# VISUAL2 User's & Programmer's Manual

Rev. 1.30
HP 9000/700 version

Bob Haimes, Mike Giles

July 17, 1992

# Introduction

VISUAL2 is an interactive graphics program for the visualization of two-dimensional data on unstructured grids. It has a FORTRAN and C programming interface, uses X11 for graphic output.

The user may optionally set an environment variable, Visual2_CP, which gives the file path to be searched for color files. For example, if the color files are /usr/local/color then it is done by the following command.

```
setenv Visual2_CP /usr/local/color
```

A final optional environment variable is Visual2_Key, which, if defined, is a character string which will be printed in the key window.

VISUAL2 requires three X fonts. The file '.Xdefaults' in the users home directory is examined for the font names and are designated "Visual*large", "Visual*med2" and "Visual*small". A sample file 'user.Xdefaults' on the distribution tape may be concatinated to the user's '.Xdefaults' file. NOTE: The X fonts loaded on any system may be examined by the command 'xlsfonts'.

The first section describes the user interface, the functionality of VISUAL2 and the way in which the user running the program can invoke the different capabilities. The second section describes the programming interface, the way in which the programmer couples VISUAL2 to the programmer's application program. The final section describes some programs for the post-processing of output files generated by VISUAL2.

The functions to be displayed can be of three types, scalar or vector functions defined on the whole two-dimensional grid, or scalar edge functions defined only at points on boundaries. At any instant there will 'active' scalar, vector and edge functions, which can be changed by pressing a key on the keyboard. The package is designed to handle data in which the functions, the underlying grid, and even the basic data structure may vary in time.

# 1 VISUAL2 User Interface

## 1.1 Windows

The user interface is divided into three different windows. As is typical of of X-window applications, the functions invoked by mouse button, keyboard or dialbox input are dependent upon the position of the cursor. Thus, different functions are available to the user depending on which window is 'active', i.e. contains the cursor. An important feature of the user interface is its help key. Pressing '?' will cause a list of the available commands for the active window to be displayed in the text window. The four different windows are:

- Text Window

  The text window is the window from which VISUAL2, or rather the application program, was started. It is a good idea to keep this window in the lower left corner of the screen where it will not be obscured by the other VISUAL2 windows. This window is used to output various messages, including the help menus, and to input filenames, numerical values etc.

- Primary Window

  This window displays two-dimensional data. Also, the user can 'interrogate' the numerical data using a variety of 'probes', to be discussed later. There are a wide varity of functions which are available when the cursor is in this window; press '?' for a list of them.

- Secondary Window

  This window is used to display one-dimensional data (produced by a 'probe') or secondary two-dimensional data (produced by the 'magnifying glass').

- Key Window

  This window displays the color index used in the primary window. When the cursor is in this window, there are many options available. These include the ability to interactively change the color scheme, load in a new color scheme, or change the length of displayed vectors.

## 1.2 Plotting in primary window

- Contour plot

  A Gouraud-shaded contour plot of the currently 'active' scalar variable is generated. This is dynamic if the grid or the function is unsteady.

- Grid plot

  The computational grid is displayed on top of whatever else is currently being plotted.

- Contour line plot

  Contour lines corresponding to the current scalar function are displayed on top of whatever else is currently being plotted.

- Vector plot

  Vectors corresponding to the current vector function are displayed on top of whatever else is currently being plotted.

- Interactive pan and zoom

  This can be under dial box or keyboard control.

- Interactive changes in function scaling and lookup tables

### 1.3 Interrogation in primary window; output in secondary window

- Point probe

  The position of the probe in the primary window is defined by the cursor. The output in the secondary window is a time history of the scalar variable being plotted in the primary window.

- Edge function probe

  This is similar to the point probe, but is for plotting edge function data. The probe is defined to be at the edge point closest to the current cursor location.

- Edge plot

  This plots the current scalar variable along the edge which is closest to the current cursor location. The output is dynamic if the data is unsteady.

- Edge function plot

  This is similar to the edge plot, but is for plotting edge function data. Again, the output is dynamic if the data is unsteady.

- Line probe

  The position of the line probe is controlled by the cursor. The output in the secondary window is the steady or unsteady scalar function values along the line. A subsidiary option for unsteady cases is to calculate and plot a time-average.

- Edge normal probe

  This is similar to the line probe, except that the line is normal to an edge. Like the edge function probe, one end of the line is defined to be at the edge point closest to the current cursor location. The line then extends normally out from the edge into the domain. The initial length is set by the initial distance of the cursor from the edge, but it can be changed by the user. The location of the whole line varies dynamically as the user moves the cursor, allowing the user to quickly scan around an entire surface examining fine scale details of data in a thin layer next to the boundary.

- Magnifying glass

  This is similar to the interactive pan and zoom, except that the primary window remains unchanged. The output in the secondary window is the magnified region.

## 1.4 Interrogation in primary window; with output in text window

- Point text probe

  This returns the coordinates and the scalar function value at the cursor location. For unsteady data this output is dynamic and will repeat until terminated by the user.

- Data structure check

  This option checks the self-consistency of the data structure and grid, and report on any inconsistencies.

## 1.5 Output files

- All line and vector plots in the 2D window can be output as line files. These files may be converted to Postscript for output to a printer.
- Rendered data in the both windows can be dumped into an output file, and post-processing programs can convert this into a Postscript file or a Macintosh TIFF file.
- All plots from the 1D window can be output as ASCII tabular files. These files may be converted to Postscript for output to a printer or used by any software that can take a table of numbers as input.

## 1.6 Input color files

A number of different color files are supplied on the distribution tape. For those who wish to define their own color files, the format of these ASCII files is as follows:

$$
\begin{array}{l}
nc\ \ nb \\
\left.\begin{array}{ccc} r & g & b \\ & . & \\ & . & \\ & . & \\ & . & \end{array}\right\} nc \\
\left.\begin{array}{ccc} r & g & b \\ & . & \\ & . & \\ & . & \\ & . & \end{array}\right\} nb
\end{array}
$$

where $nc$ is the number of colors, $nb$ is the number of background colors (0–4), and $r, g, b$ are red,green,blue intensity values (0.0–1.0). The four background colors are for window background; grid color; tuft/streamline color; contour line color. The default values which are used if $nb = 0$ are black; white; white; white. If $nb \neq 0$ then the specified colors over-ride the defaults for the first $nb$ colors.

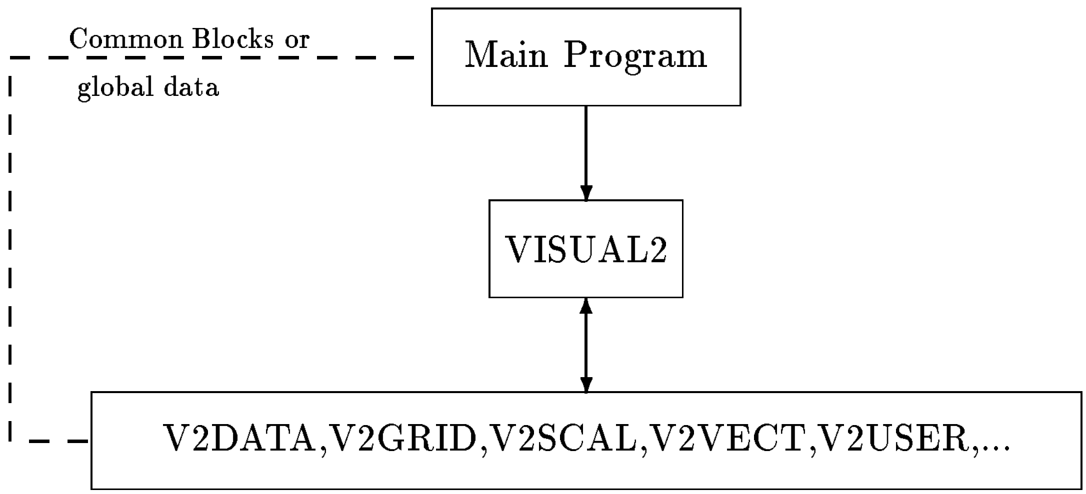

Figure 1: Flow diagram showing use of VISUAL2

# 2 Programming VISUAL2

## 2.1 Overview

The manner in which the user's program interacts with VISUAL2 is slightly unusual. As shown in Fig. 1 the main program first initializes and activates VISUAL2 by calling it as a subroutine. VISUAL2 then gets the data it requires from the programmer by calling a small number of programmer-supplied subroutines. These subroutines are connected to the main program only through COMMON blocks or global data in C. The reason for this unusual approach is that it keeps the interface between the main program and VISUAL2 as simple as possible, and allows VISUAL2 to be created as a concurrent process in parallel with the main program, while keeping the complexity of concurrency completely hidden from the programmer.

The main program will call VISUAL2 and transfer program control to it. VISUAL2 therefore remains in total control, and when it calls the programmer-supplied routines they return the grid coordinates and function values, etc. If the data is unsteady in some sense then each time VISUAL2 calls the programmerer-supplied routines they will return new values of the data. Control will only be returned to the main program when the user exits from VISUAL2.

## 2.2 VISUAL2 Data Structure

The VISUAL2 data structure has been chosen to provide the information required by VISUAL2 to perform its calculations efficiently, and to perform its graphical presentation as quickly as possible. Thus the data structure is intimately related to the low-level graphics functions provided in the PEX interface.

The VISUAL2 data structure begins with a set of grid nodes. KNODE is the number of nodes and XY(2,KNODE) are their coordinates. Scalar and vector variables defined at the nodes are given in the arrays S(KNODE) and V(2,KNODE).

For plotting the user's grid, VISUAL2 uses two different sets of PolyLines. A PolyLine is a list of node numbers which define a line of connected straight line segments. i.e., the first two node numbers, $n_1, n_2$ define the first line segment, the next (partially overlapping) pair of nodes, $n_2, n_3$, define the second line segment, and so on. If there are N nodes in a PolyLine list then these define a line with N$-$1 line segments.

The first set of PolyLines used by VISUAL2 define the interior faces of the user's grid. The one-dimensional integer array PFACE is a concatenation of all of the PolyLines which define the interior faces. PPFACE is a secondary one-dimensional integer array of pointers, which mark the end of each PolyLine list in PFACE. e.g., PPFACE(1) is the index of the last element of PolyLine 1 in the array PFACE. KPFACE is the number of PolyLines, and is also the dimension of PPFACE.

The second set of PolyLines used by VISUAL2 define the exterior edges of the user's grid in exactly the same manner as with the interior faces. The one-dimensional array PEDGE is a concatenation of the PolyLines which define the edges faces. PPEDGE is the secondary one-dimensional array of pointers, which mark the end of each PolyLine list in PEDGE, and KPEDGE is the number of PolyLines.

For contour plotting and rendering, VISUAL2 uses a set of 'PolyTri strips'. Each one of the these PolyTri strips is a list of node numbers, such that each consecutive group of three nodes define a triangle. i.e., the first three nodes numbers, $n_1, n_2, n_3$ define the corners of the first triangular cell, the next (partially overlapping) group of three nodes $n_2, n_3, n_4$ define the second triangle, and so on. If there are N nodes in the PolyTri list then these define a total of N$-$2 triangular cells. The one-dimensional array PTRI is a concatenation of all of the PolyTri strips which define the VISUAL2 grid. PPTRI is a secondary one-dimensional array of pointers, which mark the end of each PolyTri strip in PTRI. e.g., PPTRI(1) is the index of the last element of PolyTry strip 1 in the array PTRI. KPPTRI is the number of PolyTri strips, and is also the dimension of PPTRI.

The VISUAL2 grid differs from the user's original grid in that any quadrilateral cells (or higher order polygonal cells) in the original grid have been broken into triangular pieces for the VISUAL2 grid. This is necessary because the low-level PEX graphics instructions operate only on triangles.

Two additional one-dimensional arrays, of dimension KPTRI, are needed by VISUAL2 for some particular applications. The first, CTRI, provides information on the connections between PolyTri strips. For every triangle in each of the strips, its first and third nodes define a face which is not shared with the neighboring triangles in the same strip. Instead, there are two possibilities. Either the face is also a face of some other triangle, probably in some other strip, or it is an edge. In the former case, CTRI points to the other triangle. This is accomplished by setting CTRI($i$)=$j$, where $i$ is the index (in PTRI) of the second node of the first triangle, and $j$ is the index (again in PTRI) of the second node of the neighbor cell. In the latter case, CTRI points to the edge. This is accomplished by setting CTRI($i$)=$-j$, where $i$ is the index (in PTRI) of the second node of the first triangle, and $j$ is the index (in PEDGE) of the first node of the edge. The beginning and end of each PolyTri strip is handled in a similar manner. If $i$ is the index (in PTRI) of the beginning of a strip then the nodes $n_i, n_{i+1}$ form a face that is not interior to the strip, and so CTRI(i) points to the index of the neighboring cell which shares the face, or to the edge, which ever is the case. Similarly, if $i$ is the index (in PTRI) of the end of a strip then the nodes $n_i, n_{i-1}$ form a face that is not interior to the strip, and CTRI(i) points to the index of the neighboring cell or edge. The example below will hopefully be helpful in understanding these definitions for the construction of CTRI. The reason for defining CTRI is to permit the efficient calculation of pathlines which are everywhere tangent to the vector variables defined at either nodes or cells.

The second array, ITRI, identifies the original cell in the user grid from which each triangle has come. This is accomplished by setting ITRI($i$)=$j$, where $i$ is the index (in PTRI) of the second node of a triangle, and $j$ is the index of the original cell in the user grid. ITRI is undefined for the first and last nodes in each PolyTri strip. The importance of ITRI is that if two consecutive triangles in a PolyTri strip have the same values of ITRI, then they both come from the same quadrilateral cell originally. This is needed only when plotting vector variables at cell centers when one wants the vector to be located at the center of the original quadrilateral, and not at the centers of both of the VISUAL2 triangles into which the quadrilateral cell has been divided. This places a restriction upon the user to ensure that whenever an original quadrilateral cell is divided into two triangles, then those triangles are consecutive triangles in a PolyTri strip.

To understand all of the above definitions, consider an example of a data structure for the very simple $3 \times 4$ rectangular 'user grid' shown in Fig. 2. The PolyTri strips are defined by bisecting each quadrilateral cell, as shown in Fig. 3. The figure also shows the node numbering system which will be used. The VISUAL2 data to define this case is as follows:

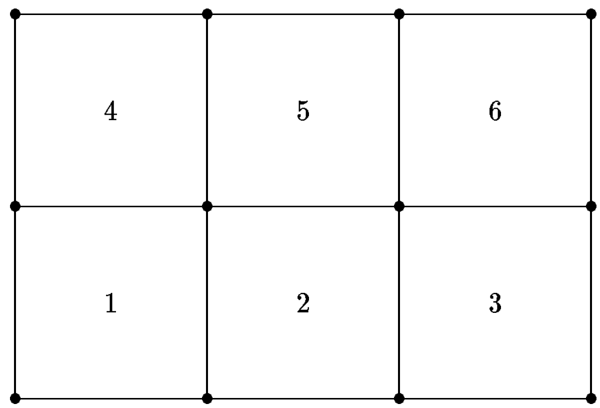


Figure 2: Rectangular user grid with cell numbering

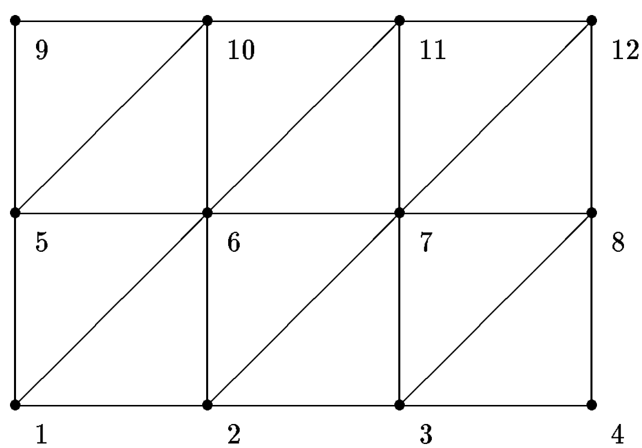


Figure 3: Triangular VISUAL2 grid and node numbering

$\mathrm{KNODE} = 12$

$$\mathrm{XY} = \begin{array}{lllllll} ( & (0.0, 0.0) & (1.0, 1.0) & (0.0, 2.0) & (0.0, 3.0) & (1.0, 0.0) & (1.0, 1.0) \\ & (1.0, 2.0) & (1.0, 3.0) & (2.0, 0.0) & (2.0, 1.0) & (2.0, 2.0) & (2.0, 3.0) \quad ) \end{array}$$

$\mathrm{KFACE} = 10$

$\mathrm{PFACE} = (\ 5\quad 6\quad 7\quad 8\quad 2\quad 6\quad 10\quad 3\quad 7\quad 11\ )$

$\mathrm{KPFACE} = 3$

$\mathrm{PPFACE} = (\ 4\quad 7\quad 10\ )$

$\mathrm{KEDGE} = 11$

$\mathrm{PEDGE} = (\ 1\quad 2\quad 3\quad 4\quad 8\quad 12\quad 11\quad 10\quad 9\quad 5\quad 1\ )$

$\mathrm{KPEDGE} = 1$

$\mathrm{PPEDGE} = (\ 11\ )$

$\mathrm{KPTRI} = 16$

$\mathrm{PTRI} = (\ 5\quad 1\quad 6\quad 2\quad 7\quad 3\quad 8\quad 4\quad 9\quad 5\quad 10\quad 6\quad 11\quad 7\quad 12\quad 8\ )$

$\mathrm{KPPTRI} = 2$

$\mathrm{PPTRI} = (\ 8\quad 16\ )$

$\mathrm{CTRI} = (\ -10\quad 11\quad -1\quad 13\quad -2\quad 15\quad -3\quad -4\quad -9\quad -8\quad 2\quad -7\quad 4\quad -6\quad 6\quad -5\ )$

$\mathrm{ITRI} = (\ 0\quad 1\quad 1\quad 2\quad 2\quad 3\quad 3\quad 0\quad 0\quad 4\quad 4\quad 5\quad 5\quad 6\quad 6\quad 0\ )$

### 2.3 Programming Overview

VISUAL2 was designed to be accessible from both FORTRAN and C. FORTRAN is more restrictive in argument passing and naming, therefore it has shaped the programming interface. Also the following routine descriptions are from the FORTRAN programmer's point of view. All subroutines internal to VISUAL2 have names which begin with 'V2_' or 'XFtn' and the common blocks have names which begin with 'V2C': such names should be avoided in an application program. It may also be helpful to look at the source code of the demo programs on the distribution tape. Each one is well commented and suitable for use as a template for other applications.

In describing the arguments of the routines in the next sections, the following notation has been used. The variable name is first followed by 'i' or 'o', indicating input and output, respectively, showing whether or not the subroutine is to set the variable. The variable is then followed by an expression indicating the variable type (L=logical, I=integer, R=real, C=character) and its dimensionality (e.g. R(2,MNODE) indicates a two-dimensional real array with the first dimension being 2 and the second MNODE, where MNODE is an integer variable).

## 2.4 Programmer-called subroutines

**V2_INIT(TITL, IOPT, CMNCOL, CMFILE, CMUNIT,**
**XYPIX, XYMIN, XYMAX,**
**NKEYS, IKEYS, TKEYS, FKEYS, FLIMS,**
**MNODE, MPTRI, MPPTRI,**
**MFACE, MPFACE, MEDGE, MPEDGE)**

This subroutine initializes VISUAL2 and defines the active keys on the keyboard.

| | |
|---|---|
| TITL:I: C | Title |
| IOPT:I: I | Control parameter |
| | IOPT=0 Steady data structure, grid and variables |
| | IOPT=1 Steady data structure and grid, and unsteady variables |
| | IOPT=2 Steady data structure, and unsteady grid and variables |
| | IOPT=3 Unsteady data structure, grid and variables |
| | IOPT>0 This is the synchronous mode in which every time VISUAL2 goes through its main processing loop it calls V2UPDATE to allow the programmer-supplied routine to update the variables and/or 'time'. In this case V2_INIT returns to the user's main program only when VISUAL2 is terminated interactively by the user. |
| CMNCOL:I: I | Maximum number of color entries to be used in color map |
| CMFILE:I: C | Name of default color map file |
| CMUNIT:I: I | Logical unit number for file operations |
| XYPIX:I: I(2) | Number of pixels in x,y directions for main window |
| XYMIN:I: R(2) | Minimum x,y values for main window |
| XYMAX:I: R(2) | Maximum x,y values for main window |
| NKEYS:I: I | Number of active keyboard keys |
| IKEYS:I: I(NKEYS) | X-keypress return code for each key |
| TKEYS:I: C(NKEYS) | Menu title for each key |
| FKEYS:I: I(NKEYS) | Type of function controlled by each key |

| | |
|---|---|
| | FKEYS()=1 Scalar function at grid nodes |
| | FKEYS()=2 Scalar function at cells |
| | FKEYS()=3 Vector function at grid nodes |
| | FKEYS()=4 Vector function at cells |
| | FKEYS()=5 Edge function at edge nodes only |
| FLIMS:I: I(2,NKEYS) | Function limits/scales |
| | FKEYS()=1,2,5 Min and max values of function |
| | FKEYS()=3,4 scale factor for vectors (second variable is unused) |
| MNODE:I: I | Maximum number of nodes |
| MPTRI:I: I | Maximum total size for all PolyTri strips |
| MPPTRI:I: I | Maximum number of PolyTri strips |
| MFACE:I: I | Maximum total size for all face PolyLines |
| MPFACE:I: I | Maximum number of face PolyLines |
| MEDGE:I: I | Maximum total size for all edge PolyLines |
| MPEDGE:I: I | Maximum number of edge PolyLines |

**V2_CURSOR(FLAG)**

This subroutine can be used by certain programmer-supplied routines when the programmer wants to add a function which needs user input from the text window.

| | |
|---|---|
| FLAG:I: L | **FLAG=.TRUE.** Move cursor to text window |
| | **FLAG=.FALSE.** Move cursor back to previous window |

## 2.5 Programmer-supplied subroutines

### 2.5.1 Required subroutines

**V2DATA(KNODE, PTRI, CTRI, ITRI, KPPTRI, PPTRI, PFACE, KPFACE, PPFACE, PEDGE, KPEDGE, PPEDGE)**

This subroutine supplies VISUAL2 with the grid data structure.

| | |
|---|---|
| KNODE:o: I | Number of nodes |
| PTRI:o: I(KPTRI) | PolyTri pointers to nodes |
| CTRI:o: I(KPTRI) | PolyTri pointers to neighboring cells. This is an optional input; if not supplied the streamline (integrated vector field lines) option is disabled. |
| ITRI:o: I(KPTRI) | PolyTri pointers to original cells |
| KPPTRI:o: I | Number of PolyTri strips |
| PPTRI:o: I(KPPTRI) | Pointers to ends of strips |
| PFACE:o: I(KFACE) | Face PolyLine pointers to nodes |
| KPFACE:o: I | Number of face PolyLines |
| PPFACE:o: I(KPFACE) | Pointers to ends of face PolyLines |
| PEDGE:o: I(KEDGE) | Edge PolyLine pointers to nodes |
| KPEDGE:o: I | Number of edge PolyLines |
| PPEDGE:o: I(KPEDGE) | Pointers to ends of edge PolyLines |

**V2GRID(XY,XYDELT,XYMINP,XYMAXP, XYMINF,XYMAXF,XYMINE,XYMAXE)**

This subroutine supplies VISUAL2 with the grid coordinates.

| | |
|---|---|
| XY:o: R(2,KNODE) | $x, y$-coordinates of grid nodes |
| XYDELT:o: R(2) | Maximum width (Dx) and height (Dy) of any cell (optional) |
| XYMINP:o: R(2,KPPTRI) | Minimum values of $x, y$ in each PolyTri strip (optional) |
| XYMAXP:o: R(2,KPPTRI) | Maximum values of $x, y$ in each PolyTri strip (optional) |
| XYMINF:o: R(2,KPFACE) | Minimum values of $x, y$ in each face PolyLine (optional) |
| XYMAXF:o: R(2,KPFACE) | Maximum values of $x, y$ in each face PolyLine (optional) |

XYMINE:O: R(2,KPEDGE) Minimum values of $x, y$ in each edge PolyLine (optional)

XYMAXE:O: R(2,KPEDGE) Maximum values of $x, y$ in each edge PolyLine (optional)

If the user supplies the optional information, VISUAL2 can perform all rendering and drawing much more efficiently. If the user chooses not to supply the information, the variables, (both min and max) should be returned with their values unchanged.

### 2.5.2 Optional subroutines

**V2SCAL(JKEY,S)**
This subroutine supplies VISUAL2 with scalar function values.

| | |
|---|---|
| JKEY:I: I | Index of key, relative to ordering specified in V2_INIT. (i.e. first key is 1, second is 2, third is 3, etc.) |
| S:O: R(KNODE),R(KPTRI) | Scalar function at nodes or cells |

**V2VECT(JKEY,V)**
This subroutine supplies VISUAL2 with vector function values.

| | |
|---|---|
| JKEY:I: I | Index of key, relative to ordering specified in V2_INIT. (i.e. first key is 1, second is 2, third is 3, etc.) |
| V:O: R(2,KNODE),R(2,KPTRI) | Vector function at nodes or cells |

**V2EDGE(JKEY,S,IEDGE)**
This subroutine supplies VISUAL2 with edge function values.

| | |
|---|---|
| JKEY:I: I | Index of key, relative to ordering specified in V2_INIT. (i.e. first key is 1, second is 2, third is 3, etc.) |
| S:O: R(*) | Scalar function at edge nodes |
| IEDGE:I: I | Number of edge line for which scalar data is required; for example, if $IEDGE = 3$ then it is the third line, and the dimension of $S$ will be $PPEDGE(3) - PPEDGE(2)$. |

**V2UPDATE(TIME)**

This subroutine is called by VISUAL2 in synchronous unsteady applications, IOPT>0, every time VISUAL2 begins the main processing loop. This allows the programmer to update variables and/or update 'time'.

| | |
|---|---|
| TIME:O: R | The current simulation time. This sets the time in VISUAL2 for bubble integration. If the time between calls does not change, VISUAL2 takes this as a indication that the data has not been updated and does not call V2SCAL, V2VECT and etc. |

**V2USER(ETYPE,ESTAT,XYCURS)**

This subroutine gives the user a method of accessing a user-supplied routine to process X-events before they are handled by VISUAL2. One particular use of this capability is to change parameters in the user's main program, when doing co-processing.

| | |
|---|---|
| ETYPE:I: I | X-event type |
| ESTAT:I: I | X-event state |
| XYCURS:I: R(2) | x,y coordinates of cursor |

**V2USERHELP**

This subroutine allows the user to write any message to the text window which is unit *. (i.e. use the Fortran statement WRITE(*,*) '....') This message will appear after VISUAL2's own help message whenever the user types a question mark.

this subroutine has no arguments

**V2STRING(STRING)**

This subroutine allows the user to provide a label for all plots, in addition to the title supplied to V2_INIT. This is particularly useful for labelling plots with the time in unsteady applications.

| | |
|---|---|
| STRING:O: C*80 | character string label |

**V2PEDGES(NUMEDGE)**

By default VISUAL2 plots all edges in the main window when generating hardcopy output.

If the user has internal grid boundaries and does not want them plotted, this routine allows the user to specify that only the first NUMEDGE edges will be plotted; the remainder will not be plotted.

| | |
|---|---|
| NUMEDGE:o: I | number of edges to be plotted |

### 2.6 QAT2V2

QAT2V2 (whose name stands for Quadrilaterals And Triangles to V2) is a subroutine which is supplied along with VISUAL2 to aid the user in setting up the data required by VISUAL2. It is not needed for the use of VISUAL2 and is simply to save the user some programming effort. Its function is to take input data which defines an unstructured grid as a mix of quadrilateral and triangular cells, and generate the PolyTri strips, Polylines, edge data and connection data needed for the VISUAL2 data structure. The recommended place to use this routine is <u>inside</u> the routine V2DATA which the user must supply to VISUAL2. This is best because VISUAL2 has already allocated memory for almost all of the necessary arrays, so this minimizes memory requirements.

The argument list is:

**QAT2V2(PCELL, WCELL, KCELL, WNODE, KNODE,WFACE,**
**MTRI, MPTRI, MEDGE, MPEDGE, MFACE, MPFACE,**
**PTRI, CTRI, ITRI, KTRI, PPTRI, KPTRI,**
**PEDGE, CEDGE, KEDGE, PPEDGE, KPEDGE,**
**PFACE, KFACE, PPFACE, KPFACE)**

| | |
|---|---|
| PCELL:I: I(4,KCELL) | cell pointers. Each entry contains the node numbers that make up the cell. For triangular cells PCELL(1,n) must equal PCELL(4,n). For quadrilateral cells the pointers must be ordered clockwise or counter-clockwise. |
| WCELL: R(KCELL) | work array |
| KCELL:I: I | number of cells |
| WNODE: R(KNODE) | work array |
| WFACE: R(6*MFACE) | work array |
| CEDGE:O: R(MEDGE) | Polyline Edge connection array analogous to CTRI, giving pointer from edge faces to neighboring cells: not needed by VISUAL2, but needed as a work array for QAT2V2 and useful output in some applications. |

All other variables are as defined in earlier subroutine descriptions. The first two lines of the argument list are inputs, the others are outputs. This really simplifies the user's task in generating a VISUAL2 data structure to simply supplying a list of unstructured cells.

### 2.7 C Programming

Using VISUAL2 with C requires that the programmer do some things that are not intuitive. This is because FORTRAN is also supported and some of VISUAL2 was written in FORTRAN.

The following rules must be followed:

- main program

  The main program must be called 'main' and have no arguments. This is so that the loader (invoked with f77) treats the program as a FORTRAN main program and initializes the FORTRAN I/O system. If the command line is required, the FORTRAN routine 'igetarg' must be used.

- routine names

  All FORTRAN subroutine and function names are lower case. FORTRAN is case insensitive and the compiler generates external names in lower-case.

  Since there are dummy entries in the VISUAL2 library for all optional routines, not getting the name of a programmer-supplied correct will still produce a complete executable. The supplied routine will not be accessed!

- argument passing

  FORTRAN expects all arguments to be passed by reference, not value. Character variables also have their length appended to the stack (end of the call) in the order that character arguments appear in the call. These lengths are passed by value.

  The only routine this effects is 'v2_init'. The call must have 3 more arguments; the length of TITL, the length of CMFILE, and the length of TKEYS. These are passed by value.

- logicals

  FORTRAN logicals are ints with a value of 0 for .FALSE. and -1 for .TRUE. .

- character strings

  FORTRAN character strings have a specified length (hence passing the length). If the string is not fully used, it is padded with blanks. The null byte that terminates a C string gets interpretted by FORTRAN as a zero character. To avoid passing the null either overpad the string with blanks (above the size used by 'v2_init'), specify the length as the position before the null, or remove the null. For programmer supplied routines with character arguments ('v2string'), the strings must be padded with blanks to the specified length.

- arrays

  FORTRAN array indexing, by default, starts at 1. Also in FORTRAN, the left-most index produces addresses that are consecutive in memory. Therefore, when filling multiple dimension arrays, reverse the order of the indices as documented and start at index zero.

  Note: key, node and cell numbering MUST start at 1. This gives an offset of 1 between the index and the number.

- linking

  Use f77 to link and produce an executable. This invokes the FORTRAN libraries. Also the math library must be serached "-lm".

# 3 Post Processing

There are three types of output files generated by VISUAL2 by appropriate function keys in different windows. The first is tabular output which is generated from the 1D window. This is an ASCII file suitable for inclusion into most line plotting or spread-sheet software. The default file name is 'visualXYZ.tab'. A post-processing program 'tab2ps' produces Postscript output.

The second file type is an image dump of an individual window. This file is written in a FORTRAN unformatted manner with the default name of 'visualXYZ.img'. This can be converted into Postscript (using 'img2ps') or Macintosh TIFF files (using 'img2tiff').

The last file type is used for line drawings. This file has the default name 'visualXYZ.lin' and is also written unformatted. A post-processing program 'lin2ps' produces Postscript output, with hidden line removal.

The following section describes the usage of the post-processors supplied with VISUAL2. It should be noted that the source and make-files for the post-processors have been included on the distribution tape. By making small modifications to the sources, other output devices can be supported with no changes to VISUAL2.

## 3.1 tab2ps

tab2ps takes as an argument a tabular output file name and produces PostScript on standard output. The output may be redirected to a file or piped directly to the printer spooler.

examples: % tab2ps visual002.tab | lp
          % tab2ps visual002.tab > tab.ps

## 3.2 img2ps

img2ps takes as the first argument the image file name. Additional arguments are options controlling the translation from True/PseudoColor to 8-bit Black and White. The output of img2ps is PostScript and is written to standard output. The output may be redirected to a file or piped directly to the printer spooler.

The options are as follows:

**colorfile** If no color file is specified, the translation takes the bit value of each pixel in the dump file and maps it linearly into a gray-scale value. If a color file is specified, the translation maps the bit value to a color, and then maps the color intensity to a gray-scale. This is particularly important when using non-linear gray-scale color maps.

**-4** produce 4-bit gray-scale output, reducing the size of the output file

**-i** invert gray-scale output intensity

examples: % img2ps visual001.img | lp
% img2ps visual001.img spec.col -4 > img.ps

Notes:
1) PostScript printers such as Apple's LaserWriter IInt or DEC's LN03 (printers with only 2 MBytes) may only be able to produce hard copy using the -4 option.

## 3.3 img2tiff

img2tiff takes as its first argument the PseudoColor image file name, the second argument is the output TIFF file name, and the third is the color file name.

examples: % img2tiff visual001.img output.tiff spec.col

## 3.4 lin2ps

lin2ps takes as an argument a VISUAL2 line output file name and produces PostScript on standard output. The output may be redirected to a file or piped directly to the printer spooler.

examples: % lin2ps visual005.lin | lp
% lin2ps visual005.lin > lin.ps