INL/EXT-12-26956

MarmotViz User Guide

Alexander Rattner
Donna Post Guillen
Srinivas Garimella
Alark Joshi

August 2012

The INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance

INL/EXT-12-26956

MarmotViz User Guide

Alexander Rattner
Donna Post Guillen
Srinivas Garimella
Alark Joshi

August 2012

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Through the INL LDRD Program
Under DOE Idaho Operations Office
Contracts: DE-FG02-97ER25308 and DE-AC07-06ID14517

This page intentionally left blank.

ABSTRACT

MarmotViz is an illustrative visualization plug-in for ParaView. It is intended for generating enhanced visualizations of time-varying datasets on unstructured connected meshes. A detailed description of the implemented algorithms and program structure can be found in the related document: *Generalized framework and algorithms for illustrative visualization of time-varying data on unstructured meshes*[*]. This document provides details for compiling/building MarmotViz, using the MarmotViz plug-in in ParaView, and extending the MarmotViz plug-in.

[*] Rattner, A.S. Guillen, D.P. Garimella, S. Joshi, A. *Generalized framework and algorithms for illustrative visualization of time varying data on unstructured meshes*. Idaho National Laboratory Report INL/EXT-12-26809, July 2012.

CONTENTS

MarmotViz User Guide

1. Overview

MarmotViz is an illustrative visualization plug-in for ParaView. It enables users to identify and track time-varying *features* in simulation datasets. It permits the application of illustrative visualization effects to these features including: selective visualization, feature coloring, boundary smoothing, haloing, silhouette outlining, speedlines, and strobe silhouettes. These techniques serve to assist in exploration and interpretation of simulation data and can be used to generate enhanced renderings for presentations. The MarmotViz plug-in is developed as a flexible framework, and can be easily extended to incorporate new region-of-interest (ROI) identification algorithms, feature matching and tracking algorithms, and illustrative visualization effects.

2. Compiling and building MarmotViz

The distributed MarmotViz source code contains a top-level directory named `MarmotVizAll`. This directory contains versions of the MarmotViz plug-in for use on workstations with ParaView (`MarmotViz`) and in client/server environments, i.e. ParaView/pvserver (`MarmotVizCAVE`). MarmotViz was developed for use on GNU/Linux environments, and has not been fully tested on Windows or OS X systems.

Before building the MarmotViz plug-ins, ensure that you are operating in an environment with ParaView 3.14.1 installed, and the correct software and environment variables for building ParaView. If ParaView 3.14.1 can be built from source code on your working environment then the MarmotViz plug-in can also be built. The ParaView source code can be found on: http://paraview.org/paraview/resources/software.php, and compilation and building instructions can be found on http://paraview.org/Wiki/ParaView:Build_And_Install.

Once ParaView 3.14.1 has been installed, MarmotViz can be built. To build the workstation plug-in, run the `MarmotVizAll/MarmotViz/makePlugin.sh` script. To build the client/server plug-in, run the `MarmotVizAll/MarmotVizCAVE/makePlugin.sh` script.

3. Using MarmotViz on workstation environments

3.1 Loading the MarmotViz plug-in

To load MarmotViz in the workstation environment, open the *Tools* ➔ *Manage Plugins* menu in ParaView. Select *Load New* and select the `libMarmotViz.so` library file that was built in the `MarmotVizAll` directory. Users can enable the *Auto Load* option to automatically load MarmotViz during ParaView startup. Introductory guides to the usage of ParaView can be found on: http://paraview.org/paraview/resources/webinars.html.

3.2 Applying the MarmotViz filter to data

Once data has been loaded into ParaView pipeline, the MarmotViz filter can be applied to generate illustrative visualizations. Source data should be defined on a connected mesh (i.e. not on point clouds) made of 3D cells. Data fields should be defined on cells – point data fields are ignored by the MarmotViz filter. To apply the filter to a data source, select the data in the *Pipeline Browser*, and then select the *MarmotViz* filter in the *Filters* menu. If the filter has not been used recently, it may be necessary to search for it by name (*Filters*➔*Search*).

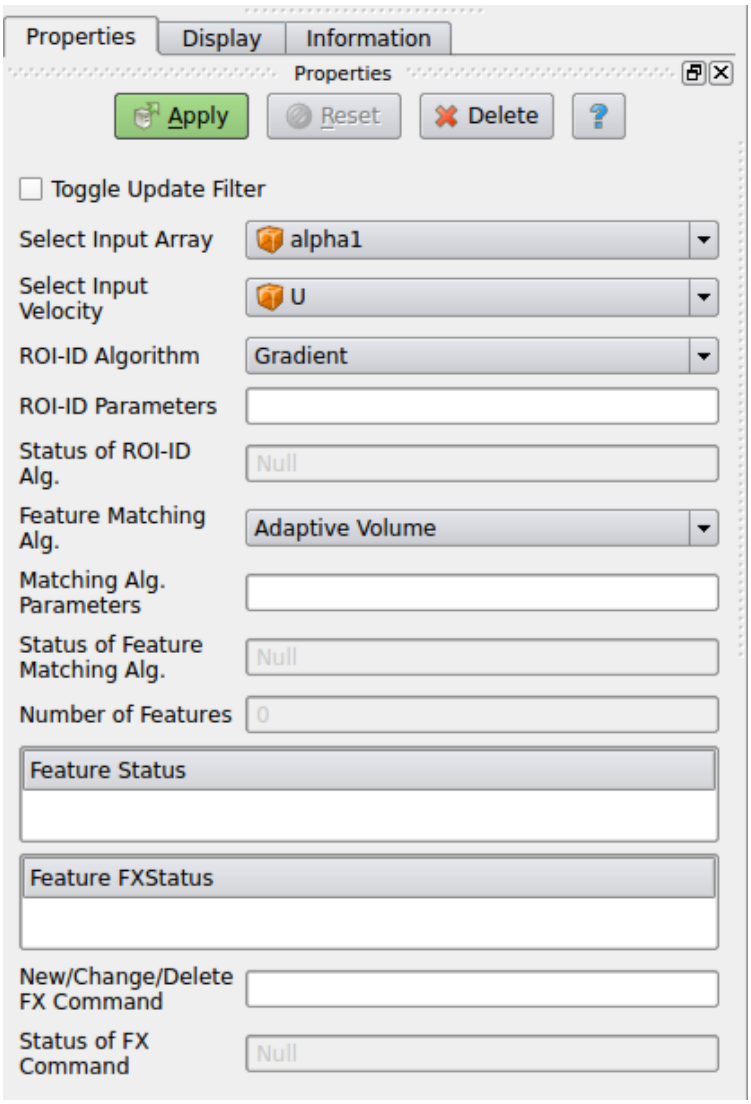

Figure 1 – MarmotViz filter GUI

3.3 Using the MarmotViz filter

The MarmotViz filter proceeds through three stages: region-of-interest (ROI) identification, feature matching, and application of illustrative visualization effects. Once the MarmotViz filter is applied to a dataset, users are presented with the interactive GUI (Fig. 1), which is used to control the filter through these stages. Descriptions of the functionality and use of the GUI elements are provided below.

- **Toggle Update Filter** – Toggling this checkbox informs the filter to update when *Apply* is pressed. This is typically used to regenerate illustrative effects when the view is changed.
- **Select Input Array** – This drop-down menu allows the user to select the input cell scalar field that will be used to identify ROIs in the input data.

- **Select Input Velocity** – This drop-down menu allows the user to select the vector field that will be used for velocity data during feature matching. If no vector field is available, feature matching will proceed without using velocity data.
- **ROI-ID Algorithm** – This drop-down menu allows the user to select the ROI identification algorithm that will be used. Currently, only the *Gradient* algorithm is available.
- **ROI-ID Parameters** – This text field is used to input comma-separated parameters for the ROI-ID algorithm.
- **Status of ROI-ID Alg.** – This textbox displays results from changing the ROI-ID parameters.
- **Feature Matching Alg.** – This drop-down menu allows the user to select the algorithm used for feature matching. Currently, only the *Adaptive Volume* algorithm is available.
- **Matching Alg. Parameters** – This text field is used to input comma-separated parameters for the feature matching algorithm.
- **Status of Feature Matching Alg.** – This textbox displays results from changing the feature-matching algorithm parameters.
- **Number of Features** – This textbox displays the number of identified features in the dataset.
- **Feature Status** – This listbox lists the identified features. Checkboxes are provided next to each feature, and unchecked features are removed from the output dataset.
- **Feature FXStatus** – This listbox lists defined feature effects. Users can toggle the checkboxes next to each effect to enable/disable the display of effects.
- **New/Change/Delete FX Command** – This text field is used to input commands to define new illustrative effects, modify existing effects, and delete effects.
    - To define a new command, provide an input string like: new,(feature number),(effect name),(comma separated effect parameters)
    - To change an existing effect, provide an input string like: change,(effect number),(comma separated effect parameters)
    - To delete an illustrative effect, provide an input string like: delete,(effect number)
- **Status of FX Command** – This textbox displays the result of the most recently applied FX command.

3.4 Common tasks in MarmotViz

Processes and guides for some common tasks in MarmotViz are described below.

3.4.1 Using the gradient-based ROI identification algorithm

The *Gradient* ROI identification algorithm finds contiguous regions in input meshes separated by regions of high gradient magnitude in the input scalar field. High gradient magnitude cells are skipped are removed from the output dataset. Additionally, the algorithm can be given a lower threshold for region cell count. Contiguous regions with fewer cells than this threshold are also removed from the output dataset.

To set the parameters for this algorithm provide input to the *ROI-ID Parameters* box as: (boundary gradient magnitude),(minimum region cell limit)

For example, the input:

```
4.5,8
```

sets the boundary gradient limit to 4.5, and minimum number of cells per region to 8.

3.4.2 Using the adaptive-volume feature matching algorithm

Feature matching algorithms are used to match identified regions at new time-steps with known features at previously evaluated time-steps. The *Adaptive Volume* feature-matching algorithm attempts to match ROIs with nearby features of similar volume using a matching criterion that relaxes over multiple passes. If velocity data is provided (in the *Select input velocity* menu), this algorithm can use this information to better track moving features. Feature matching and tracking is performed automatically as the user steps through time in ParaView. Identified features are added to the *Feature Status* list box, and feature cells are marked with the feature ID number in the output *RegionColors* field.

To set the parameters for this algorithm, provide input to the *Matching Alg. Parameters* box as: (initial similarity criterion),(relaxation parameter),(relaxation limit)

For example, the input:

```
0.7,0.8,0.3
```

sets the initial volume similarity matching criterion to 0.7, relaxes this criterion by 0.8 at each matching pass, and terminates matching if the matching criterion falls below 0.3. Unmatched ROIs become new features.

3.4.3 Selective feature visualization

Once the MarmotViz filter has been applied, features can be selectively visualized – allowing display of interior features or removal of blocking features. Boundary regions and any unchecked feature in the *Feature Status* listbox are automatically removed from the output dataset.

3.4.4 Feature smoothing illustrative effect

Once features have been identified in the MarmotViz filter, various illustrative effects can be applied. The feature-smoothing effect generates a smoothed bounding surface around features. Users can specify the number of smoothing passes (up to 3) applied by a built-in VTK subdivision filter (0 passes just applies boundary Delaunay triangulation).

To apply the smoothing effect to a feature, provide input to the *New/Change/Delete FX Command* box as:

new,(feature number),smoothboundary,(number of smoothing passes)

For example, the input:

```
new,10,smoothboundary,1
```

would apply a new smoothing filter to feature 10 with 1 subdivision smoothing pass.

If the newly created effect were assigned number 3 in the *Feature FXStatus* box, supplying the input:

```
change,3,2
```

would increase the number of smoothing passes to 2.

Similarly, supplying the input:

```
delete,3
```

would delete this effect (3).

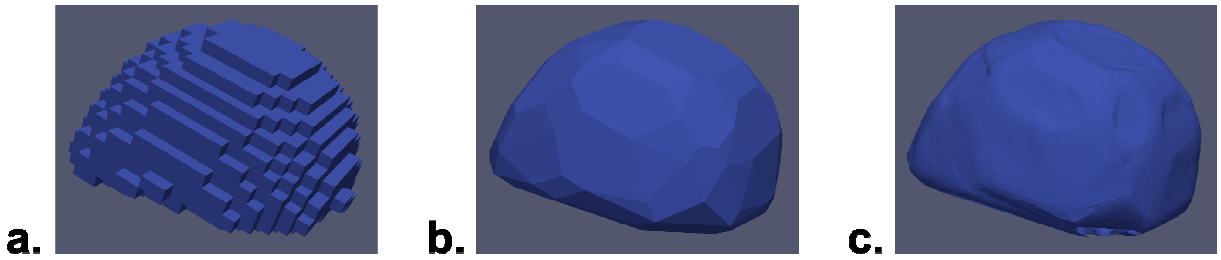

Figure 2 – Demonstration of smoothing illustrative effect. **a**. Original unsmoothed feature, **b**. Smoothed feature (triangulated, 0 smoothing passes), **c**. Feature with 2 smoothing passes

A demonstration of the feature-smoothing effect with 0, 1, and 2 smoothing passes is presented in Fig. 2.

3.4.5 Tube outline illustrative effect

The tube outline illustrative effect generates a tube contour around the boundary of an identified feature. This effect can be used to emphasize a particular feature, or clarify its boundaries.

To apply a tube outline to a specific feature, provide input to the *New/Change/Delete FX Command* box as:

New,(feature number),tubeoutline,(tube thickness),(tube intensity),(number of tube sides),(contour smoothing value)

For example, the input:

```
new,2,tubeoutline,0.002,5,10,0.2
```

would generate a tube outline around feature 2 with thickness 0.002, value 5 in the *RegionColors* field, 10 sides around each tube segment, and smooth the boundary contour to a factor of 0.2. This last parameter must range from >0 (fully smoothed) to 1 (relatively coarse outline).

Note that the generated tube is view dependent. If the camera position or view direction is changed, the tube outline can be regenerated by toggling the *Toggle Update Filter* checkbox, and pressing *Apply*.

A demonstration of the tube outline effect is presented in Fig. 3. The regeneration process for different camera angles is demonstrated in Figure 4

3.4.6 Feature halo illustrative effect

The feature halo effect generates a ribbon-like halo around the boundary of a feature. This can be used to emphasize features of interest and provide additional visual cues about the relative depths of features.

To apply the halo effect to a particular feature, provide input to the *New/Change/Delete FX Command* box as:

new,(feature number),halo,(offset thickness),(inset thickness),(halo intensity),(smoothing factor)

For example, the input:

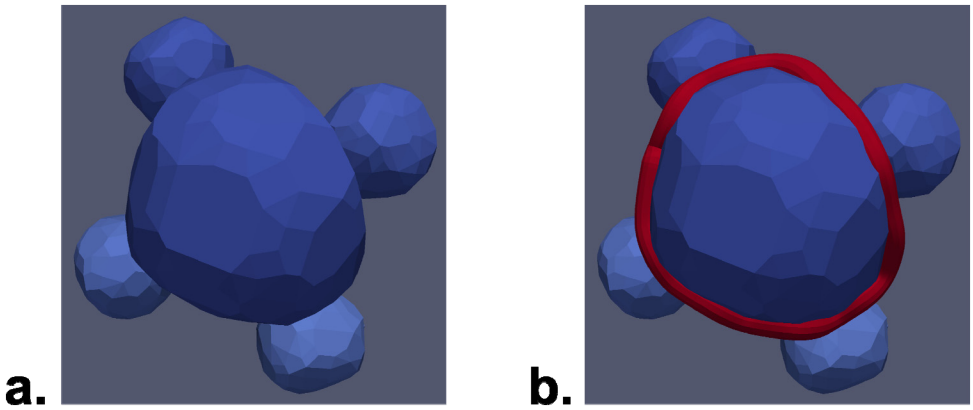

Figure 3 – Demonstration of tube outline illustrative effect to clarify the borders of a feature that blends in with background features. **a**. Original visualization, **b**. With tube outline enhancement

```
new,5,halo,0.002,0.0005,8,0.2
```

would generate a halo around feature 5 with offset thickness 0.002, inset thickness 0.0005, intensity 8 in the *RegionColors* field, and contour smoothing factor of 0.2. As with the tube outline effect, the halo can be regenerated for a new camera view by toggling the *Toggle Update Filter* checkbox.

A demonstration of the feature halo effect is presented in Fig. 5.

3.4.7 Speedlines illustrative effect

The speedlines illustrative effect generates cones on the trailing edge of a moving feature to indicate its dynamic behavior. This effect requires that a velocity field be provided in the *Select Input Velocity* drop-down menu.

To apply the speedlines effect to a particular feature, provide input to the *New/Change/Delete FX Command* box as:

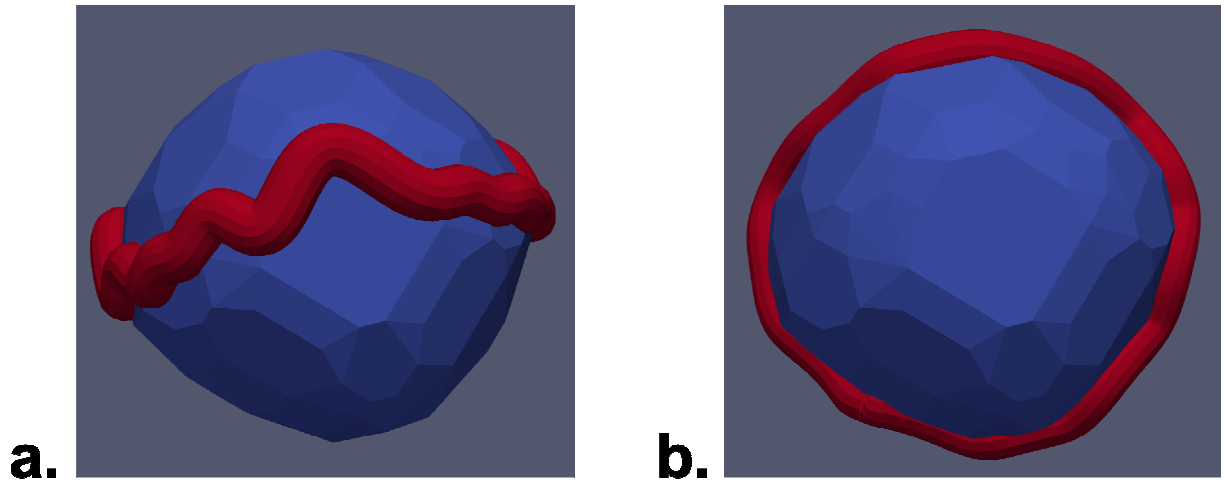

Figure 4 – When the view orientation changes, generated illustrative effects may no longer appear valid (**a**) and need to be regenerated (**b**)

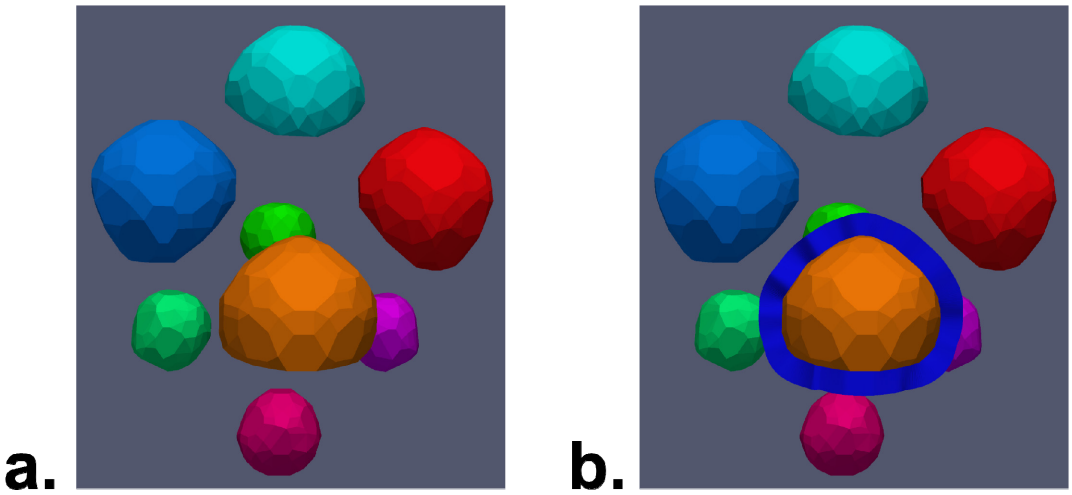

Figure 5 – Demonstration of the feature halo illustrative effect. In cases where relative positions of features may be unclear (**a**), feature halos can provide additional depth cues (**b**)

new,(feature number),speedlines,(speedlines intensity),(number of speedlines),(cone base radius),(time change for cone length),(number of facets per cone)

For example, the input:

```
new,4,speedlines,8,6,0.001,0.005,12
```

would generate 6 speedlines along the trailing edge of feature 4, with intensity 8 in the *RegionColors* field, cone base radii of 0.001, axial length of 0.005×*velocity*, and 12 facets per cone. As before, the speedlines can be regenerated for a new camera view by toggling the *Toggle Update Filter* checkbox.

A demonstration of the speedlines effect is presented in Fig. 6.

3.4.8 Strobe silhouettes illustrative effect

The strobe silhouettes illustrative effect generates multiple tube curves offset from the

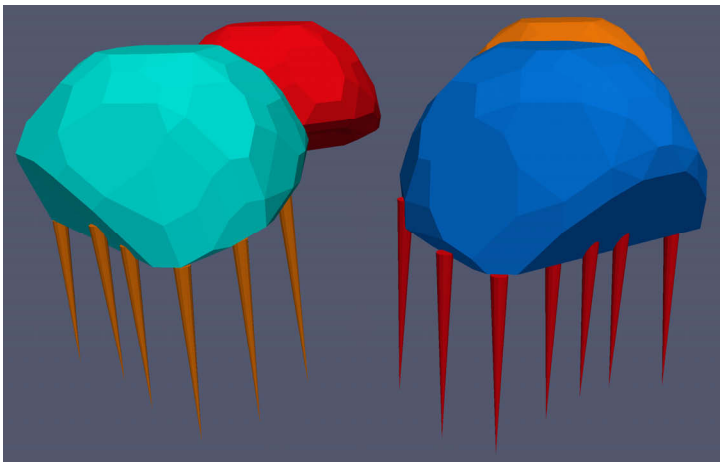

Figure 6 – Demonstration of the speedlines illustrative effect

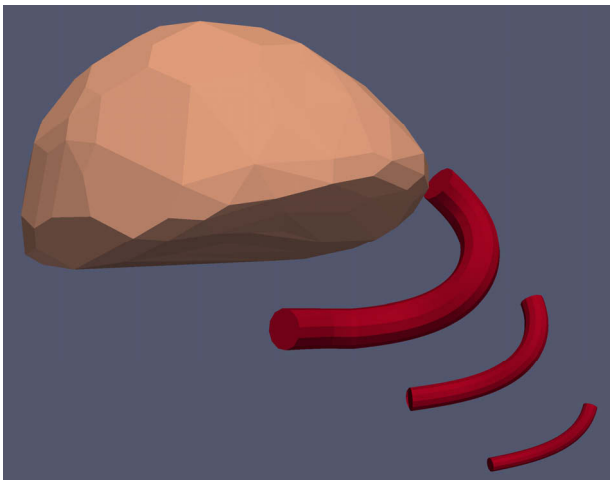

Figure 7 – Demonstration of the strobe silhouettes illustrative effect

trailing edge of a feature. As in the speedlines effect, this can be used to indicate the motion of a feature. This effect also requires that a velocity field be provided in the *Select Input Velocity* drop-down menu.

To apply the strobe silhouette effect to a particular feature, provide input to the *New/Change/Delete FX Command* box as:

new,(feature number),strobesilhouette,(initial strobe thickness),(strobe intensity),(number of facets per tube segment),(number of offset strobes),(time change for strobe positions),(relative reduction between strobes),(smoothing factor for strobes)

For example, the input:

```
new,0,strobesilhouette,0.002,6,12,3,0.001,0.75,0.1
```

would generate strobe silhouettes for feature 0, with a thickness of 0.002 on the first strobe, intensity 6 in the *RegionColors* field, 12 sides per tube segment, 3 strobes offset from the feature, spacing between strobes equal to 0.001×*velocity*, each strobe reduced by 25% (1-0.75) in length and diameter relative to the preceding strobe, and the strobe contours smoothed to a factor of 0.1. As before, the strobe silhouettes can be regenerated for a new camera view by toggling the *Toggle Update Filter* checkbox.

A demonstration of the strobe silhouettes effect is presented in Fig. 7.

4. Using MarmotViz in client/server environments

4.1 Loading the MarmotVizCAVE plug-in

First, launch ParaView and connect to a server session (pvserver). Ensure that the MarmotVizCAVE plugin is loaded in the *Tools* ➔ *Plugin Manager* menu on both the client and server sides. Introductory material on the usage of ParaView in client/server environments can be found here: http://paraview.org/Wiki/Setting_up_a_ParaView_Server and here: http://www.visualization.hpc.mil/wiki/Paraview_Client-Server_Mode.

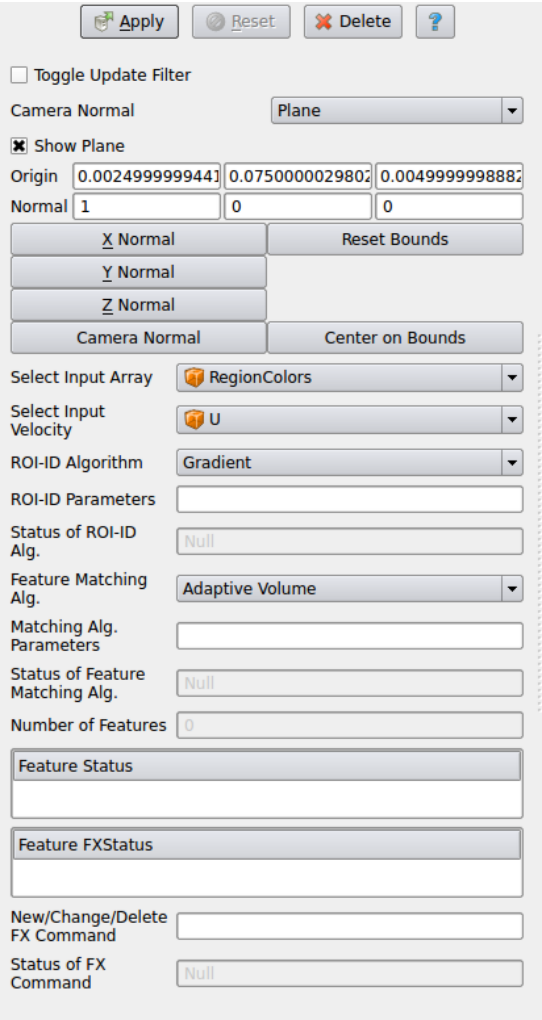

Figure 8 – MarmotVizCAVE filter GUI

4.2 Using the MarmotVizCAVE filter

Usage of the MarmotVizCAVE filter is similar to that of the workstation MarmotViz filter. The primary difference is that the MarmotVizCAVE filter cannot automatically acquire camera orientation data. The filter GUI panel is extended (Fig. 8) to allow the user to supply the camera orientation information to the filter (in the *Normal* fields) for generation of illustrative effects. The *Camera Normal* button automatically populates the *Normal* fields with the current camera view information. The *Origin* field has no effect on the filter behavior, and the *Show Plane* check box can be toggled to show or hide the camera projection plane with no effect on the MarmotVizCAVE filter. Note that the filter must be triggered to update illustrative effects for a new camera orientation (i.e. by toggling the *Toggle Update Filter* checkbox).

5. Extending the MarmotViz functionality

The MarmotViz and MarmotVizCAVE filters are intended to be generalized and extensible frameworks for illustrative visualization. As, such it is relatively simple to develop new ROI

identification algorithms, feature matching algorithms, and illustrative effects and incorporate them into these filters.

5.1 Developing new ROI identification algorithms

In the MarmotViz framework, ROI identification algorithms are called once for each visited simulation time-step. New ROI identification algorithms should extend the asrROIID_Base class found in the `SharedCode` directory. The derived class should implement:

- Constructor – call the parent class constructor and set default values for algorithm parameters.
- `const char * GetName()` – This returns the name of the algorithm.
- `void SetParameters ( vector< std::string > &Parameters, std::string &ChangeROIIDStatus)` – This sets parameters for the algorithm given a vector of input strings, and returns a result string
- `vector<asrFeatureInstant*>* FindROIs( vtkDataSet *DataSet, double Time)` – This algorithm receives an input dataset and the time value, and returns a pointer to a vector of pointers to identified ROIs (`asrFeatureInstant` objects).

Once the ROI identification algorithm class has been implemented, it must be incorporated into the following places in the source code:

1. Add an option for the new ROI identification algorithm in the `ROIIDAlgorithm` dropdown menu in the `MarmotViz/MarmotViz.xml` and `MarmotVizCAVE/MarmotVizCAVE.xml` files.
2. Add an include statement for the new ROI identification algorithm header file in the tops of `MarmotViz/vtkMarmotViz.cxx` and `MarmotVizCAVE/vtkMarmotVizCAVE.cxx` files.
3. Add a construction option in the `SetROIIDAlg(const int AlgID)` function in the `MarmotViz/vtkMarmotViz.cxx` and `MarmotVizCAVE/vtkMarmotVizCAVE.cxx` files. You may also want to prepend field names to the command string as is done for the *Gradient* algorithm in this function.
4. Add the new .cxx file name to the `OTHER_SRC` variables in the `CMakeLists.txt` files in the `MarmotViz` and `MarmotVizCAVE` directories.

The MarmotViz and MarmotVizCAVE filters can now be rebuilt with the new ROI identification algorithm following the steps from Section 2.

5.2 Developing new feature matching algorithms

To develop a new feature-matching algorithm, extend the `asrMatch_Base` class found in the `SharedCode` directory. The derived class should implement the following functions:

- Constructor – call the parent class constructor and set default values for algorithm parameters.
- `const char * GetName()` – This returns the name of the algorithm.

- `void SetParameters ( vector< std::string > &Parameters, std::string &ChangeMatchingStatus)` – This sets parameters for the algorithm given a vector of input strings, and returns a result string
- `void MatchFeatures ( vector<asrFeatureInstant*> &FeatureInstants, double Time)` – This algorithm receives a vector of pointers to new ROIs at a supplied time, and matches them to the internal vector of features in the object.

Once the feature-matching algorithm class has been implemented, it must be incorporated into the following places in the source code:

1. Add an option for the new feature-matching algorithm in the `MatchingAlgorithm` dropdown menu in the `MarmotViz/MarmotViz.xml` and `MarmotVizCAVE/MarmotVizCAVE.xml` files.
2. Add an include statement for the new feature-matching algorithm header file in the tops of `MarmotViz/vtkMarmotViz.cxx` and `MarmotVizCAVE/vtkMarmotVizCAVE.cxx` files.
3. Add a construction option in the `SetMatchingAlg(const int AlgID)` function in the `MarmotViz/vtkMarmotViz.cxx` and `MarmotVizCAVE/vtkMarmotVizCAVE.cxx` files.
4. Add the new .cxx file name to the `OTHER_SRC` variables in the `CMakeLists.txt` files in the `MarmotViz` and `MarmotVizCAVE` directories

The MarmotViz and MarmotVizCAVE filters can now be rebuilt with the new feature-matching algorithm following the steps from Section 2.

5.3 Developing new illustrative visualization effects

To develop a new illustrative visualization effect, extend the `asrIllustrativeEffect` class in the `SharedCode` directory. The new effect should implement the following functions:

- Constructor – call the parent class constructor and set default values for algorithm parameters.
- `const char * GetName()` – This returns the name of the illustrative effect. Note that the class name should be short to avoid overruns in the ParaView GUI.
- `void SetParameters ( vector< std::string > &Parameters, std::string &ChangeFXStatus)` – This sets parameters for the illustrative effect given a vector of input strings, and returns a result string
- `void ApplyEffect( UnstructuredGeometryData &Geometry, vtkDataArray* OutputArray, double Time)` – This algorithm receives an input `Geometry` object (defined in `asrIllustrativeEffect.h`), the time value, and the output data field. It should add any new effect geometry elements to the `Geometry` object and modify the output field appropriately.

Once the illustrative visualization effect class has been implemented, it must be incorporated into the following places in the source code:

1. Add an include statement for the new illustrative visualization effect header file in the tops of `MarmotViz/vtkMarmotViz.cxx` and `MarmotVizCAVE/vtkMarmotVizCAVE.cxx` files.
2. Add a construction option in the `ChangeFX(const char* InCommand)` function in the `MarmotViz/vtkMarmotViz.cxx` and `MarmotVizCAVE/vtkMarmotVizCAVE.cxx` files.
3. Add the new .cxx file name to the `OTHER_SRC` variables in the `CMakeLists.txt` files in the `MarmotViz` and `MarmotVizCAVE` directories

The MarmotViz and MarmotVizCAVE filters can now be rebuilt with the new illustrative effect following the steps from Section 2.