Patrick Brockmann <Patrick.Brockmann@cea.fr>
20/12/2004
Release 1.3

Introduction

The application VTK_Mapper described in the following lines is a prototype of an application made during the PRISM project. The initial goal was to demonstrate the high potential in designing applications using the VTK library to handle the visualization, the CDMS/COCO libraries to access and process netCDF files and the Qt library to build handy and nice user interface.

Designing and implementing such a prototype is helpful in insuring that essential features like 2D/3D rendering, vector-based quality output, batch and offscreen running mode are fully covered. Such a application is then suitable for both Low End environment use (scripting, batch and offscreen modes) and High End use (high interactivity).

Software architecture

All the code of the VTK_Mapper application has been written in the Python language which is a portable open source scripting language. This powerful programming language can be extended with compiled modules implemented in C/C++ or FORTRAN. To extend and bring higher visualization capabilities to the CDAT/VCS system, the Visualization ToolKit (VTK) has been investigated.

The Visualization ToolKit (VTK) is a highly referenced C++ library in the graphics/visualization/imaging domain. VTK consists of an API (Application Programming Interface) with more than 700 C++ classes implented with more than 350,000 lines of C++ code (110,000 executable lines) and with more than 215,000 lines of automatically generated Python wrapper code. VTK supports a wide variety of visualization algorithms including scalar, vector, tensor, texture, and volumetric methods; and advanced modeling techniques such as implicit modelling, polygon reduction, mesh smoothing, cutting, contouring, and Delaunay triangulation. In addition, dozens of imaging algorithms have been directly integrated to allow the user to mix 2D imaging / 3D graphics algorithms and data. The design and implementation of the library has been strongly influenced by object-oriented principles. VTK has been installed and tested on nearly every Unix-based platform, PCs (Windows 98/ME/NT/2000/XP), and Mac OSX.

To read the netCDF files and access to all the metadata informations, the CDMS Python module has been employed. Its capacities to get the grid and the mesh from a variable stored in a netCDF file have been greatly helpful. The management of masked values recovered from the different possible combinaisons of masks (ocean/land mask and the variable mask itself) has been made possible through the use of the CDMS/MV Python module.

To design a user interface, the Qt library and its full set of GUI (Graphical User Interface) controls has been prefered to other libraries also explored. Qt is a platform-independent set of C++ classes that can be freely used in open source projects. It also comes with development tools such as Qt Designer to visually build your application. Technically, the VTK_Mapper application has been implemented with calls to the Python bindings of the Qt toolkit, module called PyQt.

Covered features

With the software architecture used in the VTK_Mapper application, you can:

- Read netCDF files using Climate and Forecast (CF) or COARDS convention.

By using the CDMS module, the reading of variables from netCDF files and the recovery of metadata informations have been easily managed. The Climate and Forecast (CF) convention used to store grid information of rectilinear, curvilinear and generic models and the corresponding GetGrid() and GetMesh() methods from the CDMS module have been particulary useful. CDMS also offers backward compliance for netCDF files using the COARDS convention. Convertions from one grid type to another are easily implemented through calls to the toCurveGrid() and the toGenericGrid() methods.
To let a user explore repositories of model output with conventions older than the CF convention, an external gridfile is written containing all information of the grid which could be missing in models output files. The VTK_Mapper is then a really flexible and backward compliant application.

- Read remote netCDF files served by an OPeNDAP/DODS server.

By using the CDMS module compiled with the OPeNDAP netCDF API rather than the usual netCDF API, it becomes possible to read and access part of a remote netCDF file. It opens a virtually unlimited world access to models or data output repositories served by OPeNDAP/DODS servers. An IPSL OPeNDAP/DODS server has been set during the PRISM project to serve and share IPCC simulations output computed from the IPSL coupled model. It has also been used to test netCDF files from OASIS coupler and particulary their full compliance to the netCDF CF convention.

- Read CDML files (collections of netCDF files).

By using the CDMS module and the Climate Data Markup Language (CDML), you can aggregate split netCDF files and see them combined as a single dataset. CDML files are generated by the cdscan command provided with the CDAT distribution.

- Render 2D/3D objects.

By using the VTK API, it is possible to render 2D and 3D complex data structures. An orthographic projection has been computed to produce a 3D scene and a linear (plate caree) projection to produce a 2D scene. Interactive zoom and translation with easy mouse controls are the proposed features of the VTK_Mapper application to get interactivity.

The user can also interactively switch between projections passing from a 2D to a 3D rendering.

- Render large objects with appropriate level-of-details.

By using the VTK API and the use of level-of-details objects, the application can achieve acceptable rendering performance at the cost of lower-resolution representation. This is particulary useful during motions when the application renders large objects to maintain interactive frame rates.

- Produce isocontours or polygons maps.

By using the VTK API and the marching squares algorithm, it is possible to generate isocontours from generic, curvilinear and of course rectilinear grids. Switching from an isocontour rendering to a polygon map rendering is as easy as pressing a key. By this interaction, the user avoids the long cycle of changing parameters/launching application.
Additional controls to pass from a vertical level to another or from a time level to another are proposed with the VTK_Mapper application. The rendered modes (isofilled, cells, cellsbounds, isolines1, isolines2) are updated following key events.
Isocontours are generated with respect to the original topology of cells boundaries; other isocontour algorithms using cell centers have been investigated in particular Delaunay triangulation.

- Handle any type of model grid.

By using the VTK API, you can handle structured (uniform rectilinear, non-uniform rectilinear, and curvilinear grids), unstructured, polygonal and image data. The different dataset structures proposed by the VTK API cover all the needs for building a visualization application taking care to represent correctly topology and connectivity. The different model grids have been represented with the use of the vtkPolydata dataset type. This unstructured dataset type requires an explicit description of cells and points from the model grid. The connectivity is then dynamically computed with the use of a vtkCleanPolyData filter to join cells with shared boundary points.

- Probe variable values.

By using the VTK API, the user can focus on a particular zone and also probe values from the field displayed. This feature is particulary helpful when model codes are in a beta stage and when the user expects to examine the model output at its real and computed form.

- Write raster output.

By using the VTK API, it is possible to create raster images from the displayed window in different formats: Windows Bitmap (*.bmp), JPEG Images (*.jpg), PNG Images (*.png), Binary PPM (*.ppm), TIFF images (*.tif). The PNG image format has been chosen in the VTK_Mapper application because it is a recommended open source true lossless format.

- Write vector-based output.

By using the VTK API and the vtkGL2PSExporter class, it becomes possible to save rendered objects in a high quality vector PostScript (PS/EPS) or PDF file. This class uses the

GL2PS API to translate the OpenGL scene to vector format. It has some limitations since the PostScript is not an ideal language to represent complex 3D scenes but you can generate high quality vector PostScript with simple 3D scenes and most 2D plots. Thus, with a simple keypress, the VTK_Mapper application offers the user, generation of a PDF file.

- Run in batch and offscreen mode.

By using the VTK and Mesa libraries, it becomes possible to render a OpenGL scene in memory, without using hardware capacities of a graphic card. Thus, the VTK_Mapper application can produce a PDF file and a PNG file without any open window on your display or Xserver running. This feature effectively enables you to work off-line in a batch-oriented environment.

- Automate mass-production documents.

By using Python scripts and calls to the VTK_Mapper application expressed as a single line commands, you can mass produce documents. All interactive actions of the VTK_Mapper application can be retrieved as options in a unix-like command.

- Use a high level interface.

By using the Qt API, it has been possible to develop a very high level user interface. Many controls are possible with use of graphical and powerful widgets. It include a color control dialog, grid text layout to present the different variables and their attributes from the netCDF file loaded, control sliders, file selection dialogs and many others.

- Process data.

By using the COCO Python extension to CDMS API, it will be possible to process data easily. This feature is for now in a beta stage since there a small incompatibility with COCO and the CDMS generic grid structures. For now, processing is made by a simple evalution of a Python expression.

Examples of use

In the following examples, all the files arguments can be either a local file or a remote file served for example from the IPSL OPeNDAP/DODS server.
Download files from http://dods.ipsl.jussieu.fr/prism/gridsCF and access the http://dods.ipsl.jussieu.fr/fast/atlas/2L27_SE_2030_2039_output/ to find example files used.

But it will be much easier to use the remote access capability by using the following syntax:
http://dods.ipsl.jussieu.fr/cgi-bin/nph-dods/dir1/dir2/file.nc

....

```
$ mapper.py -v -p orthographic -x 2L27_SE_2030_2039_histmth.nc tsol
```

```
$ mapper.py -v -p orthographic -x sampleCurveGrid4.nc sample
```

With a remote access, this would read:

```
 $ mapper.py -v -p orthographic-x http://dods.ipsl.jussieu.fr/cgi-bin/nph-dods/prism/gridsCF/sampleCurveGrid4.nc sample
```

```
$ mapper.py -l 100:2500:100 -v -p orthographic -x sampleGenGrid3.nc sample
```

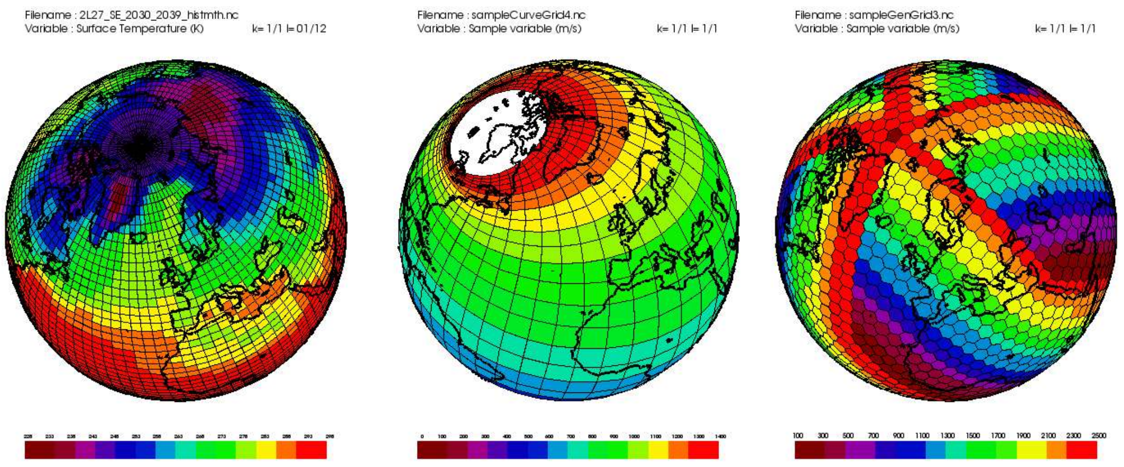

Filename : 2L27_SE_2030_2039_histmth.nc
Variable : Surface Temperature (K)
k= 1/1 l= 01/12
Filename : sampleCurveGrid4.nc
Variable : Sample variable (m/s)
k= 1/1 l= 1/1
Filename : sampleGenGrid3.nc
Variable : Sample variable (m/s)
k= 1/1 l= 1/1
100 300 500 700 900 1100 1300 1500 1700 1900 2100 2300 2500

Figure 1a,1b,1c: Different model grids: rectilinear, curvilinear, generic.

....

```
$ mapper.py -v --color color1.sty 2L27_SE_2030_2039_histmth.nc precip
```

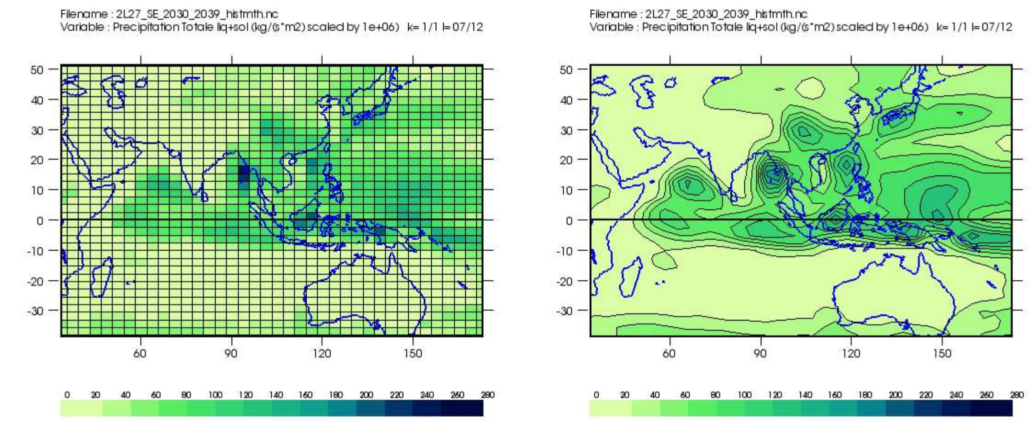

Filename : 2L27_SE_2030_2039_histmth.nc
Variable : Precipitation Totale liq+sol (kg/(s*m2) scaled by 1e+06) k= 1/1 l= 07/12

Figure 2a,2b: Switching between representation mode: cellsbounds, isocontours in a linear projection with an atmospheric model output (rectilinear grid).

....
$ mapper.py -v --operation 'var-273.15' -l -40:40:5 2L27_SE_2030_2039_output/2L27_SE_2030_2039_histmth.nc t2m

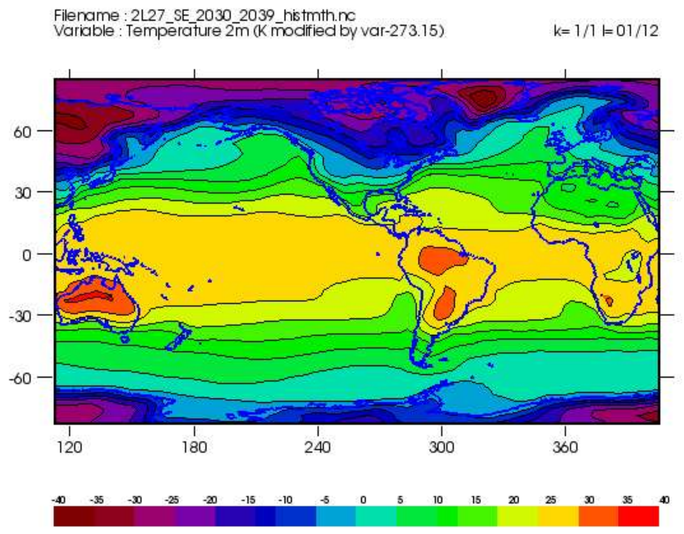

Filename : 2L27_SE_2030_2039_histmth.nc
Variable : Temperature 2m (K modified by var-273.15)
k= 1/1 l= 01/12

Figure 3: Application of an operation to an atmospheric variable in an isocontour mode in a linear projection.

....
*$ mapper.py -v --levels -2:18:1 --lindex 9 --color color3.sty --gridfile IPSL.ORCA2_gridCF.nc --projection orthographic 2L27_SE_2030_2039_grid_T.nc votempe*r

If remote access is used, this example becomes:
$ mapper.py -v --levels -2:18:1 --lindex 9 --color color3.sty --gridfile

http://dods.ipsl.jussieu.fr/cgi-bin/nph-dods/prism/gridsCF/IPSL.ORCA2_gridCF.nc --projection orthographic http://dods.ipsl.jussieu.fr/cgi-bin/nph-dods/fast/atlas/2L27_SE_2030_2039_output/2L27_SE_2030_2039_grid_T.nc votemper

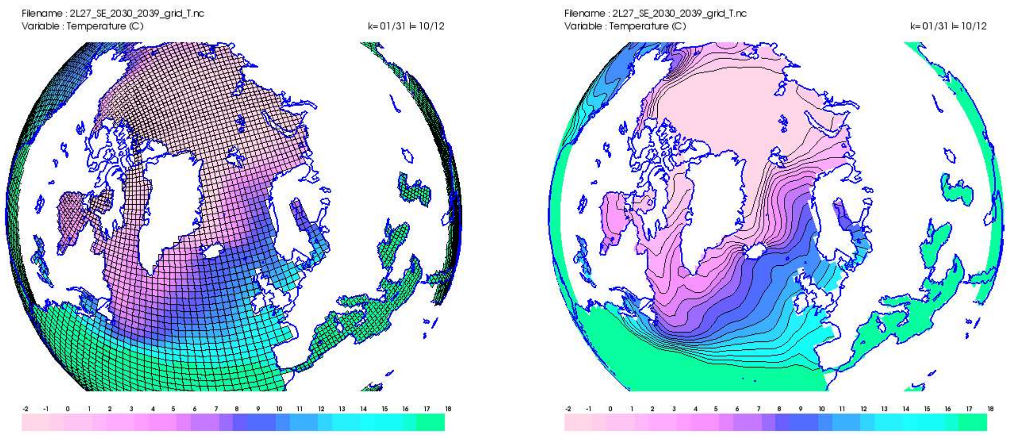
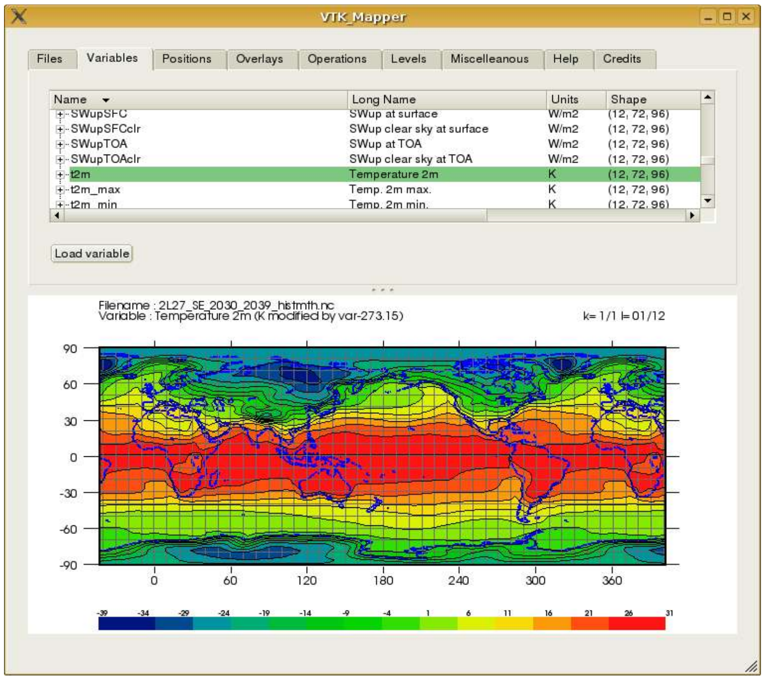


Figure 4a,4b: Switching between representation mode: cellsbounds, isocontours in an orthographic projection with an ocean model output (curvilinear grid).

....

$ mapper.py -x --actor isofill --camera camera4.sty --color color4.sty --continents --continents_color 0.0,0.0,1.0 --continents_file ./polydouble_earth_continents.nc --continents_width 2 --equator --equator_color 0.0,0.0,0.0 --equator_width 2 --grid --grid_color 0.45,0.45,0.45 --grid_delta 10 --grid_width 1 --levels_nb 15 --operation 'var-273.15' 2L27_SE_2030_2039_histmth.nc t2m



Name	Long Name	Units	Shape
SWupSFC	SWup at surface	W/m2	(12, 72, 96)
SWupSFCclr	SWup clear sky at surface	W/m2	(12, 72, 96)
SWupTOA	SWup at TOA	W/m2	(12, 72, 96)
SWupTOAclr	SWup clear sky at TOA	W/m2	(12, 72, 96)
t2m	Temperature 2m	K	(12, 72, 96)
t2m_max	Temp. 2m max.	K	(12, 72, 96)
t2m_min	Temp. 2m min.	K	(12, 72, 96)

Filename : 2L27_SE_2030_2039_histmth.nc
Variable : Temperature 2m (K modified by var-273.15)
k= 1/1 l= 01/12

Figure 5: The Qt user interface and the corresponding command to re-produce the map.

....

```
$ mapper.py -v --offscreen -l 100:1000:50 --equator –continents
2L27_SE_2030_2039_histmth.nc rhum
```

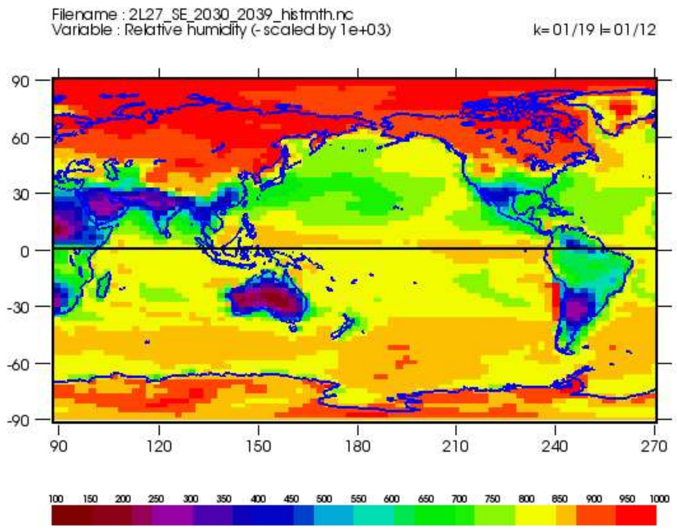


Figure 6: An offscreen call to produce a pdf file and a png file with a cells visualization mode.

Usage documentation

```
#########################################################
Usage: mapper.py [-h]
           [-p projection] [-a actor]
           [-n levels_nb] [-l min:max:delta]
           [--bg r,g,b] [--fg r,g,b]
           [--camera camera_object_file] [--color color_object_file]
           [--kindex index] [--lindex index]
           [--continents] [--continents_file file] [--continents_color r,g,b] [--continents_width width]
           [--boundaries] [--boundaries_color r,g,b] [--boundaries_width width]
           [--equator] [--equator_color r,g,b] [--equator_width width]
           [--grid] [--grid_color r,g,b] [--grid_width width] [--grid_delta delta]
           [--verbose] [--prefix prefixfilename] [--offscreen] [-x] [--interface]
           [--gridfile gridCF_file]
            var_file var
#########################################################
Options:
       -h, -?, --help, -help
                 Print this manual
         -x, --interface
                 Run the application with the GUI interface
       -p, --projection
                 Projection to choose in (linear,orthographic)
       -a, --actor
                 Actor to choose in :
                          isofill,cell,cellbounds,isoline1,isoline2
                 Other accepted syntax are:
                          isofilled,cells,,cellsbounds,isolines1,isolines2
       -v, --verbose
                 Verbose mode
       -n, --levels_nb
                 Number of levels should be in [3:100]
       -l, --levels
                 Levels expressed as minimum:maximum:delta
                 Example:      -l 2:32:4 from 2 to 32 by step of 4
                               -l 0:0:4 from min to max by step of 4
         --bg, --background
                 Background color expressed as red, green, blue values in [0:1]
                 Example: --bg 0.3,0.3,0.3
         --fg, --foreground
                 Foreground color expressed as red, green, blue values in [0:1]
                 Example: --fg 1.0,1.0,1.0
         --op, --operation
                 Operation to apply on variable (use quote)
```

Example: --op 'var*86400'
--op '(var*100)+273.15'

--camera
    Camera object file
--color
    Color object file
--kindex
    Index for the 3th dimension (vertical axis)
    of the variable to plot [1:n]
--lindex
    Index for the 4th dimension (time axis)
    of the variable to plot [1:n]
--boundaries, --continents, --equator, --grid
    Drawn if this option is present
--boundaries_color, --continents_color, --equator_color, --grid_color
    Color expressed as red, green, blue values in [0:1]
  Example: --boundaries_color 0.,0.,0.3
--boundaries_width, --continents_width, --equator_width, --grid_width
    Lines width expressed in [1:5]
--continents_file
    NetCDF continents file (CONT_LON,CONT_LAT variables)
--grid_delta
    Delta for grid lines (default=30)
--prefix
    Filename prefix used when PNG and PDF file
    are saved (default=picture)
--gridfile
    NetCDF file at the CF convention from where the mesh is read.
    If present, the "mask" variable is read and used in combinaison
    with the mask deduced from the variable.
    If gridfile not present, use only self descriptions of the variable.
--ratioxy
    Set the ratio between height and width
    for linear projection (default=1.0)
--offscreen
    Produce a PNG and a PDF file in a offscreen mode

Installation

You can download all the material from http://dods.ipsl.jussieu.fr/vtk/VTK_Mapper and follow the instructions you will find there.

Conclusion and Future Work

The initial set of defined goals has been covered. This confirms the strong advantages to build applications over the explored software architecture. Designing and implenting applications efforts will be continued in this way.

It has to be noted that the open source Paraview will be an excellent confirmation for the use of VTK since Paraview also uses this toolkit as the data processing and rendering engine and is a major keystone for large visualisation projects.
For information, the open source project Paraview is presented as follows:
"Paraview is an application designed with the need to visualize large data sets in mind. The goals of the ParaView project include the following:
- Develop an open-source, multi-platform visualization application.
- Support distributed computation models to process large data sets.
- Create an open, flexible, and intuitive user interface.
- Develop an extensible architecture based on open standards.

ParaView runs on distributed and shared memory parallel as well as single processor systems and has been succesfully tested on Windows, Linux and various Unix workstations and clusters. Under the hood, ParaView uses the Visualization Toolkit as the data processing and rendering engine.
ParaView is being developed by Kitware in conjunction with the Advanced Computing Laboratory at Los Alamos National Laboratory. ParaView is funded by the US Department of Energy ASCI Views program as part of a three-year contract awarded to Kitware, Inc. by a consortium of three National Labs - Los Alamos, Sandia, and Livermore. The goal of the project is to develop scalable parallel processing tools with an emphasis on distributed memory implementations."

Links to related materials mentionned

- VTK_Mapper application
  http://dods.ipsl.jussieu.fr/vtk/VTK_Mapper
- VTK toolkit
  http://www.vtk.org
- CDAT/CDMS (Climate Data Management System)
  http://esg.llnl.gov/cdat
- COCO (CDMS overloaded for CF Objects)
  http://prism.enes.org/WPs/WP4a/ProcessingLib
- Mesa for offscreen rendering
  http://www.mesa3d.org/
- PyQt (Python bindings for Qt) for the interface of the application
  http://www.trolltech.com/qt and http://www.riverbankcomputing.co.uk/pyqt
- Paraview
  http://www.paraview.org