NASA-CR-197461

Visualization of Unsteady Computational Fluid Dynamics

Final Techincal Report
for
Grant # NAG2-884

Submittted

by

Robert Haimes

Computational Aerospace Sciences Laboratory
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology
Cambridge, MA 02139

November 1994

Performance Report for:
Visualization of Unsteady CFD

1. Introduction

The current compute environment that most researchers are using for the calculation of 3D unsteady Computational Fluid Dynamic (CFD) results is a super computer class machine. The super computer as well as Massively Parallel Processors (MPPs) and clusters of workstations acting as a single MPP (by concurrently working on the same task) provide the required computation bandwidth for CFD calculations of transient problems. The cluster of Reduced Instruction-Set Computers (RISC) is a recent advent based on the low-cost and high-performance that the workstation vendors are providing. The cluster, with the proper software can act as a MIMD (Multiple Instruction/Multiple Data) machine.

Work is in progress on a new set of software tools designed specifically to address visualizing 3D unsteady CFD results in these compute environments. The parallel version of **Visual3**, **pV3** required splitting up the unsteady visualization task to allow execution across a network of workstation(s) and compute servers. In this computing model, the network is always the bottleneck so much of the effort involved techniques to reduce the size of the data transferred between machines. **pV3** revision 1.00 has been released and delivered to NASA Ames.

The software used for the movement of data across the network is currently **PVM** from Oak Ridge National Laboratory. **PVM** (parallel virtual machine) is public domain software that provides the mechanisms required to transform a (heterogeneous) network of machines to one parallel computer. **PVM** provides all the required *hooks* including master/slave paradigms as well as peer-to-peer models, message passing, synchronization and the ability to target certain machines to specific tasks.

2. Progress on pV3

pV3 is a long term project that is funded by a number of sources. The bulk of the time spent on this contract has been devoted to getting **pV3** to the base level of functionality. Revision 1.0 was released toward the end of this contract and is currently being used on the NAS IBM SP2.

The following design goals for **pV3** were met:

- High Performance
  Take advantage of the proper hardware to get the best performance out of the entire compute arena. It is viewed as foolish to have overall compute performance of gigaflops and only the ability to do 2D graphics. **pV3** requires graphics hardware so that scene rendering time is not a limitation and the data presented to the investigator is of high quality and timely.

- Interactive
  The goal of any scientific visualization package should be to allow the assimilation of the vast amounts of data produced by the models and solvers in order to better understand the underlying physics. The ultimate goal, with this new knowledge, is to affect design and produce a better car, aircraft, gas-turbine engine, etc. This can only be done by poking and probing into the data to interrogate areas of interest

- Co-processing
  An important part of **pV3** is the ability to visualize the data as the solver or model progresses in time. It is also designed to allow the solver to run as independently as possible. If the solution procedure takes hours to days, **pV3** can *plug-into* the calculation, allow viewing of the data as it changes, then can *unplug* with the worst side-effect being the temporary allocation of memory and a possible load imbalance.

  This also has the advantage that all the data need not be stored and then played back to get a continuous and smooth viewing of the data. The data required can be 10s to 100s of gigabytes putting a huge storage equipment (and financial) burden on the compute facility. If the solver is fast enough (on the order of an iteration a second or less), then only a coarse sampling of data in time need be placed on disk.

- **Visual3** functionality
  pV3 provides the same kind of functionality as **Visual3** with the same suite of tools and probes. The data represented to the investigator (the 3D, 2D and 1D windows with cursor mapping) is the same. Also the same Graphical User Interface (GUI) is used.

- **Visual3**-like programming
  Another goal for **pV3** that has been met is that the programming be very **Visual3**-like. For the desired flexibility and the merging of the visualization with the solver, some programming is required. The coding is simple; like **Visual3**, all that is required of the programmer is the knowledge of the data. Learning the details of the underlying graphics, data extraction, and movement (for the visualization) is not needed. If the data is distributed in a cluster of machines, **pV3** deals with this, resulting in few complications to the user.

Though not directly associated with building **pV3**, work was also done on the suitability of various integration schemes for tracking unsteady particle paths. Multi-step, multi-stage, and some hybrid schemes were considered. It was shown that due to initialization errors, many particle path integration schemes are limited to third order accuracy in time. Multi-stage schemes require at least three times more data storage than multi-step schemes of equal order. However, for timesteps within the stability bounds, multi-stage schemes are generally more accurate. A linearized analysis reveals that the stability of these integration algorithms are determined by the eigenvalues of the local velocity tensor. Thus, the results can be interpreted with concepts typically used in critical point theory. Based on the linear analysis and practical experience, some approximate rules can be given for the timestep size necessary for accurate particle path integration. A not surprising result is that the timestep limitation for the particle integration is not unlike the timestep required for an explicit CFD solver. This makes **pV3**'s co-processing a very important feature.

3. pV3 Status

The current state of **pV3** is that it has almost the same functionality as **Visual3**. The following denotes the differences:

Planar Cut - **pV3** allows 2D viewing during positioning.
Structure Unsteady - not supported by **Visual3**.
Cell Based Visualization - **pV3** will not support these tools.
StreamLines - completely revised. Allows instantaneous SLs for unsteady cases.
Particles - more general seeding. Allows material lines. Coloring by time.
Histogramming - not implemented, not targeted for Rev 1.00.
Line output files - not implemented, not targeted for Rev 1.00.

3.1 Supported Machines

At **pV3** Rev 1.00, the following machines are supported as 'clients' (the computers containing the volume of data and performing the solver):

- CONVEX
- DEC Alphas running OSF/1
- DEC Stations (MIPS) running ULTRIX
- HP 9000/700 series at HP-UX 9.0 (or higher)
- IBM RS/6000s including the SP1s and SP2s
- KSR-1 or KSR-2
- SGI 4D Series, PI, Indigo, Indy, Power Series, Crimson, Onyx or Challenge running IRIX 5.0 (or higher)
- SUN (Sparc 2 or Sparc 10)

Currently, the only machines supported as the **pV3** server are SGI workstations with 3D graphics support. This is because SGI supplies good 3D graphics performance and supports 'multi-threading'. The server was designed to run as two independent execution streams that share the same address space. This allows the graphics to run concurrently with the network support required for distributed visualization. SUN SPARCs, DEC Alphas (under OSF/1) and IBM RS/6000s at AIX 4.1 support 'multi-threading', so these machines are possible candidates for future server ports.

3.2 Presentations

pV3: A Distributed System for Large-Scale Unsteady CFD Visualization, AIAA 94-0321 AIAA Aerospace Siences Meeting & Exhibit, Reno, January 1994.

An Analysis of Particle Path Integration Algorithms for Unsteady Data, NAS New Technology Seminar series, NASA Ames, August 1994. Submitted to AIAA CFD Conference, San Diego, July 1995.

3.3 pV3 Demonstrations

AIAA Aerospace Sciences Meeting & Exhibit, Reno, January 1994.
HPCN '94, Munich, April, 1994.
CFD '94 & SS '94, Toronto, June, 1994.
Supercomputing '94, Washington DC, November 1994.

4. Manuals

The three **pV3** manuals follow. The *Server User's Reference* is used by the investigator sitting in front of the visualization workstation. The *Programmer's Guide* informs the person grafting **pV3** to the solver what needs to be changed. The *Advanced Programmer's Guide* aids in the customization of the **pV3** system including adding new tools and probes.

pV3 Server User's Reference Manual

Rev. 1.00
for use with Silicon Graphics workstations

Bob Haimes
Massachusetts Institute of Technology

September 1, 1994

License

Contents

1 Introduction

pV3 is the newest in a series of graphics and visualization tools to come out of the Department of Aeronautics and Astronautics at MIT. Like it's predecessors **Visual3**, **Visual2** and **Grafic**, **pV3** is a software package aimed at aiding in the analysis of a particular suite of problems. In this case it is the real time visualization of 3D large scale solutions of transient (unsteady) systems.

pV3 (which stands for parallel **Visual3**), is a completely new system, but builds heavily on the technology developed for **Visual3**. It has been designed specifically for co-processing visualization of data generated in a distributed compute arena. It is also designed to allow the solver to run as independently as possible. If the solution procedure takes hours to days, **pV3** can 'plug-into' the calculation, allow viewing of the data as it changes, then can 'unplug' with the worst side-effect being the temporary allocation of memory and a possible load imbalance.

pV3 provides the same kind of functionality as **Visual3** with the same suite of tools and probes. The data represented to the investigator (the 3D, 2D and 1D windows with cursor mapping) is the same. Also the same Graphical User Interface (GUI) is used.

pV3 programming is very **Visual3**-like. For the desired flexibility and the merging of the visualization with the solver, some programming is required. The coding is simple; like **Visual3**, all that is required of the programmer is the knowledge of the data. Learning the details of the underlying graphics, data extraction, and movement (for the visualization) is not needed. If the data is distributed in a cluster of machines, **pV3** deals with this, resulting in few complications to the user.

1.1 pV3 Startup

The **PVM** daemon(s) and with co-processing, the solver, must be executing. Without the **pV3** server running, every time the solution is updated, a check is made for the number of members in the group *ServerPV3*. If none is found, no action is taken. When the **pV3** server starts (interactively from an xterm window on the graphics workstation), it enrolls in that group. The next time the solution is updated, an initialization message is processed and the session begins. Each subsequent time in the solver completes a time step, visualization state messages and *extract* requests are gathered, the appropriate data calculated, collected and sent to the server.

When the user is finished with the visualization, the server sends a termination message and exits. The clients receive the message and clean up any memory allocations used for the visualization. Then the scheme reverts to looking for server initialization if termination was not specified at **pV3** client initialization.

1.1.1 Server Startup

The server, *pV3Server*, takes two arguments at the command line (both optional). The first is the *setup* file with the default of 'pV3.setup'. The second argument is the *color* file to be used at startup (the default is 'spec.col').

```
examples: % pV3Server case.setup
          % pV3Server pV3.setup bw.col
          % pV3Server
```

1.1.2 Environment Variables

The **pV3** server uses three Unix environment variables. Some are the same as the ones used for **Visual3**. The variable 'Visual3_CP' defines the file path to be searched for color files, if they are not in the user's current directory. This allows all of the color files to be kept in one system directory.

The second variable, 'Visual_KB' is optional. This variable, if defined, must point to a file that contains alternate keyboard bindings for the special keys used by **pV3**. The file is ACSII. The first column is the key name (10 cahracters) and the second is the X-keysym value in decimal (use 'xev' to determine the appropriate values for the key strokes).

The third is 'pV3_TO' and should be used to change the internal Time-Out constant. If the variable is set, it must be an integer string which is the number of seconds to use for the Time-Out constant (the server's default is 60). This may be required if the time between solution updates is long. See the section on *Time-Outs and Error Recovery*.

1.1.3 Special Files

- .Xdefaults

  If the SGI window manager is used, '4DWm' must be told what to do with **pV3**s windows. These commands must be placed in the file '.Xdefaults'. See the file 'user.Xdefaults'.

  pV3 requires three X fonts. The file '.Xdefaults' in the users home directory is examined for the font names and are designated "Visual*large", "Visual*medium" and "Visual*small". The sample file 'user.Xdefaults' comes with the distribution and may be concatinated to the user's '.Xdefaults' file.

  The X fonts loaded on any system may be examined by the command 'xlsfonts'.

- 'twm' Setup File

  If the 'twm' window manager is used, when a user begins a session, twm reads an initialization file '.twmrc' in the user's home directory. This file defines certain key bindings and window attributes. It is necessary for correct **pV3** operation for the user to modify the standard '.twmrc' file as shown in the 'user.twmrc' file on the **pV3** distibution, or just use this file as '.twmrc'.

- Setup File

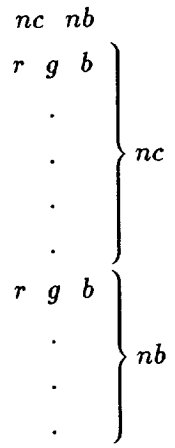
  When the **pV3** server starts, it looks in the user's current directory for the setup file specified as the first argument on the command line. This is an ASCII file which contains a number of useful defaults that the user may want to set to different values than **pV3**'s initial defaults. It also contains a set of viewing positions and cutting plane positions. Normally this file is generated by **pV3** when the user wants to store certain favorite parameters and viewing positions so that they can be used again on another data set, which is particularly useful when the user wishes to directly compare two different data sets with the same computational grid or geometry. However, an experienced user can also generate this file from scratch. NOTE: This file is NOT compatible with **Visual3**'s setup file.

- Lock File

  If the server is running on a multi-processor workstation (PowerSeries or Onyx) a file is used for the coordination of the 2 threads generated during execution. This file has the name '.pV3.locks' and is open in the current directory. It should be noted that running two invokations of the **pV3** server from the same directory will NOT work. Both will use the same file for the lock and semaphore arena!

- Color Files

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



  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/ribbon 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.

  When **pV3** searches for named color files, it first looks in the current directory, and then follows the color file path specified by the environment variable 'Visual3_CP'.

1.2 pV3 Time-Outs and Error Recovery

The **pV3** system was designed to be as error-free and as robust as possible. Because the client software runs closely coupled to the software generating the data, extra care has been taken to avoid causing any errors or problems related to this visualization system.

If the **pV3** server aborts, the client side software will NOT hang waiting for the completion of some message stream. In general, the server sends a series of request messages framed by an 'end-of-requests' message. All messages are received by the client software by either a tight loop or timed-out receive. In the tight loop, there is first a check to see if the server is still active. If so, the the next message is pulled of the message queue. If there is none, the server's presence is checked again, and so on until a message is received. Control is returned to the solver when either the 'end-of-requests' message is received or the server terminates (either gracefully or just disappears).

In the timed receive, an amount of time is specified for the receive. If this time is exceeded before a message is collected, it is assumed that something is wrong and the client shuts down the visualization (just as if the user terminated the session).

The type of message handling is controlled by the server's time-out constant. This is set by the environment varaible 'pV3_TO'. If the constatnt is set to 0 (zero), then the client and server do not time out, and are in hard loops. If the value is some positive number, both the client(s) and server will time out if something gets hung. If the time-out constant is negative, the server will time-out after the (negated) number of seconds and the client(s) are put in a hard loops. The default is 60 (seconds).

The choice of what to use depends on the network, the type of machine(s), whether the machines are dedicated, and the cost of putting software in tight loops waiting for some response.

If you are using tight loops and **pV3** seems to hang, the only way to free things up is to abort the server. If the server seems hung, hitting *Ctrl-C* in the window that started the server should work.

NOTE: The server runs as two threads (there are two unique PIDs for the task). If for some reason the IO thread aborts, it may leave the windows up but give a prompt in the window where the server was started. This can usually be fixed by hitting 'Esc' in the 3D window or by *kill*ing the process left. If the graphics thread aborts (the windows will disappear) interrupt the IO thread by hitting *Ctrl-C*.

2 The pV3 Server Users Interface

In trying to understand the following description of the user interface, it may be helpful to run one of the demo programs found in the example directory of the distribution. It is necessary to get **PVM** going then startup one of the example clients. After the client is running the server (*pV3Server*) can be started.

2.1 Surfaces

pV3 deals with three different types of surfaces. The first category is *domain* surfaces. These are surfaces that are defined by the client program(s) during **pV3** initialization, and they typically correspond to the surfaces which bound the computational domain. A subset of this first class, are *mapped domain* surfaces, for which there is a mapping from points on the surface to an (x', y') coordinate system. This allows plotting of surface quantities in a 2D setting.

The second category is *dynamic* surfaces. These are surfaces whose orientation and position, relative to the computational domain, can be changed interactively by the user. Although there are several types of *dynamic* surfaces, only one *dynamic* surface can exist at one time. Also, a *dynamic* surface cannot be activated when a *mapped domain* surface is being plotted in the 2D window.

The third category is *static* surfaces. These are surfaces which at one time were *dynamic*, but then transferred into the surface database, along with the *domain* surfaces. These *static* surfaces are then treated in almost the same way as the *unmapped domain* surfaces.

NOTE: In **Visual3**, any *static* surface in a *grid unsteady* application deformed with the grid movement. The surface was associated with the cells and not physical space. In **pV3** the *static* surface acts like it did when *dynamic* – correctly for *grid unsteady* and *structure unsteady* cases.

2.2 Windows

The user interface is divided into six different windows. As is typical 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 six different windows are:

- Text Window

  The text window is the window from which the **pV3** server (*pV3Server*) 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 **pV3** server windows. This window is used to output various messages, including the help menus, and to input filenames, numerical values, etc.

- 3D Window

  The 3D window displays data on three-dimensional surfaces. It also displays three-dimensional lines such as tufts and streamlines, and other objects that are discussed later. These objects in the 3D window can be rotated, translated and enlarged using the dialbox (or pseudo-dials). All motion is relative to screen coordinates and not object coordinates. One set of key strokes allow the user to store particular viewing positions, and later restore then using the numeric keypad. The *setup* file retains this data so that it can be used at server restart or with other data-sets.

- 2D Window

  The 2D window is used to display data on a *mapped domain* or *dynamic* surface (for which there exists a mapping to a (x', y') coordinate system.)

- 1D Window

  The 1D window is used to display one dimensional data which is generated by various functions in the 2D window or from mapping instantaneous streamlines.

- Key Window

  This window displays the color scheme used in the 3D and 2D windows. 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.

  A *traffic light* (which is red when the **pV3** server is computing, yellow when **pV3** waiting for some specific user input, and green when ready to accept new user input) is active in the left-most portion of this window. Tied to the *traffic light* color is the color of the cursor. When the cursor is yellow **pV3** is expecting mouse button presses.

- Dialbox Window

  The dialbox window serves two functions. The first is to display the functions associated with each of the dials of the dialbox. Pressing the middle mouse button will switch the dialbox display between the 3D, 2D and key windows for which the dials are active, and in each case eight dials are displayed together with labels describing

the dial function (pan, zoom, rotate, etc.) when the cursor is in that window. If there is no dialbox the dials can be rotated by putting the cursor on the appropriate dial and holding down the left or right mouse button; in this case the window that is changed is the one labelled in the base of the dialbox window. Holding down the mouse button in the center of the dial will cause faster movement. In this mode the affected window will go into *fast-drawing* mode where only the surface edges are drawn. Under many circumstances this produces more interactive positioning than using the physical dialbox.

Pressing 's' switches the dialbox window to, and from, its secondary role, which displays the state of the surface and then streamline database. The surface database lists all of the *domain* and *static* surfaces, and for each one has four small boxes which give the status of the surface attributes. The first box gives the rendering status (white=ON, grey=translucent, black=OFF), the second is the grid status (white=ON, black=OFF), the third is the grey surface status (white=ON with colored contours, grey=ON, black=color) and the fourth is the thresholding status (white=ON, black=OFF). The meaning of some of these terms will become clearer later when discussing the scalar visualization tools. The user can change the attributes by pressing any mouse button when the cursor is on one of the boxes. Doing this, or pressing a mouse button on the surface label, also causes that surface to become the *active* surface. The *active* surface is highlighted in the database, and is important for certain plotting options discussed later.
NOTE: In **Visual3** it was not possible to have contouring on without the underlying surface also rendered. In **pV3**, the user can accomplish this by having the rendering box black and grey surface box white.

The streamline database lists all of the streamline objects, and next to each there are four small boxes which give the status of important streamline attributes. The first box gives the rendering status (white=colored, grey=ON, black=OFF), the second is the direction (white=backward, grey=both directions, black=foreward), the third is the streamline type (white/cross=tubes with twist, white=tubes, grey=ribbons, black=streamlines) and the fourth is the particle seeding status (white=ON, black= OFF). The meaning of some of these terms will become clearer later with the discussing the vector visualization tools. The user can change the attributes by pressing any mouse button when the cursor is on one of the boxes. Doing this, or pressing a mouse button on the streamline label, also causes that object to become the *active* streamline object. This object is highlighted in the database, and is important for certain plotting and control options discussed later.
NOTE: This is completely new for **pV3** and does not exist in **Visual3**!

2.3 Scalar Visualization Tools

The one of the data types in **pV3** is scalar data, which is simply scalar information defined at each node of the computational grid(s). **pV3** does not know anything about the data other than an associated function number and label. For example, in fluid dynamic applications, a function may have label *pressure* and a second scalar function may have label *Mach number*. An important concept in **pV3** is the notion of *active* functions, and the *active scalar function* is the function and associated label which corresponds to the scalar data currently used by **pV3**. The user can switch to a different scalar function by hitting a particular key on the keyboard which is *bound* by the **pV3** client initialization procedure. For example, key 'p' may be bound to the function, labelled *pressure*.

The following is a list of plotting functions available for use with scalar data:

2.3.1 Surface rendering

Gouraud-shaded (smooth color shading) surface contours of the scalar function can be rendered on any, or all, of the *domain* and *static* surfaces. Column 1 of the surface database is used to select which surfaces are to be displayed. With the cursor in the key window, there is a variety of options to interactively change the color scheme used for the rendering. Key 'l' loads a new color file, while key 'r' restores the original color file. Two dials on the dialbox (or pseudo-dials) change the upper and lower bounds of the scalar function range spanned by the color scheme.

If the *active* surface (as defined earlier) is a *mapped domain* surface (with an associated mapping to a (x', y') coordinate system), then Gouraud-shaded contours can also be plotted in the 2D window, by pressing F5 in the 3D window. The active surface is highlighted in the surface database, but if one is unsure of which it is in the 3D window then pressing F1 will cause the rendering on that surface to blink off and on.

Another option with *mapped domain* surfaces is F6 in the 3D window, which performs a surface rendering in both the 2D and 3D windows of the active *surface scalar* function, a function that is only defined on mapped domain surfaces as specified by the programmer. The color map used to render this surface function can be viewed by toggling key 's' in the key window.

2.3.2 Planar Cutting plane

The cutting plane is a *dynamic* surface, a true planar surface cutting through the 3D field. The cutting plane is initialized by pressing F3 in the 3D window. This puts the 3D window into a special mode in which the 3D object is held fixed and the user can use the dialbox

to rotate the cutting plane into the desired orientation. When ready, pressing F3 again switches off the planar movement mode and turns on the regular cutting plane mode.

Once the cutting plane is activated, it is controlled from the 2D window, meaning that it responds to keys and dials that are active when the cursor is in the 2D window. Using dials, the cutting plane can be moved from side to side, up and down, in a direction normal to the plane (using the *scan* dial) and rotated in its own plane. Function key F9 toggles (switches on and off) rendering in the 2D window, while F6 toggles rendering in the 3D widow. F10 toggles the display in the 2D window of the grid defined by the intersection of the cutting plane and the 3D computational grid faces.

The cutting plane can be turned off and on using F2 in the 3D window. The cutting plane position can also be stored, like the 3D viewing position, by pressing the Ctrl key and one of the ten numbers in the numeric keypad on the right-hand-side of the keyboard. It can be restored later by pressing just the number.

2.3.3 Program-defined cutting plane

This is similar to the last option, but instead of being a truely planar surface, it is a surface corresponding to $z' = const$, where z' is a programmer-defined function (in the clients); the *scan* capability varies the value of *const* interactively. This allows the programmer to define conical, cylindrical or other surfaces not otherwise defined by the **pV3** server. The client programmer also must have defined a mapping to (x', y') coordinates so that plotting is possible in the 2D window. The program-defined cutting plane is activated by F4 in the 3D window. The other options in the 2D window are the same as for the regular cutting plane.

2.3.4 Iso-surfaces

An iso-surface is a *dynamic* surface with a uniform value of the currentlty active scalar variable, and is activated by F7 in the 3D window. The iso-surface value is displayed in the key window and can be varied interactively (in the key window) using the dialbox to *scan* the value of z', or key 'z' to set its value, or the right mouse button to pick a value from the color key.

2.4 Vector Visualization Tools

The second **pV3** data type is vector data. This is a set of 3D vector values for each grid node. As with the scalar function, this vector data is associated with an *active* vector function, which can be changed by pressing a key that is *bound* to another vector function. The following is a list of plotting functions for vector data:

2.4.1 Bubbles

Bubbles are unsteady particle paths. This tool provides the same effect as hydrogen bubbles in experimental techniques. Bubbles are active with all **pV3** unsteady modes (unless the simulation is *paused*). A single bubble path may be spawned by simply pressing a mouse button in the 2D window (assuming seeding is off) which provides an initial point to start the integration in 3D space. If bubble coloring is on (F12 in the 3D window), the particle location will be rendered by the current scalar; if not, the current location is rendered with the default streamline color. The spheroid may also be colored by the time that the bubble was spawned. This is accomplished by hitting 's' in the key window. The time limits will probably have to be adjusted (hitting 'f' in the key window).

Several bubble paths may be started from a line or circle by using key F11 in the 2D window. Similarly, a grid of bubbles may be spawned using F12 in the 2D window.

If the particle streamer is on (Tab key in the 2D window), bubbles will be continuously spawned from the current cursor location at every time step. This mimics the experimental technique of streaklines where dye is continuously injected at a spot in the flow field. With the streamer on and the boundary layer or line probe is on, particles are emitted along the line every snapshot in time (the number of particles is the last set by spawning a line of bubbles – F11 in the 2D window). And, finally if the streamer is on and the tufts are active, a grid of bubbles is seeded each time step at the tuft locations.

2.4.2 Instantaneous Streamlines

Streamlines are curved 3D lines which are everywhere parallel to the local vector field. They are obtained by numerical integration of the vector field along a line starting at some chosen location. Instantaneous streamlines may only be activated for steady-state cases or when the seeding toggle in on (the key '|' in the 2D window). The starting point is initially determined by use of a surface plotted in the 2D window. A point in the 2D window maps back to a corresponding point on the *dynamic* surface in the 3D window and so can be used to seed instantaneous streamlines; pressing one of the mouse buttons does this and (depending on which button is pressed) produces a streamline going upstream and/or

downstream. Alternatively, key F11 (and subsequent mouse actions which are requested) defines a line or circle in the 2D window, which is then used to specify a set of streamlines in the 3D window. Key F12 initiates an object of streamlines from a regular grid of points in the 2D window, like it would spawn a grid of bubbles with the seeding toggle off.

Using the streamline database (in the dial window), groups of streamlines can be plotted either as lines of constant color (usually white), or colored according to the value of the local *active* scalar function. In the latter case, it is helpful to enable the *grey* status for background surfaces (using column 3 in the surface database) so that instead of being rendered in color they are instead rendered in solid grey, making the colored streamlines clearer.

Each object in the streamline database can also be reset as to the direction (downstream, upstream or both) and if bubbles are to be seeded from the same locations. It should be noted that in most cases seeding particles in this manner (and with the instantaneous streamlines not rendered) is faster than doing it interactively with an active cut in the 2D window.

The streamline object database also allows each group to be drawn as a line or the following (by using column 3 in the streamline database):

- Ribbons

  Stream ribbons are streamlines that have been given some width. One edge is the true instantaneous particle path, the other edge is constructed by rotating a constant length normal vector about the path tangent according to the local streamwise angular rotation rate. The result is a ribbon whose twist illustrates the streamwise vorticity of the flow.

  If the streamlines are colored, the ribbon is rendered in the default streamline color (usually white), otherwise the ribbon is colored with the current scalar.

  The width of the ribbon may be adjusted by using the dials when the cursor is in the key window. A specific width may be entered by hitting 'w'. Also, the ribbons may be rotated by using the dials with the cursor in the Key Window.

- Tubes

  A tube is a streamline with a circular crossflow area. The radius of the cross-section is derived from the local crossflow divergence. The crossflow divergence measures the local crossflow expansion rate. Thus, the resulting tube displays the local expansion/compression of the current vector field.

  If the streamline coloring is on, the tubes will be colored with the current scalar. Otherwise, the default streamline color is used.

The width of the tube may be adjusted by using the dials when the cursor is in the key window. A specific width (and maximum radius) may be entered by hitting 'w' in the key window. The maximum radius is useful to limit the size of the tube in stagnation regions of a flow field where the radius can become exponentially large.

- Tubes with Twist

  The rotation and divergence effects can be rendered simultaneously by placing lines on the surface of tube which twist with the local rotation rate. This effectively combines the functionality of the ribbons and tubes. The final image displays the streamline, the rotation rate, the crossflow divergence, and scalar variations.

  Again, if the streamline coloring is on, the tubes will be colored with the current scalar and the lines will be in the default streamline color. Otherwise, the tube is the default color and the lines are the current scalar.

  As with ribbons and tubes, the rotation angle, the tube size, and the tube maximum may all be controlled from the key window using the dial box or by the appropriate key strokes.

NOTES:
(1) If key F5 in the 3D window is in effect, rendering a *mapped domain* surface in the 2D window, then the instantaneous streamlines which are generated are similar to regular streamlines except that they are lie in the surface, by taking the projection of the local vector field onto the surface. Similarly, if key F6 is in effect, rendering a *mapped domain* surface with the surface scalar function, then the streamlines are generated using the surface vector function.
(2) The streamline accuracy is reduced for the segment that crosses interface regions. Invoking this option also slows down the overall integration speed.

2.4.3 Tufts

Tufts are similar in concept to streamlines. A regular grid of points in the 2D window map to a corresponding grid of points on the surface in the 3D window. At the points in the 3D window, tufts are drawn which are short lines with magnitude and direction correponding to the local vector field. At the points in the 2D window the tufts correspond to the projection of the 3D vector field onto the 2D plane. Key 'Tab' in the 3D window toggles tufts on and off. One of the dials in the key window allows interactive change in the scaling parameter which relates the vector magnitude to the tuft size, and key 'a' allows this scaling parameter to be input from the keyboard.

2.4.4 Arrows

Arrows are the same as tufts except that they are defined only at 2D nodes (the intersection of a cut and cell edges). To emphasise that they are different, they are drawn as lines with heads in the 2D window, whereas tufts are drawn as lines with a cross base. Arrows are shown on cutting planes as well as iso-surfaces and are displayed in the 3D window as line segments. Arrows are toggled on and off by key F7 in the 2D window.

2.4.5 Vector Clouds

Vector clouds display the local vector field at nodes which meet the current threshold limits. The vector cloud technique is useful for locating interesting flow features and displaying the vector fields in these regions. Vector clouds are invoked by hitting F8 in the 3D window and are always rendered with the current scalar.

Note: Selecting this option (and not carefully pre-setting the thresholding limits) can produce an enormous amount of network traffic! 7 words are transmitted for each node within the system that meets the threshold criteria.

2.5 Grid

The computational grid can be displayed on any, or all, of the *static* and *dynamic* surfaces. For the *static* surfaces this is controlled through column 2 of the surface database. For *dynamic* surfaces and data plotted in the 2D window, grid display is controlled by function key F10 (in the 2D window). The grid lines that are displayed correspond to the intersection of the plotting surfaces and the faces of the computational grid(s).

2.6 Thresholding

A threshold function is another scalar function which is set, or changed, by pressing the appropriate keys. The purpose of this function, when enabled, is to restrict all *domain* and *static* surface plotting to only those parts of the surface on which the thresholding function lies within a certain range. The user can interactively vary the upper and lower threshold bounds. The user can also select, through column 4 of the surface database, the surfaces that are to be thresholded.

If the threshold function is chosen to be the same as the scalar function, then this provides a means to plot the part of a surface on which the scalar function is within certain limits. If the threshold function is chosen to be geometric (e.g. x) then this produces a dynamic *cutaway*, in which the surface is only plotted within a certain geometric volume.

The threshold function can be set in two ways. Pressing a key on the keyboard that has been defined by the programmer to be associated with a threshold function loads that functions data into the threshold array. Alternatively, pressing F9 in the 3D window loads the current scalar function data into the threshold storage.

The thresholding limits, within which plotting will be performed, can be varied interactively using dials in the key window, or input manually using key 't'.

2.7 Probes

There are a variety of *probes* which are available when plotting in the 2D window or from streamline objects:

- Point (2D - F1)

  The point probe is located at the cursor position, and returns, in the text window, the point's coordinates and the value of the active scalar and vector functions.

- Strip chart (2D - F2)

  The strip chart is similar to the point probe, except that instead it produces a plot in the 1D window of the current scalar function against time.

- Line (2D - F3)

  When the line probe is invoked the user is asked to input two points using the mouse. These define a line in the 2D window, and the output is a plot in the 1D window showing the variation of the current scalar function along that line.

- Edge Plot (2D - F4)

  The edge plot is similar to the line plot, except that in this case the line in the 2D window is the edge line closest to the cursor when this option is invoked.

- Surface Layer (2D - F5)

  This option produces a line plot in the 1D window of the current scalar function along a line placed normal to an edge in the 2D window, at the edge position which is closest to the cursor. As the user moves the cursor, the normal line moves accordingly.

- Streamline Probe (Dial - '|')

  The streamline probe may be started any time there are streamline objects. The current object is mapped to the 1D window. When the cursor is in the 1D window a cross-hair or disc appears in the 3D window marking the closest position on the active streamline. The size of the disc mimics the stream tube thickness. For surface streamlines the mapped cursor is a cross-hair displayed in both the 2D and 3D windows. This allows the user to both know which streamline is mapped and probe the streamline. Notes:
  1) Tabular output files (visualXYZ.tab) created when this probe is active also contain the coordinate triads for the streamline.
  2) When tubes are on, the disc size is 150% of the tube thickness.

2.8 Help Menus

2.8.1 3D Window

The help menu that is printed when one types '?' in the 3D window is as follows:

```
3D Window


Mouse Buttons:
m - Center View @ Cursor

Key Strokes:
~       - write visual.img File       +         - Box blow-up
F1      - Show Active Surface         F2        - Toggle Cutting Plane
F3      - Cutting Plane positioning   F4        - Toggle Program Cut Plane
F5      - Toggle Surface Display      F6        - Toggle Disp. w/Surface Fn
F7      - Toggle Iso-Surface          F8        - Toggle Vector Clouds
F9      - Set Scalar as Threshold     F10       - Animate StreamLines
F11     - Bubble Render Toggle        F12       - Bubble Color Toggle
Delete  - Delete Bubbles              \         - Material Line Toggle
Insert  - Save Dynamic Surface        ^         - Shading Toggle
Tab     - Tufts Toggle                Home      - Reset View
PageUp  - Reset Clipping              PageDown  - Depth Cueing Toggle
NumPad  - Set view from position #    Ctrl-NumP - Store view in position #
!       - 3D Window Status            End       - Terminate 2D modes
|       - Ribbon/Tube Toggle          Pause     - Freeze the action
/       - Edge Outline Toggle         Esc       - Terminate pV3 Server
```

Comments:

1) At the top of the help menu in real applications there would be a list of the scalar, vector and threshold function variables and their associated keys, as defined by the application program.

2) A toggle is a switch that is either on or off, and so pressing the key changes it to the other status.

3) Clipping is similar to a geometric thresholding. It displays the part of the 3D object that is behind a plane held parallel to the screen.

4) In the 'NumPad' and 'Ctrl-NumP' descriptions, 'NumPad' and 'NumP' refer to one of the ten numbers on the numerical keypad on the right of the keyboard. This number is then referred to as #. This option allows the storing and recall of ten different viewing

positions and any cutting planes that are active. The 'NUM LOCK' light must be on for these key-strokes to be acknowledged.
5) Displaying the active surface, F1, will only work if the surface has some render attribute on (Box 1 in the surface database). The surface will flash on and off.
6) Depth Cueing only works on those machines that support *fog*.

2.8.2 2D Window

The help menu that is printed when one types '?' in the 2D window is as follows:

```
2D Window

Mouse Buttons:
l - Bubble/StreamLine going upstream
m - Bubble/StreamLine going up/downstream
r - Bubble/StreamLine going downstream

Key Strokes:
~       - write visual.img File        +      - Box blow-up
F1      - Point Probe                  F2     - Strip Chart
F3      - Line Probe                   F4     - Edge Plot
F5      - Surface Layer Scan           F6     - 3D Window Render Toggle
F7      - Arrow Toggle                 F8     - Contour Toggle
F9      - Render Toggle                F10    - Grid Toggle
F11     - Line/Circle of StreamLines   F12    - Grid of StreamLines
Delete  - Flip X in Window             Tab    - Bubble Streamer Toggle
End     - Terminate Line Plot          |      - StreamLine Seed Toggle
t       - Dynamic Surf Thresh Toggle   !      - 2D Window Status
```

Comments:
1) the 'Delete' option reverses the sign of the x'-coordinate in the 2D window, effectively *turning over* the 2D window. This is helpful when the cutting plane surface you are seeing in the 3D window is the reverse side of the 2D window.
2) 'End' ends all plotting in the 1D window.
3) The StreamLine Seed Toggle allows what would spawn off Bubble(s), to add objects to the StreamLine database.

2.8.3 1D Window

The help menu that is printed when one types '?' in the 1D window is as follows:

```
1D Window

Mouse Buttons:
m   - Set Cut Plane w/ StreamLine Probe (positioning on)
any - Seed SL/Bubble w/ Edge Plot on

Key Strokes:
r       - add Reference line          s          - Volume/Surface Fn Toggle
x       - Change X scaling            y          - Change Y scaling
End     - Terminate Line Plot         PrintScrn  - Tabular Output
```

Comments:

1) To set a planar cut perpendicular to the streamline at a given postion, first turn the streamline probe on and select the appropriate streamline, then turn planar cut positioning on (F3 in the 3D window). Move the cursor in the 1D window to the correct position, and to finish, press the middle mouse button.

2) The *Reference line* is an additional line placed in the 1D window along with the results of a probe. This line is read from a file in the **pV3** tabular file output format and displayed in grey.

3) The Volume/Surface function toggle allows the specifying of what surface functions are used for integrations and rendering of surface particles and streamlines when there are special surface functions. This allows the choice between the special functions and the normal volume scalar/vector fields.

2.8.4 Key Window

The help menu that is printed when one types '?' in the key w ndow is as follows:

```
KEY Window

Mouse Buttons:
m - Set new color at cursor position
r - Set Iso-Surface value

Key Strokes:
 a      - (Re)Set arrow/tuft size        c      - Set # of Contours
 f      - (Re)Set function limits        l      - Load new color file
 m      - Set S.L. Animations            r      - Reset color scheme
 s      - Toggle color schemes           t      - (Re)Set thresh lims
 w      - Set Tube/Ribbon width          z      - Set ZPrime
 !      - Key Window Status
```

Comments:
1) Option 's' allows one to toggle the display of the color schemes, between the color scheme that is used for all standard scalar rendering, the scheme that is used to display scalar surface functions, and the color map used for time rendering of particles.
2) The iso-surface and cutting planes correspond to a surface on which $z' = const.$ Option 'z' allows one to explicitly specify the value of this constant.

2.8.5 Dials Window

```
Dials Window (Dials)

Mouse Buttons:
l - Move Dial Clockwize
m - Change Window Mapping
r - Move Dial CounterClockwize

Key Strokes:
` - write ImageFile of Screen        c - comparison window
d - Dial Sensitivity                 s - Surface List Toggle
S - Send Clients a String            M - Mirror Toggle
```

Dials Window (Surface List)

Mouse Buttons: any - Select

Key Strokes:

~	- write ImageFile of Screen	c	- comparison window
d	- Dial Sensitivity	s	- Surface List Toggle
PageUp	- Move Surface List Up	PageDown	- Move Surface List Down
Delete	- Remove Current Surface	S	- Send Clients a String
M	- Mirror Toggle		

Box:	1	2	3	4
black	not rendered	no grid	scalar rendered	no thresholding
grey	translucent		grey surface	
white	opaque	grid on	grey w/ contour	thresholding on

Dials Window (SL-Particle List)

Mouse Buttons: any - Select

Key Strokes:

~	- write ImageFile of Screen	c	- comparison window
d	- Dial Sensitivity	s	- Surface List Toggle
PageDown	- Move SL-Part List Up	PageUp	- Move SL-Part List Down
Delete	- Remove Current SL-Part	|	- StreamLine Probe
S	- Send Clients a String	M	- Mirror Toggle

Box:	1	2	3	4
black	off	foreward	streamline	no particles
grey	on	both	ribbon	
white	colored	backward	tube	particles
cross			tube w/ twist	

Comments:

1) This window has two modes; dials and surface/streamline list (database). In dials mode it displays or emulates the functions of the dialbox dials (if a dialbox) exists. In surface list mode it displays the surface database, a list of all of the *domain* and *static* surfaces and their attributes. In streamline database mode it displays a list of all of the streamline objects and their attributes.

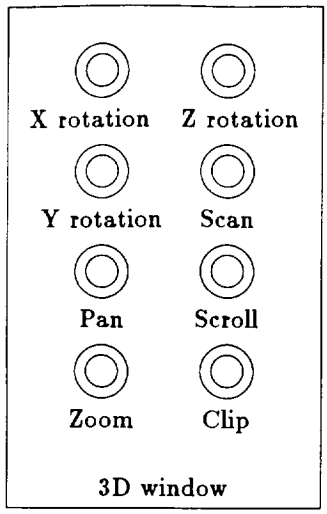
2) In dials mode, pressing the middle button displays in succession the meaning of the dials for each of the principal windows. If dialbox emulation is being used pressing the right or left mouse buttons has the effect of turning the appropriate *pseudo-dial.* The middle circle moves the dial faster than the outer portion of the dial. Holding the button down will cause **pV3** to go into *outline* mode for faster motion. This can be defeated by holding down the 'Shift' key while pressing on the button.

3) In surface list mode, boxes 1,2,3,4 refer to the four columns of the surface list which are labelled on the screen as Render/Grid/Grey/Thres.

4) In streamline list mode, boxes 1,2,3,4 refer to the four columns of the list which are labelled on the screen as Render/Dir/Type/Seed.

5) The *comparsion window* is an additional window placed on the screen. This window displays of the contents of an existing **pV3** image file. It is open in the upper-left corner of the screen and may be moved by using the appropriate window manager functions. The keystroke 'End' hit in this window, closes it and deallocates any memory.

2.9 Dialbox Functions



X rotation: rotate about X-axis

Z rotation: rotate about Z-axis

Y rotation: rotate about Y-axis

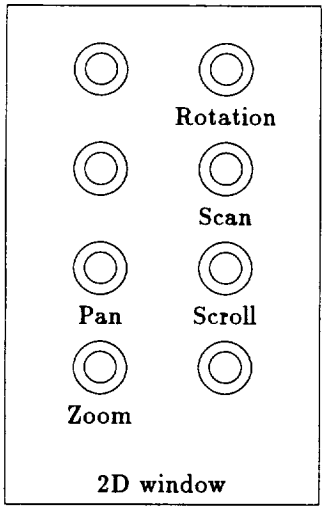
Scan: move cutting plane or iso-surface

Pan: move right/left

Scroll: move up/down

Zoom: enlarge/reduce

Clip: move clipping boundary



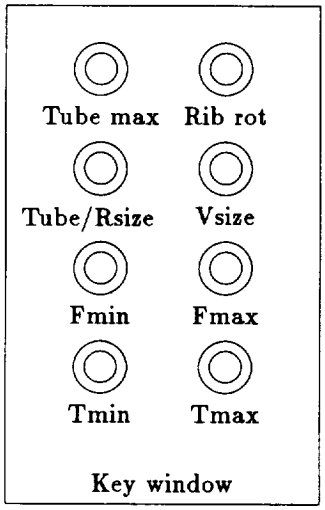
Rotation: rotate

Scan: move cutting plane

Pan: move right/left

Scroll: move up/down

Zoom: enlarge/reduce



Tube max: maximum tube size

Rib rot: ribbon rotation

Tube/Rsize: change tube/ribbon size

Vsize: change scaling of tufts/arrows

Fmin: change minimum scalar function value

Fmax: change maximum scalar function value

Tmin: change minimum thresholding value

Tmax: change maximum thresholding value

3 Post-Processing

There are two types of output files generated by **pV3** activated by the appropriate keys in different windows. These files are compatible with the post-processors supplied with **Visual3**. The first type 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 the entire screen or 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') and viewed on the screen (using 'img2X').

The following section describes the usage of the post-processors supplied with **pV3** and **Visual3**. It should be noted that the source and make-files for the post-processors have been included with the distribution. By making small modifications to the sources, other output devices can be supported with no changes to **pV3**.

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 img2tiff

img2tiff takes as its first argument the image file name, and the second argument is the output TIFF file name. The TIFF file can then be transferred to a Macintosh and be used by any application that can take TIFF as input, i.e. Adobe Photoshop.

example: % img2tiff visual001.img output.tiff

3.3 img2ps

img2ps takes as the first argument the image file name. Additional arguments are options controlling the translation from TrueColor to 8-bit grey scale. 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:

-cps produce color PostScript output - this is the only option that will produce color, the default and the other options produce grey scale output

-r produce a red color separation

-g produce a green color separation

-b produce a blue color separation

-cxxxxxxxx color mapping where "x" is either r, g, or b.

-i inverse intensity

-4 4-bit grey scale

examples: % img2ps visual001.img | lp
% img2ps visual001.img -r -i > img.ps
% img2ps visual001.img -cps | lpr
% img2ps visual001.img -crgbrgbrg | lp

Notes:

1) Options -r, -g, -b, -c and -cps are mutually exclusive.

2) If options -r, -g, -b, -c or -cps are not selected, the color translation defaults to 3 bits red, 3 bits green and 2 bits blue (-crrrgggbb).

3) 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 from the 2D window. Use the -4 option for the 3D window. Do not attempt full screen dumps unless your printer has alot of memory!

3.4 img2X

img2X may be used to view **pV3** image file(s) on the screen. It can display as many a 10 images and also write a **pV3** image file of the entire screen. img2X may have as many as 10 arguments, each should be the name of an image file. Optionally, each image can be compressed before drawn. This is done by appending '/n' to the end of the file-name (where n is the compression factor).

examples: % img2X visual001.img visual002.img/3
% img2X visual001.img/2

Notes:

1) The compession scheme used is very simple. If n=2, every other pixel is displayed, n=3 picks every third pixel.

2) An image window may be closed by hitting the key 'x' in that window.

3) An image file of the entire screen may be generated by hitting '~ ' in any image window.

4) img2X is properly terminated by clicking any mouse button while the cursor is in an image window.

4 User Interface Differences with Visual3

The following is a list of differences that an experienced **Visual3** user should note:

- Planar Cut

  During reorientation of the plane (F3 in the 3D window) cut data is plotted in the 2D window in **pV3**. This cut data may lag behind the current position as accurately shown in the 3D window. Note: this may cause some confusion, remember to turn off the re-positioning (F3 in the 3D window) when the desired orientation is reached.

- Cell Visualization Tools

  pV3 does not support the plotting options for Cell Based Scalars. **Visual3** should be used to view output of these functions.

- Histogramming

  pV3 does not support histogramming or any function that requires the server to look at the entire 3D data-set. Therefore querying the limits ('q' in the key window) and auto-scaling are also not supported.

- Line output files

  pV3 does not produce line files from the 3D and 2D windows for post-processing.

- Streamlines

  The methods used for dealing with streamlines in **pV3** are much more general than in **Visual3**. This includes:

  - A streamline object database
  - Streamline probe is initiated from the database and may contains as many lines as streamlines in the object
  - Each object has independent attributes (like the surface database)
  - Streamlines are active during unsteady (non*paused*) states
  - No streamer

  See the Instantaneous Streamline section in Vector Visualization Tools.

- Bubbles

  The seeding methods used for particle paths are more numerous in **pV3**. These include:

  - Seeding from streamline start locations (not requiring an active cut). Allows *material lines* if the object is built from a line.
  - Streamer (cursor) location
  - Line/boundary layer probe locations
  - Tuft locations

  Notes:
  (1) *Ghost* bubbles are not plotted.
  (2) Bubbles can be colored by the seed time in **pV3**. See the Bubble section in Vector Visualization Tools.

- Surface Streamlines and Bubbles

  In **Visual3**, surface integrations were initiated when a *domain* surface was mapped and seeding was accomplished in the 2D window. Once the surface was unmapped, all streamlines/bubbles were deleted. In **pV3**, seeding surface streamlines and surface bubbles follow the above rules for a *domain* surface when it is mapped. The seed points/particles are not deleted when the surface is unmapped. Also, seeding can be accomplished without mapping by using any mouse click in the 1D window with an Edge Plot active. This is VERY usefull when the surface is too complex to map to the 2D window. The control of what vector/scalar fields (special surface or volume) are used for the integration/rendering of the surface streamlines and particles is performed by the surface function toggle ('s' in the 1D window).

5 Warning and Error Messages

- Warning: No Time-Out Set!

  The time-out constant was set to no time-out, hard loops are used for both the server and client(s).

- Warning: Visual**xxx* not defined in .Xdefaults!

  The font specified is not known to the X windows system. See the section on Special Files.

- Warning: pV3Client group size = *xxx*, only *yyy* task(s) active!

  The **PVM** client group has members that are no longer active. This is usually do to some clients exiting ungracefully.

- Warning: Current ServerpV3 group size = *xxx*!

  The **PVM** server group has members that are no longer active. The server must have aborted earlier during the session.

- Warning: *xxx* clients in group but only using: *yyy*

  An 'init' message was only received from *yyy* clients but the total number of active tasks in the **PVM** group is *xxx*. It is possible that a task is not calling **pV_Update** (see the **pV3** Programmer's Guide. Also, it is possible that you may have to increase the Time-Out constant. See the section on Evironment Variables.

- Warning: Illegal Message # *xxx* from *PVMtid*

  A non-**pV3** message was received.

- Warning - MIRROR multi-client mismatch!

  MIRROR specified in pV_Init does not match between the various clients. Mirroring is turned off.

- Warning - FLIMS multi-client mismatch!

  FLIMS specified in pV_Init does not match between the various clients. The first received is used.

- Warning: pV3 client at different Rev than server! - tid = *PVMtid*

  A client/server mismatch. It is a good idea to rebuild the clients with the library that matches the server!

- Warning: pV3 MAX clients exceeded - set to: *xxx*

  The internal maximum number of clients was exceeded. The case will run with only *xxx* clients.

- Warning: Bad Status from client: *PVMtid*

  The client specified has had some type of problem. The data from this iteration will not be plotted.

- Warning - New Client trying to connect! *PVMtid*

  A client is attempting to connect to a running visualization session. The request is ignored and the session continues.

- Warning: Max Segs for SL # *xxx*

  The maximum number of streamline segments has been reached for the specified streamline. The integration is aborted and the streamline is displayed unfinished.

- Warning: ACK from client: *PVMtid*

  An acknowledgement from the specified client has come at an incorrect time.

- Warning: Bad client: *PVMtid*

  A client that is not part of this session has set a **pV3** message. This should not happen!

- Warning: Double Fill from client: *PVMtid*

  The specified client has sent two data streams for this iteration.

- Warning: Clients reporting different times!

  Not all clients are reporting the same simulation time.

- Warning - Pending particle/SL inserts!

  A Streamline group delete was requested while the server is processing inserts. The delete request is ignored.

- Warning - StreamLine deletes pending!

- Warning - Particle delete pending!

  Inserting a group of Streamlines is invalid while the server waits for the pending deletes to complete. The request is ignored.

- Warning - Another surface delete pending!

Only one surface delete can be specified during an iteration. Additional requests are ignored.

- Warning - Window did not produce Expose Event!

  The server waited for a window to give an X Expose event and it did not happen!

- Warning (*routine*): ***xxx*** can not be transfered from *PVMtid* to *PVMtid*

  During an integration, a request has been made to continue to a task that does not exist or for some reason can not be reached.

- Warning (*routine*): Bad client tid: *PVMtid*

  A non-existant task was targetted for an integration transfer.

- Warning (*routine*): partID ***xxx*** out of range.

  An illegal particle number was encountered during a interclient transfer.

- Warning (*routine*): partID ***xxx*** NO History!

  A particle that has had no prior history is requesting a transfer.

- Warning (*routine*): SLXferID ***xxx*** out of range.

  An illegal streamline number was encountered during a interclient transfer.

- Warning (*routine*): SLXferID ***xxx*** NO History!

  A streamline that has had no prior history is requesting a transfer.

- Warning (*routine*): SLXferID ***xxx***, client *yyy*, unmark <= 0

  An error occured removing a streamline from the active list.

- Error - Cannot attach to lock arena!

  The **pV3** server cannot set-up the lock areana to the file '.pV3.locks'. Is there write access in the current directory?

- Error - getting new lock!

  There is some problem initiating a new lock. Try removing the file '.pV3.locks'.

- pV3 Error - No response from any clients!

  The **pV3** clients currently running have not responded within the time-out constant.

- Error Starting Thread for multi-processing!

  The new thread for the visualization could not be initialized.

- Error - Client(s) have exited!

  One or more clients have exitted from a running visualization.

- Error in Memory Allocation!

  The server has requested a block of memory and has been refused. This is usually do to the problem's size. Either wait until the workstation was fewer tasks running or find a bigger (more swap space) machine.

- Error - NPGCUT multi-client mismatch!

- Error - TPGCUT multi-client mismatch!

- Error - NKEYS multi-client mismatch!

- Error - IKEYS multi-client mismatch!

- Error - FKEYS multi-client mismatch!

  The specified pV_Init data does not match between the various clients for a case that has more than one client.

- Error during Particle Initialization!

  The server has requested memory for the particle tracking and has been refused. Either wait until the workstation was fewer tasks running or find a bigger (more swap space) machine.

- Error during SL Transfer Initialization!

  The server has requested memory for the streamline tracking history and has been refused. Either wait until the workstation was fewer tasks running or find a bigger (more swap space) machine.

- Singular matrix.

  This occurs while processing the view transformation matrix. It is considered illegal to have a singular transformation matrix. The server should not produce this condition. It usually happens when the setup file has been corrupted. This can also happen if all the coordinate data passed to the server is identical (the same XYZ position for all nodes).

- Error - File does not exist!

  The requested file does not exist.

- Error - Not a TrueColor Image!

  The image file requested for the comparison window is not a TrueColor image (it is PseudoColor).

- Error - Image depth mismatch!

  The image file requested for the comparison window has the wrong color depth.

- Error in ImageFile!

  An error has occured during the image file read.

- Error in Reference File - NOT in Tabular form!

  A reference line file was specified that was not in **Visual3** *.tab* format.

- ERROR in (SUB-)EXTRACT allocation

  Memory could not be allocated for the (sub)extract subsystem.

- Extract NSEG ERROR: sub = *xxx*, nseg = *yyy*, max = *zzz*

  A streamline sub-extract has been received with an illegal segment number. This is not fatal but something is wrong!

- Extract TID ERROR: sub = *xxx*, nseg = *yyy*, ic = *zzz*

  A streamline sub-extract has been received with a *PVMtid* for segment *yyy*, but a previous message for that segment had the client number *zzz*. This is not fatal but something is wrong!

- Extract Client ERROR: sub = *xxx*, tid = *PVMtid*

  The client tid is not in the active list. This is not fatal but something is wrong!

- Extract ERROR: type = *www*, sub = *xxx*, size = *yyy*, len = *zzz*

  A sub-extract has been allocated for *yyy* words, but the message has *zzz* words. This is not fatal but something is wrong!

- ERROR: Timed out waiting for Init Hand-Shake!

- ERROR: Timed out waiting for Client Hand-Shake!

  The time-out constant has been exhausted before responce from the client(s). Increase the constant by the environment variable 'pV3_TO'.

- ERROR - ColorMap OverFlow!

  The requested number of colors specified in the colormap file exceeds the internal colormap storage.

- ERROR - ColorMap File Error!

  An error occured parsing the colormap file.

- KeyBoard File Does NOT Exist!

  The environment variable 'Visual_KB' has been set and the file indicated does not exist.

- ERROR reading KeyBoard File!

- ERROR E-O-F in KeyBoard File!

  The environment variable 'Visual_KB' has been set and there has been an error parsing the data in the file.

- pV3: ERROR pvmd Not running!

  The **PVM** system has not been initiated.

- pV3: ERROR No pV3 Clients running!

  The server finds no clients.

- pV3: ERROR Server Already running!

  Another server is already running. **pV3** currently can have only one active server.

- pV3: bufinfo error: *xxx*, *yyy*

  This is not a fatal error but a receive message buffer is giving an error indication. The message is ignored.

pV3 Programmer's Guide

Rev. 1.00
Client Side Programming

Bob Haimes
Massachusetts Institute of Technology

September 1, 1994

License

This software is being provided to you, the LICENSEE, by the Massachusetts Institute of Technology (M.I.T.) under the following license. By obtaining, using and/or copying this software, you agree that you have read, understood, and will comply with these terms and conditions:

Permission to use, copy, modify and distribute, this software and its documentation for any purpose and without fee or royalty is hereby granted, provided that you agree to comply with the following copyright notice and statements, including the disclaimer, and that the same appear on ALL copies of the software and documentation:

THIS SOFTWARE IS PROVIDED "AS IS", AND M.I.T. MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, M.I.T. MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

The name of the Massachusetts Institute of Technology or M.I.T. may NOT be used in advertising or publicity pertaining to distribution of the software. Title to copyright in this software and any associated documentation shall at all times remain with M.I.T., and USER agrees to preserve same.

Contents

1 Introduction

pV3 is the newest in a series of graphics and visualization tools to come out of the Department of Aeronautics and Astronautics at MIT. Like it's predecessors **Visual3**, **Visual2** and **Grafic**, **pV3** is a software package aimed at aiding in the analysis of a particular suite of problems. In this case it is the real time visualization of 3D large scale solutions of transient (unsteady) systems.

pV3 (which stands for parallel **Visual3**), is a completely new system, but builds heavily on the technology developed for **Visual3**. It has been designed specifically for co-processing visualization of data generated in a distributed compute arena. It is also designed to allow the solver to run as independently as possible. If the solution procedure takes hours to days, **pV3** can 'plug-into' the calculation, allow viewing of the data as it changes, then can 'unplug' with the worst side-effect being the temporary allocation of memory and a possible load imbalance.

pV3 provides the same kind of functionality as **Visual3** with the same suite of tools and probes. The data represented to the investigator (the 3D, 2D and 1D windows with cursor mapping) is the same. Also the same Graphical User Interface (GUI) is used.

pV3 programming is very **Visual3**-like. For the desired flexibility and the merging of the visualization with the solver, some programming is required. The coding is simple; like **Visual3**, all that is required of the programmer is the knowledge of the data. Learning the details of the underlying graphics, data extraction, and movement (for the visualization) is not needed. If the data is distributed in a cluster of machines, **pV3** deals with this, resulting in few complications to the user.

In most cases, the calls or routines provided are identical to the **Visual3** programming interface. For someone familiar with **Visual3**, programming of **pV3** requires little new knowledge.

Changes in the programming interface were required to support added functionality, and the distributed nature of the compute. Some changes were due to the separation of the display workstation (the server) from the volume of data (residing in the client or clients). **Visual3** programmers must pay particular attention to the routines pV_Init, pV_Update, pVSurface and pVBlank.

Because the **pV3** server does not contain the entire volume of data (but only the *extracts*), compatibility with **Visual3**'s advanced programming could not be perserved. See the **pV3** Advanced Programmers Guide for details.

2 pV3 in the PVM Environment

The software used for the movement of data across the network is **PVM** from Oak Ridge National Laboratory. **PVM** (parallel virtual machine) is public domain software that provides the mechanisms required to transform a (heterogeneous) network of machines to one parallel computer. **PVM** provides all the required 'hooks' including efficient data transfers, message passing, synchronization and the ability to target certain machines to specific tasks. **PVM** is available directly from Convex and IBM for their cluster offerings and will also be available from the traditional Massively Parallel Processor vendors.

2.1 Message Passing

pV3 requires **PVM** version 3.3.0 or higher. For co-processing in a cluster of workstations (multiple clients) certain rules must be followed so that messages for the visualization and the compute do not interfere with each other.

- Open Send Buffer

  It is assumed by **pV3** that the default send buffer is free and available for use. This should not be a restriction because either pV_Init or pV_Update should be called at times when interclient communication is at a completed stage.

- Broadcasts

  Broadcasts should be avoided. You will end up sending messages to the **pV3** server. The server will report then ignore messages without the proper signature. In general, any client need not send the **pV3** server any messages (all the data communication necessary is handled internally by **pV3**).

  If a broadcast facility is required, use the multiple-cast send, and send only to known tasks.

- Receives

  Do not use a wild-card for the task id in the receive calls to **PVM**. You will get **pV3** requests. The message collection in the routine pV_Update only takes messages from the **pV3** server, leaving other client message traffic alone.

2.2 pV3 Startup

The **PVM** daemon(s) and with co-processing, the solver, must be executing. Without the **pV3** server running, every time the solution is updated and pV_Update is called, a check is made for the number of members in the group *ServerPV3*. If none is found, this routine returns. When the **pV3** server starts (interactively from an xterm window on the graphics workstation), it enrolls in that group. The next time pV_Update is called, an initialization message is processed and the session begins. Each subsequent time in pV_Update, visualization state messages and *extract* requests are gathered, the appropriate data calculated, collected and sent to the server. Like in **Visual3**, when the code calls pV_Update, additional routines provided by the programmer are called to supply **pV3** with data about coordinates, scalar and vector fields.

When the user is finished with the visualization, the server sends a termination message and exits. The clients receive the message and clean up any memory allocations used for the visualization. pV_Update reverts to looking for server initialization if termination was not specified at **pV3** initialization.

2.2.1 Server Startup

The server, *pV3Server*, takes two arguments at the command line (both optional). The first is the *setup* file with the default of 'pV3.setup'. The second argument is the *color* file to be used at startup (the default is 'spec.col').

```
examples: % pV3Server case.setup
          % pV3Server pV3.setup bw.col
          % pV3Server
```

2.2.2 Environment Variables

The **pV3** server uses three Unix environment variables. Some are the same as the ones used for **Visual3**. The variable 'Visual3_CP' defines the file path to be searched for color files, if they are not in the user's current directory. This allows all of the color files to be kept in one system directory.

The second variable, 'Visual_KB' is optional. This variable, if defined, must point to a file that contains alternate keyboard bindings for the special keys used by **pV3**. The file is ACSII. The first column is the key name (10 cahracters) and the second is the X-keysym value in decimal (use 'xev' to determine the appropriate values for the key strokes).

The third is 'pV3_TO' and should be used to change the internal Time-Out constant. If the variable is set, it must be an integer string which is the number of seconds to use for the

Time-Out constant (the server's default is 60). This may be required if the time between solution updates is long. See the section in the **pV3** Server User's Reference Manual on *Time-Outs and Error Recovery.*

2.2.3 Special Files

- .Xdefaults

  If the SGI window manager is used, '4DWm' must be told what to do with **pV3**s windows. These commands must be placed in the file '.Xdefaults'. See the file 'user.Xdefaults'.

  pV3 requires three X fonts. The file '.Xdefaults' in the users home directory is examined for the font names and are designated "Visual*large", "Visual*medium" and "Visual*small". The sample file 'user.Xdefaults' comes with the distribution and may be concatinated to the user's '.Xdefaults' file.

  The X fonts loaded on any system may be examined by the command 'xlsfonts'.

- 'twm' Setup File

  If the 'twm' window manager is used, when a user begins a session, twm reads an initialization file '.twmrc' in the user's home directory. This file defines certain key bindings and window attributes. It is necessary for correct **pV3** operation for the user to modify the standard '.twmrc' file as shown in the 'user.twmrc' file on the **pV3** distibution, or just use this file as '.twmrc'.

- Setup File

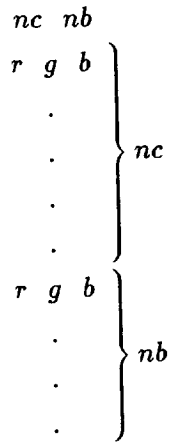
  When the **pV3** server starts, it looks in the user's current directory for the setup file specified as the first argument on the command line. This is an ASCII file which contains a number of useful defaults that the user may want to set to different values than **pV3**'s initial defaults. It also contains a set of viewing positions and cutting plane positions. Normally this file is generated by **pV3** when the user wants to store certain favorite parameters and viewing positions so that they can be used again on another data set, which is particularly useful when the user wishes to directly compare two different data sets with the same computational grid or geometry. However, an experienced user can also generate this file from scratch. NOTE: This file is NOT compatible with **Visual3**'s setup file.

- Lock File

  If the server is running on a multi-processor workstation (PowerSeries or Onyx) a file is used for the coordination of the 2 threads generated during execution. This file has the name 'pV3.lock' and is open in the current directory. It should be noted that running two invokations of the **pV3** server from the same directory will NOT work. Both will use the same file for the lock and semaphore arena!

- Color Files

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



  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/ribbon 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.

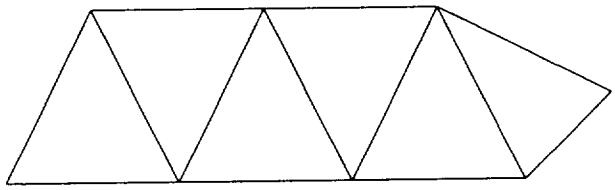
  When **pV3** searches for named color files, it first looks in the current directory, and then follows the color file path specified by the environment variable 'Visual3_CP'.

3 Programming pV3

3.1 Programming Overview

Before presenting the subroutine argument lists in detail it is helpful to discuss in general terms the data structures which the programmer supplies to **pV3**. The programmer gives **pV3** a list of unconnected cells, poly-tetrahedral strips and structured blocks. The disjoint cells are of four types; tetrahedra, pyramids, prisms and hexahedra. This element generality covers almost all data structures being used in current computational algorithms. Any special cell type which is different must be split up into some combination of these primitives by the programmer. Linear interpolation is used throughout **pV3**, so high order elements must be also be subdivided so that the linear interpolation assumption is valid.

Poly-tetrahedral strips are 'structured' collections of tetrahedra. The strip is started by a triangular face, one node is added to produce the first tetrahedron, another is added to produce the second cell (with the previous 3 nodes) and so forth. See Figure 1. Currently, no one is using this concept for calculating results but there is more than a factor of two savings in the storage required to represent a complete tetrahedral mesh.



tetra	#1	tetra	#2	tetra	#3	tetra	#n (last)
nodes:	1 2 3 4	nodes:	2 3 4 5	nodes:	3 4 5 6	nodes:	n n+1 n+2 n+3
face	nodes	face	nodes	face	nodes	face	nodes
1	1 2 3	1	-	1	-	1	-
2	-	2	-	2	-	2	n+1 n+2 n+3
3	3 4 1	3	4 5 2	3	5 6 3	3	n+2 n+3 n
4	4 1 2	4	5 2 3	4	6 3 4	4	n+3 n n+1

Figure 1: Ploy-Tetrahedral Strip

3.1.1 Node Numbering

The node numbering used within **pV3** is local. For multiple processor cases, this numbering need not have any reference outside the data on the client.

The node numbering used differentiates between the nodes in the non-block regions (formed by the disjoint cells and poly-tetrahedral strips) and the structured blocks. Figure 2 shows a schematic of the node space. **knode** is the number of nodes for the non-block grid. Each structured block (m) adds $NI_m * NJ_m * NK_m$ nodes to the node space (where NI, NJ and NK are the number of nodes in each direction). The node numbering within the block follows the memory storage, that is, (i,j,k) in FORTRAN and [k][j][i] in C. The **pV3** node number = $base + i + (j-1) * NI_m + (k-1) * NI_m * NJ_m$.
Note: all indices start at 1.

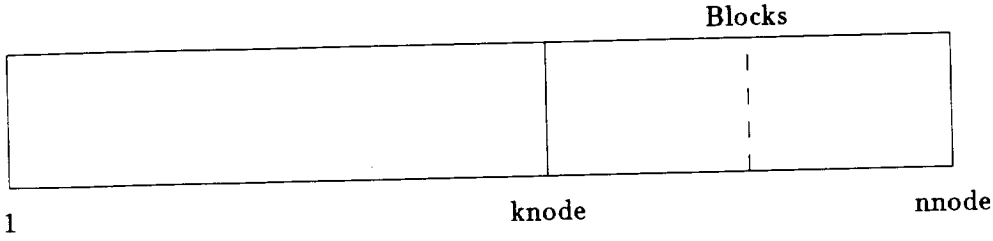


Figure 2: Node Space

3.1.2 Cell Numbering

The non-block cell types may contain nodes from the non-block and the structured block volumes. The cell numbering used within **pV3** orders the cells by type. Figure 3 shows a schematic of the cell space. The programmer explicitly defines all non-block cells by the call pVCell. Again the cells within the blocks are defined by the block size. Each structured block (m) adds $(NI_m - 1) * (NJ_m - 1) * (NK_m - 1)$ cells to cell space. The cell numbering within the block follows the memory storage so that a **pV3** cell number $= base + i + (j-1) * (NI_m - 1) + (k-1) * (NI_m - 1) * (NJ_m - 1)$.
Note: i goes from 1 to $NI_m - 1$, j goes from 1 to $NJ_m - 1$, and k goes from 1 to $NK_m - 1$.

Again, the numbering is local to the client for multiple processor applications.

3.1.3 Connectivity

In order to calculate streamlines and particle paths from vector fields **pV3** requires information about which are cells are neighbors, i.e. share a common face. There are two options; (1) either the programmer gives **pV3** these connections by setting IOPT (of pV_Init) negative and supplying the routine pVConnect, or (2) **pV3** calculates this information by processing all cells with exposed faces. This process compiles a list of all of these faces and checks

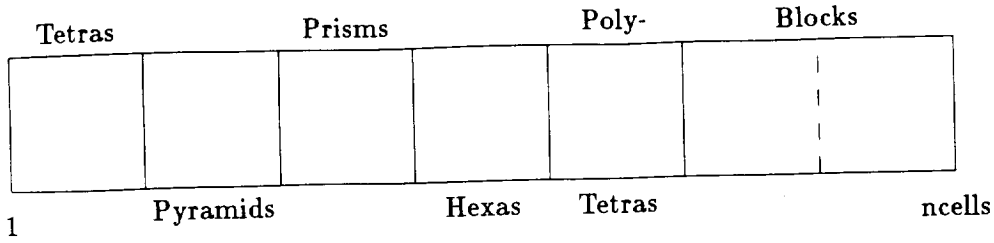


Figure 3: Cell Space

whether the face appears on another cell. If it does appear twice, then it is an interior face, and a streamline or particle can pass through from one cell to its neighbor. If it does not appear twice, then it is a surface face on the boundary of the computational domain, and a streamline will terminate when it hits the face.

Face matching between structured blocks is not possible using this automatic scheme. The node numbers that make up a face are different in both blocks even if they match in 3 space. The concept of 'node equivalency' allows the face matching to patch between regions. Node equivalency is simply a list of matching node numbers that get used only during this face matching procedure. This concept is generalized in **pV3** to allow equivalency to nodes anywhere in the local node space.

Anytime a streamline crosses a structured block, blanked or domain boundary and the integrator is able to continue (based on 'node equivalency' or other specified options), the accuracy is reduced for that segment.

3.1.4 Blanking

Blanking is an option (see the description of pV_Init) and only used with structured blocks to indicate that some region of the block is 'turned off'. This information is also used to give **pV3** an indication of how multi-block grids are connected in these areas. A part of a block is deactivated by flagging the appropriate nodes as invalid. This is done by an IBLANK array. An invalid node is never used. A cell with an invalid node is considered not to exist and therefore cuts and iso-surfaces through that cell will not be plotted. Also streamlines will not pass through cells with invalid nodes.

When blanking is used, all the nodes (**nnode** – **knode**) in the structured block space are given a value; zero corresponds to an invalid mesh point, any non-zero value indicates an existing node point. The value of one is the indication of an interior point. A negative value means that the physical space 'continues' in the block number that is the absolute

value of this IBLANK entry.

pV3 uses an algorithm for integrating streamlines and particle paths accross blanked boundaries by finding the closest node to the required position in the target block that has an IBLANK entry equal to the original block. If the negative blanked region is at the boundary of the block and all IBLANK entries of the exitting face are the same (and the case is not grid unsteady) the connectivity information is updated with the connected cell if found. Future integrations in the current session that pass through this face will not require the IBLANK node searching.

In the case of C-meshes and other topologies where the blocks abut, it is advisable to use 'node equivalency', if appropriate. Streamlines and particle paths are always faster going through a volume that has had face matching.

When the visualization is grid or structure unsteady, a new IBLANK array is requested for each snap-shot in time (after the coordinate triads are retrieved). If the blanking has not changed, the data need not be updated, and pVBlank should just return.

3.1.5 Surfaces

In principle, all surface faces could be grouped together to form one bounding surface for plotting purposes. However, in many applications it is more useful to split the bounding surface into a number of pieces, referred to earlier as *domain surfaces*. For example, the outer bounding surface of a calculation of airflow past a half-aircraft (using symmetry to reduce the computation) would typically be split into four pieces, the far-field boundary, the symmetry plane, the fuselage and the wing. If the programmer specifies each of these as separate *domain surfaces* by grouping the appropriate faces, then **pV3** can offer the capability of plotting on just one or two of the surfaces (e.g. the fuselage and wing) and not on the others (far-field and symmetry plane).

Internal surfaces are those that get created when the computational domain is sub-divided and and placed in multiple computational clients. These surfaces need only be defined to allow the passage of data (during integrations) from one domain to another.

3.1.6 Calling Sequences

pV3 supports steady-state visualization as well as three types of unsteady visualization. In a multi-client simulation, each client can have a different mode of unsteadyness. Each mode causes a different internal calling sequence. In general, the application must first call pV_Init to initialize the **pV3** client subsystem and then call pV_Update after every time the solution space has been updated.

- Steady-State

Call	Calls in Sequence
pV_Init	pVCell (optional)
	pVSurface
	pVEquiv (optional)
	pVGrid
	pVBlank (optional)
	all others as needed
pV_Update	NOT required

pV_Init does not return until the visualization session is over.

- Data Unsteady

Call	Calls in Sequence
pV_Init	pVCell (optional)
	pVSurface
	pVEquiv (optional)
	pVGrid
	pVBlank (optional)
pV_Update	pVScal
	pVVect (optional)
	all others as needed

- Grid Unsteady

Call	Calls in Sequence
pV_Init	pVCell (optional)
	pVSurface
	pVEquiv (optional)
pV_Update	pVGrid
	pVBlank (optional)
	pVScal
	pVVect (optional)
	all others as needed

- Structure Unsteady

Call	Calls in Sequence
pV_Init	NONE
pV_Update	pVStruc
	pVCell (optional)
	pVSurface
	pVEquiv (optional)
	pVGrid
	pVBlank (optional)
	pVScal
	pVVect (optional)
	all others as needed

3.1.7 Programming Notation

pV3 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 **pV3** have names which begin with 'pV_' or 'XFtn' and the common blocks have names which begin with 'PVC': such names should be avoided in an application program. It may also be helpful to look at the source code of the demo programs in the 'examples' directory of the distribution. 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 (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).

3.2 Programmer-called subroutines

3.2.1 pV_Init

PV_INIT(TITL, IOPT, NPGCUT, TPGCUT,
NKEYS, IKEYS, TKEYS, FKEYS, FLIMS, MIRROR,
KNODE, KEQUIV, KCEL1, KCEL2, KCEL3, KCEL4,
KNPTET, KPTET, KNBLOCK, BLOCKS, KSURF, KNSURF,
ISTAT)

This subroutine initializes **pV3**. This process involves enrolling the task in the **PVM** group 'pV3Client' which is required for connection to the graphics workstation. Calling this routine also defines the type of case, the sizes of various parameters and the types of functions defined. Returns immediately for all cases except steady-state ($IOPT = 0$).

TITL:i: C	Title (up to 80 characters used)
IOPT:i: I	Unsteady control parameter
	IOPT=-3 structure unsteady with connectivity supplied
	IOPT=-2 unsteady grid/data with connectivity supplied
	IOPT=-1 steady grid and unsteady data with connectivity supplied
	IOPT=0 steady grid and data
	IOPT=1 steady grid and unsteady data
	IOPT=2 unsteady grid and data
*NPGCUT:i: I	Number of programmer-defined cuts
*TPGCUT:i: C(NPGCUT)	Title for each cut (up to 32 characters used)
*NKEYS:i: I	Number of active keyboard keys
*IKEYS:i: I(NKEYS)	X-keypress return code for each key
*TKEYS:i: C(NKEYS)	Title for each key (up to 32 characters used)
*FKEYS:i: I(NKEYS)	Type of function controlled by each key:
	FKEYS()=1 Scalar
	FKEYS()=2 Vector
	FKEYS()=3 Surface scalar
	FKEYS()=4 Surface vector
	FKEYS()=5 Threshold

*FLIMS:i: R(2,NKEYS)	Function limits/scales **FKEYS()=1,3,5** Min and max values of function **FKEYS()=2,4** Arrow/tuft scaling (only the first element is used)
*MIRROR:i: I	Mirror flag: **MIRROR=0** No mirroring **MIRROR=1** Mirror about the plane X=0.0 **MIRROR=2** Mirror about the plane Y=0.0 **MIRROR=3** Mirror about the plane Z=0.0
KNODE:i: I	Number of non-block nodes
KEQUIV:i: I	Number or node equivalency pairs
KCEL1:i: I	Number of tetrahedra
KCEL2:i: I	Number of pyramids
KCEL3:i: I	Number of prisms
KCEL4:i: I	Number of hexahedra
KNPTET:i: I	Number of poly-tetrahedral strips
KPTET:i: I	Number of cells in all poly-tetrahedra
KNBLOCK:i: I	Number of structured blocks NOTE: A negative value indicates that blanking will be supplied for the blocks.
BLOCKS:i: I(3,KNBLOCK)	Structured block definitions: **BLOCKS(1,m)** = NI **BLOCKS(2,m)** = NJ **BLOCKS(3,m)** = NK
KSURF:i: I	Number of domain surface faces NOTE: A negative value is a flag to indicate that faces not connected to cells should be allowed.
KNSURF:i: I	Number of domain surface groups

ISTAT:i/o: i

On input this sets the startup/terminate state:

ISTAT=0 do not wait for graphics workstation (server) to start

ISTAT=1 wait for server to startup for the first time

ISTAT=2 do not wait for server / terminate with server

ISTAT=3 wait for server to startup / terminate with server

Only $ISTAT = 1$ and $ISTAT = 3$ are valid for steady-state cases ($IOPT = 0$).

On output any non-zero value is the indication of a startup error and the task is not included in the **pV3** client pool. See the Appendix for a list of the error codes.

Notes:

*) Multi-client cases: these parameters must match in all clients!

1) The X-keypress return codes for alphanumeric keys is identical to their usual ASCII integer codes.

2) A domain surface group is a collection of faces, which do not have to form a single connected surface, but form instead a logical grouping referred to earlier as being a *domain surface*. **pV3**'s initialization phase ($IOPT = 0, 1, 2$) examines each exposed face of each primitive, and determines whether or not it is shared with a neighboring cell. If not, it must be a surface face, but the user may choose to not declare it as such (see pVSurface). In this case, **pV3** takes all undeclared surface faces, splits them up into disjoint collections and calls the collection the 'Others' surface group. Therefore, the final number of domain surface groups can exceed KNSURF by one.

3) If NKEYS is negative, the absolute value of NKEYS is used for the number of keys and streamline/ribbon/tube/bubble calculations are disabled. This frees up a large amount of memory for $IOPT = 0, 1, 2$ cases (the cell connection information is discarded).

4) For structure unsteady cases, the parameters that describe the sizes of the node and cell space are the maximum sizes used during the simulation. The current sizes are set by a call to pVStruc from within pV_Update.

5) There will probably be a synchronization switch added to account for multi-client unsteady calculations that are not time-accurate. This will be added to ISTAT.

3.2.2 pV_Update

PV_UPDATE(TIME)

This subroutine must be called after the solver has updated the solution space. This is when the data is extracted and communication between the client(s) and the graphics workstation is done. The call to this routine is not needed if $IOPT = 0$.

TIME:i: R	The current simulation time. This sets the time in **pV3** for particle integration.

This routine is where all interaction with the graphics workstation is performed. Therefore the overall response and the interactive latency depends on how often this routine is called. About one call per second is optimal. If the soultion is updated significantly faster then most of the compute cycles will be used for the visualization, moving the solution slowly forward in time. In this case it is advisable to call pV_Update only every N times the solver updates the solution.

The more difficult case is when the solution update rate is much slower than optimal. In this situation, there are two choices; (1) live with large lags between user requests and screen response or (2) setup another task between the solver and the **pV3** server. This software's responsibility is to communicate with the solver. It should be the task to make all **pV3** calls.

This secondary task can communicate with the solver using **PVM** (and therefore must be on the same machine to avoid large network transfers). Or, if the machine supports multi-threading, the task can be a second thread and perform double-buffering of the solution space, so no data need be transferred. These methods are a trade-off of memory usage for interactivity. Multi-thread examples can be seen in the distribution in the directory 'examples/mthread'.

3.2.3 pV_Stat

PV_STAT(ISTATE)

This subroutine allows the programmer to query the status of the **pV3** system.

ISTATE:o: I	the status:
	ISTATE < 0 : error code from **pV3** - positive errors are designated by $-(1000 + code)$
	ISTATE=0 client not initialized
	ISTATE=1 no server
	ISTATE=2 server active

3.2.4 pV_Console

PV_CONSOLE(STRING)

This subroutine allows the programmer to have a string printed in the text window of the running **pV3** server application. The string will be prefaced by the **PVM** tid for multi-client cases.

STRING:i: C*80	character string to be output to the server

If the server is not running, this call does nothing.

3.2.5 pV_Termin

PV_TERMIN

This subroutine gracefully removes the client from the **pV3** system by leaving the group 'pV3Client' and deallocating associated memory.

No Arguments	pV_Init must be called again to use **pV3**

3.3 Programmer-supplied subroutines

The first set routines are the principal ones which will be used in most applications.

3.3.1 pVCell

PVCELL(CEL1, CEL2, CEL3, CEL4, NPTET, PTET)

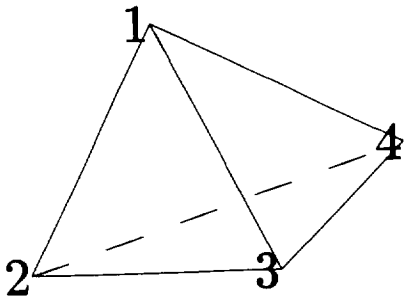
This subroutine supplies **pV3** with the grid data structure. It is not required for a grid that contains only structured blocks.

CEL1:o: I(4,KCEL1)	Node pointers for tetrahedral cells
CEL2:o: I(5,KCEL2)	Node pointers for pyramid cells
CEL3:o: I(6,KCEL3)	Node pointers for prism cells
CEL4:o: I(8,KCEL4)	Node pointers for hexahedral cells
NPTET:o: I(8,KNPTET)	Poly-Tetrahedra strip header: **NPTET(1,n)** = the pointer to the end of the strip n, i.e. it points to the last entry in PTET for the poly-tetrahedral strip **NPTET(2,n)** = the first node in the poly-tetrahedra **NPTET(3,n)** = the second node in the poly-tetrahedra **NPTET(4,n)** = the third node in the poly-tetrahedra **NPTET(5-8,n)** are used by **pV3**
PTET:o: I(KPTET)	The rest of each poly-tetrahedra, 1 node per cell

Notes:

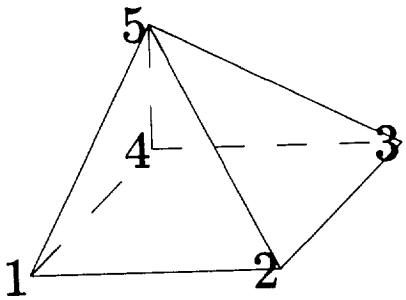
1) If KCELn is zero, the corresponding CELn must NOT be filled. And the same holds true for NPTET and PTET.

2) The correct order for numbering nodes for the four disjoint cell types is shown in Fig. 4. The Poly-Tetrahedra numbering is shown in Fig. 1.



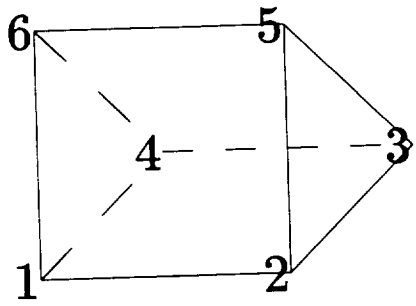
face	nodes
1	1 2 3
2	2 3 4
3	3 4 1
4	4 1 2

Tetrahedron



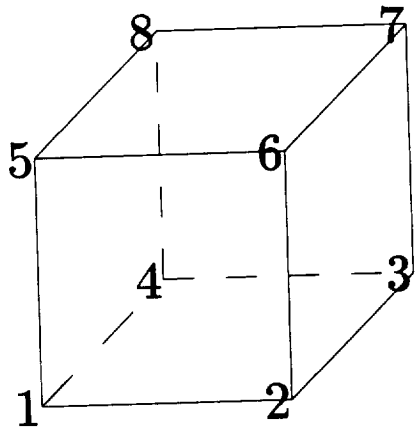
face	nodes
1	1 2 3 4
2	2 3 5
3	3 4 5
4	4 5 1
5	5 1 2

Pyramid



face	nodes
1	1 2 3 4
2	2 5 6 1
3	3 4 6 5
4	4 6 1
5	5 2 3

Prism



face	nodes
1	1 2 3 4
2	2 3 7 6
3	3 4 8 7
4	4 8 5 1
5	5 6 7 8
6	6 5 1 2

Hexahedron

Figure 4: Disjoint cell types and node/face numbering

3.3.2 pVSurface

PVSURFACE(NSURF, SCON, SCEL, TSURF)

This subroutine supplies **pV3** with the surface data structure.

NSURF:o: I(3,KNSURF)

NSURF(1,n) is the pointer to the end of domain surface group n, i.e. it points to the last entry in both SCON and SCEL for that group.

NSURF(2,n) is the startup drawing/mapping state (the following are additive):

0 off

1 render

2 grid

4 grey flag

8 thresholded

16 contours

32 translucent

256 2D mapping is provided (see note 1)

NSURF(3,n) is the global surface number (needed for multi-client cases only). A non-positve number is the indication that the surface is an internal boundary caused by domain decomposition. The number must be the **PVM** tid (negated) or zero. Zero is a special flag (along with $SCON = -1$) to allow an integration to try all other clients. The global surface number must be less than 3599!

SCON:o: I(KSURF)

The cell number to connect. This is the cell number in the local cell space of the **PVM** tid (specified above) for the connecting cell. If the value is -1, then an attempt is made to pass the particle into that domain or if NSURF(3,n) is zero all domains (except the current). A -1 if NSURF(3,n) is greater than zero, or for single client cases indicates that a re-enter attempt should be tried in this volume of data. A value of zero signals that there is no connection.

SCEL:o: I(4,KSURF)

node numbers for surface faces. For quadrilateral faces SCEL must be ordered clockwise or counter-clockwise; for triangular faces, SCEL(4,n) must be set to zero.

TSURF:o: C*20(KNSURF) titles for domain surfaces (optional)

Notes:

1) If the 2D mapping bit is set in NSURF(2,n), that is a flag to indicate that this surface has a 2D mapping. pVSurf, pVXYSurf, and optionally pVSSurf and pVVSurf will be provided and will respond to this surface.

2) The correct order for numbering faces for the four disjoint cell types is shown in Fig. 4. The face definitions for Poly-Tetrahedral cells is displayed in Fig. 1. For structured blocks; face #1 is for exposed cells with cell index $k = 1$, face #2 is for $i = NI_m - 1$, face #3 is for cells with $j = NJ_m - 1$, face #4 is for $i = 1$, face #5 is associated with $k = NK_m - 1$, and face #6 is for $j = 1$.

3) See Appendix B for a table of NSURF(3,n) and SCON options.

3.3.3 pVEquiv

PVEQUIV(LISTEQ)

This subroutine supplies **pV3** with node equivalency data. Required for $KEQUIV \neq 0$ cases.

LISTEQ:o: I(2,KEQUIV) Node equivalency pairs.

Notes:

1) For multiple (more than 2) node matching, always have the lowest node number in each entry. e.g, for a node at an edge between 4 blocks that line up, 3 node equivalency pairs are required, each matching the lowest node number with the others.

3.3.4 pVBlank

PVBLANK(IBLANK, TBCON)

This subroutine supplies **pV3** with blanking data. Required for $KNBLOCK < 0$ cases.

IBLANK:o: I(NNODE-KNODE) Blanking data:

< 0 node is valid and connected to this local block number (negated) - fill TBCON for multi-client cases.

$= 0$ off, invalid node

> 0 on

TBCON:o: I(NNODE-KNODE) **PVM** tid for block number. Zero indicates this client (multi-client cases only)

3.3.5 pVGrid

PVGRID(XYZ)

This subroutine supplies **pV3** with the grid coordinates.

XYZ:o: R(3,NNODE)	(x, y, z)-coordinates of grid nodes, using left-handed coordinate system. If right-handed coordinates are desired reverse sign of the z values.

3.3.6 pVScal

PVSCAL(JKEY,S)

This subroutine supplies **pV3** with scalar function values (FKEY=1).

JKEY:i: I	Key index, relative to ordering specified in pV_Init. (i.e. first key is 1, second is 2, third is 3, etc.)
S:o: R(NNODE)	Scalar function values.

3.3.7 pVThres

PVTHRES(JKEY,XYZ,T)

This subroutine supplies **pV3** with threshold function values (FKEY=5).

JKEY:i: I	Key index
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes
T:o: R(NNODE)	Threshold function values

Notes:

1) XYZ is passed by **pV3** to the user subroutine, in case it is needed to calculate T but, for storage reasons, the user's program has not kept a copy of XYZ. XYZ must not be changed by pVThres.

3.3.8 pVVect

PVVECT(JKEY,V)

This subroutine supplies **pV3** with vector function values (FKEY=2).

JKEY:i: I	Key index
V:o: R(3,NNODE)	Vector function values (Vx, Vy, Vz). If right-handed coordinates are desired reverse sign of the Vz values.

3.3.9 pVStruc

PVSTRUC(KNODE, KEQUIV, KCEL1, KCEL2, KCEL3, KCEL4,
KNPTET, KPTET, KNBLOCK, BLOCKS,
KSURF, KNSURF, HINT)

This subroutine is required for structure unsteady cases ($IOPT = -3$) only. This routine supplies the sizes of the current state of the problem.

KNODE:o: I	Number of non-block nodes / static flag
KEQUIV:o: I	Number or node equivalency pairs
KCEL1:o: I	Number of tetrahedra
KCEL2:o: I	Number of pyramids
KCEL3:o: I	Number of prisms
KCEL4:o: I	Number of hexahedra
KNPTET:o: I	Number of poly-tetrahedral strips
KPTET:o: I	Number of cells in all poly-tetrahedra
KNBLOCK:o: I	Number of structured blocks
BLOCKS:o: I(3,KNBLOCK)	Structured block definitions: **BLOCKS(1,m)** = NI **BLOCKS(2,m)** = NJ **BLOCKS(3,m)** = NK
KSURF:o: I	Number of domain surface faces NOTE: A negative value is a flag to indicate that faces not connected to cells should be allowed.
KNSURF:o: I	Number of domain surface groups
HINT:o: I	Hint on what to do with particle locations: **0 - nothing** - if the old cell number is still valid, use it to start **1 - nearest node** - use the nearest node to find a valid cell **2 - nearest surface node** - use the nearest surface node to find a valid cell **3 - use supplied info** - call pVLocate to get the new cell

Notes:

1) If KNODE is -1 that is a special flag to indicate that the structure has NOT changed for this iteration. With this flag set, no other paramaters should be modified, in that **pV3** reverts to the grid unsteady calling sequence.

2) Gets called every time in pV_Update even if no visualization is active.

3) Flag for blanking must be set in pV_Init.

4) Performance is enhanced by using HINT = 3. The other options exist in the case that a translation from the old structure does not exist. In these cases, pick the option that, in general, gives a cell number closest to the target. HINT = 2 is initially less compute intensive, but may require many cell *walks* to get to the actual location. Calls to pVConnect will be used to locate the the actual cell once the integration begins.

3.3.10 pVLocate

PVLOCATE(PXYZ,KCOLD,KCNEW)

This subroutine supplies **pV3** with the cell locations for particles in an $IOPT = -3$ and $HINT = 3$ case. This will be called for each active particle and many of the StreamLine seed positions.

PXYZ:i: R(3)	The current (pre-integrated) position of the particle.
KCOLD:i: I	Cell number in the old structure that contained the point PXYZ.
KCNEW:o: I	The new cell number that contains PXYZ, used to continue (or start) the integration (this does not have to be the actual cell that contains the point, but should be close). A zero indicates that there is no new cell (the domain no longer exists at the location). If the location is now in another client, return a valid cell number (in this client) that will cause the integration to continue to the target client when pVConnect is called.

3.3.11 pVConnect

PVCONNECT(KCOUT,KFOUT,KCIN,IDTIN)

This subroutine supplies **pV3** with cell connectivity for cases where $IOPT < 0$. This is not called for the interior of structured blocks or poly-tetrahedral strips.

KCOUT:i: I	Exitting cell number for the integration
KFOUT:i: I	The face number of the cell
KCIN:o: I	The entering cell number inorder to continue the integration. A zero indicates that there is no entering cell (the entire domain has been exitted). A negative number is an indication that the integration should attempt to re-enter the domain. The number must be the index to SCEL of the exitting face for single client cases or when IDTIN is -1. This insures that a re-entry will not occur to that face.
IDTIN:o: I	Multi-client only. A -1 is the flag that the cell number is in this client. Zero flags an attempt to reenter any other client's domain (all values of KCIN except zero are ignored). Any other value specifies the **PVM** tid for the client that has the cell. Re-entries (negative KCINs) will be attempted in this client.

Notes:

1) See Appendix B for a table of IDTIN and KCIN options.

The next two routines are needed for the programmer-defined cutting planes.

3.3.12 pVZPrime

PVZPRIME(IDCUT,XYZ,NNODE,ZP,ZPRIME,XPC,YPC,HALFW)

This subroutine is called when the programmer-defined cutting plane is initialized, to set up the 2D data.

IDCUT:i: L	Selected cut number (1 to NPGCUT). It will be positive to request ZPRIME, XPC, YPC and HALFW.
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes (as set in pVGrid)
NNODE:i: I	total number of nodes
ZP:o: R(NNODE)	z' values
ZPRIME:o: R	starting z' value
XPC:o: R	desired x' value at center of 2D window
YPC:o: R	desired y' value at center of 2D window
HALFW:o: R	desired half-width for square 2D window

Notes:

1) The last four variables should be set only if IDCUT > 0.

2) For IOPT=+/-1, it is assumed that ZP does not change with time.

3.3.13 pVXYPrime

PVXYPRIME(ZPRIME,KN,XYZ,N,XYP)

This subroutine supplies **pV3** with the (x', y') values at selected dynamic surface nodes.

ZPRIME:i: R	current z' value
KN:i: I(N)	set of pointers to surface nodes
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes (as set in pVGrid)
N:i: I	number of selected surface nodes
XYP:o: R(2,N)	(x', y') values at surface nodes

Notes:

1) There is no ordering of the nodes in KN, and in fact it may contain the same node more than once.

The next two routines are needed only for the mapping of domain surfaces.

3.3.14 pVSurf

PVSURF(ISURF,XPC,YPC,HALFW)

This subroutine is called once when a mapped 2D surface is required. This will only be called for those surfaces that have been described as mapped (see pVSurface, Note 1).

ISURF:i: I	the global surface number
XPC:o: R	desired x' value at center of 2D window
YPC:o: R	desired y' value at center of 2D window
HALFW:o: R	desired half-width for square 2D window

3.3.15 pVXYSurf

PVXYSURF(KN,XYZ,N,XYP)

This subroutine supplies **pV3** with the (x', y') values at selected mapped domain surface nodes.

KN:i: I(N)	set of pointers to surface nodes
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes (as set in pVGrid)
N:i: I	number of selected surface nodes
XYP:o: R(2,N)	(x', y') values at surface nodes

Notes:

1) There is no ordering of the nodes in KN, and in fact it may contain the same node more than once.

The next two routines are needed for mapped domain surfaces and for surface integrations using the special surface functions.

3.3.16 pVSSurf

PVSSURF(JKEY,KN,XYZ,N,S)

This subroutine supplies **pV3** with surface scalar values (FKEY=3) at selected mapped domain surface nodes.

JKEY:i: I	Key index
KN:i: I(N)	set of pointers to surface nodes
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes (as set in pVGrid)
N:i: I	number of selected surface nodes
S:o: R(N)	Scalar function values at surface nodes

Notes:

1) There is no ordering of the nodes in KN, and in fact it may contain the same node more than once.

3.3.17 pVVSurf

PVVSURF(JKEY,KN,XYZ,N,V)

This subroutine supplies **pV3** with surface vector values (FKEY=4) at selected mapped domain surface nodes.

JKEY:i: I	Key index
KN:i: I(N)	set of pointers to surface nodes
XYZ:i: R(3,NNODE)	(x, y, z)-coordinates of grid nodes (as set in pVGrid)
N:i: I	number of selected surface nodes
V:o: R(3,N)	Vector function values at surface nodes

Notes:

1) There is no ordering of the nodes in KN, and in fact it may contain the same node more than once.

The following routines are used to communicate within the pV3 system.

3.3.18 pVString

PVSTRING(STRING)

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

STRING:o: C*80	character string label

3.3.19 pVCatch

PVCATCH(STRING)

This routine allows the the client to get a text string from the the running **pV3** server application. This is usefull for steering.

STRING:i: C*80	character string sent from the server

3.4 C Programming

Using **pV3** with C requires that the programmer do some things that may not be intuitive. This is because FORTRAN is supported with the same bindings.

The following rules must be followed:

- 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.

- 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, specify the length as the position before the null, or remove the null. For programmer supplied routines with character arguments 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.

- defalut types

  The translation between FORTRAN INTEGERs and REALs and their C counterparts depends on the FORTRAN default size. In all cases *float* may be used for REALs. *int* (or *long*) must be used for INTEGER.

4 Portability

4.1 FORTRAN Programming

For the most part FORTRAN source that runs on one implementation of **pV3** will work on others.

4.2 C Programming

Unfortunately **pV3** is not source code compatible for C across all machines. This has to do with supporting FORTRAN.

The C programmer that wishes to have their **pV3** application running on many platforms should be aware that:

- main program

  On IBM and HP workstations, normal C conventions apply. For DEC and SGI ports the name of the main program must be 'MAIN__', on KSR machines the name is 'main_', and on all others the name is 'MAIN_'.

- routine names

  For IBM and HP ports, all **pV3** entry points are the FORTRAN names in lower-case. On all other platforms, the external entries are lower-case with an underscore ('_') appended to the end.

- integers

  pV3 uses the default FORTRAN INTEGER for all integer arguments. This almost always corresponds to an *int*. KSR is the only exception. In this case all **pV3** integers must be *long*!

See the file 'osdepend.h' in the examples subdirectory of the distribution for a method to avoid these problems.

A Error Codes

Any error generated at the call to pV_Init invalidates the client for inclusion in the visualization.

If any of these errors are generated during a Structure Unsteady visualization, that volume is invalid for the current time step. The user will see no data coming from this client. The structure is checked again at the next call to pV_Update.

The following codes report a more detailed message to standard output:

-1	Maximum Number of Surface Faces Exceeded! The size for the temporary face structure was too small. To increase the storage, give KSURF a larger value so that the face matching procedure can run to completion.
-2	Degenerate face! A degenerate face was found. If the cell is really degenerate (has the same node numbers in multiple entries) use the appropriate cell type!
-3	Face Hit by 3 (or more) Cells/Surfaces! The is part of more than 2 objects. Any face must be touched by either 2 cells (an internal face) or a cell and a surface face. If you are trying to put a *domain surface* where **pV3** thinks the face is internal, change the node numbering on one of the cells to indicate a new face. This will probably require that additional nodes be added to the node space.
-4	Surface Face with no Connecting Cell! A face has been specified that is not part of the volume, it is not a face of a cell.
-5	Maximum Number of Surface Faces Exceeded while constructing the surface 'Others'! This can occur if the number of cells is small compared to the number of surface faces. To increase the storage, give KSURF a larger value so that the face matching procedure can run to completion.
-6	Memory Allocation Error!
-7	Structure Check error. A more detailed list is given on standard output.
-8	Edge Table Overflow! The size for the temporary edge structure was too small. To increase the storage, give

	KSURF a larger value so that the edge matching procedure can run to completion.
-9	Maximum Number of Edge Lines Exceeded!

The following codes generate no additional information:

-102	The client library was not built properly! You should not see this error.
-101	Some PVM error was encountered during initialization! This happens when either the PVM daemon is not running, or the PVM group subsystem cannot be initialized.
-100	The pV3 client system is already initialized!
1	Control Parameter Out of Range! IOPT is greater than 2 or less than -3.
2	NKEYS is less than 1!
3	NKEYS is greater that the maximum (currently 64)!
4	FKEYS(i) out of range! An entry in FKEYS is less than one or greater than five.
5	FLIMS(1,i) equals FLIMS(2,i)! A set of entries in FLIMS have the same value. This is not legal for **pV3**.
6	Memory Allocation Error! **pV3** has requested a block of memory and has been refused. This is usually do to the problem's size.
7	Degenerate Block! One of the block sizes is less than two.
8	KPTET is less than zero!
9	NPTET(1,MAX) <> KPTET! The last entry in NPTET does not match the value KPTET.
10	KNODE <> 0 but no non-structured block cells!
11	NSURF(1,KNSURF) > KSURF! The last entry in NSURF is larger than the size given at initialization.
12	Structure Unsteady - no change flag set without any prior definition!
13	Structure Unsteady - current size greater than initially specified.

B Multi-client Connectivity Options

There are a limitted number of segments allowed for streamlines. When that limit is reached (currently 4 times the number of clients) a warning is displayed and that streamline will terminate.

If the interface between clients is ragged (such as found with tetrahedra meshes), a command to enter the domain (cell number = -1) may fail.

An attempt to (re)enter (cell number = -1) a client is **MUCH** more compute intensive than specifying the cell number (or even a cell that is close to the target)!

B.1 pVConnect

When the programmer is responsible for the connectivity (any negative IOPT in pV_Init) the routine pVConnect must be supplied. It is called during particle integrations to move data from one cell to the next. There are many options based on where the the target cell resides.

- IDTIN = -1

  The integration continues in the current client.

KCIN	Comment
negative	try to re-enter in this client
	value must be index to SCEL of exitting face (negated)
zero	stop - hit boundary
positive	continue to this cell number

- IDTIN = 0

  Try to enter into all other clients. KCIN is ignored unless zero.

- IDTIN = positive number

  Continue the integration into client with this **PVM** tid.

KCIN	Comment
-1	try to enter in the client specified by IDTIN
zero	stop
positive	continue to this cell in IDTIN

B.2 pVSurface

The programmer can also specified the multi-client topology by the data returned from the routine pVSurface. In this case *internal surfaces* must be constructed to patch regions together. The surfaces define the connectivity with the following options:

- NSURF(3,n) = negative number

  The absolute value of the number is the **PVM** tid to continue the integration.

SCON(i)	Comment
-1	try to enter in the client specified by -NSURF(3,n)
zero	stop - hit boundary
positive	continue to this cell in -NSURF(3,n)

- NSURF(3,n) = 0

  Try to enter into all other clients. SCON(i) must be -1.

- NSURF(3,n) = positive number

  The integration continues in the current client.

SCON(i)	Comment
-1	try to re-enter in this client
zero	stop

Advanced Programmer's Guide

for

pV3 Rev. 1.00

Bob Haimes

August 3, 1994

DISCLAIMER: This programming interface is in its infancy. The goal is to mimic **Visual3***s advanced programming interface as much as possible. Attempts will be made not change anything already defined in this manual.*

License

Contents

1 Introduction

The control of the **pV3** visualization session is driven by the user. Much time and effort were placed in making that interaction *real-time*. But there are many instances when you may want the control of what is rendered on the screen to be specified by a program. Examples of this are demonstrations, visual results from expert-systems, macros and etc.

In that the **pV3** server is an **X** windows application, control is driven by **X** events. Figure 1 shows the server's normal processing loop. In general, all **X** events are extracted from the **X** event queue and the internal **pV3** state information is adjusted appropriately. When all events are exhausted, any changes to state that require getting data from the application cause the server to request various extracts from the clients. After this information collection phase is done, the windows that require updating are redrawn during the rendering phase. Then the event queue is checked for more user interaction, and this continues for the life of the **pV3** server.

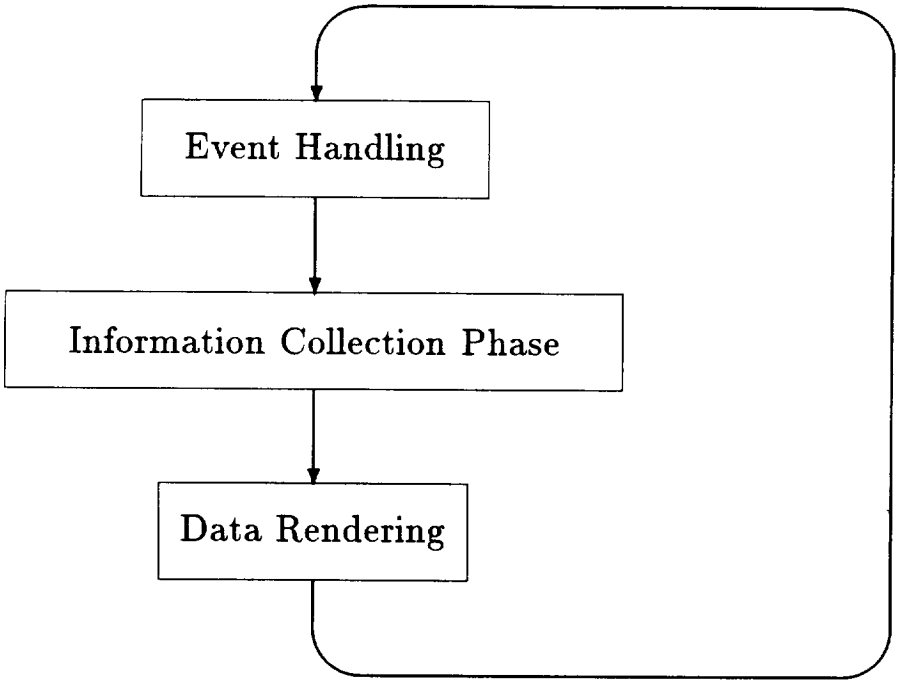


Figure 1: The **pV3** Server's Internal Loop

Also, to simplify the data handling, **pV3** only exposed it's internal data to the programmer when it required data on the client side. The data generated to produce cut surfaces, streamlines and other objects was hidden. This document describes how the advanced **pV3** programmer can access this extract information on the server side.

2 pV3 Server Event Handling

One of the goals of allowing programming control in **pV3** is to maintain the current action as the default. Also this must allow the programmer to either modify the effects of an event or completely take over.

Before attempting to program **pV3** at this level it is important to understand the server's program structure. This includes the internals of the event handling phase.

The details of the **pV3** server event handling phase can be seen in Figure 2. First, an **X** event is pulled of the **X** event queue. The event is described by a window ID - *IWIN* (the window in which the event occurred), the type of event (e.g., KeyPress, ButtonPress, and etc.), the location in the window of the event, and finally any state data (e.g., which button or key was pressed). There is a special window ID flag of -1 which indicates that the event queue was empty.

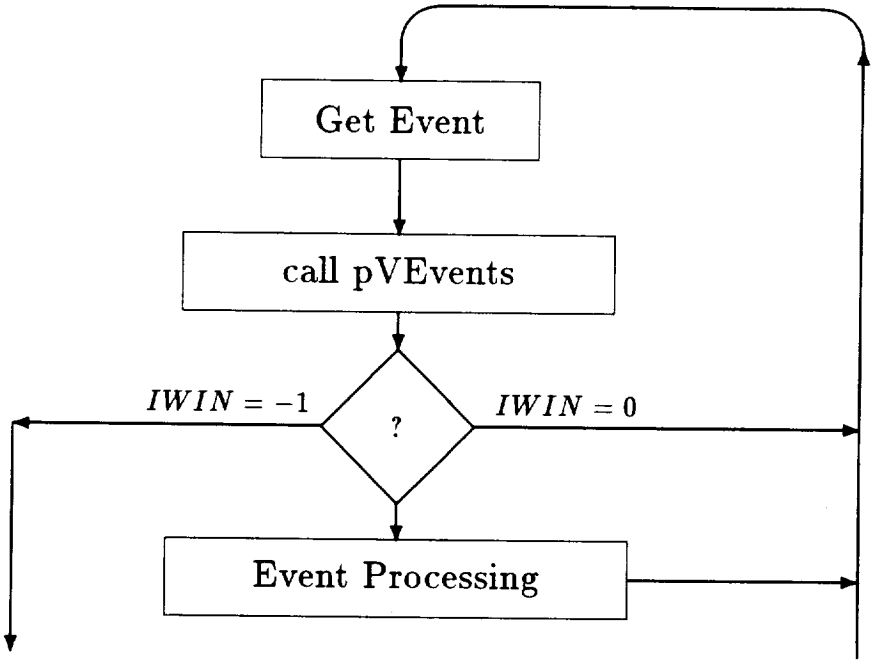


Figure 2: **pV3** Server's Event Handling

Next a programmer-supplied routine **pVEvents** is called with the event data. **pVEvents** has the option of passing the event on (doing nothing), performing some action based on the event, changing the event (by modifying the arguments) or having the event ignored (by setting *IWIN*, the window ID, to 0). The other routines described in this document may be called within **pVEvents** to inquire and set many of the **pV3** internal state variables.

On the basis of the value of $IWIN$, as returned by **pVEvents**; the event processing section of **pV3** is called when $IWIN > 0$, the event is ignored if **pVEvents** sets $IWIN = 0$, or this phase is terminated with $IWIN = -1$. When changing the action of the **pV3** server it is important to remember that nothing will be updated in the windows until **pVEvents** returns with $IWIN = -1$. Also, if **pVEvents** was called with $IWIN = -1$ and an event is constructed and passed through to the event processing section, **pVEvents** will be re-entered (possibly with $IWIN \neq -1$), before the screen gets updated.

Examples of **pVEvents**, that perform various functions, are contained within the distribution.

NOTE:
The **pV3** server is distributed in two forms. First, as a complete application. This is what would be executed if no server modifications are desired. Second, it is distributed in library and object module format. This is required if a different release of **PVM** was used to build the server than what is currently used at your site. This allows the re-building of the server without supplying the source. Because the **pV3** server is already distributed in a library form, all that is required to modify the server (as described in this manual), is to *make* the server including the compiled routines that you supply.

3 The Use of Pointers in the pV3 server using FORTRAN

Some of the data that gets returned from **pV_GetPointer**, **pV_GetSub** and **pV_GetExtract** is in the form of pointers to blocks of memory. The pointers can be treated as INTEGER*4 variables. Traditionally, this information cannot be used by the FORTRAN programmer. But, a mechanism exists on all major workstation's FORTRAN (including SGI) to allow the pointer to be passed to a SUBROUTINE or FUNCTION and then have the memory treated as a normally declared vector or array. This is done by the VAX extension '%val(pointer)' used in the CALL or function invocation. When the pointer is passed to the sub-program by 'value', it is equivalent to passing a variable by 'reference' (the FORTRAN method). That is, in both cases, the address of the memory of interest is placed on the stack!

Using this mechanism (or the POINTER statement), sophisticated **pV3** server enhancements may be performed using FORTRAN. To complete this picture, the following routines can be used to allocate and free memory blocks.

3.1 MALLOC

PTR = MALLOC(NBYTES)

This function is equivalent to the C routine 'malloc'. It allocates a block of memory and returns the pointer to the block.

NBYTES:i: I	The number of bytes to allocate.
PTR:o: I	The address of the block (0 is an error indicator).

3.2 FREE

FREE(PTR)

This function is equivalent to the C routine 'free'. It deallocates a block of memory. NOTE: Only free up blocks of memory that YOU allocate!

PTR:i: I	The address of the block.

4 Server Side Extracts

The following section describes the internal data stored in the **pV3** server. This is the data used to produce the graphics objects that get rendered to make the scene. Each tool generates a different type of *extract* from the 3D data in the client. The data gets transmitted to the server and is stored for as long as it is needed. Each *extract* consists of a number of *sub-extract* types, and there is a complete collection of *sub-extracts* for each client. Note: each clients data is stored separately.

4.1 Surfaces

This data is generated by the **pV3** scalar tools (planar cuts, programmed cut surfaces, iso-surfaces and domain surfaces). This data is exposed so that new 'probes' may be easily generated. The size of many of these arrays (and therefore the pointers) will change during the execution of **pV3**, so when using this data, get the current pointers before accessing the memory.

Extract	Type	Valid Sub-Extracts
2	Planar Cut	0 1 2 3 4 5 6 7 9 10*
4	Geometric Cut	0 1 2 3 4 5 6 7 8 9 10 11
5	Domain Surface	0 1 2 3 4 5 6 7 8 9 10 11
7	Iso-Surface	0 1 2 3 4 5 6 7 10 11

4.1.1 0 - Surface Sub-Extract *Tris*

The following data defines the disjoint triangle space. Where the number of triangles in the structure is KTRI.

TRIS: I(4,KTRI) — disjoint triangle definitions.

TRIS(1,n) = first node index for the triangle.

TRIS(2,n) = second node index for the triangle.

TRIS(3,n) = third node index for the triangle.

TRIS(4,n) = the parent 3D cell number (in the client).

4.1.2 1 - Surface Sub-Extract *Quads*

The following data defines the disjoint quadrilateral space. Where the number of quadrilaterals in the structure is KQUAD.

QUADS: I(5,KQUAD) disjoint quadrilateral definitions.

QUADS(1,n) = first node index for the quadrilateral.
QUADS(2,n) = second node index for the quadrilateral.
QUADS(3,n) = third node index for the quadrilateral.
QUADS(4,n) = fourth node index for the quadrilateral.
QUADS(5,n) = the parent 3D cell number (in the client).

4.1.3 2 - Surface Sub-Extract *XYZ*

The following data defines the 3D coordinates for the nodes (and therefore also the number of nodes) that support the surface. The number of nodes in the structure is KXYZ.

XYZ: R(3,KXYZ) (x, y, z)-coordinates for the nodes.

4.1.4 3 - Surface Sub-Extract *Mesh*

The following data defines the disjoint lines that make-up the intersection of the cell edges and the cutting surface. The number of line segments in the structure is KFACE.

FACE: I(2,KFACE) disjoint line definitions.

FACE(1,n) = first node index for the line.
FACE(2,n) = second node index for the line.

4.1.5 4 - Surface Sub-Extract *Outline*

The following data defines the disjoint lines that make-up the outline of the surface. The number of line segments in the structure is KEDGE.

EDGE: I(3,KEDGE) disjoint line definitions.

EDGE(1,n) = first node index for the line.
EDGE(2,n) = second node index for the line.
EDGE(3,n) = the parent surface face number (in the client).

4.1.6 5 - Surface Sub-Extract *Scalar*

The following data defines the current scalar for the nodes (and therefore also the number of nodes) that support the surface. The number of nodes in the structure is KS and is the same as KXYZ.

S: R(KS) scalar functional values for the nodes.

4.1.7 6 - Surface Sub-Extract *Vector*

The following data defines the current vector for the nodes (and therefore also the number of nodes) that support the surface. The number of nodes in the structure is KV and is the same as KXYZ.

V: R(3,KV) vector values (Vx, Vy, Vz) for the nodes.

4.1.8 7 - Surface Sub-Extract *Threshold*

The following data defines the current threshold values for the nodes that support the surface. The number of nodes in the structure is KT and is the same as KXYZ.

T: R(KT) threshold functional values for the nodes.

4.1.9 8 - Surface Sub-Extract *2D Mapping*

The following data defines the 2D mapping for the nodes that support the surface. The number of nodes in the structure is KXY and is the same as KXYZ.

XY: R(2,KXY) raw (x', y')-coordinates as specified by the client.

Notes:
(1) The 2D mapping for planar cuts is implicit and not required from the client.
(2) There is no 2D mapping for iso-surfaces.

4.1.10 9 - Surface Sub-Extract *Transformed 2D Mapping*

The following data defines the 2D mapping used to draw the surface into the 2D window, where the viewed range is between -1.0 and 1.0 in both x' and y'. This is a simple transformation of XY. The number of nodes in the structure is KXYP and is the same as KXYZ.

XYP: R(2,KXYP) transformed (x', y')-coordinates.

Notes:
(1) This data is generated at the server and not transfered from the clients.
(2) If KYXP is zero, the transformation has not yet been computed.
(3) There is no 2D mapping for iso-surfaces.

4.1.11 10 - Surface Sub-Extract *Tri normals*

The following data defines the normals used (for each disjoint triangle). This data is used in the lighting model for drawing the surface in the 3D window. The number of normals in the structure is KTRIN and is the same as KTRI.

TRIN: R(3,KTRIN) (x, y, z)-normals for the triangle.

Notes:
(1) This data is generated at the server and not transfered from the clients.
(2) If KTRIN is zero, the normals have not yet been computed.
(3) The normals for planar cuts are implicit and the same for all triangles and therefore this sub-extract is not used (but the index is reused for clipping).

4.1.12 11 - Surface Sub-Extract *Quad normals*

The following data defines the normals used for each disjoint quadrilateral. This data is used in the lighting model for drawing the surface in the 3D window. The number of normals in the structure is KQUADN and is the same as KQUAD.

QUADN: R(3,KQUADN) (x, y, z)-normals for the quadrilateral.

Notes:
(1) This data is generated at the server and not transfered from the clients.
(2) If KQUADN is zero, the normals have not yet been computed.
(3) The normals for planar cuts are implicit and the same for all quadrilaterals and therefore this sub-extract is not used.

4.1.13 10* - Surface Sub-Extract *Planar clipping*

The following data defines the clipping index used for each node. The number of entries in the structure is KCLIP and is the same as KXYZ.

CLIP: I(KCLIP)

0 - not clipped, otherwise the following are additive:

1 - transformed x' less than -1.0

2 - transformed x' greater than 1.0

4 - transformed y' less than -1.0

8 - transformed y' greater than 1.0

Notes:

(1) This data is generated at the server and not transfered from the clients.

(2) If KCLIP is zero, the clipping index has not yet been computed.

(3) This sub-extract index is only used for planar cuts so that there is no conflict with triangle normals.

4.2 StreamLines

This data is generated by the **pV3** clients during the integration of instantaneous streamlines. The size of many of these arrays (and therefore the pointers) will change during the execution of **pV3**, so when using this data, get the current pointers before accessing the memory. Unlike all other Extracts, the number of sub-extracts is not a function of the number of clients but of the maximum allotted streamline segments (that is greater than the number of clients). This allows a streamline to reenter a client more than once.

4.2.1 0 - StreamLine Sub-Extract *Cell*

The following data contains the 3D cell number for the position of the point for this segment (used for the point probe). The number of entries in the structure is KCELL and is the same as KXYZ.

CELL: I(KCELL) — the parent 3D cell number (in the client).

4.2.2 1 - StreamLine Sub-Extract *Time*

The following data defines the integration pseudo-time for the point (used for streamline animation). Where the number of elements in the structure is KTIME and is the same as KXYZ.

TIME: R(KTIME) — integration time (from the seed position).

4.2.3 2 - StreamLine Sub-Extract *XYZ*

The following data defines the 3D coordinates for the points that support this poly-line segment. The number of nodes in the structure is KXYZ.

XYZ: R(3,KXYZ) — (x, y, z)-coordinates for the points.

4.2.4 3 - StreamLine Sub-Extract *Div*

The following data defines the cross-flow divergence felt by each point during the integration. Where the number of elements in the structure is KDIV and this is the same as KXYZ.

DIV: R(KDIV) — used for streamtube rendering, where the size of the tube is based on a starting size mutiplied by **e** to this power.

4.2.5 4 - StreamLine Sub-Extract *Angle*

The following data contains the curl for each point, calculated during the integration, in this segment of the streamline Where the number of entries in the structure is KANG and this is the same value as KXYZ.

ANG: R(KANG) angle of the twist for ribbons in degrees.

4.2.6 5 - StreamLine Sub-Extract *Scalar*

The following data defines the current scalar for the points that support the line in this segment. The number of points in the structure is KS and this is the same as KXYZ.

S: R(KS) scalar functional values for the points.

4.2.7 6 - StreamLine Sub-Extract *Vector*

The following data defines the current vector for the points that make up this segment of the streamline. The number of elements in the structure is KV and this is the same as KXYZ.

V: R(3,KV) vector values (Vx, Vy, Vz) for the points.

4.2.8 7 - StreamLine Sub-Extract *Threshold*

The following data defines the current threshold values for the points that support the poly-line. The number of entries in the structure is KT and is the same as KXYZ.

T: R(KT) threshold functional values for the points.

4.3 Particles

This data is updated by the **pV3** clients during the bubble integration at each time-step. The size of many of these arrays (and therefore the pointers) will change during the execution of **pV3**, so when using this data, get the current pointers before accessing the memory.

4.3.1 0 - Particle Sub-Extract *Number*

The following data contains the unique particle number for each bubble in that client. The number of entries in the structure is KNUM and this is the same as KXYZ.

NUM: I(KNUM)	the global particle number.

4.3.2 1 - Particle Sub-Extract *Time*

The following data defines the start time for each bubble. The number of elements in the structure is KTIME and this number is the same as KXYZ.

TIME: R(KTIME)	bubble simulation time when the particle was seeded.

4.3.3 2 - Particle Sub-Extract *XYZ*

The following data defines the current 3D coordinates for the particles. The number of nodes in the structure is KXYZ.

XYZ: R(3,KXYZ)	(x, y, z)-coordinates for the bubbles.

4.3.4 3 - Particle Sub-Extract *Div*

The following data defines the cross-flow divergence currently felt by each bubble. Where the number of elements in the structure is KDIV and this is the same as KXYZ.

DIV: R(KDIV)	optionally used for bubble rendering, where the size of the particle is based on a starting size mutiplied by **e** to this power.

4.3.5 5 - Particle Sub-Extract *Scalar*

The following data defines the current scalar for the particles in this client. The number of points in the structure is KS and this is the same as KXYZ.

S: R(KS) scalar functional values for the bubbles.

4.3.6 6 - Particle Sub-Extract *Vector*

The following data defines the current vector for the particles. The number of elements in the structure is KV and this number is the same as KXYZ.

V: R(3,KV) vector values (Vx, Vy, Vz) for the bubbles.

4.3.7 7 - Particle Sub-Extract *Threshold*

The following data defines the current threshold values for the particles. The number of entries in the structure is KT and is the same as KXYZ (the number of bubbles).

T: R(KT) threshold functional values for the bubbles.

4.4 Vector Clouds

4.4.1 2 - VC Sub-Extract *XYZ*

The following data defines the coordinates for the 3D nodes that satisfy the threshold limits within each client. The number of nodes in the structure is KXYZ.

XYZ: R(3,KXYZ)	(x, y, z)-coordinates for the vector cloud.

4.4.2 5 - VC Sub-Extract *Scalar*

The following data defines the current scalar for the vector cloud The number of points in the structure is KS and this number is the same as KXYZ.

S: R(KS)	scalar functional values for the 3D nodes.

4.4.3 6 - VC Sub-Extract *Vector*

The following data defines the current vector for each node in the client that satisfies the threshold limits. The number of elements in the structure is KV and this number is the same as KXYZ.

V: R(3,KV)	vector values (Vx, Vy, Vz) for the vector cloud.

5 Server Supplied Routines and Calls

Consistant with the **pV3** naming convension, the routines that are part of **pV3** server are prefixed with 'pV_', those that are supplied by the programmer start with 'pV'.

5.1 pVEvents

PVEVENTS(IWIN,TYPE,XE,YE,STATE)

This subroutine is called by the **pV3** server immediately after an **X** event has been pulled of the queue. This routine may perform certain functions based on the event, change the event status, have the server ignore the event or just do nothing.

IWIN:i/o: I	The **X** window ID of the event or: **IWIN=-1** No more events to process, therefore pass the control of the server to the data collection phase **IWIN=0** Output only. Ignore this event and pull the next one off the queue
TYPE:i/o: I	The event type. The event types that **pV3** windows can generate are: **TYPE=2** KeyPress **TYPE=3** KeyRelease **TYPE=4** ButtonPress **TYPE=5** ButtonRelease **TYPE=12** Expose **TYPE=14** NoExpose **TYPE=33 X** ClientEvent
XE:i/o: I	The X pixel location in the window of the event (0 is the left)
YE:i/o: I	The Y pixel location in the window of the event (0 is the top of the window)
STATE:i/o: I	The event state. For Key events, the state is the **X** KeySym number (usually the ASCII code except for the special keys). For mouse button events, the state is the button number.

5.2 pV_Cursor

PV_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

5.3 pV_GetPointer

PV_GETPOINTER(OPT,PTR)

Returns the internal **pV3** server structure. This routine can be called from any programmer-supplied code.

OPT:i: I	Option set to specify what data to get.
PTR:o: I	Integer (in FORTRAN) used as a pointer to the structure.

- OPT = 0 - **X** Event Structure:

  PTR = *XEvent - Useful for building another Graphical User Interface (GUI) on the **pV3** server. Allows passing the event information to tool-kits such as Motif. This need only be called once. The same structure is used for all **pV3** event handling.

- OPT = 1 - **X** window structure for the 1D Window

  PTR = *Window - for 1D Window

- OPT = 2 - **X** window structure for the 2D Window
- OPT = 3 - **X** window structure for the 3D Window
- OPT = 4 - **X** window structure for the Dails Window
- OPT = 5 - **X** window structure for the Key Window
- OPT = 6 - **X** window structure for the Text Window
- OPT = 7 - **X** window structure for the Comparison Window (0 means the window is not open)
- OPT = 8 - **X** window structure for the Root Window

5.4 pV_GetState

PV_GETSTATE(OPT,IVEC,RVEC,STRING)

Returns the internal **pV3** state. This **pV3** server routine can be called from any programmer-supplied code linked with the server.

OPT:i: I	Option set to specify what data to get
IVEC:i/o: I()	Integer data returned based on OPT (length also determined by OPT). Some OPTs control actions on input.
RVEC:o: R()	Real data returned based on OPT (length also determined by OPT)
STRING:o: C	Character data returned based on OPT

- OPT = -3 - Get client data:

  IVEC(1) = Client index - input (1 to total number of clients)

  IVEC(2) = PVM tid - the **PVM** task id of the client index

  IVEC(3) = IOPT - unsteady option for the client index

  IVEC(4) = Number of clients - the total number of clients

- OPT = -2 - Get unsteady info:

  IVEC(1) = IOPT - maximum for all clients

  IVEC(2) = Pause State - 0 not paused, 1 paused

  RVEC(1) = Time - simulation time currently displayed

- OPT = -1 - Get special key bindings. IVEC is filled with 32 integers that are the **X** KeySyms for the non-ASCII keys used by **pV3**. A file 'KeyBoard' contains the default **pV3** server KeySyms and the associated labels. These values may be modified and used by the server if the environment variable 'Visual_KB' is set to point to the file containing the new bindings.

- OPT = 0 - Get Rev number:

  IVEC(1) = pV3 flag always -1

  RVEC(1) = Revision Number

- OPT = 1 - Get the current transformation matrix for the 3D view. 12 words of RVEC are used.

- OPT = 2 - Get the current planar cut coordinates. 9 words of RVEC are used for 3 of the 4 corners of the cut plane in 3 space.
- OPT = 3 - Get the current scalar tool status.

  IVEC(1) = Current Tool :

  0 - No tool

  2 - Planar cut

  3 - Planar cut positioning on

  4 - Programmed cut

  5 - Mapped surface

  6 - Mapped surface with surface functions

  7 - Iso-surface

  RVEC(1) = Current ZPRIME - for 2, 4 and 7 above

  RVEC(2) = XPC - for 4, 5 and 6

  RVEC(3) = YPC - for 4, 5 and 6

  RVEC(4) = HALFW - for 4, 5 and 6

  RVEC(5) = 2D Rotation Angle in degrees - for 4, 5 and 6

- OPT = 4 - Lighting and Mirroring state:

  IVEC(1) = Mirror status :

  0 - No Mirroring

  1 - Mirror about X = 0.0

  2 - Mirror about Y = 0.0

  3 - Mirror about Z = 0.0

  IVEC(2) = Number of Lights maximum is 8

  RVEC(1-3) = Ambient light color (r,g,b) - values between 0.0 and 1.0

- OPT = 5 - Scalar state:

  IVEC(1) = Binding index

  RVEC(1) = Minimum for display - fmin shown in the key window

  RVEC(2) = Maximum for display - fmax shown in the key window

  STRING = Scalar title - maximum of 32 characters used.

- OPT = 6 - Vector state:

  IVEC(1) = Binding index

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  STRING = Vector title - maximum of 32 characters used.

- OPT = 7 - Threshold state:

  IVEC(1) = Binding index - negative value indicates the appropriate scalar index

  IVEC(2) = Vector Cloud status - 0 = off, 1 = on

  IVEC(3) = Dynamic Threshold status - 0 = off, 1 = on

  RVEC(1) = Minimum for display - tmin shown in the key window

  RVEC(2) = Maximum for display - tmax shown in the key window

  STRING = Threshold title - maximum of 32 characters used.

- OPT = 8 - User response status:

  IVEC(1) = First mouse button action - 0 = no button press

  IVEC(2) = X pixel location for action

  IVEC(3) = Y pixel location for action

  IVEC(4) = Second mouse button action

  IVEC(5) = X pixel location for action

  IVEC(6) = Y pixel location for action

  STRING = Text answer

  The data in IVEC is valid for those operations that require snapping a line/circle or a 'rubber-band' box.

- OPT = 9 - Light Status:

  IVEC(1) = Light number - input (1 to the number of lights)

  RVEC(1-3) = Light color (r,g,b) - values between 0.0 and 1.0

  RVEC(4-6) = Light normal

- OPT = 10 - Current domain/static surface state:

  IVEC(1) = Surface index

  IVEC(2) = Current drawing state - see the Appendix

  IVEC(3) = Surface type :

  1 - Domain surface

  2 - Planar cut
  RVEC is filled with the 9 words (for the corners) at the time of the cut. This is like OPT = 2.

  4 - Programmer-defined cut
  RVEC(1) contains the ZPrime value.

  7 - Iso-surface
  RVEC(1) contains the iso-surface value.

  IVEC(4) = Maximum number of surfaces

  STRING = Surface title - maximum of 20 characters used.

- OPT = 11 - Current streamline state:

  IVEC(1) = Current streamline group number - 0 no streamlines active

  IVEC(2) = Current drawing state - See Appendix

  IVEC(3) = Stream group number

  IVEC(4) = Maximum number of streamline groups

  IVEC(5) = Global Surface number - 0 indicates a volume streamline

  RVEC(1) = Ribbon rotation in degrees

  STRING = Streamline title - maximum of 20 characters used.

- OPT = 12 - NOT Used

- OPT = 13 - Current probe state:

  IVEC(1) = 1D plot type :

  0 - no plot

  1 - StreamLine Probe

  2 - Strip Chart

  3 - Line Probe

  4 - Edge Plot

  5 - Surface Layer Scan

  6 - Programmer-defined probe

  IVEC(2) = Point Probe status - 0 (off) or 1 (on)

  IVEC(3) = Reference line status - 0 (off) or 1 (on)

  RVEC(1) = X Minimum for plot

  RVEC(2) = X Maximum for plot

  RVEC(3) = Y Minimum for plot

  RVEC(4) = Y Maximum for plot

- OPT = 14 - Global drawing state:

  IVEC(1) = Edge Outline status - 0 (off), 1 (on - default), 2 (no internal boundary edges)

  RVEC(1) = Object Radius - The radius of the sphere that encloses the object(s).

  RVEC(2) = Line Adjustment - The amount that lines are displaced (towards the viewer) so that the surfaces do not obscure them.

  RVEC(3) = Screen Z Clip value - data added to screen Z to move object(s) into the front or back clipping plane.

- OPT = 15 - Dynamic surface state:

  IVEC(1) = Current drawing state - see the Appendix

  IVEC(2) = Number of tufts - the number of tufts on a side

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  RVEC(2) = Vector scale for surface functions - the scale factor used with tufts and arrows for special surface functions

- OPT = 16 - StreamLine/Bubble drawing state:

  IVEC(1) = StreamLine representation (when seeded):

  0 - StreamLine

  1 - Ribbon

  2 - StreamTube

  3 - Tubes with twist

  IVEC(2) = Bubble rendering :

  0 - off

  1 - Rendered in foreground color

  2 - Rendered using scalar color

  3 - Rendered with surface function (for Surface particles only)

  IVEC(3) = Current Surface/Volume setting :

  0 - Volume Scalar/Volume Vector

  1 - Surface Scalar/Volume Vector

  2 - Volume Scalar/Surface Vector

  3 - Surface Scalar/Surface Vector

- OPT = 17 - Key and Colormap state:

  IVEC(1) = Key status :

  0 - Scalar

  1 - Special Surface Scalar

  2 - Time

  IVEC(2) = Number of entries in colormap

  IVEC(3) = Number of entries in background colormap - between 1 and 4

- OPT = 18 - Get colormap entry in current Key colormap:

  IVEC(1) = index - set on input:

  -2 - Foreground: used for outlines and pseudo-cursors, usually white

  -1 - Midground: used for 1D cursor and solid surfaces, usually grey

  0 - Background: used for Dial and 1D window background, usually black

  1-on - Colormap/background colormap entry. The background colormap indices follow the colormap entries.

  RVEC(1-3) = The color (r,g,b) - values between 0.0 and 1.0

- OPT = 19 - Contouring state:

  IVEC(1) = Dynamic contour status - 0 = off, 1 = on

  IVEC(2) = Number of contour levels

5.5 pV_SetState

pV_SETSTATE(OPT,IVEC,RVEC,STRING)

Sets the internal **pV3** server state. This routine can only be called from **pVEvents**.

OPT:i: I	Option set to specify what internal state data to change
IVEC:i: I()	Integer data set based on OPT (length also determined by OPT)
RVEC:i: R()	Real data set based on OPT (length also determined by OPT)
STRING:i: C	Character data set based on OPT

- OPT = -1 - Force a 3D and 2D window update. This tells **pV3** to do a complete redraw including the static objects.
- OPT = 0 - NOT Used
- OPT = 1 - Set the current transformation matrix for the 3D view. 12 words of RVEC are used.
- OPT = 2 - Set the current planar cut coordinates. 9 words of RVEC are used for 3 of the 4 corners of the cut plane in 3 space.
- OPT = 3 - Set the current scalar tool values:

  RVEC(1) = ZPRIME

  RVEC(2) = XPC

  RVEC(3) = YPC

  RVEC(4) = HALFW

  RVEC(5) = 2D Rotation Angle in degrees

  NOTE:
  If you don't wish to change all of the above call pV_GetState(3,IVEC,RVEC,STRING) first!

- OPT = 4 - Set Lighting and Mirroring state:

  IVEC(1) = Mirror status :

  0 - No Mirroring

  1 - Mirror about X = 0.0

  2 - Mirror about Y = 0.0

  3 - Mirror about Z = 0.0

  IVEC(2) = Number of Lights maximum is 8

  RVEC(1-3) = Ambient light color (r,g,b) - values between 0.0 and 1.0

- OPT = 5 - Scalar state:

  IVEC(1) = Binding index - allows the changing of the current scalar function

  RVEC(1) = Minimum for display - fmin shown in the key window

  RVEC(2) = Maximum for display - fmax shown in the key window

  STRING = Scalar title - maximum of 32 characters used (blank string indicates no change)

- OPT = 6 - Vector state:

  IVEC(1) = Binding index allows the setting of the current vector field

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  STRING = Vector title - maximum of 32 characters used (blank string indicates no change)

- OPT = 7 - Threshold state:

  IVEC(1) = Binding index - negative value indicates the appropriate scalar index

  IVEC(2) = Vector Cloud status - 0 = off, 1 = on

  IVEC(3) = Dynamic Threshold status - 0 = off, 1 = on

  RVEC(1) = Minimum for display - tmin shown in the key window

  RVEC(2) = Maximum for display - tmax shown in the key window

  STRING = Threshold title - maximum of 32 characters used (blank string indicates no change). You cannot change the title of a scalar threshold with this function.

- OPT = 8 - Enter user response:

  IVEC(1) = First mouse button action - 0 = no button press

  IVEC(2) = X pixel location for action

  IVEC(3) = Y pixel location for action

  IVEC(4) = Second mouse button action

  IVEC(5) = X pixel location for action

  IVEC(6) = Y pixel location for action

  STRING = Text answer

  The data in IVEC is required for those operations that snap a line/circle or a 'rubber-band' box.

- OPT = 9 - Set Lights:

  IVEC(1) = Light number - 1 to the number of lights

  RVEC(1-3) = Light color (r,g,b) - values between 0.0 and 1.0

  RVEC(4-6) = Light normal

- OPT = 10 - Domain/static surface state:

  IVEC(1) = Surface index (-) value indicates that the numbered surface should be deleted without effecting what is the current surface

  IVEC(2) = Current drawing state - value < 0 doesn't change the state. See the Appendix.

  STRING = Surface title - maximum of 20 characters used (blank string indicates no change). Only for Domain surfaces.

- OPT = 11 - Streamline state:

  IVEC(1) = Streamline group index (-) value indicates that the numbered streamline group should be deleted without effecting what is the current streamline index

  IVEC(2) = Current drawing state - value < 0 doesn't change the state. See the Appendix.

  RVEC(1) = Ribbon rotation in degrees

- OPT = 12 - NOT used

- OPT = 13 - Set probe state:

  IVEC(1) = 1D plot type :

  0 - turn off plot

  6 - start Programmer-defined probe

  Note: use the proper 'events' for initiating the other probes.

- OPT = 14 - Set global drawing state:

  IVEC(1) = Edge Outline - 0 (off), 1 (on), 2 (no internal boundary edges)

  RVEC(1) = Object Radius - The radius of the sphere that encloses the object(s).

  RVEC(2) = Line Adjustment - The amount that lines are displaced (towards the viewer) so that the surfaces do not obscure them.

  RVEC(3) = Screen Z Clip value - data added to screen Z to move object(s) into the front or back clipping plane.

- OPT = 15 - Set dynamic surface state:

  IVEC(1) = Current drawing state - see the Appendix

  IVEC(2) = Number of tufts - the number of tufts on a side

  RVEC(1) = Vector scale - the scale factor for tufts and arrows

  RVEC(2) = Vector scale for surface functions - the scale factor used with tufts and arrows for special surface functions

- OPT = 16 - Set StreamLine/Bubble drawing state:

  IVEC(1) = StreamLine seed representation :

  0 - StreamLine

  1 - Ribbon

  2 - StreamTube

  3 - Tubes with twist

  IVEC(2) = Bubble rendering :

  0 - off

  1 - Rendered in foreground color

  2 - Rendered using scalar color

  3 - Rendered with surface function (for Surface particles only)

  IVEC(3) = Current Surface/Volume setting :

  0 - Volume Scalar/Volume Vector

  1 - Surface Scalar/Volume Vector

  2 - Volume Scalar/Surface Vector

  3 - Surface Scalar/Surface Vector

- OPT = 17 - Set Key and Colormap state:

  IVEC(1) = Key status :

  0 - Scalar

  1 - Special Surface Scalar - if any Surface Scalars

  2 - Time

  IVEC(2) = Number of entries in colormap

  IVEC(3) = Number of entries in background colormap - between 1 and 4

  This forces a Key Window redraw. When changing the colors, first set the Key state (Scalar, Surface Scalar or Time) if not already correct, then set the new colormap entries (calls with OPT = 18) and finally (again) make this call.

- OPT = 18 - Set colormap entry in current Key colormap:

  IVEC(1) = index :

  -2 - Foreground: used for outlines and pseudo-cursors - Key independent

  -1 - Midground: used for 1D cursor and solid surfaces - Key independent

  0 - Background: used for Dial and 1D window background - Key independent

  1-on - Colormap/background colormap entry. The background colormap indices follow the colormap entries.

  RVEC(1-3) = The color (r,g,b) - values between 0.0 and 1.0

- OPT = 19 - Set Contouring state:

  IVEC(1) = Dynamic contour status - 0 = off, 1 = on

  IVEC(2) = Number of contour levels - must be greater than 1

5.6 pVSafe

PVSAFE()

This programmer-supplied routine is called when the graphics thread of the server is stalled. This is the time where calls can be made that require neither thread to be active. The buffers have not been swapped, so that queries of extracts will look at the last state.

No Arguments

5.7 pV_Register

PV_REGISTER(INDEX,NAME,SUBTYPE,SUBSIZE,SUBOPT,SUBLOC, ROUTINE,EXNUM)

Registers a programmer-defined extract with the **pV3** server. This routine should only be called when the threads are sync'ed, therefore the only valid place to execute this routine is within **pVSafe**.

INDEX:i: I — The extract index. This number must be greater that 100 and defines an extract.

NAME:o: C*20 — Extract name.

SUBTYPE:i: I(12) — The subextract types. Each extract is composed of up to 12 subextracts for each client. This vector defines whether the subextract is an integer (0) or a real (1).

SUBSIZE:i: I(12) — The subextract size per length. For example, if the subextract is for the 3D coordinates (X,Y,Z) that support the extract, the size would be 3.

SUBOPT:i: I(12) — The level of unsteadyness that requires the data at every time-step. Valid entries are 0 to 2. The following table specifies what action is taken with an existing subextract:

Client's OPT − >	0	1	2	3
SUBOPT = 0	leave	refill	refill	refill
SUBOPT = 1	leave	leave	refill	refill
SUBOPT = 2	leave	leave	leave	refill

SUBLOC:i: I(12) — The subextract's locality. If this subextract comes from the clients then the value is -1. If this subextract is local to the server and it's length is set by another subextract, then SUBLOC must contain the index (0 biased) to that extract. SUBOPTs for local subextracts must match that of the keyed subextract.

ROUTINE:i

This is the programmer-supplied routine that will be called to render the data associated with this extract. FORTRAN programmers must declare the subroutine 'EXTERNAL' in the calling module.

EXNUM:o: I

This is a status return. If the value is zero or greater, that indicates success. The value is the number used for multiple allocations of extracts with the same INDEX (that use the same rendering routine). If the number is negative it is an indication of an error:

-1 - Invalid INDEX number

-2 - Invalid SUBTYPE in one of the entries

-3 - Invalid SUBSIZE in one of the entries

-4 - Invalid SUBOPT in one of the entries

-5 - Invalid SUBLOC in one of the entries

-6 - SUBTYPE mismatch for subsequent calls using INDEX

-7 - SUBSIZE mismatch for subsequent calls using INDEX

-8 - SUBOPT mismatch for subsequent calls using INDEX

-9 - SUBLOC mismatch for subsequent calls using INDEX

-10 - ROUTINE mismatch for subsequent calls using INDEX

-11 - Allocation error

-12 - Routine not called from **pVSafe**

-13 - SUBOPTs mismatch for local subextract

ROUTINE must have the form:

ROUTINE(ISTAT,EXNUM,PLOTMASK,TID,CNT

LEN0,SUB0,....LEN11,SUB11)

The programmer-supplied render routine for a registered extract. This routine is called for each active **pV3** client. And, this routine will be called during the 3 stages of the 3D window's rendering. The drawing should be done by calls to **pV_Object3D**. See **pVDraw3D**.

ISTAT:i: I	Rendering stage: **0** - rendering for static object(s) **1** - rendering for dynamic object(s) **2** - rendering for translucent object(s)
EXNUM:i: I	The extract number.
PLOTMASK:i: I	The value that describes the drawing state of the extract.
TID:i: I	PVM tid for this set of subextracts.
CNT:i: I	The client count (always starts at zero).
LENn:i/o: I	The length of the subextracts. If the value is zero for a non-local extract than there is no data for this subextract. If the value is zero for a local subextract (and the keyed length is non-zero) then this routine can fill the subextract and set the length to the size of the keyed subextract. If the value is -1 for a local subextract, then there was some problem in allocation. Do NOT fill the subextract in this case!
SUBn:i/o	The subextracts. The size and type are determined by the call to register the extract.

5.8 pVSetExtract

PVSETEXTRACT(INDEX,EXNUM,PLOTMASK,REQMASK,IVAL,RVEC)

This routine gets called for each registered extract. The programmer must supply data associated with the status for the next set of requests from the clients.

INDEX:i: I	The extract index. This number must be greater that 100 and defines an extract.
EXNUM:i: I	The extract number associated with INDEX.
PLOTMASK:o: I	Programmer defined integer that specifies the plot attributes for the extract. This is passed to the render routine.
REQMASK:o: I	The request mask. Each bit specifies which subextracts are required to statisfy the plotting attributes. For example, 5 requests subextract 0 and subextract 2. If the most-significant bit is set all subextracts are requested, even if based on SUBOPT, the data exists (i.e. some state has changed).
IVAL:o: I	An integer sent to the clients associated with this extract.
RVEC:o: R(9)	A real vector of data sent to the clients with the request for this extract.

5.9 pV_GetExtract

PV_GETEXTRACT(EX,TYPE,NUM,IVEC,RVEC,NAME,NEXTEX)

Returns the internal **pV3** extract structure info. This routine can be called from any server programmer-supplied code. The extracts form a linked list. This routine allows the scanning of all active extracts by continually calling the routine until the desired extract is found.

EX:i/o: I	Extract pointer (integer in FORTRAN). On input, this is the desired extract. The special case of the first extract is indicated by a 0 or NULL and is updated with the actual extract pointer.
TYPE:o: I	The extract type or index.
NUM:o: I	The extract number. This is a unique number for the type.
IVEC:o: I(10)	Integer data set based on TYPE (length also determined by TYPE)
RVEC:o: R(12)	Real data set based on TYPE (length also determined by TYPE)
NAME:o: C*20	Extract name.
NEXTEX:o: I	Extract pointer (integer in FORTRAN) to the next extract. 0 or NULL indicates that this is the last extract. This can be used in the next call to **pV_GetExtract** (argument EX) to continue scanning the list.

- Planar Cut - TYPE = 2

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  RVEC(1-9) = Cut corners - Three of the 4 corners that denote the plane

  RVEC(10-12) = Plane normal

- Geometric Cut - TYPE = 4

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

IVEC(4) = Threshold index

IVEC(5) = Cut index

RVEC(1) = Z prime

- Domain Surface - TYPE = 5

IVEC(1) = Plot Mask

IVEC(2) = Scalar field index

IVEC(3) = Vector field index

IVEC(4) = Threshold index

IVEC(5) = Mapping flag

IVEC(6) = Special surface scalar index

IVEC(7) = Special surface vector index

- Iso-Surface- TYPE = 7

IVEC(1) = Plot Mask

IVEC(2) = Scalar field index

IVEC(3) = Vector field index

IVEC(4) = Threshold index

RVEC(1) = Z prime

- StreamLine - TYPE = 18

IVEC(1) = Plot Mask

IVEC(2) = Scalar field index

IVEC(3) = Vector field index

IVEC(4) = Threshold index

IVEC(5) = StreamLine Group number

IVEC(6) = PVM tid for client with seed location

IVEC(7) = Cell index in client to start StreamLine

IVEC(8) = Minimum StreamLine number for group

IVEC(9) = Maximum StreamLine number for group

IVEC(10) = Number of StreamLine segments

RVEC(1-3) = Seed location (XYZ)

- Particles - TYPE = 19

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

- Vector Cloud- TYPE = 20

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  RVEC(1) = Threshold minimum

  RVEC(2) = Threshold maximum

- Programmer-defined - TYPE > 100

  IVEC(1) = Plot Mask

  IVEC(2) = Scalar field index

  IVEC(3) = Vector field index

  IVEC(4) = Threshold index

  IVEC(5) = IVAL

  RVEC(1-9) - Real values assoctated with the extract

5.10 pV_GetSub

PV_GETSUB(EX,SUBEX,NUMCS,PTR,LEN,TID)

Returns the internal **pV3** sub-extracts. This routine can be called from any server programmer-supplied code.

EX:i: I	Extract pointer (integer in FORTRAN) as returned by **pV_GetExtract**.
SUBEX:i: I	Sub-extract number (0-11 based on TYPE).
NUMCS:i: I	Client index or StreamLine segment number (0 biased).
PTR:o: I	Integer (in FORTRAN) used as a pointer to the structure. In C this returns the pointer to the block of memory. A 0 (zero) indicates that the memory block is not allocated.
LEN:o: I	Length of structure. A 0 (zero) indicates that the structure is not currently filled.
TID:o: I	**PVM** tid for the client that produced the segment (StreamLines Only).

The following routines are used for drawing programmer defined objects in the 3D window:

5.11 pVDraw3D

PVDRAW3D(ISTAT)

This programmer supplied subroutine is called by the **pV3** server at different times during the rendering phase. It is the responsibility of this routine to specify the object(s) to be plotted based on the rendering status. The object(s) are defined by calls to **pV_Object3D**. Calls to **pV_GetState**, **pV_GetExtract**, **pV_GetSub** and **pV_GetPointer** are valid but invocations of **pV_SetState** should be avoided.

ISTAT:i: I — Rendering stage:

0 - rendering for static object(s)

1 - rendering for dynamic object(s)

2 - rendering for translucent object(s)

The options listed above reflect the manner that the server draws the 3D scene. If the case is steady-state (or unsteady and *pause* is in effect) and the viewing transformation matrix is NOT changing, static objects only get rendered once (until the transformation matrix is changed again). *Snap-Shot Rendering* is used to improve drawing performance so the result of the rendering of the static objects is saved in a secondary *pixmap* and *ZBuffer*. These items are used to start the scene generation for subsequent animation frames.

Dynamic objects are those that will change or move from frame to frame even if the underlying (static) object(s) are not moving. An example of this is any dynamic surface (cut plane or iso-surface) or StreamLine animation using 'blobs' to display the integration pseudo-time.

Translucent objects, even if static, must be rendered last in order to get the scene to look correct. The ZBuffer must be updated with the translucent surfaces last.

If the application is unsteady (and not *paused*), the static objects as well as the dynamic and translucent objects are rendered each time step.

5.12 pV_Object3D

PV_OBJECT3D(ITYPE,ICOLOR,XYZP,RADII,COL,NP)

This routine must only be used from within **pVDraw3D** or the registered routine for programmer defined extracts. **pV_Object3D** defines additional plotting for the 3D window. Multiple objects may be drawn by multiple calls to this routine from within **pVDraw3D** or the extract rendering routine.

ITYPE:i: I — Type of plotting primitive:

- **0** - disjoint quadrilaterals - NP must be 4 times the number of quads
- **1** - disjoint triangles - NP must be 3 times the number of tris
- **2** - polytriangle strip - NP must be the length of the strip
- **3** - disjoint line segments - NP must be 2 times the number of segments
- **4** - polyline - NP must be the length of the line
- **5** - spheroids - NP must be the number of shpere-like objects to plot

ICOLOR:i: I — The method used for coloring the object:

- **-1** - draw the object white
- **0** - draw the object in grey
- **1** - color the object from the data in COL

XYZP:i: R(3,NP) — The (x, y, z)-coordinates of the points that support the primitive.

RADII:i: R(NP) — The spheroid radius (in user coordinates). For ITYPE = 5 only. Points are plotted (and are MUCH faster) if RADII(1) is 0.0

COL:i: R(NP) — Color scaling for the data points (used only for ICOLOR = 1). The values must be in the range 0.0 to 1.0 and the scalar field color map is applied to that range to compute the color for the point.

NP:i: I — Object length (the number of points)

The following routines are used for drawing programmer defined objects in the 2D window:

5.13 pVDraw2D

PVDRAW2D(ISTAT)

This programmer supplied subroutine is called by the server at different times during the 2D window rendering phase. It is the responsibility of this routine to specify the object(s) to be plotted based on the rendering status. The object(s) are defined by calls to **pV_Object2D**. Calls to **pV_GetState**, **pV_GetExtract**, **pV_GetSub** and **pV_GetPointer** are valid but invocations of **pV_SetState** should be avoided.

ISTAT:i: I	Rendering stage:
	0 - rendering for static object(s)
	1 - rendering for dynamic object(s)

Like rendering for the 3D window, the 2D window scene generation has multiple phases. Because translucency is not supported (nor does it make much sense) in the 2D window and there is no ZBuffer, the translucent phase is not implemented.

The only 2D dynamic object that **pV3** currently uses is the mapped cross-hair cursor.

For a program controlled 2D window, the programmer must perform a forced update (**pV_SetState** OPT = -1) for a static draw or should render everything during the dynamic phase.

5.14 pV_Object2D

PV_OBJECT2D(ITYPE,ICOLOR,XYW,COL,NP)

This routine must only be used from within **pVDraw2D**. It defines additional plotting for the 2D window. Multiple objects may be drawn by multiple calls to this routine from within **pVDraw2D**.

ITYPE:i: I

Type of plotting primitive:

0 - disjoint quadrilaterals - NP must be 4 times the number of quads

1 - disjoint triangles - NP must be 3 times the number of tris

2 - polytriangle strip - NP must be the length of the strip

3 - disjoint line segments - NP must be 2 times the number of segments

4 - polyline - NP must be the length of the line

5 - points - NP must be the number of dots

ICOLOR:i: I

The method used for coloring the object:

-1 - draw the object white

0 - draw the object in grey

1 - color the object from the data in COL

XYW:i: R(2,NP)

X and Y window values for the points that make up the drawing primitive. The plotting in the 2D window for X ranges from -1.0 (left side) to 1.0 (the right side of the window). The Y range is from -1.0 (bottom) to 1.0 (the top of the window).

COL:i: R(NP)

Color scaling for the data points. Used only for ICOLOR = 1. The values must be in the range 0.0 to 1.0 and the scalar field color map is applied to that range to compute the color for the point.

NP:i: I

Object length in points

The following routines are used for Programmer-defined probes:

5.15 pVProbe

PVPROBE(FLAG,XAXIS,YAXIS)

This programmer supplied subroutine is called by the server at the end of the rendering phase if the programmer-defined probe is activated. It is the responsibility of this routine to specify the line(s) to be plotted and return the axis annotation. The line or lines are defined by calls to **pV_LIine**. Calls to **pV_GetState**, **pV_GetExtract**, **pV_GetSub** and **pV_GetPointer** are valid but invocations of **pV_SetState** should be avoided.

FLAG:i: L	logical flag
	.TRUE. first call after probe has been activated
	.FALSE. subsequent calls
XAXIS:o: C*32	X-Axis label for the plot
YAXIS:o: C*32	Y-Axis label for the plot

Note: Currently the only way to initiate the programmer-defined probe is by a call to **pV_SetState** with OPT = 13 from **pVEvents**.

5.16 pV_Line

PV_LINE(X,Y,NP)

This routine must only be used from within **pVProbe**. It defines a line to be plotted in the 1D window. Multiple lines may be drawn by multiple calls to this routine from within **pVProbe**. Auto-scaling is performed on the first line for the first call to **pVProbe** after the probe has been started. Afterwards user or programmed events can control the scaling.

X:i: R(NP)	X values for the line
Y:i: R(NP)	Y values for the line
NP:i: I	Line length

5.17 pVInit

PVINIT(NPIX1,NSX1,NSY1,NPIX2,NSX2,NSY2,NPIX3,NSX3,NSY3)

This routine is called at server initialization before the windows are open. It allows the customized selection of window sizes and placement as well as allowing any advanced programming that should only be performed at initialization (such as memory allocation).

NPIX1:i/o: I	The size of the 1D window. Enters with **pV3**s default size.
NSX1:i/o: I	The suggested X pixel location (in the root window) for the start position of the 1D window. Enters with **pV3**s default location.
NSY1:i/o: I	The suggested Y pixel location (in the root window) for the start position of the 1D window. Enters with **pV3**s default location.
NPIX2:i/o: I	The size of the 2D window. Enters with **pV3**s default size.
NSX2:i/o: I	The suggested X pixel location (in the root window) for the start position of the 2D window. Enters with **pV3**s default location.
NSY2:i/o: I	The suggested Y pixel location (in the root window) for the start position of the 2D window. Enters with **pV3**s default location.
NPIX3:i/o: I	The size of the 3D window. Enters with **pV3**s default size.
NSX3:i/o: I	The suggested X pixel location (in the root window) for the start position of the 3D window. Enters with **pV3**s default location.
NSY3:i/o: I	The suggested Y pixel location (in the root window) for the start position of the 3D window. Enters with **pV3**s default location.

6 Client Side Programming

6.1 The Use of Pointers in FORTRAN for the Clients

Some of the data that gets returned from **pV_GetStruc** is in the form of pointers to blocks of memory. The pointers are treated as INTEGER*4 variables on all machines except DEC ALPHAs and KSRs where they are INTEGER*8. Traditionally, this information cannot be used by the FORTRAN programmer. But, a mechanism exists on all major FORTRANs (SGI, DEC, IBM, HP and SUN) to allow the pointer to be passed to a SUBROUTINE or FUNCTION and then have the memory treated as a normally declared vector or array. This is done by the VAX extension '%val(pointer)' used in the CALL or function invokation. When the pointer is passed to the sub-program by 'value', it is equivalent to passing a variable by 'reference' (the FORTRAN method). That is, in both cases, the address of the memory of interest is placed in the stack!

Using this mechanism (or the POINTER statement supported by some f77s), sophisticated **pV3** client enhancements may be performed using FORTRAN. To complete this picture, two additional entries to the **pV3** client library have been added to allow the FORTRAN programmer to allocate and free up blocks of memory.

6.2 MALLOCPV

PTR = MALLOCPV(NBYTES)

This function is equivalent to the C routine 'malloc'. It allocates a block of memory and returns the pointer to the block.

NBYTES:i: I	The number of bytes to allocate.
PTR:o: I	The address of the block (0 is an error indicator).

6.3 FREEPV

FREEPV(PTR)

This function is equivalent to the C routine 'free'. It deallocates a block of memory. NOTE: Only free up blocks of memory that YOU allocate!

PTR:i: I	The address of the block.

6.4 Client Structures

In general, this data is the information returned from the programmer-supplied routines, though the connectivity tables are internally generated (with the proper IOPT). The structures are exposed so that multiple copies of the data need not be kept.

6.4.1 Node based

The following data is based on the 3D node space:

XYZ: R(3,NNODE)	(x, y, z)-coordinates of grid nodes.
S: R(NNODE)	Scalar function values.
T: R(NNODE)	Threshold function values.
V: R(3,NNODE)	Vector function values (Vx, Vy, Vz).
Z: R(NNODE)	Current z' values for planar or programmed cuts.
IBLANK: I(*)	IBLANK values for structured blocks. The length is the number of nodes in the structured blocks. For a case that is only structured blocks the length is NNODE. If the pointer to this array is 0, then no IBLANKing is active.
TBCON: I(*)	PVM tid number for block connectivity outside current domain.

6.4.2 Cell based

The following data is based on the 3D cell space:

CEL1: I(4,KCEL1)	Node pointers for tetrahedral cells.
CEL2: I(5,KCEL2)	Node pointers for pyramid cells.
CEL3: I(6,KCEL3)	Node pointers for prism cells.
CEL4: I(8,KCEL4)	Node pointers for hexahedral cells.
NPTET: I(8,KNPTET)	Poly-Tetrahedra strip header: **NPTET(1,n)** = the pointer to the end of the strip n, i.e. it points to the last entry in PTET for the poly-tetrahedral strip **NPTET(2,n)** = the first node in the poly-tetrahedra **NPTET(3,n)** = the second node in the poly-tetrahedra

NPTET(4,n) = the third node in the poly-tetrahedra

NPTET(5,n) = the fourth node in the poly-tetrahedra

NPTET(6,n) = the last node in the poly-tetrahedra

NPTET(7,n) = first connection (face 1 of the first teta)

NPTET(8,n) = last connection (face 2 of the last tetra)

PTET: I(KPTET) — The rest of each poly-tetrahedra, 1 node per cell. The first and last node numbers in PTET are replaced by a flag to indicate the start and end of the strip. The flag is the negative of the strip number so that if you are in a strip, you can find which strip it is.

BLOCKS: I(6,KNBLCK) — Structured block definitions:

BLOCKS(1,m) = NI

BLOCKS(2,m) = NJ

BLOCKS(3,m) = NK

BLOCKS(4,m) = cell number that terminates the block

BLOCKS(5,m) = node number that terminates the block

BLOCKS(6,m) = index into CBLOCK that starts the connectivity table for the block.

6.4.3 Domain Surfaces

The following data is based on the surface space:

NSURF: I(3;KNSURF) — NSURF(1,n) is the pointer to the end of domain surface group n, i.e. it points to the last entry in both SCON and SCEL for that group.
NSURF(2,n) is the startup drawing state.
NSURF(3,n) is the global surface number.

SCON: I(KSURF) — connecting cell number for internal boundaries.

SCEL: I(4,KSURF) — node numbers for surface faces. For quadrilateral faces SCEL must be ordered clockwise or counter-clockwise; for triangular faces, SCEL(4,n) must be set to zero.

TSURF: C*20(KNSURF) — titles for domain surfaces.

6.4.4 Domain Surface Edges

The following data is based on the surface edge space:

NSED: I(2,KNSED)	NSED(1,n) is the pointer to the end of the surface edge n, i.e. it points to the last entry in SED for that group. NSED(2,n) is the surface group number associated with this edge. NOTE: a domain surface may have as few a zero edges (the surface closes totally upon itself) and can have more than one edge!
SED: I(*)	pointers to nodes that make the surface edge. Each node connects to the last (except for the first entry) in a polyline structure.

6.4.5 Connectivity

The connectivity tables are used by the particle path algorithms and other tools that require neighborhood data. Any connection (cell number) that is negative indicates that there is no neighboring cell and the value is the index into SURF, SCON and SCEL for the surface face. If the pointers to all of these structures are 0, then this information has been deallocated.

CCEL1: I(4,KCEL1)	cell indices for the neighbors touching each of the four faces of the tetrahedron.
CCEL2: I(5,KCEL2)	cell indices for the neighbors touching each of the five faces of the pyramid.
CCEL3: I(5,KCEL3)	cell indices for the neighbors touching each of the five faces of the prism.
CCEL4: I(6,KCEL4)	cell indices for the neighbors touching each of the six faces of the hexahedon.
CPTET: I(2,KPTET)	The neighbor for face 3 and face 4 for each tetrahedra in the strip. Face 1 and face 2 are implicit (the last and next cells) except for the beginning and end of the strip where the connectivity can be found in the header NPTET.
CBLOCK: I(*)	A vector that contains the connectivity data (the cell index) for the husk of the block. The start index for a particular block is in BLOCKS(6,m).

face 1 - $k = 1$

first $(NI-1)*(NJ-1)$ entries indexed in a nested set of loops with j as the outer and i as the inner.

face 2 - $i = NI-1$

next $(NJ-1)*(NK-1)$ entries indexed in a nested set of loops with k as the outer and j as the inner.

face 3 - $j = NJ-1$

next $(NI-1)*(NK-1)$ entries indexed in a nested set of loops with k as the outer and i as the inner.

face 4 - $i = 1$

next $(NJ-1)*(NK-1)$ entries indexed in a nested set of loops with k as the outer and j as the inner.

face 5 - $k = NK-1$

next $(NI-1)*(NJ-1)$ entries indexed in a nested set of loops with j as the outer and i as the inner.

face 6 - $j = 1$

next $(NI-1)*(NK-1)$ entries indexed in a nested set of loops with k as the outer and i as the inner.

SURF: I(KSURF) — cell number that contains the face. Zero indicates a face without a cell.

CCEL: I(4,KSURF) — surface indices for the neighbors touching each of the three or four faces of a surface face.

6.5 pV_GetStruc

PV_GETSTRUC(OPT,PTR,LEN)

Returns the internal **pV3** client side structure. This **pV3** routine can be called from any programmer-supplied code.

OPT:i: I	Option set to specify what data to get.
PTR:o: I	Integer (in FORTRAN) used as a pointer to the structure. In C this returns the pointer to the block of memory. A 0 (zero) indicates that the memory block is not allocated. FORTRAN NOTE: this is an INTEGER*8 value on machines with 64 bit pointers such as DEC ALPHAs.
LEN:o: I	Length of structure.

- Node Structures:

OPT	PTR - Pointer to	length of structure
301	XYZ	NNODE
302	S	NNODE
303	T	NNODE
304	V	NNODE
305	Z	NNODE
306	IBLANK	number of nodes in blocks
307	TBCON	number of nodes in blocks

- Cell Structures:

OPT	PTR - Pointer to	length of structure
311	CEL1	KCEL1
312	CEL2	KCEL2
313	CEL3	KCEL3
314	CEL4	KCEL4
315	NPTET	KNPTET
316	PTET	KPTET
317	BLOCKS	KNBLCK

- Domain Surface Structures:

OPT	PTR - Pointer to	length of structure
321	NSURF	KNSURF
322	TSURF	KNSURF
323	SURF	KSURF
324	SCEL	KSURF
327	SCON	KSURF

- Domain Surface Edge Structures:

OPT	PTR - Pointer to	length of structure
325	NSED	KNSED
326	SED	Number of ployline edge points - NSED(1,KNSED)

- Connectivity Structures:

OPT	PTR - Pointer to	length of structure
331	CCEL1	KCEL1
332	CCEL2	KCEL2
333	CCEL3	KCEL3
334	CCEL4	KCEL4
336	CPTET	KPTET
337	CBLOCKS	len of husks
338	CCEL	KSURF

6.6 pVExtract

PVEXTRACT(INDEX,EXNUM,REQMASK,IVAL,RVEC)

It is the responsibility of this routine to calculate and send back to the server any requested subextracts. The sending of messages to the server must be done by calls to either **pV_SendXi** or **pV_SendXr** based on the type of the data (specified during server registration - SUBTYPE).

INDEX:i: I	The extract index. Always greater than 100.
EXNUM:i: I	The extract number associated with INDEX.
REQMASK:i: I	The request mask. Each bit specifies which subextracts are required to statisfy the plotting attributes. For example, 5 requests subextract 0 and subextract 2.
IVAL:i: I	An integer associated with this extract.
RVEC:i: R(9)	A real vector of data associated with this extract request.

6.7 pV_SendXi

PV_SENDXI(INDEX,EXNUM,SUBINDEX,SUBSIZE,LEN,IVEC)

Sends an integer subextract back to the server.

INDEX:i: I	The extract index. Always greater than 100.
EXNUM:i: I	The extract number associated with INDEX.
SUBINDEX:i: I	The subextract index (0-11).
SUBSIZE:i: I	The subextract size per length. For example, if the subextract is to define a set of disjoint tris that support the extract, the size would be 3. This must match the size specified in the server code during registration.
LEN:i: I	Length of structure.
IVEC:i	The array of data to be passed to the server. The actual number of integers transferred is LEN * SUBSIZE.

6.8 pV_SendXr

PV_SENDXR(INDEX,EXNUM,SUBINDEX,SUBSIZE,LEN,RVEC)

Sends a real subextract back to the server.

INDEX:i: I	The extract index. Always greater than 100.
EXNUM:i: I	The extract number associated with INDEX.
SUBINDEX:i: I	The subextract index (0-11).
SUBSIZE:i: I	The subextract size per length. For example, if the subextract is for the 3D coordinates (X,Y,Z) that support the extract, the size would be 3. This must match the size specified in the server code during registration.
LEN:i: I	Length of structure.
RVEC:i	The array of data to be passed to the server. The actual number of floating point values transferred is LEN * SUBSIZE.

A Plotting Masks

The following are additive (or-able) so that the proper attibutes can be specified.

A.1 Cut Surfaces

This mask controls the **pV3** scalar tools (planar cuts, programmed cut surfaces, iso-surfaces and domain surfaces) attributes.

1 - Render - Surface rendering on

2 - Grid - Mesh display on

4 - Grey - Surface colored with grey

8 - Threshold - Surface is thresholded according to the threshold function and limits

16 - Contour - Contour lines are plotted on the surface

32 - Translucent - Plot surface using the translucent attribute

64 - Arrows - Arrow drawing on

128 - Tufts - Grid of tufts on (dynamic only)

256 - Mapping - A 2D mapping exists for this surface (domain only)

512 - Probing - 2D probing is active

1024 - Outline - Outline drawing is requested (with the mask equal to only this flag)

A.2 StreamLines

This mask controls the **pV3** streamline plotting attributes and therefore the requested sub-extracts.

1 - Render - StreamLine rendering on

2 - Tube - Tube rendering on

4 - Grey - StreamLine drawn with default color

8 - Threshold - StreamLine is thresholded according to the threshold function and limits (not currently implemented)

16 - Back - StreamLine is backward going (can not be active with 32)

32 - Fore - StreamLine goes down stream (can not be active with 16)

64 - Ribbon - Ribbon rendering on (with 2 makes tubes with twist)

2048 - Probing - StreamLine probe currently active for this StreamLine

A.3 Particles

This mask controls the **pV3** bubble rendering attributes and therefore the requested sub-extracts.

1 - Render - Bubble rendering on

2 - Size - Bubble size based on divergence like tubes - currently not used

4 - Grey - Bubble colored with default color

16 - Time - Bubbles are colored with the time of spawning

32 - Material Lines - Plot lines between particles in the same group

A.4 Vector Clouds

1 - Render - Vector cloud rendering on

4 - Grey - Vector cloud colored with default color