 shows a 3-dimensional threshold visualizer with the ruler displayed.

In the surface representation, the ruler cannot indicate the coordinates of elements in the vertical axis, since they change depending on the height of each element. However, you can press the **Shift** key and left-click as described above to display the coordinates and value of an element.

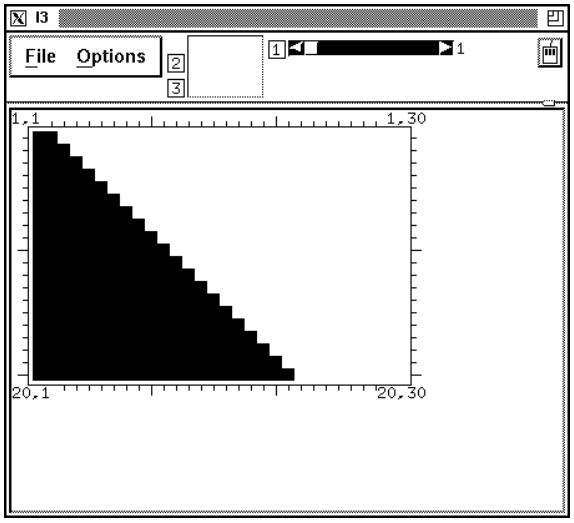

Figure 27. A threshold visualizer with a ruler.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

Displaying Statistics

Choose **Statistics** from the **Options** menu to display a window containing statistics and other information about the variable being visualized. The window contains

- the name of the variable
- its type and number of dimensions
- the total number of elements the variable contains, and the total number of active elements, based on the context you set within Prism (see the next section for a discussion of setting the context)
- the variable's minimum, maximum, and mean; these statistics reflect the context you set for the visualizer

Figure 28 gives an example of the **Statistics** window.

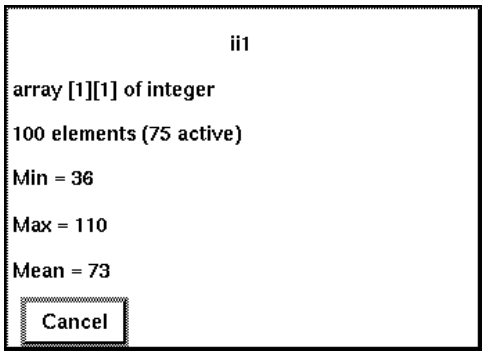

Figure 28. Statistics for a visualizer.

For complex numbers, Prism uses the modulus.

Using the Set Context Dialog Box

Choose **Set Context** from the **Options** menu to display a dialog box in which you can specify which elements of the variable are to be considered active

Copyright © 1991–1994 Thinking Machines Corporation

and which are to be considered inactive. Active and inactive elements are treated differently in visualizers:

- In text, graph, surface, and vector visualizers, inactive elements are grayed out.
- In colormap visualizers, inactive elements by default are displayed as gray. You can change this default; see Section 9.3.2.
- Context has no effect on dither and threshold visualizers.

Figure 29 shows the **Set Context** dialog box.

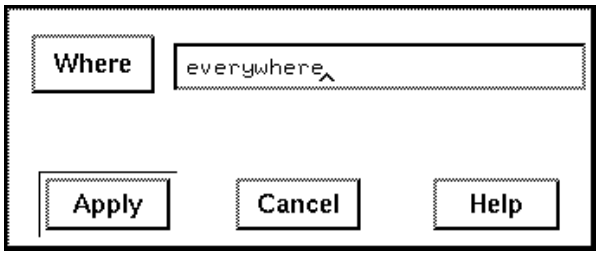

Figure 29. The Set Context dialog box.

By default, all elements of the variable are active; this is the meaning of the **everywhere** keyword in the text-entry box. To change this default, you can either edit the text in the text-entry box directly, or you can click on the **Where** button to display a menu. The choices in the menu are **everywhere** and **other**.

- Choosing **everywhere**, as mentioned above, makes all elements active.
- Choose **other** to erase the current contents of the text-entry box. You can then enter an expression into the text-entry box.

In the text-entry box, you can enter any valid expression that will evaluate to true or false for each element of the variable.

The context you specify for printing does not affect the program's context; it just affects the way the elements of the variable are displayed in the visualizer.

See "Setting the Context" above for more information on context. See Section 2.9 for more information on writing expressions in Prism.

Copyright © 1991–1994 Thinking Machines Corporation

Click on **Apply** to set the context you specified. Click on **Cancel** or press the **Esc** key to close the dialog box without setting the context.

5.3.5 Updating and Closing the Visualizer

If you created a visualizer by issuing a **Display** command, it automatically updates every time the program stops execution.

If you created the visualizer by issuing a **Print** command, its display window is grayed out when the program resumes execution and the values in the window are outdated. To update the values, choose **Update** from the visualizer's **File** menu.

To close the visualizer, choose **Close** from the **File** menu, or press the **Esc** key.

5.4 Visualizing Structures

If you print a pointer or a structure (or a structure-valued expression) in a window, a *structure visualizer* appears. One exception: C* parallel structures are displayed in a regular visualizer, because they can't contain pointers.

Figure 30 shows an example of a structure visualizer.

Copyright © 1991–1994 Thinking Machines Corporation

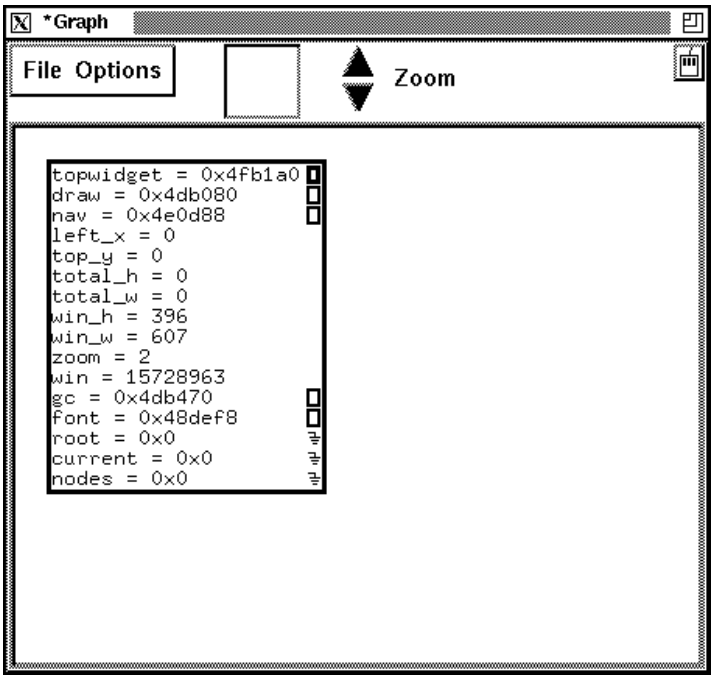

Figure 30. A structure visualizer.

The structure you specified appears inside a box; this is referred to as a *node*. The node shows the fields in the structure and their values. If the structure contains pointers, small boxes appear next to them; they are referred to as *buttons*. Left-click on a node to select it. Use the up and down arrow keys to move between buttons of a selected node.

You can perform various actions within a structure visualizer, as described below.

5.4.1 Expanding Pointers

You can expand scalar pointers in a structure to generate new nodes. (You cannot expand a pointer to a parallel variable.)

To expand a single pointer:

- **With a mouse:** Left-click on a button to expand the pointer. For example, clicking on the button next to the **`nav`** field in Figure 30 changes the visualizer as shown in Figure 31.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

- **From the keyboard:** Use the right arrow key to expand and visit the node pointed to by the current button. If the node is already expanded, pressing the right arrow key simply visits the node. Use the left arrow key to visit the parent of a selected node.

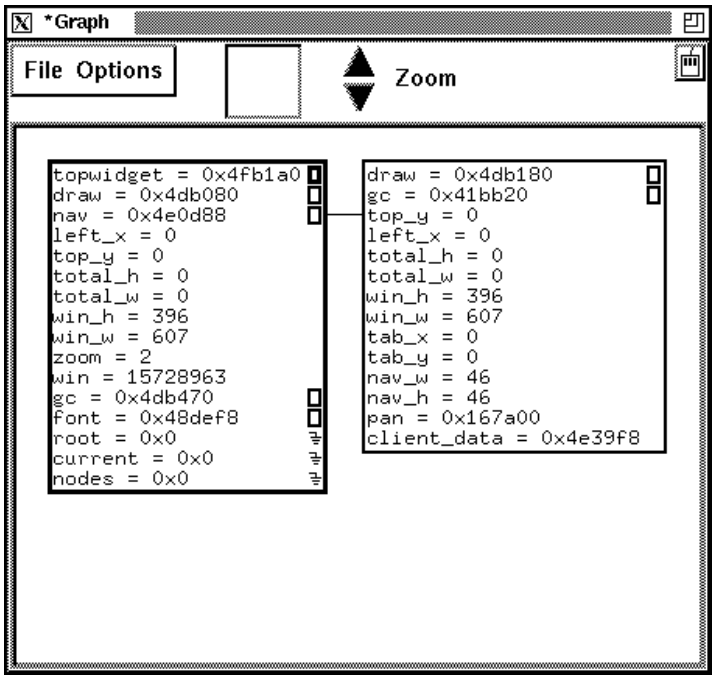

Figure 31. A structure visualizer, with one pointer expanded.

To expand all pointers in a node:

- **With the mouse:** Double-click or **Shift**-left-click on the node.
- **From the keyboard:** Press the **Shift** key along with the right arrow key.
- **From the Options menu:** Click on **Expand**. The cursor turns into a target; move the cursor to the node you are interested in and left-click.

To recursively expand all pointers from the selected node on down:

- **With the mouse:** Triple-click or **Control**-left-click on the node.
- **From the keyboard:** Press the **Control** key and the right arrow key.

Copyright © 1991–1994 Thinking Machines Corporation

- **From the Options menu:** Click on **Expand All**. The cursor turns into a target; move the cursor to the node you are interested in and left-click.

5.4.2 Panning and Zooming

You can left-click and drag through the data navigator or the display window to pan through the data, just as you can with visualizers; see Sections 5.3.1 and 5.3.2.

You can also "zoom" in and out on the data by left-clicking on the **Zoom** arrows. Click on the down arrow to zoom out and see a bird's-eye view of the structure; click on the up arrow to get a closeup. Figure 32 shows part of a complicated structure visualizer in which we have zoomed out.

Left-click on a node in a zoomed-out structure visualizer to pop up a window showing the full contents of the node.

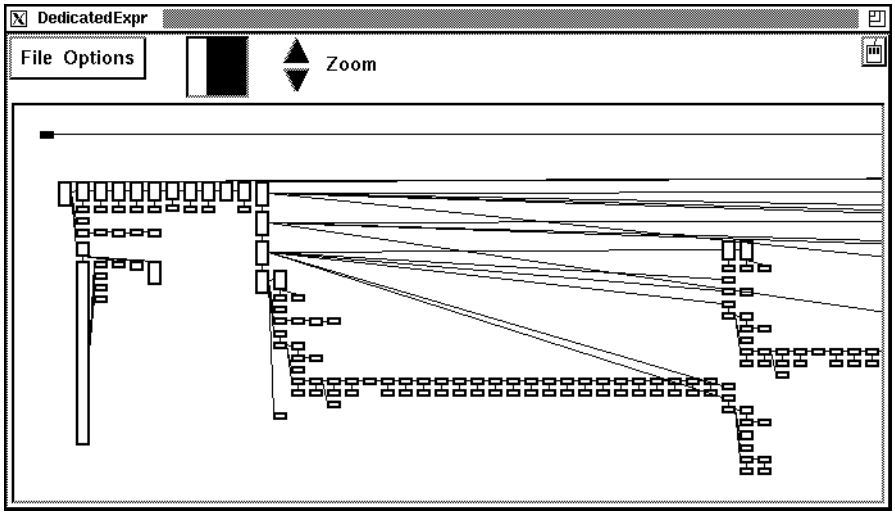

Figure 32. Zooming out in a structure visualizer.

The selected node is centered in the display window whenever you zoom in or out.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

5.4.3 Deleting Nodes

To delete a node (except the root node):

- **With the mouse:** Middle-click on a node (except the root node).
- **From the Options menu:** Click on **Delete**. The cursor turns into a target; move the cursor to the node you want to delete and left-click.

Deleting a node also deletes its children (if any).

5.4.4 More about Pointers in Structures

Note the following about pointers in structure visualizers:

- Null pointers — for example, `root` in Figure 31 — have "ground" symbols next to them.
- If a pointer has previously been expanded, it has an arrow next to its button; you can't expand the pointer again. (This prevents infinite loops on circular data structures.)
- A pointer containing a bad address has an **X** drawn over its button.

5.4.5 Updating and Closing a Structure Visualizer

Left-click on **Update** in the **File** menu to update a structure visualizer. When you do this, the root node is re-read; Prism attempts to expand the same nodes that are currently expanded. (The same thing happens if you re-print an existing structure visualizer.)

Left-click on **Close** in the **File** menu to close the structure visualizer.

5.4.6 Visualizing Dynamic Arrays and Union Members

You can include special functions in your program to describe data structures to Prism in more detail than the compiler alone can provide. This

Copyright © 1991–1994 Thinking Machines Corporation

allows Prism's structure visualizer to display the additional information. Specifically, you can tell Prism:

- which union member of a structure is currently valid
- which pointers in a structure are really dynamically sized arrays

Note these points about the functions discussed in this section:

- Providing these functions is optional. Prism's structure visualizer still expands structure nodes without them, but the dynamic arrays look like pointers, and every union member is printed instead of only the currently valid member.
- You never call these functions directly; rather, Prism calls them as necessary.

Calling the Functions

You can provide a special function for each C structure type whose definition you wish to augment. The function has this definition:

```
int prism_define_name (pointer)
struct name *pointer;
```

where *name* is the type name of the structure.

When visiting a node of a structure, the structure visualizer first checks for the existence of such a function. If the function does not exist, the node is expanded normally. If the function does exist, it is called with one argument: a pointer to the specific instance of the structure type being expanded.

Your **`prism_define_`***name* function must call several auxiliary functions, defined in the Prism run-time library, to augment the structure's definition. These functions are:

- **`void prism_struct_init()`**

  Call this at the top of the function.

- **`int prism_struct_return()`**

  The function should end with

  ```
  return prism_struct_return();
  ```


Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

- **void prism_add_array (char *fieldname, int nelements)**

  Call this for each pointer field in the structure that is really a dynamic array. **fieldname** is the name of the pointer field of the structure, and **nelements** is the current length of the array pointed to by **fieldname**.

- **void prism_add_union (char *union_name, char *union_element)**

  Call this for each field of a structure that is of type **union**. **union_name** is the name of the member that is a union. **union_element** is the name of the field in the union that is currently valid.

An Example

This sample program shows the use of these functions.

```
struct Graph {
    int nlines;          /* length of lines array */
    float *lines;
    int nvertices;       /* length of vertices array */
    float *vertices;
};

struct Value {
  int type;         /* gives type of currently valid
                       union member */
  union {
    int ival;
    float fval;
    double dval;
  } val;
};

#define INTEGER 0
#define FLOAT 1
#define DOUBLE 2

prism_define_Graph(g)
struct Graph *g;
{
    prism_struct_init();
```


Copyright © 1991–1994 Thinking Machines Corporation

```
    prism_add_array( "lines", g->nlines);
    prism_add_array( "vertices", g->nvertices);

  return prism_struct_return();
}

prism_define_Value(v)
struct Value *v;
{
  prism_struct_init();

  if (v->type == INTEGER)
    prism_add_union( "val", "ival");
  else if (v->type == FLOAT)
    prism_add_union( "val", "fval");
  else if (v->type == DOUBLE)
    prism_add_union( "val", "dval");

  return prism_struct_return();
}
```

The functions **prism_define_Graph** and **prism_define_Value** provide additional information about the structure types **Graph** and **Value**, respectively.

5.5 Printing the Type of a Variable

Prism provides several methods for finding out the type of a variable.

From the menu bar: Choose the **Whatis** selection from the **Debug** menu. The **Whatis** dialog box appears; it prompts for the name of a variable. Click on **Whatis** to display the information about the variable in the command window.

From the source window: Select a variable by double-clicking on it or by dragging over it while pressing the left mouse button. Then hold down the right mouse button; a popup menu appears. Choose **Whatis** from this menu. Information about the variable appears in the command window.

From the command window: Issue the **whatis** command from the command line, specifying the name of the variable as its argument.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

5.5.1 What Is Displayed

Prism displays the information about the variable in the command window. If a CM Fortran array is CM-resident, that information is included. For example:

```
whatis primes
(CM based) logical primes(1:999)
```

5.6 Modifying Data

You can use the **assign** command to assign new values to a variable or an array. For example,

```
assign x = 0
```

assigns the value 0 to the variable **x**. You can put anything on the left-hand side of the statement that can go on the left-hand side in the language you are using — for example, a variable, a Fortran array section, or a C* left-indexed parallel variable.

If the left-hand side is a CM Fortran array or array section, the right-hand side must be a conformable expression. For example, if **array1** and **array2** are conformable,

```
assign array1 = array2 + 2
```

adds 2 to each element of **array2** and assigns the result to **array1**.

Similarly, if the left-hand side is a C* parallel variable, the right-hand side must be of the same shape. For example, if **p1** and **p2** are parallel variables of the same shape,

```
assign p1 = p2
```

assigns the value of each element of **p2** to the corresponding element of **p1**.

If the right-hand side does not have the same type as the left-hand side, Prism performs the proper type coercion.

Copyright © 1991–1994 Thinking Machines Corporation

5.7 Changing the Radix of Data

Use the command *value* = *base* to change the radix of a value in Prism. The value can be a decimal, hexadecimal, or octal number. Precede hexadecimal numbers with **0x**; precede octal numbers with **0** (zero). The base can be **D** (decimal), **X** (hexadecimal), or **O** (octal). Prism prints the converted value in the command window.

For example, to convert 100 (hex) to decimal, issue this command:

```
0x100=D
```

Prism responds:

```
256
```


Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation