# Visual3: Interactive Unsteady Unstructured 3D Visualization

Robert Haimes and Michael Giles

Computational Fluid Dynamics Laboratory

Department of Aeronautics and Astronautics
Massachusetts Institute of Technology, Cambridge, MA 02139

## Abstract

A new graphics package has been developed for the visualization of three-dimensional engineering and scientific data. Unlike existing packages, it can handle a wide variety of structured and unstructured grids, and deals with generic scalar and vector data, making it suitable for a wide range of physical applications. An important feature of the package is the ability to view unsteady data in either post-processing or co-processing modes.

## 1 Introduction

The field of scientific visualization is young. PLOT3D [1], the first major visualization program for three-dimensional CFD results, was developed approximately five years ago, shortly after the introduction of the Silicon Graphics Iris graphics workstations with dedicated hardware for the rapid manipulation of three-dimensional objects. At the time, block-structured grids and flow solvers were the standard for three-dimensional calculations, and so PLOT3D was designed to handle just this one data type. Similarly, the hardware at the time had limited computing and graphical rendering capability, and so PLOT3D was designed more for the quality of the final images than for interactive speed, and the rendering was restricted to computational coordinate surfaces.

In the last five years there have been great changes in both CFD research and computer hardware. Unstructured grid generation methods [2, 3, 4] and corresponding flow solvers have evolved rapidly. Progress has also been rapid in adaptive flow solvers which may start with a structured grid but finish with a semi-structured or unstructured grid [5, 6]. These methods need entirely new visualization software. At the same time, graphics workstations have advanced to the point where high-end 'graphics superworkstations' have a compute speed on the order of 15MFLOPS and a graphics rendering speed of around 100,000 Gouraud-shaded, Z-buffered, 3D-transformed triangles per second. This level of performance makes truly interactive visualization possible, including iso-surfaces and various forms of cutting planes.

Beginning two years ago, driven by the need for visualization of steady flows on three-dimensional unstructured grids and unsteady flows on two-dimensional grids, we have been developing two packages, Visual2 and Visual3, for the visualization of two- and three-dimensional data, respectively. This paper discusses the program architecture, screen layout, tools and use of Visual3 in both steady-state and unsteady applications. A discussion of algorithmic issues can be found in another paper by Giles and Haimes [7]. This reference also discusses the earlier development of the two-dimensional package, Visual2.

## 2 Visual3 Architecture

### 2.1 Design Goals

The following were the design goals for Visual3, together with their motivation and implications.

- High performance

  Visual3 was designed to take advantage of the latest advances in graphics workstation hardware. Instead of aiming the code to be portable across the vast majority of existing graphics workstations (including SGI, DEC, SUN, HP and IBM) and accepting the current limitations of these machines, Visual3 was instead written for a machine in the graphics supercomputer workstation class. The Stardent GS series of machines was chosen for the development of Visual3 because of its rendering speed and raw computational capability. Also, the graphics system is based on X with an extended server that performs the 3D drawing. This paradigm is very similar to the proposed standard PEX (PHIGS Extension to X) and so our planned port of the code to PEX in the future should be straightforward.

  Our expectation is that within another five years the vast majority of high-end graphics workstations from the vendors listed above will have 24-bit graphics, Z-buffers, Gouraud-shading hardware and PEX software, and thus will be suitable platforms for Visual3. As the rendering and compute capabilities improve in the future, larger data sets with faster animation rates will be possible, without any modification to Visual3.

- Highly interactive research tool

  Visual3 was designed primarily as an interactive tool to aid researchers in understanding and interpreting their computational results. The production of glossy presentation graphics was very much a secondary role. To improve its usefulness as an investigative tool, considerable attention was devoted to providing a wide variety of tools and 'probes' which the researcher can use to extract information from the solution. Also, the design emphasized performance in both the graphics and compute portions to maximize the degree of user interactivity.

- Animation of time varying data

  Although at present most three-dimensional calculations are steady-state, there is a growing number of unsteady calculations, such as the stator/rotor interaction calculations of Rai [8] and Chen [9]. As computer speeds increase we will reach a point at which such calculations become possible on a more routine basis, and will have to face the need for good visualization tools. Therefore, Visual3 was designed to handle unsteady data in both co-processing and post-processing modes.

- Support of unstructured grids

  Visual3 was designed to support as many different CFD grids as possible, including both block-structured and unstructured grids. Visual3 views any grid as a heterogeneous collection of cells of four types: hexahedra, tetrahedra, prisms and pyramids. These four types support almost all current CFD codes. The hexahedra are for structured, block-structured [10] and adaptive hexahedral programs [6]. The tetrahedra are for unstructured tetrahedral programs [2, 3, 4]. The prisms are for viscous boundary layer elements used in conjunction with unstructured tetrahedral elements [11]. Finally, the pyramids (with a quadrilateral base and four triangular faces) can be used for the decomposition of larger cells types; any cell with quadrilateral and triangular faces can be split into pyramids and tetrahedra by forming an additional node at its center, and connecting it to all of the other nodes.

- Generality

  It was decided that Visual3 should not be limited to simple CFD applications, and should instead be useful in areas such as chemically reacting flows, aeroacoustics, structural analysis, electomagnetics. This was accomplished through designing Visual3 to handle scalar and vector data in a generic fashion, with the responsibility on the programmer to supply that data.

- Simple programming interface

  Any graphics post-processing package requires some programming, although it may be as simple as converting the flow solver's data to a form that the package can read. A goal for Visual3 was to keep any programming required as simple as possible, expecting that the typical programmer/user (at least in the CFD community) would be a knowledgable FORTRAN programmer with no desire to learn graphics programming such as X-windows. Accordingly, Visual3 was designed to be used from FORTRAN, both with a C interface available for generality. Visual3 itself is written primarily in FORTRAN, with certain crucial parts (particularly graphics, sorting and dynamic memory allocation) in C.

### 2.2 Visual3 Structure

The standard approach to a visualization system is a monolithic, end-user program. The user/programmer converts the data into a form that the program expects and then executes the code. This cannot meet the design goals. At the other extreme, a traditional graphics library can supply the generality and flexibility but requires the programmer to have a great deal of graphics knowledge and perform a substantial amount of programming.

The approach taken with Visual3 is a library, but not a library in the usual sense: there are not hundreds of low level calls. The programmer makes a call to initialize Visual3 and also supplies routines that return data to Visual3. The programmer need not know anything about internal data structures, graphics or windowing; all this is handled by Visual3. Therefore the program which the programmer must write is only a little more complex that the conversion program required by the monolithic approach.

### 2.3 User Interface

Visual3's user interface is also different from most packages which are being developed currently. Other packages are emphasizing a 'point-and-click' Graphical User Interface (GUI) with a hierarchy of pop-up/pull-down menus. This approach was considered and rejected in favor of an interface using mouse clicks and dialbox and keyboard input. There were two reasons for this. The first is our opinion that an experienced user can invoke the options in the visualization package more quickly with single keystrokes than using 'point-and-click' with multiple menus. For the novice user, there is a help menu listing the available options in each window, and printed reference lists are also recommended. The second reason is that pop-up/pull-down menus require considerable amounts of screen space. Often more than half of the area on the screen is dedicated to this input task, whereas in Visual3 approximately three quarters of the area is dedicated to the output task of displaying images. This is particularly important because of the desire for separate windows to display 3D, 2D and 1D data.

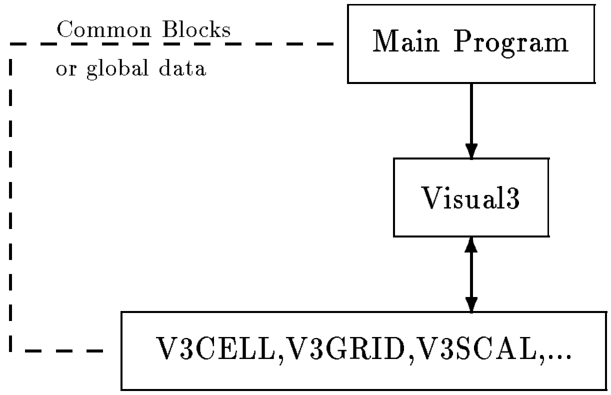


Figure 1: Flow diagram showing use of Visual3

# 3 Programming Interface

## 3.1 Overview

Fig. 1 shows how the programmer interacts with Visual3. The application program first activates and initializes Visual3 by calling V3_INIT. Visual3 then gets the data it requires by calling a small number of programmer-supplied subroutines. These subroutines are connected to the main program only through COMMON blocks in FORTRAN or through global data in C.

The program control in this interaction between the main program and Visual3 depends on whether Visual3 is being used for post-processing or co-processing. In a post-processing mode, the results of some calculation are already known. The main program will call Visual3 and transfer program control to it. Visual3 therefore remains in total control; when it calls the programmer-supplied routines, they return the grid coordinates, function values, etc. Control is only returned to the main program when the user exits from Visual3.

In co-processing modes, the results are being calculated concurrently by the main program. In this mode, when the main program calls Visual3, this initializes it and generates a new process (actually a 'thread', the distinction being that threads share global memory and so data transfer is simpler than using sockets, the standard UNIX method of communication between processes) which runs the graphics, before returning to the main program. At this point there are now two processes (threads) running, one executing the main program, and the other executing Visual3. To avoid data conflicts between the two, special locking mechanisms have been set up to ensure that the main program does not update its data at the same time that Visual3 is fetching it by calling the programmer-supplied routines.

When describing the mesh, the programmer gives Visual3 a list of unconnected cells of four types; tetrahedra, pyramids, prisms and hexahedra. These four types cover almost all data structures being using in current computational algorithms. Any cell type not covered, or any element using a higher order interpolant must be split up into a combination of these four where a linear assumption is valid.

In order to calculate streamlines from vector fields Visual3 requires information about which cells are neighbors, i.e. share a common face. Rather than impose on the programmer the burden of computing these connections and passing the information to Visual3, instead Visual3 calculates this information by processing all cells, compiling a list of all of their faces and checking for each face whether it appears on another cell. If it does appear twice, then it is an interior face, and a streamline 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. It should be noted that Visual3 only requires connectivity for the streamline algorithm.

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 as 'domain surfaces'. For example, the outer bounding surface of a calculation of airflow past a half-aircraft (using symmetry to reduce the computation) might 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 Visual3 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).

## 3.2 Subroutines

The following is a list of Visual3 subroutines with a very brief description of their function:

Programmer-called subroutines:

- V3_INIT: initializes Visual3, allocates required memory based on number of nodes, cells and surface faces, opens the windows
- V3_STAT: handles synchronization in co-processing applications
- V3_CURSOR: moves the cursor to the text window (and back) when the program asks the user questions

Programmer-supplied routines:

- V3CELL: returns the cell and domain surface data structures
- V3GRID: returns the grid coordinates (XYZ triads) for all nodes
- V3SCAL: returns the requested scalar function values for all nodes
- V3VECT: returns the requested vector function values (optional)

- V3THRES: returns the requested threshold function values (optional)
- V3ZPRIME: defines the programmer-defined cutting plane (optional)
- V3XYPRIME: defines the associated $(x', y')$ values for programmer-defined cutting plane (necessary if V3ZPRIME exits)
- V3SURF: initializes plotting of mapped domain surfaces (optional)
- V3XYSURF: defines $(x', y')$ values for mapped domain surfaces (necessary if V3SURF exists)
- V3SSURF: returns the surface scalar function values (optional)
- V3VSURF: returns the surface vector function values (optional)
- V3EVENT: allows programmer processing of X-events (optional)
- V3EVENTHELP: additional help info associated with V3EVENT (optional)
- V3STRING: additional label for plots - label time in unsteady applications (optional)
- V3UPDATE: updates time and/or data in unsteady post-processing applications (optional)

### 3.3 Visualization of Steady Data

A simple steady-state visualization application program is a main routine that reads the data from disk and calls V3_INIT, plus four other subroutines, V3CELL, V3GRID, V3SCAL, and V3VECT, that are called by Visual3. Figure 2 shows the flow diagram for the control of Visual3. In the initialization phase V3CELL and V3GRID are called once to obtain the data structure and grid coordinates, which do not change. After initialization, Visual3 enters a large animation loop. At the beginning of the loop the first step is to process the list of X-events (keystrokes, dial movements, and cursor movements) that have occurred since the beginning of the previous loop. Processing this list can result in changes in the transformation matrix being used to view the objects in the 3D and 2D windows, or a change of desired scalar or vector variable, or the invokation of some new plotting function. The second step is to call V3SCAL if the scalar variable has changed, and V3VECT if the vector variable has changed. The third step is to construct any 2D structures required for cutting planes and iso-surfaces. The final step is to perform the rendering, the drawing of the images in all of the graphics windows. The third and fourth steps are usually the most computationally and graphically demanding, and are optimized by keeping careful note of what has changed since the previous loop.

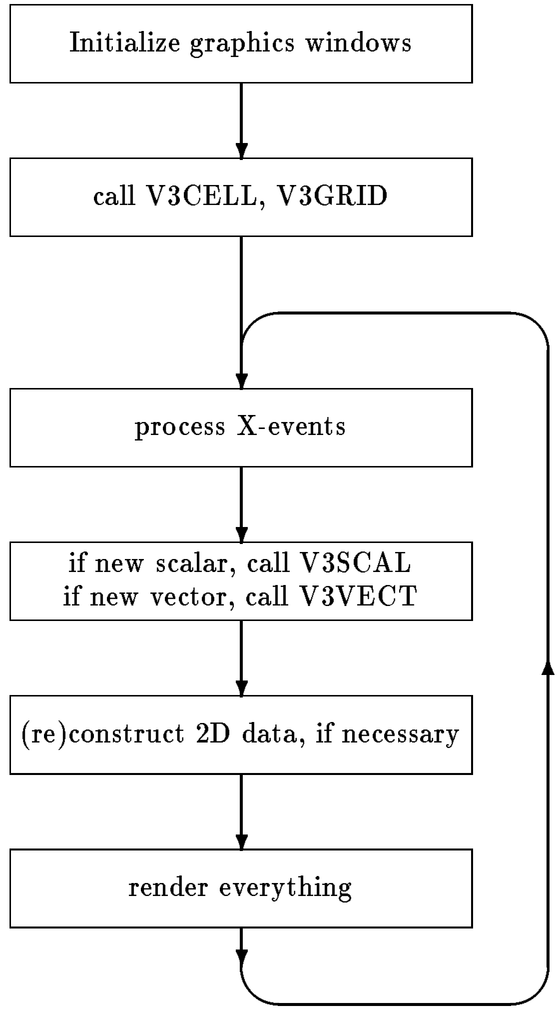


Figure 2: Flow diagram of Visual3 for steady application

### 3.4 Post-processing Unsteady Visualization

There are currently two supported types of unsteadiness. The first type is 'data unsteady', in which the grid is fixed but the data changes in time. The second is 'grid unsteady' in which the grid coordinates and the data are time varying. At present the grid structure is not allowed to be unsteady. If the number of nodes, number of cells or cell connectivity changes then Visual3 must be re-initialized by terminating and restarting it.

The modifications to the control loop in Visual3 to handle both 'data unsteady' and 'grid unsteady' modes are very minor, as shown in Figure 3.

Before the data collection phase, the programmer-supplied routine V3UPDATE is called. This allows the programmer to update the data as appropriate for the problem. As can be noted by this approach, the time varying data can exist in any form and reside in memory or on disk, meeting many of the design goals.

The other two changes are that V3GRID is called if it

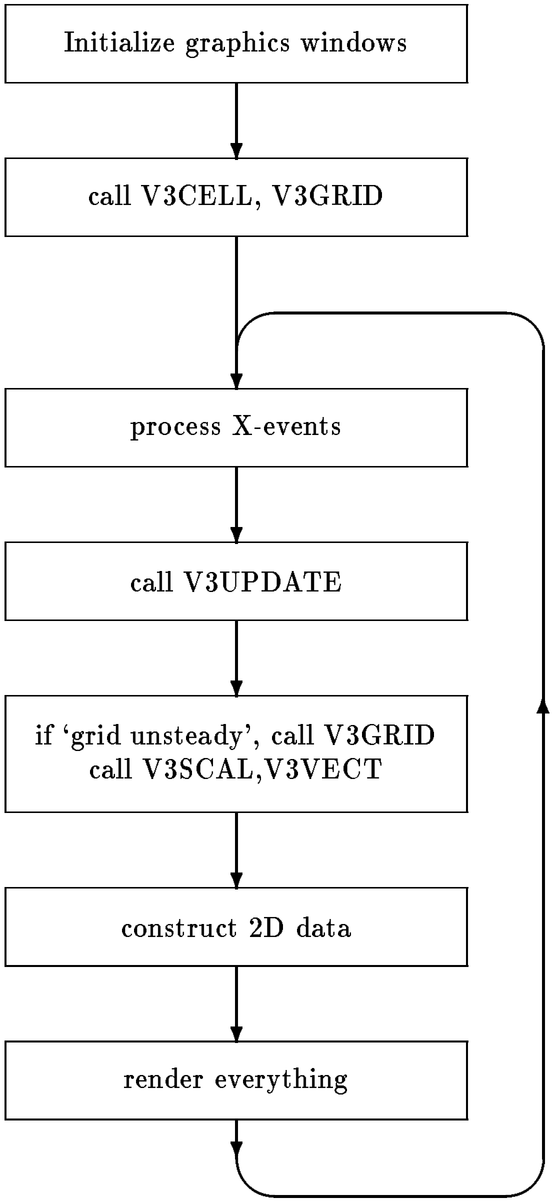


Figure 3: Flow diagram of Visual3 for post-processing unsteady application

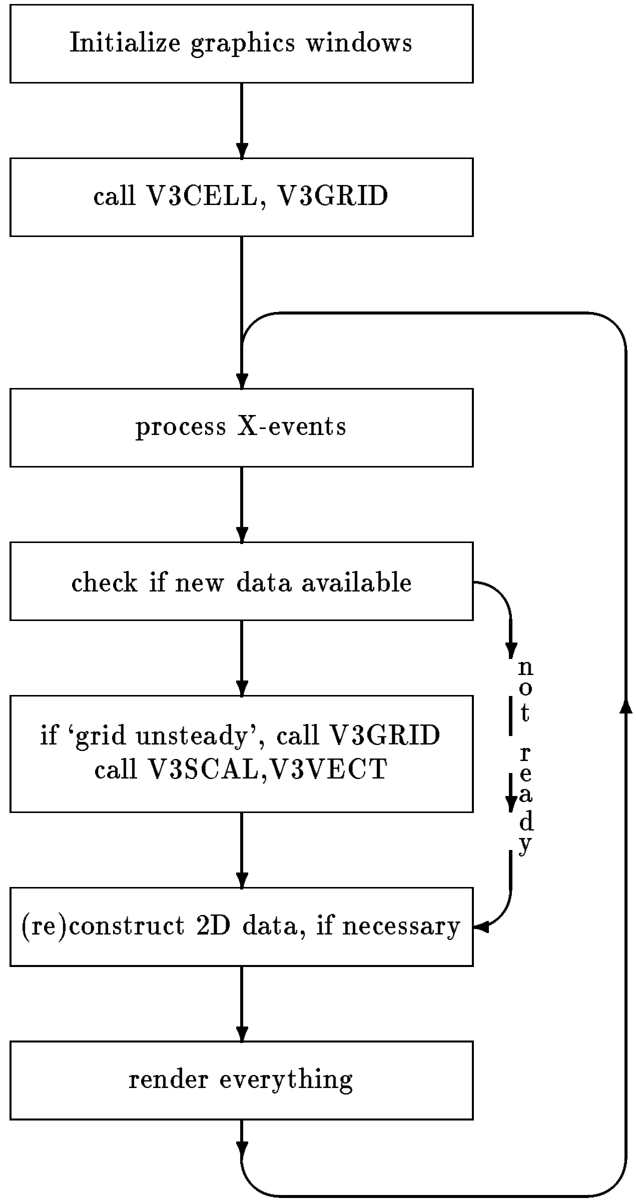


Figure 4: Flow diagram of Visual3 for co-processing unsteady application

is 'grid unsteady' and so new grid coordinates must be obtained, and V3SCAL and V3VECT are always called, even if their definition has not changed, because their values will have changed.

### 3.5 Co-processing Unsteady Visualization

In co-processing, the programmer controls when new data is available to Visual3, by calls from the main program to V3_STAT. When the main program wants to update the flow variables, at the end of one timestep for example, it calls V3_STAT to notify Visual3 that the data is being updated and is not currently available. If Visual3 is in the process of accessing the data, then V3_STAT waits until it is completed before seizing control. Once the main program has finished updating the data, it calls V3_STAT to notify Visual3 that the data is available.

From the point of view of Visual3, the modifications to the control loop are shown in Figure 4. After processing the X-event list, and before updating the data, it checks to see if the data is available. If it is not, then Visual3 jumps to the rendering phase.

With this approach, the precise details of the implementation of the multiple threads and locking mechanisms are hidden from the programmer, who simply deals with standard FORTRAN and C code.

# 4 Visualization Tools

## 4.1 Windows

A multi-windowing setup in concert with reduction of dimensionality is useful in understanding complex 3D data. Trends are much easier to comprehend if shown on a 2D map. Linear cuts in 2D can be displayed in a normal XY plot. In Visual3 each type of display has its own window and a function can be active concurrently in the 3D, 2D and 1D windows.

The screen 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 Visual3, or rather the application program, was started. 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, either from the bounding domain or from cuts and iso-surfaces. It also displays three-dimensional lines such as tufts and streamlines. The objects in the 3D window can be rotated, translated and enlarged using the dialbox.

- 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). The user can independently adjust the view in the this window. When pointing (with the cursor) in the 2D window, the user is actually pointing in 3-space by reversing the 3D to 2D mapping. This is a simple and powerful concept. Without a multi-window system pointing in 3D is, at best, difficult.

- 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 streamlines.

- Key Window

  This window displays the color index 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.

- Dialbox Window

  The dialbox window serves two functions. The first is to display the functions associated with each of the dials of the dialbox. 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 cursor button.

  In this window's secondary role, it displays the state of the surface database. The database lists all of the surfaces, and for each one has four small boxes which give the status of the surface attributes. The attributes are rendering status (on, off or translucent), the second is the grid intersection status (on or off), the third is the gray status (on or color by scalar) and the fourth is the thresholding status (on or off). The user can change the attributes by pressing any mouse button when the cursor is on one of the boxes.

## 4.2 Scalar Visualization Tools

The most important data type in Visual3 is scalar data, which is simply scalar information defined at each node of the computational grid. Visual3 does not know anything about the data other than an associated function number and label. An important concept in Visual3 is the notion of 'active' functions, and the 'active scalar function' is the function number and associated label which corresponds to the scalar data currently stored by Visual3. Its label is displayed in the key window. The user can switch to a different scalar function by hitting a particular key on the keyboard which is 'bound' at initialization. This causes V3SCAL to be called with the function number.

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

- Surface rendering

  Gouraud-shaded (smooth color shading) surface contours of the scalar function can be rendered on any, or all, of the surfaces. Domain surfaces may be mapped to 2D for further probing.

- Cutting plane

  The user may interactively set a plane at any orientation through the 3D field. The resultant planar cut is displayed in both the 2D and 3D windows. The scalar field may be either shaded or contoured with lines. The user may interactively move the plane parallel to the current location.

- Program-defined cutting plane

  This is similar to the last option, but instead of being a truly planar surface, it is a surface corresponding to $z' = const$, where $z'$ is a programmer-defined function; the 'scan' capability varies the value of $const$ interactively. This allows the programmer to define conical, cylindrical or other surfaces not otherwise defined by Visual3. The programmer also must define a mapping to $(x', y')$ coordinates so that plotting is possible in the 2D window.

- Iso-surfaces

  An iso-surface is a 3D surface with a uniform value of the current scalar variable. There is no 2D mapping. The iso-surface value is displayed in the key window and can be varied interactively using the dialbox to 'scan' the value of $z'$. An exact value may also be set.

### 4.3 Vector Visualization Tools

The second 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, causing V3VECT to be called.

There are four plotting functions for vector data:

- Streamlines

  'Streamlines' are curved 3D lines which are everywhere parallel to the local vector field. They are obtained by numerical integration [7] of the vector field along a line starting at some chosen location. The starting point is determined by use of a surface plotted in the 2D window. Starting a single streamline is as easy as clicking a mouse button. Lines, circles and grids of streamlines may also be generated by pointing in the 2D window.

  The streamlines can be plotted either as lines of constant color, or rendered according to the value of the local scalar function.

- Ribbons

  Stream ribbons are streamlines that have been given some width. One edge is the true 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. The width of the ribbons may be inteactively adjusted, also the ribbons may be rotated in real-time by the user.

- 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.

- Arrows

  Arrows are the same as tufts except that they are defined only at 2D nodes. To emphasise that they are different, they are drawn as arrows, whereas tufts are drawn as straight lines with a cross base. They are shown only on cutting planes and are not displayed in the 3D window.

### 4.4 Thresholding

A threshold function [12], when enabled, restricts all surface plotting to only those parts of the surface on which the values lie 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, which surfaces 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.

### 4.5 Probes

There are a variety of 'probes' which are available when plotting in the 2D window or from streamlines:

- point probe

  The point probe is located at the cursor position in the 2D window, and returns, in the text window, the point's coordinates and the value of the active scalar and vector functions.

- strip chart

  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 probe

  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

  The edge plot is similar to the line plot, except that in this case the line in the 2D window is the edge (intersection of a domain surface and the cut) closest to the cursor when this option is invoked.

- surface layer probe

  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

  The streamline probe may be started any time there are streamlines. The last streamline is mapped to the 1D window in a line plot of the current scalar against the length of the streamline. When the cursor is in the 1D window a cross-hair cursor appears in the 3D window. This allows the user to both know which streamline is mapped and probe the streamline. Hitting a key selects another streamline.

## 5 Conclusions

This paper has discussed some of the design features of a new system for the visualization of scientific and engineering data. The key components include:

- generic scalar and vector data
- unstructured grid with four cell types
- unsteadiness allowed in both post-processing and co-processing modes
- high performance with a variety of interactive 'probes'
- use of multiple windows for the simultaneous presentation of 3D, 2D and 1D data.


## Acknowledgements

Prof. Earll Murman, Mr. David Modiano, Mr. David Darmofal and Dr. John Dannenhoffer have all contributed greatly to this research through critical review of the software design at an early stage, many suggestions for improvements at later stages, and ideas on new 'probes' and interactive capabilities.


## References


[1] P. Buning and J. L. Steger. Graphics and flow visualization in computational fluid dynamics. AIAA Paper 85-1507, 1985.

[2] T. J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained delauney triangulation. *Engineering with Computers*, 5:161–175, 1989.

[3] R. Löhner, K. Morgan, J. Peraire, and O. C. Zienkiewicz. Finite element methods for high speed flows. AIAA Paper 85-1531, 1985.

[4] J. Peraire, J. Peiro, L. Formaggia, K. Morgan, and O. Zienkiewicw. Finite element euler computations in three dimensions. *International Journal for Numerical Methods in Engineering*, 26:2135–2159, 1988.

[5] J. F. Dannenhoffer III and J. R. Baron. Robust grid adaptation for complex transonic flows. AIAA Paper 86-0495, 1986.

[6] S. Samant, J. Bussoletti, F. Johnson, R. Burkhart, B. Everson, L. Melvin, D. Young, L. Erickson, M. Madson, and A. Woo. Tranair: A computer code for transonic analyses of arbitrary configurations. AIAA Paper 87-0034, 1987.

[7] M. B. Giles and R. Haimes. Advanced Interactive Visualization for CFD. *Computing Systems in Engineering*, 1(1):51–62, 1990.

[8] M. M. Rai. Unsteady three-dimensional Navier-Stokes simulations of turbine rotor-stator interaction. AIAA Paper 87-2058, June 1987.

[9] Y. S. Chen. 3-d stator-rotor interaction of the ssme. AIAA Paper 88-3095, 1988.

[10] L.-E. Eriksson, R. Smith, M. Wiese, and N. Farr. Grid generation and inviscid flow computations about cranked-wing airplane geometries. AIAA Paper 87-1125, June 1987.

[11] K. Nakahashi. A finite-element method on prismatic elements for the three-dimensional navier-stokes equations. Proceedings of the 11th International Conference on Numerical Methods in Fluid Dynamics, June 1988.

[12] R. F. Weston. Color graphics techniques for shaded surface displays of aerodynamic flowfield parameters. AIAA Paper 87-1182, 1987.

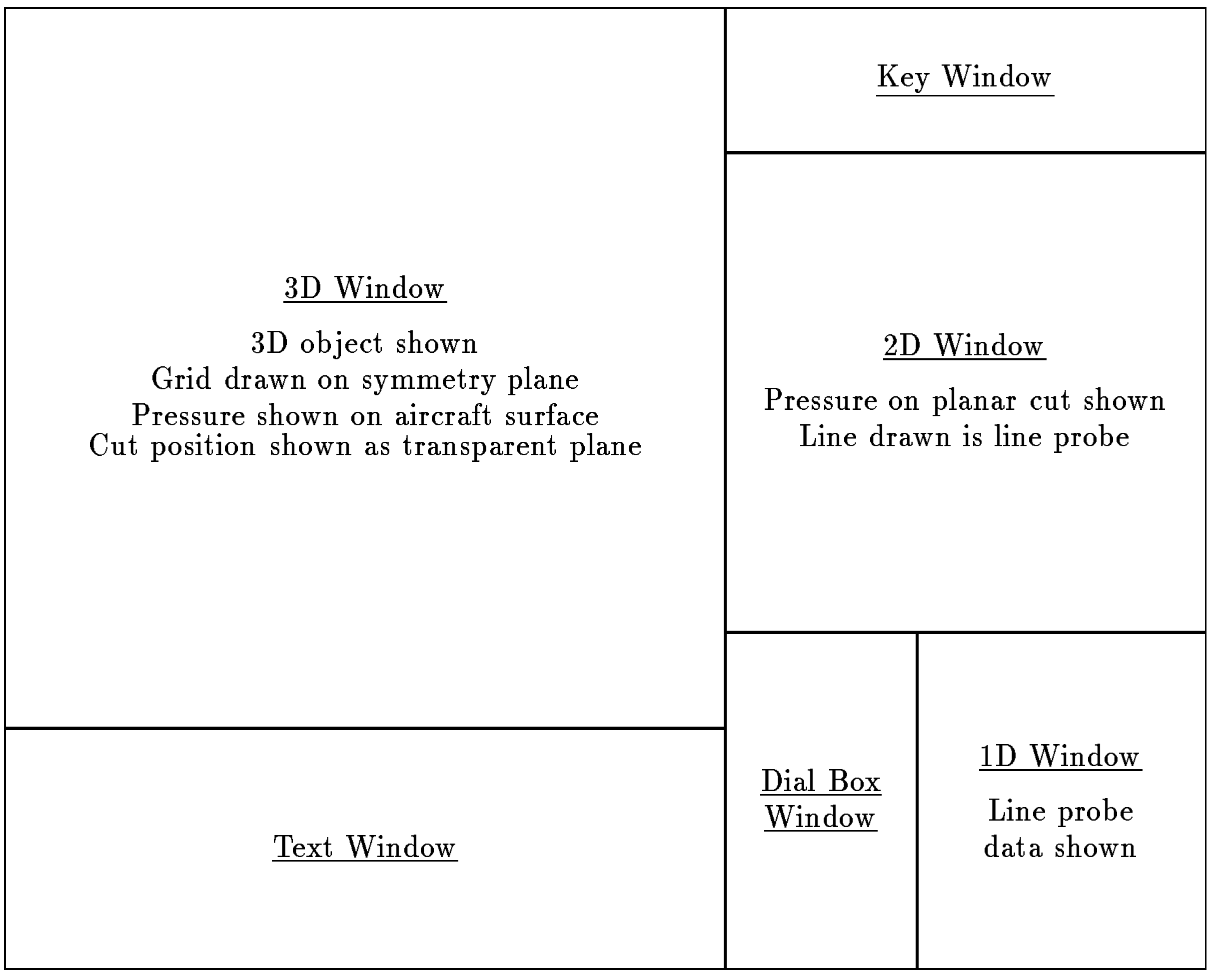


Figure 5: Screen layout