# TPview: An On-line Visualization Tool and Its Application to CFD on the Cray T3D

*Steve Williams*, Service Informatique Central, Ecole Polytechnique Fédérale de Lausanne, Switzerland, *David Cobut* and *Mark L. Sawley*, Institut de Machines Hydrauliques et de Mécanique des Fluides, Ecole Polytechnique Fédérale de Lausanne, Switzerland

**ABSTRACT:** *An on-line visualization tool has been developed to overcome the problem of storing the large amount of data generated by numerical simulations on a high-performance parallel computer system. Data distributed across multiple processors are sent via UNIX sockets to a remote graphics workstation for post-processing during the computation cycle. The tool employs the inter-process communication library for processing the data by the Tecplot visualization software. Examples of the use of this tool for Computational Fluid Dynamics simulations on the Cray T3D will be presented and various issues associated with its performance addressed.*

## Introduction

In the domain of high-performance computing of scientific simulations, productivity can be greatly increased through the use of easy-to-use visualization tools. The researcher may be overwhelmed with the quantity of data produced, and has little desire to manipulate files of various formats or to learn to use a variety of tools. In fields such as Computational Fluid Dynamics (CFD), access to graphic representations of results is imperative in order to verify the correctness of solutions, to gain physical insight into the computed flows, and as a means to present this information to other people.

The process of visualization can be divided into five discrete phases:

1. Computation
2. Data Selection or Filtering
3. Data Transfer (if necessary)
4. Rendering
5. Display

Different visualization approaches can be categorized according to how each of the above phases is accomplished. In particular, for a heterogeneous computer network such as considered in the present study, it is important to distinguish on which platform each phase is to be performed.

For lack of a more convenient method, users often resort to a *batch visualization* approach. This consists of saving datasets at various times during the simulation. After the simulation has completed, the files are manipulated into an appropriate format, and the datasets entered one at a time into a visualization package, usually on a graphics workstation. This approach has a number of drawbacks, associated with the time needed to manipulate the data into the appropriate format, and the amount of disk space required to save all the intermediate steps. In addition, if the results of the simulation that requires many minutes or hours are incorrect at the beginning, batch visualization does not provide a detection means that allows termination of the execution to avoid wasting computational resources. Finally, the analysis of CFD simulations often requires an animation of the computed flow and not simply static images.

This paper describes the development of a tool, TPview, to allow *on-line visualization* (or run-time visualization) for which the computation and visualization are performed concurrently. In our case, data computed on the Cray T3D is rendered on a remote graphics workstation (Silicon Graphics Indigo2). The goals were to build a versatile tool that would:

- have as little impact as possible on the original simulation, that is, the execution time overhead due to the visualization be minimal,

- require the user to make as few changes as possible to the simulation code,
- be composed of as many “off-the-shelf” components as possible,
- enable the user to influence interactively the behaviour of the simulation.

## TPview Selection Considerations

***Computation and Data Selection***

The computation phase is to be done by the user's code on the Cray T3D. Since the data to be visualized is usually distributed amongst the processors, the data selection phase can also be accomplished in parallel with a low overhead.

Particularly with large datasets, often only a subset of the data from the simulation is to be viewed. A versatile method of indicating the minimal amount of data to send is therefore required. This is achieved using a concept borrowed from PORTAL, a tool developed at the Argonne National Laboratory [1]. The user selects the data to send with a call to a library routine in which he specifies the size, type and location of the data. This has the advantage of being rather versatile, and allows the calls to be added or removed from the code without any modification to the rest of the program.

***Data Transfer***

Since the data selection and rendering phases are performed on different computer systems, a data transfer phase must be undertaken. Transfer of data from the T3D to the visualization computer system on the local network is undertaken using UNIX sockets [2]. This method was chosen over message passing (see e.g. [3]) for the following reasons:

- sockets can be manipulated without any explicit use of the front-end machine of the T3D (i.e. Y-MP) – the system calls are still handled by the Y-MP, but in a completely transparent manner,
- the message passing library needs to be installed on each machine that communicates with others – sockets are a part of standard UNIX, and are therefore available on all UNIX machines,
- message passing uses sockets as its underlying transfer mechanism, so the direct use of sockets should avoid the extra overhead required for buffers and daemons.

Using sockets for data transfer between the T3D and a remote workstation over Ethernet, transfer rates have been measured (4 Mbits per second) that are approximately half of the maximum available bandwidth. Further discussion on communication issues involved with the TPview tool can be found in [4].

In our local T3D configuration, it is currently not possible to open more than 16 output sockets at one time. This means that at most 16 (and often fewer) processors can transfer data to the remote workstation per iteration. This problem has been overcome by re-grouping the entire dataset onto a subset of processors, which is then involved in the data transfer. This idea can in fact be exploited by having those processors that have less calculation to perform open the sockets and communicate with the remote machine. The user has control over how many sockets are opened by setting the value of the TP_SEND_PES environment variable.

***Rendering***

It may appear appropriate to perform the rendering phase on the T3D (see e.g. [5]). However, no parallel renderers are currently available that provide the versatility that is required for our applications. A second problem with this approach is that performing a rendering phase could greatly increase the total execution time of the simulation. This is not desirable when sharing the computing resource amongst many users. Visualization of data from only a subset of the processors in our partition may lead to a further problem of load balancing. Dedicating a subset of the allocated processors to rendering may be considered, however this would have a considerable impact on the writing of the code, contrary to one of the above-mentioned design goals.

If the rendering is to performed on another machine, the choice exists between the front-end of the T3D (Y-MP) or a graphics workstation connected to the network. For the present work, the second of these choices has been selected for a number of reasons:

- there are more visualization packages available on workstations, and thus a higher probability that a suitable one already exists,
- at our site, the Y-MP is heavily loaded, whereas essentially dedicated workstations are available,
- if the rendering was done on the Y-MP, the image would still have to be transferred elsewhere for displaying
- compatibility of IEEE floating-point formats on the T3D and workstations (the Y-MP uses Cray floating-point format),
- the successor to the T3D system does not require a front-end,
- as a design feature, this tool should be made available on as large a number of different machines as possible,
- graphics workstations are quite common at sites that have parallel machines, and it should be possible to take advantage of their hardware that often has rendering power equivalent to many general purpose processors such as on the T3D.

When searching for a suitable rendering tool, an obvious candidate was the commercial visualization package Tecplot [6]. This package has been chosen by the Fluid Mechanics Laboratory at the EPFL as its standard visualization tool, since it is capable of generating the types of plots needed with a convenient input format, and because it is available on virtually all common single-processor machines. Tecplot can use datasets comprised of multiple zones, corresponding directly to data distributions employed on a parallel system, each processor containing a “block” or multiple blocks of the global dataset.

A non-negligible advantage of Tecplot for the rendering phase lies in the fact that the immediate target user-community

for TPview is already familiar with Tecplot. Our experience has also demonstrated that Tecplot has a higher throughput than commonly employed Modular Visualization Environments (MVE), such as AVS or Explorer [7]. Tecplot is delivered with an Inter-Process Communication (IPC) library that allows a user-written program to employ Tecplot as a slave, by sending it commands and by passing data generated at run-time.

***Display***

Ideally the simulation results should be displayed either on a framebuffer connected to the T3D, or on an X-Windows display. The use of a framebuffer provides potentially higher visualization throughput. However, while a framebuffer would generally be located at a distance from the user's office, the X-Windows display can be placed on the user's desk. In addition, a framebuffer may not be available on the parallel system. (Indeed, the T3D system at the EPFL employed for the present study does not have an attached framebuffer.) For these reasons, and also since a goal of the project was to produce a tool that is accessible to as many users as possible, the display phase is handled using X-Windows.

## TPview Description

A schematic diagram of the layout of the TPview tool is presented in Fig. 1. As introduced above, three different computer systems are employed: a high-performance parallel system (Cray T3D), a graphics workstation for rendering and an X-Windows display.

Before describing the operation of the system, some terminology needs to be defined. By *parameter*, we mean any variable in the user's code that is modifiable through the user interface. A *display value* is a variable, either common to all processors (in which case it is called a *global* value), or whose value can be different on different processors (a *local* value), that is updated on the screen at each send iteration.

As a first step, the user describes the user interface through calls to the TPV library. This includes details on items such as: the set of parameters available for modification during the visualization by means of sliders, and the set of display values. Data selection is then undertaken, again using a series of calls to the TPV library.

The visualization process starts by one processor of the T3D sending a request to the TPview daemon on the remote workstation, which uses the arguments of this request to start the TPview module. This program then starts, initializes a shared memory segment between Tecplot and itself using the IPC library, and starts the user interface module. It then initializes a socket from itself to a T3D processor, and waits for socket connections to arrive from the various sending processors. At this stage, data output sockets to TPview are initialized. All initialization done at this step, including the manipulation of sockets, is transparent to the user.

At each send iteration, the T3D application makes a library call to send a new data subset via the sockets to TPview, which then writes the data into the shared memory segment and updates any parameters or display values. It then instructs Tecplot to render the newly received data. At any time during processing, the user can interact via the user interface module to send any user input to TPview, which will then forward it to the T3D using its output socket. All output from the user interface module and from Tecplot can be displayed on any X-Windows display by setting the appropriate value for the `DISPLAY` environment variable.

The TPV library also has the following features:

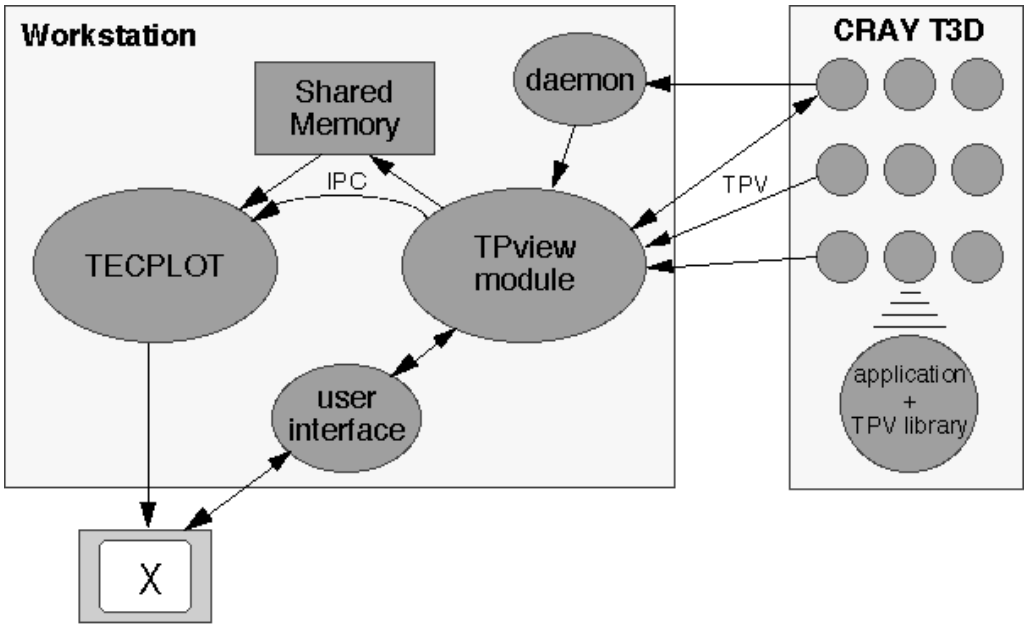


**Figure 1: Schematic diagram of TPview.**

- in a case where only the data from certain processors are of interest, the user can work with a subset of the entire dataset; this could leave some processors free to do other work,
- 8-byte entities from the T3D are converted into 4-byte entities before sending to the remote machine, since workstations normally work with 32-bit IEEE data – this also decreases the transmission time,
- an option exists to suppress any user interface, in which case the workstation is treated as a slave, and no synchronization between it and the T3D is necessary,
- callable from C and Fortran code.

To illustrate the use of TPview, a code segment containing calls to the TPV library is shown in Fig. 2. In this example, each processor has a subset (corresponding to a zone or block) of a distributed array containing a multi-component physical variable. The data is stored in a local array called `viz_vector` containing five two-dimensional variables, each of size `xdim` by `ydim`.

During the visualization loop, one T3D processor needs to communicate with the workstation in order to receive feedback from the user. This processor will then broadcast this information to all processors in the group. This processor is called the *master* processor. The call to **TPMaster** determines which of the processors will be the master. For load balancing reasons, it should normally be the processor that has the least computational work to perform. If the user omits the call to TPMaster, a master is automatically selected. The argument given can either be a PVM task id, or (in the case of the T3D) a processor number.

The call to **TPParameter** fully describes the manner in which the user may interact with the chosen variable. In this case, a slider is created with values ranging from 1.0 to 20.0 (with 1 decimal place), which will automatically control the value of the variable `cfl`. The title of the slider will be set to the string "CFL".

```
      INCLUDE 'tpview.inc'
      REAL viz_vector(5,xdim,ydim)

      CALL TPMaster(pvm_tid(1))
      CALL TPParameter(200,10,1,cfl,"CFL")
      CALL TPDisplayValue(2,TP_LOCAL,TP_RMS,
     &                    resid,"Residual")
      CALL TPDisplayValue(2,TP_GLOBAL,TP_AS_IS,
     &                    csq,"C squared")
      CALL TPZone(blk_no, TP_REAL,xdim,ydim,1,
     &            TP_FIRST_DIM,5,3,viz_vector)
      CALL TPSetSendInterval(15)
      CALL TPControl(TP_ON)
      CALL TPStart(TP_ACTIVE,"~user/tpv/TPCON-
    FIG")
      DO n_iter=1,n_iter_max
        CALL computation(update viz_vector)
        CALL TPSendData(n_iter,result)
        IF (result.NE.1) CALL TPEnd()
        IF (n_iter.EQ.1) THEN
          CALL TPSendCommand(WRITEBITDUMP,
     &                       "~user/cascade.rm")
        ELSE
          CALL TPSendCommand(APPENDBITDUMP,0)
        END IF
      ENDDO
      CALL TPEnd()
```

**Figure 2: Code segment using the TPV library.**

A display value is defined with the next call to **TPDisplayValue**. In this case, a local variable `resid` is contained on each processor. This value is to be displayed with the title "Residual", with at each send iteration the root mean square calculated and displayed to two decimal places. A second display value is defined by the code segment in Fig. 2 for the global variable `csq`.

The next call to **TPZone** involves the data selection phase. The arguments are defined as follows: the zone or block number; the data type to be sent (i.e. real, integer, etc.); the X, Y and Z dimensions of the source array; a flag to indicate the dimension in which the variables can be found in the source array; the size of the first dimension of the source array; the number of variables to send from the source array (in this case, variables 1, 2 and 3 are sent, corresponding to the pressure and the X and Y components of the velocity vector); and finally a pointer to the source array. Multiple zones may be defined per processor, and each processor may have a different number of zones.

The **TPSetSendInterval** routine call sets the frequency at which datasets are actually sent to the workstation. Usually the time necessary to calculate one iteration is much smaller than the time needed for Tecplot to render one image. If the send interval is set too small, the simulation will wait until the rendering is completed before sending a new dataset. Thus it is the user's responsibility to estimate an appropriate refresh rate. The send interval is by default always available as a parameter. The default send interval is 1, however its value can also be set via a TPParameter call.

The call to **TPControl** determines whether any special user interface is created; by default it is on. In this mode, communications between the workstation and the T3D are synchronized. If the user chooses to have no control, no communication between the master and TPview is necessary, so the simulation data is sent whenever it is ready without having to wait for the previous dataset to be processed. If, however, the user sends a new dataset before TPview is ready to receive, the Y-MP front-end will buffer the data until its buffers are full. When this happens, if a new dataset is sent, the T3D process will have to wait until buffer space has been freed on the Y-MP. This phenomenon is termed *saturation*.

At this point, the system can be started with the **TPStart** call. The TPview module is created, which in turn starts Tecplot and the user interface module. The data collected in the previous TPV library calls are then transmitted to the workstation, and the socket connections are made. At this stage there can be a delay

of a few seconds during which the simulation waits for a message from the TPview module, confirming that it has finished its initialization. TPStart has two arguments: the processor state, and the path of a configuration file.

The configuration file contains the name of the machine on which to start TPview and the path of a configuration file on the remote workstation, which TPview needs to configure itself.

The possible states of a T3D processor are represented in Fig. 3. A processor can either be ACTIVE or NOT_ACTIVE. If it is NOT_ACTIVE, it will not be involved further in the visualization process. The set of ACTIVE processors can further be divided into two groups: SENDERs and DISTRIBUTORs. A DISTRIBUTOR will have local data but no socket connection. A SENDER is a processor that will send its local data (if any) plus (optionally) any data received from a DISTRIBUTOR through its socket to the workstation. For each send iteration, data is re-grouped onto the set of SENDERs before being sent on to TPview. Note here that it is not required for a SENDER to have any of its own data to send, and that the master processor can be a member of any one of the sets.

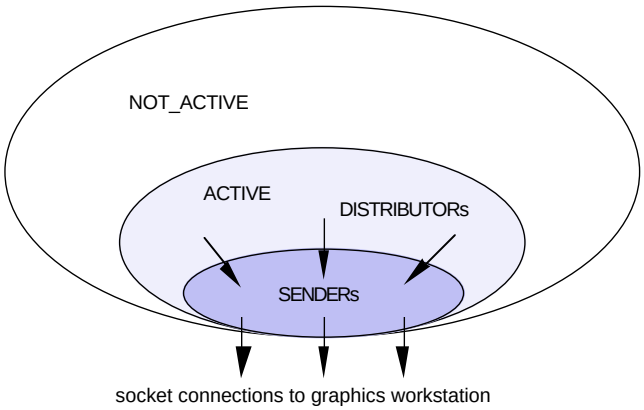


**Figure 3. Processor states defined for the TPV library.**

If the user does not explicitly define the set of SENDERs and DISTRIBUTORs, it will automatically be done for him. This is undertaken by choosing TP_SEND_PES (an environment variable) SENDERs from the set of ACTIVE processors. Currently, the selection criterion is the processors that have the largest amount of local data. The data on the non-sending processors are then distributed to the SENDERs so that each SENDER has approximately the same amount of data to transmit to the workstation.

Following this initialization stage, the program now runs as usual, updating the contents of `viz_vector`. When it is required that data be sent, a call to **TPSendData** is made, with an argument specifying the corresponding iteration number. It is at this point that the dataset is re-grouped on the sending processors, sent on the sockets, and also when any parameters or control from the user interface are updated. TPSendData is in fact implemented as two distinct steps – one for data output and one for data input. These steps can be called individually if, for example, the user wants to update the parameters more frequently than he wants to send data.

TPSendData returns a value which informs the simulation whether TPview has exited. This would be the case if the user selects Quit through the user interface, or some unexpected error occurs. At this point, any calls to the TPV library (other than TPEnd – see below) are ignored. Thus, the simulation continues in its normal mode, without visualization.

At any time after initialization, the user may send commands directly to Tecplot via a call to **TPSendCommand**. The code segment in Fig. 2 requests that a bitdump of the Tecplot window be created for the first iteration, and that bitdumps created for subsequent iterations be appended to it. This is often used when running a long simulation to produce a movie of the results. The list of possible commands is contained in Chapter 27 of [6].

When TPview is no longer required, each active processor calls **TPEnd**, which performs some cleaning and closes the socket connections. At this point, the simulation exits normally, and the user is left with a usual session of Tecplot with the last received dataset displayed.

To illustrate the type of graphical interface the user would observe, the code segment in Fig. 2 has been implemented into a parallel multi-block solver [8] to compute the incompressible flow through a turbine cascade. For this example, 16 ACTIVE processors were employed with 8 SENDERs. A pressure contour plot with superimposed flow streamlines for a two-dimensional slice through an inter-blade channel is rendered for each send iteration, according to a user-defined style file. (The style file can be changed during the course of the simulation by sending a READSTYLESHEET command with the TPSendCommand call.)

For the present example, the computational mesh does not change during the simulation. To avoid unnecessary data transfer, the mesh coordinates are therefore stored in a file on the workstation that is read during initialization. Little effort would be required to handle adaptive mesh applications if this is desired.

In addition to the standard window, a separate window is observed on the left hand side of Fig. 4, containing a menu of commands, a set of display values and a set of parameters. Each parameter is controlled by a sliding scale. The user is free to modify the corresponding variable on the T3D, and if the T3D program changes the value of that variable, the scale value will also change. The iteration number of the currently displayed image is shown as the first entry of the display values.

Each user-defined display value has a title field, a value field, a method menu and a plot menu. The value is calculated according to the current method selected, and depending on whether the corresponding variable is local or global. The rms value of the local variable labelled “Residual” and the value of the global variable “C squared” are displayed in Fig. 4. The plot menu allows the user to follow the variation of these values during the simulation by invoking separate plot windows.

Currently, a global value can be plotted versus the iteration number, while a local value can be plotted versus the iteration number, or the values for each zone can be plotted versus the zone number.

The menu of commands consists of buttons with the following functionalities:

- Detach/Attach: by pressing this button, a signal is sent to the T3D to tell it to stop sending any data except the iteration number. When *Detach* is pressed, the button changes to *Attach*, which, if pressed, will instruct the simulation to recommence sending data.
- Step: pause at the current image (this also pauses the simulation on the T3D) when pressed for the first time. For each subsequent press the next iteration dataset will be sent and rendered.
- Pause/Continue: *Pause* is the equivalent to the first *Step* press. After pressing *Step* or *Pause*, this button becomes *Continue*. Pressing *Continue* returns the simulation to normal processing mode (i.e. send data without waiting for a message from TPview).
- Quit: close all connections with the T3D, release control of, and exit TPview. After pressing *Quit*, the simulation continues execution but will no longer send information to TPview. Tecplot remains open, so the user can manipulate the data from the last image rendered.

## Performance Issues

As described above, TPview appears to fulfil the functionality design goals set out at the beginning of the project. The influence the visualization has on the execution time of the simulation has also been investigated.

In the case of TPview, the bottleneck of the system is generally the time needed for Tecplot to render an image. The user therefore has to make an estimation of how long Tecplot needs to produce an image, and the time needed for an iteration of the simulation on the T3D. An appropriate send frequency can then be chosen so that the time needed for the simulation to calculate the corresponding number of iterations is at least as long as the time required by Tecplot. If this is not the case, the execution time of the simulation will increase significantly.

Studies conducted to date have indicated that for operation when saturation does not occur, the overhead due to visualization is minimal. However, saturation – due to e.g. an excessive time for rendering a complex image – can result in a significant overhead (up to 30% has been measured). When dealing with large datasets, the visualization overhead can become very large even without saturation, thus the interest in sending only a subset of the dataset.

Calls to the TPV library may also require tuning to take into account:

- whether the simulation can be balanced so that processors with a greater computational work can employ other processors to send their data (this is done by designating certain processors as SENDERs or DISTRIBUTORs explicitly),
- whether by increasing the number of sending processors (TP_SEND_PES), the overhead is decreased (for a discussion on this topic, see [4]),
- whether there is a certain buffer size above which send performance decreases.

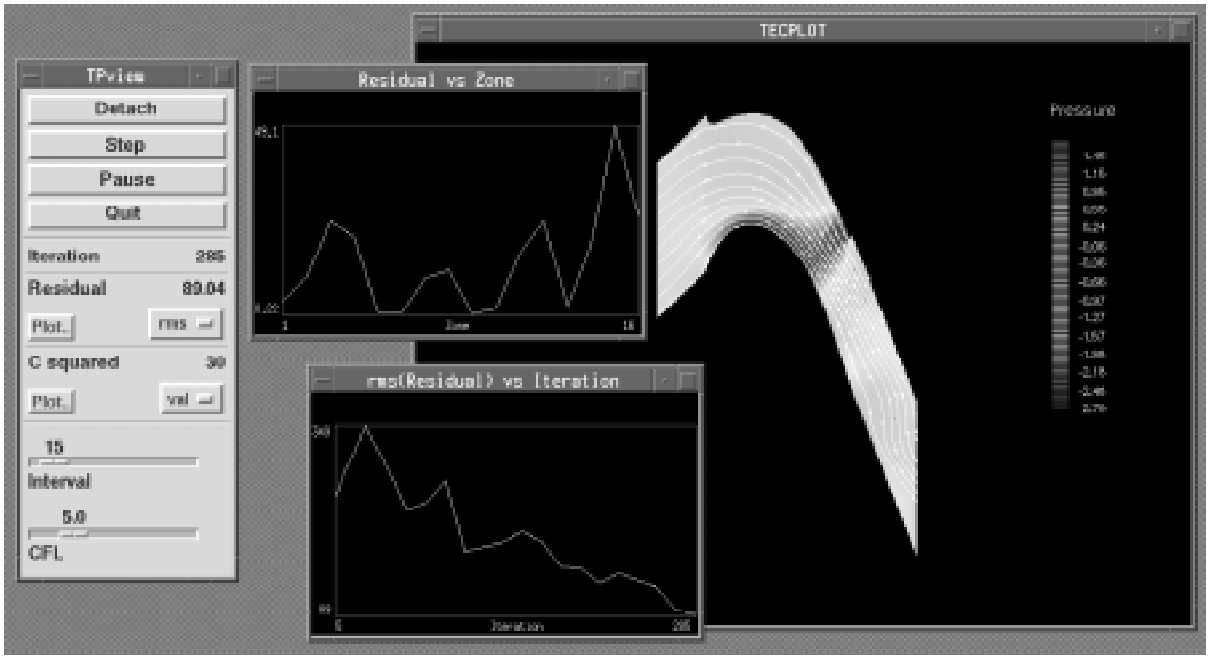


**Figure 4: TPview user interface**

When dealing with very large datasets, the time required by to render the image can become overwhelming, and it may therefore be preferable to consider the following options:

- reduce the volume of data to render by only displaying a particular area of interest, or by sending fewer dataset variables,
- reduce the complexity of the image to be rendered,
- increase the rendering speed by running on a more performant computer system,
- perform the rendering on the parallel system (subject to software availability),
- perform batch visualization.

## Future Improvements

Currently, communication internal to the T3D (i.e. between processors) is performed using PVM. It is desirable for the TPV library to be able to use the optimum communication protocol according to the hardware on which it is running. For this an MPI version is also envisaged, plus a socket version for communication within a network of workstations. A previous version of TPview, using PVM, has been ported to Silicon Graphics and Hewlett Packard workstations, in addition to the T3D.

Further experimentation with different algorithms for the choice of the sending and distributing processors may be useful. The TPV library user should be provided with a choice of methods in order to minimize the visualization overhead and thus optimize the overall performance of his simulation code.

The possibilities available in the user interface should be expanded to allow, for example, a user to manipulate a dataset (e.g. zoom, rotate) after having paused at a given image. Currently, once all the zones are defined and initialization has commenced, it is not possible to define new zones or send other datasets than that which is already being sent. We would like to be able to define collections of zones with a given number of variables, and to change the zones being viewed by the press of a button.

TPview, in its current form, is not designed to render very large datasets, or multiple frames per second. Parallel machines will always be able to produce more data than workstations can handle. To solve these more visualization intensive problems, we need to have the functionality and ease of use of Tecplot on the parallel computer system.

## Conclusions

It is considered that the on-line visualization tool, TPview, that has been developed fulfils the original design goals. It has minimal impact on the application code, in terms of both code modification and execution time, while being sufficiently flexible and easy to use. In particular, current users need acquire only minimal additional expertise to employ TPview.

The concept of distributed visualization described in this paper could be adapted in a relatively straightforward manner for other applications for which the datasets are not mesh based. For example, a molecular dynamics tool could be developed by replacing Tecplot with a visualization tool that renders molecules.

While TPview is not designed to be the optimal tool for all visualization needs, experience to date indicates that it should be very useful for a wide range of scientific visualization. In particular, it has been found to reduce considerably the inherent difficulty in the analysis of large datasets generated by numerical simulations on a high-performance parallel computer system.

## Acknowledgements

This study was undertaken within the framework of the Cray Research – EPFL Parallel Application Technology Program, and was performed using the T3D system at the EPFL.

Information on this project can be obtained at the URL:
http://imhefwww.epfl.ch/lmf/software/tpview.html

## References

[1] J.S. Rowlan, B.T. Wightman, *PORTAL: A communication library for run-time visualization of distributed, asynchronous data*, Proceedings of the Scalable High-Performance Computing Conference (Knoxville, May 1994), pp. 350-356.

[2] *SunOS 5.3 Network Interfaces Programmer's Guide*, Chapter 8, Sun Microsystems, Inc., November 1993.

[3] R. Haimes, *pV3: A distributed system for large-scale unsteady CFD visualization*, AIAA Paper 94-0321 (Reno, January 1994).

[4] S. Williams, M.L. Sawley, D. Cobut, *On-line visualization for scientific applications on the T3D*, EPFL Supercomputing Review, **7**, Nov. 1995, pp. 45-50.

[5] C. Kirchhof, *Cray Animation Theater*, Proceedings of the 34th Cray Users Group (Tours, October 1994), pp. 114-118.

[6] *Tecplot User's Manual, Version 6*, Amtec Engineering, Inc., 1994.

[7] *Modular Visualization Environments: Past, Present and Future*, Computer Graphics, **29**, No.2, May 1995.

[8] O. Byrde, D. Cobut, J.-D. Reymond and M.L. Sawley, *Parallel multi-block computation of incompressible flows for industrial applications*, Proceedings of Parallel CFD '95 (Pasadena, June 1995).