INL/EXT-12-26809

Generalized Framework and Algorithms for Illustrative Visualization of Time-Varying Data on Unstructured Meshes

Alexander S. Rattner
Donna Post Guillen
Srinivas Garimella
Alark Joshi

December 2012

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

INL/EXT-12-26809

Generalized Framework and Algorithms for Illustrative Visualization of Time-Varying Data on Unstructured Meshes

Alexander S. Rattner and Srinivas Garimella (Georgia Institute of Technology), Donna Post Guillen (Idaho National Laboratory), and Alark Joshi (Boise State University)

December 2012

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

ABSTRACT

Photo- and physically-realistic techniques are often insufficient for visualization of simulation results, especially for three-dimensional and time-varying datasets. Substantial research efforts have been dedicated to the development of nonphoto-realistic and illustration-inspired visualization techniques for compact and intuitive presentation of such complex datasets. While these efforts have yielded valuable visualization results, a great deal of work has been reproduced in studies as individual research groups often develop purpose-built platforms. Additionally, interoperability between illustrative visualization software is limited because of specialized processing and rendering architectures employed in different studies. This report proposes a generalized framework for illustrative visualization and implements it in MarmotViz, a ParaView plug-in, enabling its use on a variety of computing platforms with various data file formats and mesh geometries. This report gives detailed descriptions of the region-of-interest identification and feature-tracking algorithms incorporated into this tool. Implementations of multiple illustrative effect algorithms are presented to demonstrate the use and flexibility of this framework. By providing a framework and useful underlying functionality, the MarmotViz tool can act as a springboard for future research in the field of illustrative visualization.

ACKNOWLEDGEMENTS

The authors acknowledge generous financial support from the U.S. Department of Energy through the Krell Institute (contract DE-FG02-97ER25308) and the DOE Idaho Operations Office (contract DE-AC07-05ID14517).

CONTENTS

NOMENCLATURE

bc	bounding contour around a feature
c	cell
$\vec{c}, \vec{C}$	centroid of a cell or region (m)
d	distance of contour points from a central axis (m)
e	edge
f	feature
g	graph constructed from mesh
h	thickness for feature halos (m)
$\underline{\underline{J}}$	products of inertia tensor for a region (m^5)
$\vec{L}$	volumetric angular momentum of a region (m^5 s^{-1})
n	number of unmatched regions-of-interest
$\hat{n}$	camera projection plane normal
ot	octree used for feature matching
$\vec{p}$	point on a contour (m)
q	queue used for feature matching
r	region-of-interest
$\vec{r}$	position vector (m)
s	strobe silhouette curve
t	time (s)
T	user-specified threshold
$\vec{u}, \vec{U}$	velocity of a cell, or volume-average velocity of a region (m s^{-1})
$\vec{v}, \vec{V}$	volume of a cell or region (m^3)
x,*y*,*z*	spatial coordinates (m)

Subscripts

0	initial value
b	bounding contour around a feature
back	rear point on a contour
c	cell count threshold for regions-of-interest
g	gradient threshold for regions-of-interest
i,j,k	counting indices
l	left-most point on a contour
in	inset contour around a feature

min, max	minimum/maximum values along a contour
o	offset contour around a feature
r	right-most point on a contour

Superscripts

*	estimated value

Greek characters

α	similarity parameter for feature matching (-)
δ	user defined time step (s)
η	similarity criterion for feature matching (-)
θ	relaxation parameter for feature matching (-)
ω	hard lower limit for the feature matching criterion (-)
$\vec{\Omega}$	volumetric average angular velocity of a region (s^{-1})

Generalized Framework and Algorithms for Illustrative Visualization of Time-Varying Data on Unstructured Meshes

1. INTRODUCTION

This report (1) proposes and implements a generalized framework for illustrative visualization of time-varying datasets in MarmotViz, a ParaView plug-in, (2) develops detailed formulations for a region-of-interest identification algorithm and an adaptive feature-matching algorithm, and (3) describes implementations for various illustrative visualization effects in MarmotViz, including: feature coloring, selective visibility, feature smoothing, tube outlines, feature halos, speedlines, and strobe silhouettes.

1.1 Limitations of Photo- and Physically-Realistic Visualization Techniques

The challenge of presenting intelligible and useful visualizations of data has attracted substantial attention in recent years because of the growing prevalence of large-scale computational simulations across most fields in science and engineering. Such simulations are tackling increasingly ambitious problems that require unsteady modeling in complex three-dimensional (3D) geometries or consider inherently unsteady 3D physics such as large-eddy simulations (LES) and direct numerical simulation (DNS) turbulence models in fluid mechanics, necessitating considerations of multiple physical and temporal scales. Various *physically-accurate* or *photo-realistic* visualization techniques have been employed in the investigation and presentation of general simulation data, such as:

- **Vector plots** – Arrows are drawn throughout the simulation domain to indicate the direction and magnitude of vector fields (typically velocity). Arrows can be colored to indicate other scalar fields. This technique is somewhat limited in its ability to present 3-D vector data because arrows are projected to a 2-D view plane in most display systems [14] and may introduce substantial visual clutter, especially for 3-D vector fields.
- **Streamlines/surfaces** – Lines or surfaces (in 3-D) are drawn tangent to vector fields (typically velocity or vorticity) to indicate directions of motion or other properties. While streamlines or generalized *tangent-lines* indicate the path of vectors, they don't reveal magnitude or direction (forward or backward). Similarly, *tangent-surfaces* don't reveal the in-plane direction or magnitude of represented vector fields. These techniques can be augmented with superimposed arrows to present directional and magnitude information [8] at the cost of increased visual clutter. Again, projection of 3-D streamlines and surfaces to a viewing plane leads to a loss of depth information.
- **Contour/colorplots** – The domain is divided into regions by ranges of values of some scalar field. This technique can provide information about the general behavior of regions in a 2-D dataset, but an observer will only be able to view contour/color plots on surface slices of a 3-D dataset, and may miss important information that lies between slices.

The presentation of 3-D data incurs the additional challenge of visualizing internal features of a dataset. Conventional photo-realistic approaches to this problem include:

- **Isosurface rendering** – The 3-D dataset is reduced to a set of surfaces that lie along some threshold value of a scalar field. This technique can help observers identify regions of interest (such as "hot-spots" in a heat transfer simulation), but obscures internal features enclosed by isosurfaces.

- **Volumetric rendering** – Simulated light sources are specified, and illuminate the view-plane following some transfer-function that describes the attenuation (or more general visual properties) of each cell based on input data fields. Volume rendering simulates the passage of light through 3D data where some light is absorbed by the material and some light passes through. The properties of the material are specified through a function called the transfer function. The transfer function can also be used to specify color assignments (red, green blue) for various materials in the 3D volumetric data. This approach can reveal internal features of a simulation using well tuned-transfer functions, but may lead to ambiguous visualizations. For example, clustered particles in a simulation may appear as one large particle, and an observer may have difficulty interpreting the relative depth of particles.
- **Cutaways** – Sections of the domain are removed to reveal new internal surfaces. This approach can assist in the visualization of internal fields, but finding appropriate cut-surfaces can be difficult.

Time-varying simulations present the additional visualization challenge of indicating histories and development of data in a simulation. Two photo-realistic approaches for visualization of time-varying data are animations and image-series.

- **Animations** – A series of images from different time-steps of the simulation are rendered using some combination of above techniques and composed into a video. This technique offers an intuitive approach to presenting time-varying phenomena, when supported by the presentation platform. The value of animations may be limited for large simulations, as research in human vision indicates that viewers have difficulty simultaneously tracking more than five moving objects [15].
- **Image series** – Time varying data is often presented as multiple side-by-side images from different time-steps of a simulation. Clearly, this approach is amenable to more presentation environments than animations, but can take longer to interpret than animations [16]. Additionally, individual image size is often limited in a time series, reducing clarity of the overall visualization.

This report highlights a number of the limitations of photo- or physically-realistic visualization techniques including: visual clutter, difficulty in presenting depth information (on 2-D presentation platforms), obscuring internal features in 3-D datasets, and inability to clearly present complex time-evolving phenomena.

1.2 Illustrative Visualization Techniques

The development of nonphoto- or nonphysically-realistic rendering approaches is motivated by the fact that all visualization techniques are essentially qualitative. A vector-plot arrow may be rendered with a precise orientation and scale, but an observer will only be able to interpret such values approximately. Following this principle, the value of visualizations of simulation data is in presenting qualitative representations of data. Precise quantities of interest, such as peak shear stress or total heat production, are best evaluated on a computer and presented numerically.

The field of *illustrative* visualization has primarily been inspired by historical work in scientific illustration. Born et al. [8] highlighted drawings by Dallman [2] that qualitatively demonstrate key properties of vortical flow using techniques like half-toning to indicate interior surfaces and dashed lines for hidden sections as shown in Figures 1a and b. Similarly, Correa et al. [5] discussed work in medical and anatomical illustrations that reveal internal details clearly by deforming bodies and coloring tissues in vivid fashion. Born et al. [8] summarized the objectives of illustrative visualization as: “simplification of complex contexts, concentration on relevant features, and neglect of details that obstruct understanding.”

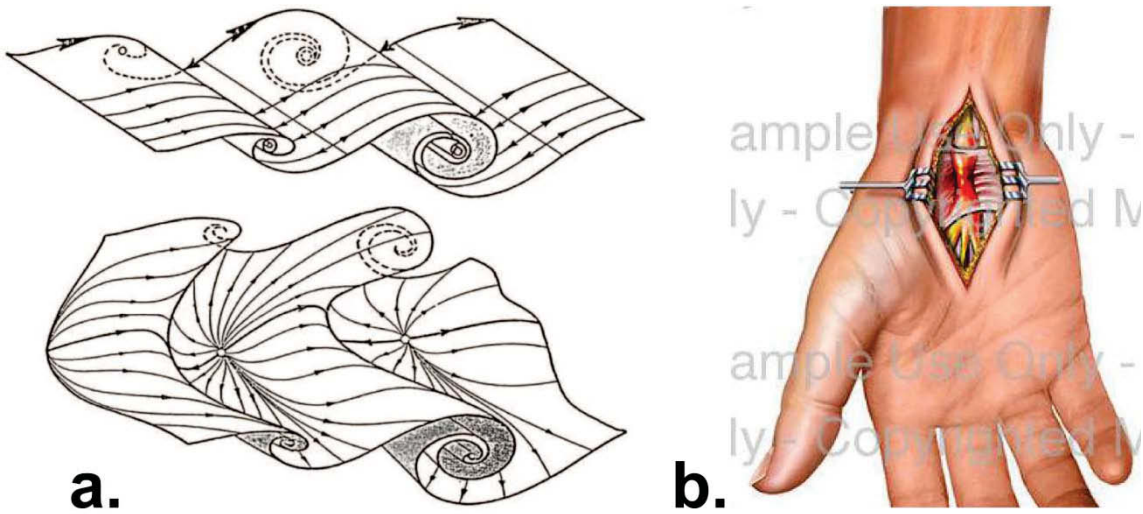

Figure1. Examples of scientific illustrations: **a** vortical flow employing half-toning and hidden sections [2] and **b.** anatomical illustration employing deforming bodies and vivid coloring of tissues [7].

While simulations often model continuum mechanics by considering numerous cells or nodes, human analyses tend to divide domains into regions such as boundary layers, vortices, or stress concentrations. This approach is encapsulated in intuitive analytic modeling techniques, such as ideal flow analysis - where flow fields are built up from superpositions of primitive elements (uniform flow, sources/sinks, doublets). Similarly, in the method of matched asymptotic expansions, domains are subdivided into regions corresponding to different governing mechanics. Thus, effective illustrative visualizations should present representations that *facilitate this qualitative partitioning and interpretation*.

Similarly, many unsteady simulation approaches are oblivious to certain critical developments in time. For example, in an interface-capturing multiphase flow simulation, bubbles may develop, coalesce, or collapse without any special notification to, or treatment by, the solver software. Again, human analyses of such unsteady phenomena focus on histories and development of individual features. Illustrative visualization tools should therefore have mechanisms for tracking the histories of features of interest.

1.3 Illustrative Visualization Approach

The goals and desirable features discussed in Section 1.2 guide the development of a framework for illustrative visualization of time-varying simulation data on unstructured meshes. In this framework, source simulations are assumed to record one or multiple data fields (such as pressure, velocity, temperature, etc.) for each point or cell in mesh geometries at discrete time-steps. Following the formulation of Joshi and Rheingans [3], the illustrative visualization process can be divided into three stages for each simulation time-step:

1. **Region-of-interest identification** – *Regions-of-interest* (ROI) in the present time-step of the simulation are identified and assessed.
2. **Feature matching and tracking** – Identified ROIs in the present time-step are matched with corresponding nearby ROIs in other time-steps. A *feature* is defined as a time-series of such matched ROIs, and could represent a physical object such as a bubble, shock-front, vortex, or thermal hot spot, depending on the simulation of interest.

3. **Illustrative Visualization** – Illustrative techniques or *effects* are applied to specific features (or the entire domain), and the complete visualization is rendered and presented to the user.

Reviews of research conducted in these three tasks for illustrative visualization are presented in Sections 1.3.1–1.3.3.

1.3.1 Prior Work in Region-of-Interest Identification in Simulation Data Sets

Substantial effort has been invested in researching the related machine vision problem of image/photograph segmentation. The specific ROI identification problem considered here is distinct in that it considers 3-D data obtained directly from simulations, so physical issues resulting from camera projection, visual noise, or lighting can be neglected. The unstructured mesh geometries considered here also have explicit connections between cells by faces, edges, and nodes. As such, connectivity between identified cells of interest can be harnessed to assist in ROI identification.

Monga et al. [17] classified ROI identification algorithms into two groups: those that group *homogeneous* cells and those that identify boundaries around regions of interest. Monga et al. argued that it was difficult to find suitable homogeneity conditions for their target applications in medical imaging, so they pursued boundary detection methods. Their study presented formulations for first and second order boundary detection algorithms. Both begin with application of smoothing filters to input data. The application of smoothing may not be needed in simulation data, because of the absence of noise from physical mechanisms. In first order approaches, the gradient magnitude is evaluated for each mesh cell, and boundary cells are defined as those with values above some threshold (which may vary depending on nearby cells). In second order approaches, the Laplacian is computed for each cell, and *zero-crossing* cells are assigned to the boundary. The authors note that it is difficult to select suitable gradient threshold values for first order boundary detection. Rather, they recommend selecting high threshold values, and attempting to close incomplete boundary surfaces in a later stage. Second order boundary detection methods also tend to yield unclosed surfaces. The authors proposed a method for boundary closing, but it is limited to rectilinear meshes.

Banks and Singer [18] investigated the problem of identifying vortices in flow simulations. Given the specialized problem domain, physical insights could be employed as the basis for ROI identification techniques. Their literature review discussed a number of approaches, including:

- Searching for low-pressure regions in incompressible turbulent flows, which tend to contain vortex cores (isopressure surfaces can approximate the boundaries of such vortices)
- Seeking regions where the velocity gradient tensor has complex eigenvalues, indicating that changes in velocity from rotation dominate those from shearing
- Assuming that vortex cores lie along points of maximum enstrophy (squared magnitude of vorticity), which can be found with gradient-climbing searches
- Assuming that vortices are approximately cylindrical, and searching for cylindrical regions of high average vorticity.

Banks and Singer developed a seed-and-growth algorithm for finding vortices based on physical criteria (low-pressure and high-vorticity). A physically inspired corrector was developed in which the vortex core axis is adjusted to intersect the pressure minimum of perpendicular planes (effectively transported by centripetal force). Such physically inspired approaches to ROI identification can yield accurate and *sensible* results. However, they are often domain specific, and may necessitate additional information not available in all data sets.

Silver and Wang [19] presented a number of criteria for identifying homogeneous regions, including threshold intervals, vector directions, and neighborhood connectivity. However, these approaches often

rely on fine-tuning by a user, and may be geometrically specific (for connectivity-based approaches). In summary, a variety of approaches for ROI identification have been investigated from general Laplacian zero-crossing and threshold-interval approaches to specialized physically motivated techniques. Performance of individual algorithms may be domain dependent.

1.3.2 Prior Work in Feature Matching and Tracking in Simulation Data Sets

The feature-matching or *correspondence* problem has also received significant attention in the computer vision community for image processing applications. As in the ROI identification problem, 3-D simulation datasets offer certain simplifications compared to 2-D images and photographs. For example, an object rotated about an axis parallel to the image plane may be pathologically unmatchable between two photographs. The view-independent nature of 3-D datasets eliminates this particular challenge. However, 3-D datasets introduce an additional dimension of complexity, and features identified in simulations can significantly deform or change in volume between output time-steps.

Reinders et al. [20] identified two classes of methods for feature matching: pixel- and feature-based approaches. Pixel-based methods measure correspondence of individual cells between nearby ROIs in two dataset time-steps. This approach may be prohibitively computationally expensive for large simulations, and requires small time steps to enable accurate cell matching. Feature-based approaches operate at a higher level of abstraction, and could range from methods that compare few bulk attributes (such as volume, mass, and/or products of inertia) to those that compare certain cells in known ROIs. Such methods are less computationally expensive, but yield less information about the development of features, such as the path of particular segments comprising a feature.

Meyer-Spradow et al. [14] developed a voxel-based matching technique. In their approach, each voxel and its surrounding neighbors in one frame are matched to most similarly valued voxels in a second frame. The computational cost of their algorithm is reduced by only searching in a small window around the starting voxel location and neglecting background areas of low intensity. Their approach can track the motion of individual cells in a feature, but is somewhat limited as presented, since it only supports datasets with single features on uniform structured rectilinear grids.

Kalivas and Sawchuk [21] developed a feature-based matching algorithm that measured the difference between 3-D regions in a relatively sparse fashion. In their approach, regions are projected into a 2-D plane and matched based on minimization of best-fit affine transforms.

Silver and Wang [22] presented a feature-based algorithm for datasets on regular meshes. Their matching algorithm assumes that subsequent time-steps are sufficiently close, so ROIs corresponding to the same feature at different times overlap in space, significantly reducing the computational cost from searches. ROIs from two time steps are matched if the relative volume of intersection exceeds some threshold, and pairs with the greatest volume of intersection are matched in the case of multiple overlaps. Good feature tracking performance was demonstrated with test datasets. However, the algorithm is limited by the requirement that corresponding regions must overlap spatially, and the use of a fixed threshold criterion, which may need to be manually tuned for different datasets. Silver and Wang [23] published a follow-on study in which they generalized their algorithm to unstructured static mesh geometries and achieved a speed-up by using different data structures to compare features.

Reinders et al. [20] developed a matching algorithm that represented individual ROIs in terms of few simple attributes including centroid, volume, and mass. In their algorithm, ROIs in newly considered time steps are matched by extrapolating attribute values from known feature histories. Matching is performed in multiple forward and backward passes with relaxing tolerances, yielding rapid matching of closely corresponding ROIs, and later matching of less-similar ones. The cost of later passes reduces, as fewer features remain to compare. While this algorithm supports matching with large time-steps between dataset

frames, it may yield inaccurate results in certain cases with clusters of similar features. Reinders et al. also discuss the incorporation of an additional *event detection* component to the illustrative visualization process. This stage would occur after feature matching, and concerns detection of discrete events that may occur to features such as: birth, death, splitting, or merging.

This survey of feature-matching algorithms highlights a number of approaches that range from individual cell- to bulk-attribute-level comparisons. In general, these approaches trade computational cost for additional feature development data and matching accuracy.

1.3.3 Prior Work in Illustrative Visualization Effects

A wide variety of illustrative visualization effects have been explored in the literature, ranging from relatively simple techniques that adjust colors to complex approaches that may deform or alter the underlying mesh geometry. This section presents a survey of these effects.

Pagendarm and Walter [4] investigated simplified visualizations for flow simulations. Their goal was to find compact and distinct representations of various phenomena to enable presentation of multiple results in a single visualization. In one figure of hypersonic flow over a wing as shown in Figure 2a, they presented vortices (using stream-ribbons), shock-lines, and skin-friction lines. Using this approach, the authors also presented both experimental and numerical results in the same visualization, permitting qualitative comparison. While this study does not demonstrate the same level of artistic liberty found in later efforts, it indicates a transition when researchers began to seek more expressive visualization techniques for simulation data.

Post et al. [11] presented a visualization approach that performs feature extraction and uses icons as symbolic representations of data. The goal in their study was to replace original complex feature data with clear and compact representations. An example visualization from their study reduces general features with complex boundaries to best fit ellipsoids as shown in Figure 2b.

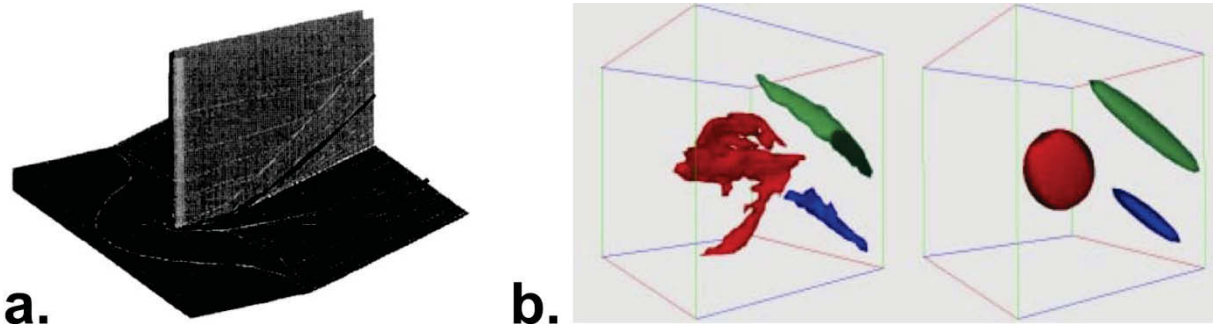

Figure 2. **a.** Simultaneous presentation of multiple data fields through stream-ribbons, shock-lines, and skin-friction lines [4], and **b.** simplified visualization of complex feature geometries [11].

Silver and Wang [19] proposed a number of illustrative techniques for time varying data for surface- and volumetric-rendering paradigms, including:

- **Enhanced surface visualization** – Features are assigned distinct surface colors to assist in tracking over multiple time-steps, and preventing confusion between nearby features. The authors also suggest rules for coloring features generated from the coalescence or breakup of other features. For example, a coalesced feature could be colored to match its larger source feature, or by mixing colors (e.g., red and blue features combine into a purple feature). Similarly bifurcated features can be colored to match their parent feature.

- **Feature isolation** – A feature of interest can be emphasized by reducing the color saturation or opacity of nearby features.
- **Enhanced volume rendering** – Similarly to enhanced surface visualization, matched features (perhaps from the same bifurcated source feature) can be assigned specific transfer functions for volumetric rendering.

Ebert and Rheingans [24] developed a number of illustrative visualization techniques for volumetric rendering. Their formulations do not operate on specific features, but instead, are incorporated into complex transfer functions that are applied to the entire volumetric data/data set, reducing visualization flexibility and possibly increasing computational cost. Illustrative techniques presented in this study include:

- **Boundary enhancement** – The opacity of cells around the boundary of a feature (determined by high gradients of a scalar field) are increased to assist in distinction between features (see implementation of this technique by Stompel et al. [10] in Figure 3a). Strobe silhouettes and tone shading are other methods to make a boundary stand out.

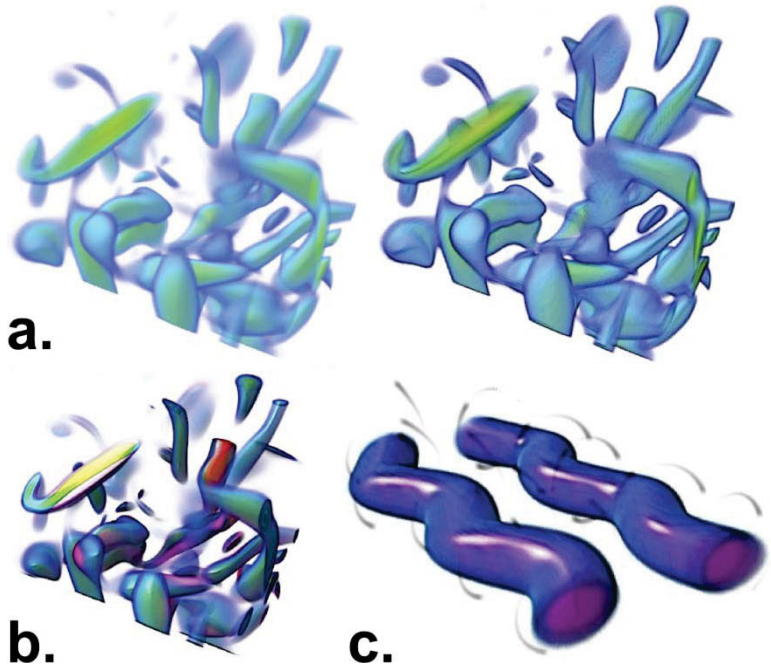

Figure 3. Illustrative visualization techniques developed by Stompel et al. [10]: **a.** original visualization on left, and with boundary enhancement on right; **b.** temporal enhancement to highlight regions with high variations in time; and **c.** multivariate visualization of vorticity field (volumetrically rendered) and velocity field (rendered with strokes).

- **Feature halos** – The region around features, parallel to the view plane, is assigned increased opacity to produce *halos*. These halos provide depth cues to the viewer, as features partially obscured by halos are easily recognized to be in the background.

Stompel et al. [10] presented a number of illustrative feature enhancement techniques for both surface and volume rendering paradigms, including:

- **Warm-to-cool shading** – Surface segments are colored relative to the lighting/view direction. Generally surfaces facing the observer are assigned warmer colors, and those oriented away from the

viewer are assigned cooler colors. This technique can assist in shape perception of 3-D features projected into the view plane.

- **Depth enhancement** – Cells closer to the view plane are rendered with warmer colors to provide additional depth cues to the viewer.
- **Temporal enhancement** – Cells or surface segments with high temporal derivatives can be rendered in warm colors to indicate regions of rapid change. This approach requires relatively small time differences between data frames for sensible visualizations (see Figure 3b).

Stompel et al. also investigated multivariate visualizations using multiple rendering approaches in a single image. For example, in a CFD application, the vorticity field could be rendered volumetrically and the velocity field can be indicated using sparse strokes (see Figure 3c).

Joshi and Rheingans [3] presented a number of visualization techniques geared toward time-varying datasets, particularly those with moving features. Investigated techniques include:

- **Speedlines** – Superimposed lines and streaks are rendered to indicate feature motion and history as shown in Figure 4a.
- **Opacity modulation** – Instances of features from previous time-steps are presented at corresponding locations with transparency and blurriness increasing backwards in time. This permits the observer to view the approximate time-history of a feature in a single image as shown in Figure 4b.
- **Strobe silhouettes** – The receding sections of feature boundaries are offset multiple times along the path of motion with reduced thickness and detail. This technique also provides visual cues of feature histories as shown in Figure 4c.

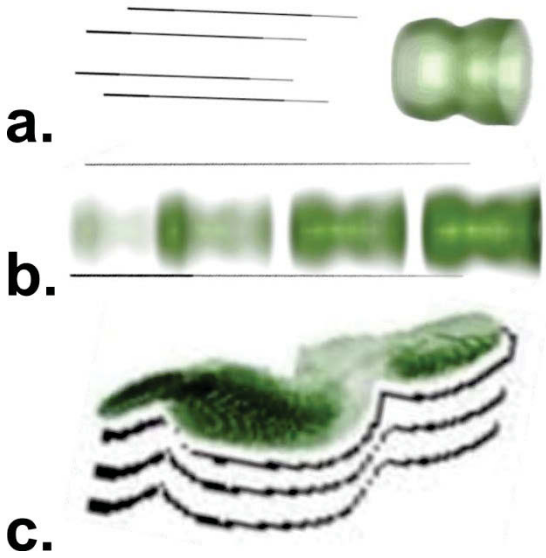

Figure 4. Illustrative visualization techniques from Joshi and Rheingans [3]: **a.** speedlines, **b.** opacity modulation, and **c.** strobe silhouettes.

Viola and Gröller [1] presented a survey of numerous illustrative visualization techniques, including:

- **Importance-driven feature enhancement** – Features are assigned *importance* values, and low-importance features are rendered transparently or with reduced detail if they obscure or lie near more important features.

- **Viewpoint-dependent distortion of 3-D data** – Geometry may be distorted to increase the relative size of features of interest, and occluding features may be moved aside as shown in Figure 5a.
- **Volume splitting and leafer deformation** – Outer layers of nested features are split and shifted to permit simultaneous visualization of inner and outer regions as shown in Figure 5b.

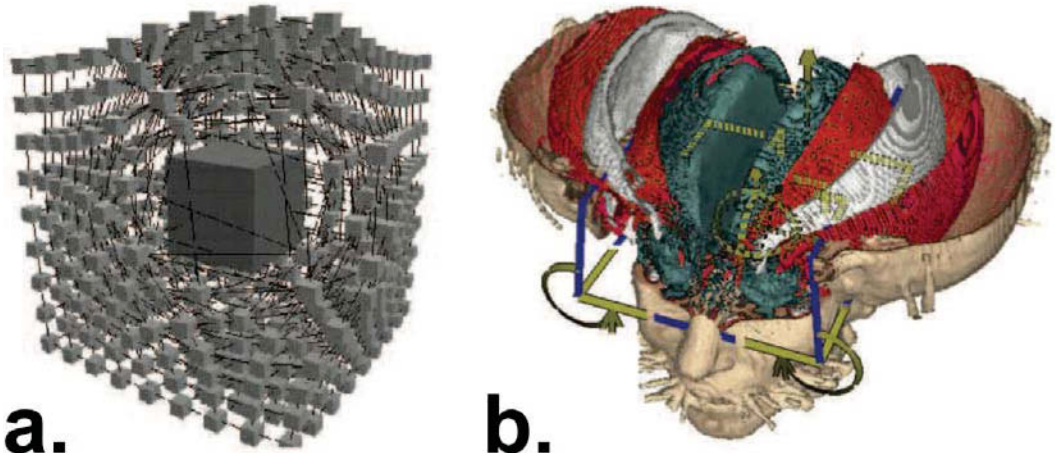

Figure 5. Illustrative visualization effects discussed by Viola et al. [1]: **a.** viewpoint dependent distortion of graph to reveal and highlight internal feature, and **b.** volume splitting and leafer deformation to reveal multiple nested features and internal details [9].

Svakhine et al. [12] investigated a number of illustrative visualization techniques geared toward the specialized problem of visualizing computational flow simulation data, including:

- **Mixed data fields for boundary enhancement** – Different data fields (or the gradient, Laplacian, etc., of a single field) may be used to alter different aspects of boundary cell properties. For example, boundary color could be determined from temperature data and opacity from velocity data as shown in Figure 6a.
- **Two dimensional transfer functions** – *Soft* tangent-lines can be generated automatically in a feature by using a periodic function (such as: $\sin(f)^k$ where f is the displayed data field) to modulate the cell opacity as shown in Figure 6b.

Svakhine et al. also developed computational techniques to simulate Schlieren as shown in Figure 6c and shadowgraph *photographic* techniques as shown in Figure 6d.

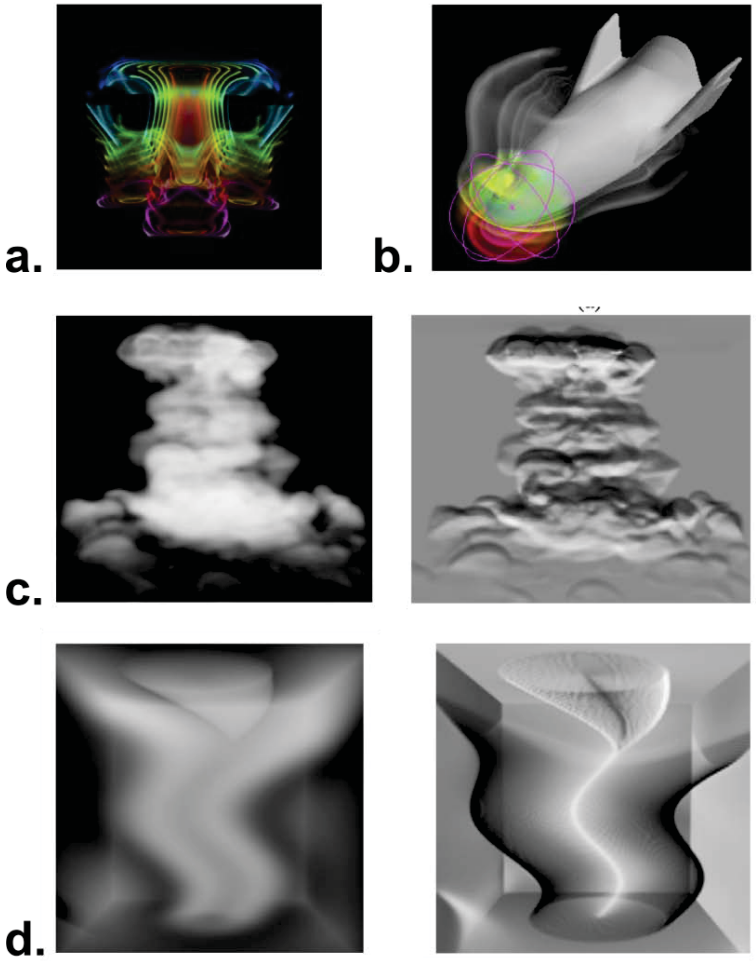

Figure 6. Illustrative visualization techniques from Svakhine et al. [12]: **a.** mixed data field boundary enhancement (natural convection flow with boundary color defined by temperature and opacity by velocity), **b.** periodic transfer functions to display *soft* contour lines in features, **c.** simulated Schlieren photographic effect of a water vapor cloud (volumetric rendering on left and Schlieren rendering on right), and **d.** simulated shadowgraph technique of a tornado (volumetric rendering on left and shadowgraph rendering on right).

Correa et al. [5] presented a formulation for illustrative visualization motivated by illustrations found in anatomy texts. Their algorithms simulate common surgical and dissection tools such as peelers, retractors, or pliers as shown in Figure 7a. This approach is distinct in that it deforms the model geometry directly, rather than modifying cell colors and opacities or adding simple elements (curves, arrows, etc.) to the domain. Correa et al. also demonstrated the value of using feature data for generating cut-away views of 3-D volumetric data. Both axis-aligned (defined by global coordinates) and surface-aligned (defined by distances from identified isosurfaces) cuts may remove portions of ROIs. However, by using feature-based (segment-aligned) cuts, obscuring features can be removed without impinging on the ROI geometry as shown in Figure 7b.

Lu and Shen [6] developed a tool to present 3D time varying data in a storyboard format with snapshots of the inputs data from different times as shown in Figure 8. An algorithm was developed to compare datasets from different times and identify key frames, reducing the number of snapshots needed in the storyboard. Individual snapshots were simplified to accelerate the difference calculation, and reduce visual clutter.

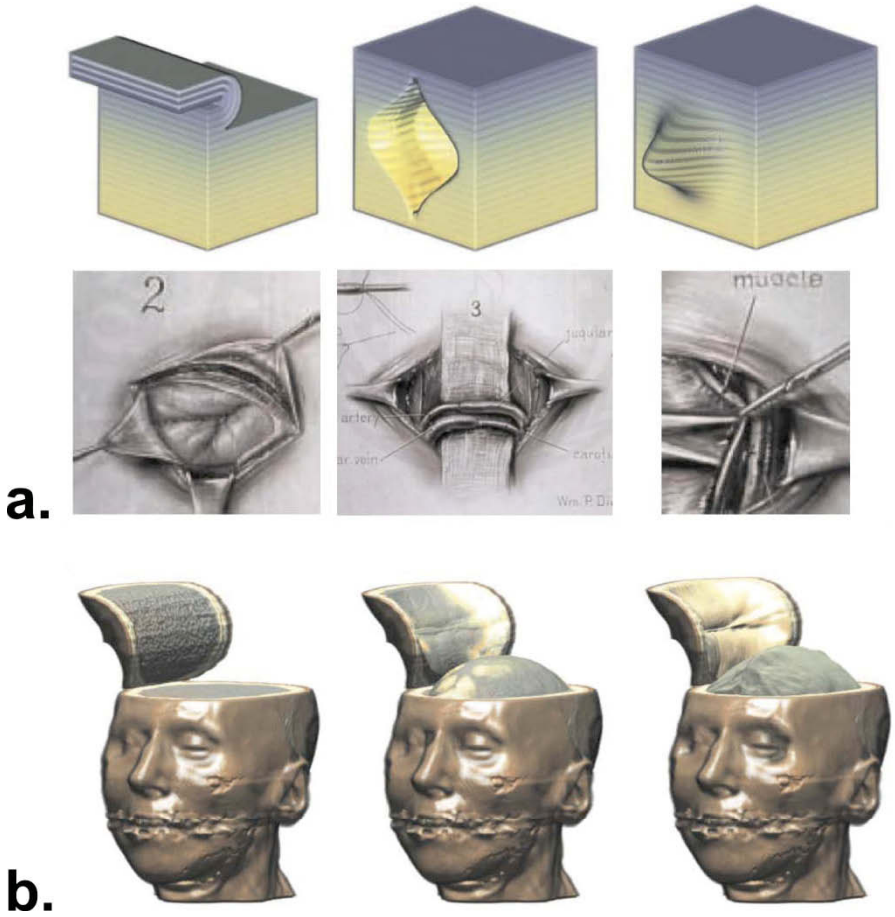

Figure 7. Illustrative visualizations presented by Correa et al. [5]: **a.** simulation of medical and dissection instruments: peelers, retractors, and pliers (hand-drawn illustrations from the U.S. National Library of Medicine), and **b.** volume splitting and leafer deformation to reveal multiple nested features and internal details [9].

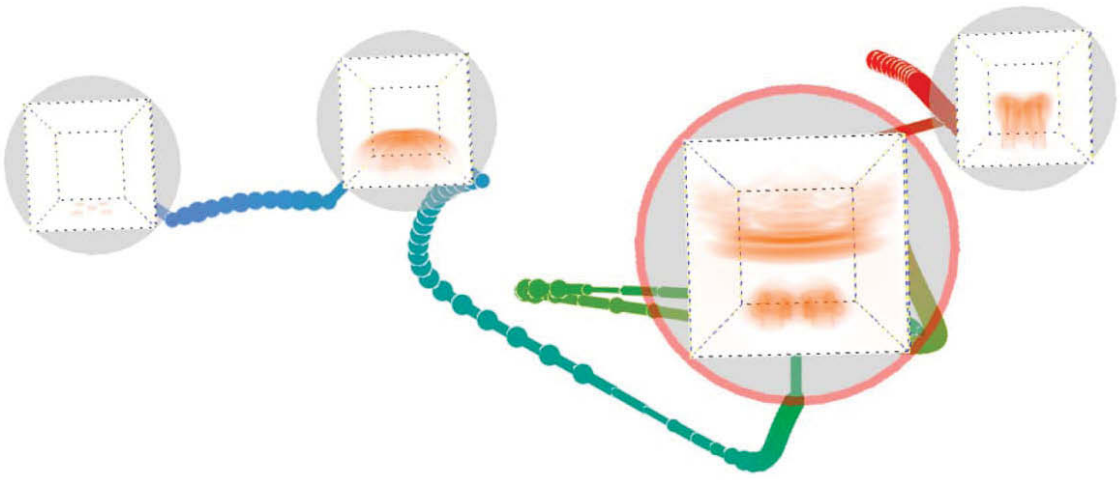

Figure 8. Storyboard visualization of time-varying data [6].

Born et al. [8] investigated visualization of stream surfaces. Stream surfaces often wrap spirally, so naive surface rendering techniques may cause significant self-occlusion. The authors presented a number of illustrative techniques, including some proposed in earlier studies such as feature/boundary lines (called boundary enhancement by Ebert and Rheingans [24]) and cut-aways. Additionally, they proposed the use of halftones and hatching to help distinguish faces of wound stream surfaces as shown in Figure 9.

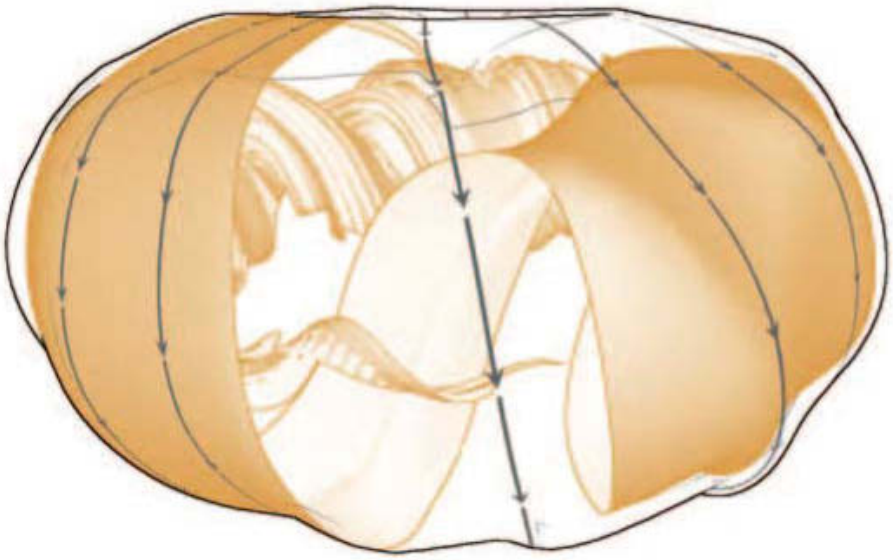

Figure 9. Wound stream surface using cut-aways to reveal interior features and halftones to distinguish sides of surface [8].

Hsu et al. [13] developed an approach to visualize time varying data by presenting multiple time-instances of a feature at once, similar to the opacity modulation technique proposed by Joshi and Rheingans [3]. Older instances of features are indicated by reduced color saturation, increased transparency, or are rendered as line drawings as shown in Figure 10.

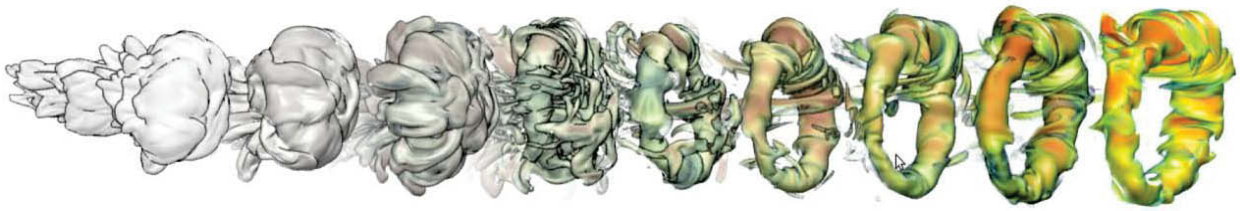

Figure 10. Evolving gas ring visualized using opacity modulation with reduced color saturation and line-rendering techniques at earlier instances [13].

Readers are also directed to a review of illustrative visualization techniques by Rautek et al. [25] that discusses the history of the field and perspectives on its future.

1.4 Present Study

The three primary tasks in the illustrative visualization process (region identification, feature matching, and effect visualization) have received considerable attention in the literature. However, many of these investigations focus on specific geometries (often uniform rectangular grids) or are geared towards specific physical problems, such as aerodynamic flows with vortices or human anatomy. Additionally, a substantial amount of software development effort has been duplicated by researchers

working on proprietary or specialized tools. The objective of the present study is to develop a modular framework for the illustrative visualization of time-varying computational simulations on general mesh geometries. It is not feasible to support all illustrative visualization approaches considered in the literature, but by providing a flexible framework, users can harness a variety of ROI identification algorithms, feature matching and tracking algorithms, and illustrative effects and quickly implement additional techniques.

The developed framework, is implemented as a plug-in filter, MarmotViz, for the ParaView [26] environment. ParaView was selected for this effort because it is an extensible open source visualization tool that supports numerous data formats and mesh geometries, and operates on a wide variety of computing platforms. ParaView was developed using the Visualization Toolkit (VTK) [27] library, which provides access to useful functionality for many illustrative effects such as gradient filters, computational geometry routines, and specialized data structures. Paraview operates in a pipelined fashion, so MarmotViz receives generalized input data from a file reader or preprocessing filters, and passes output down the pipeline, potentially to other post-processing filters, and eventually to the rendering and display system. Thus, the developed plug-in can be used in conjunction with other visualization tools and filters available in ParaView.

ParaView provides data to the MarmotViz filter at individual time-steps in a simulation, potentially out-of-order, depending on user input. Thus, at each update call, the MarmotViz filter performs the following process:

1. Read new user inputs to the MarmotViz GUI (update algorithm parameters, disable/enable features and effects, create/change/delete illustrative effects applied to particular features).
2. Receive dataset from new time-step, if the time-step has been processed previously, skip to Step 3:
    a. Identify ROIs in the time-step using the user-selected algorithm and parameters. Assess ROIs and evaluate key quantities
    b. Match identified ROIs to known features from other time-steps using the user-selected algorithm and parameters.
3. Construct the output data field with cell values corresponding to feature indices.
4. Apply illustrative effects to the output dataset (mesh geometry and field data).
5. Pass the output dataset to the next stage in the ParaView pipeline.

A schematic of the MarmotViz program structure is presented in Figure 11.

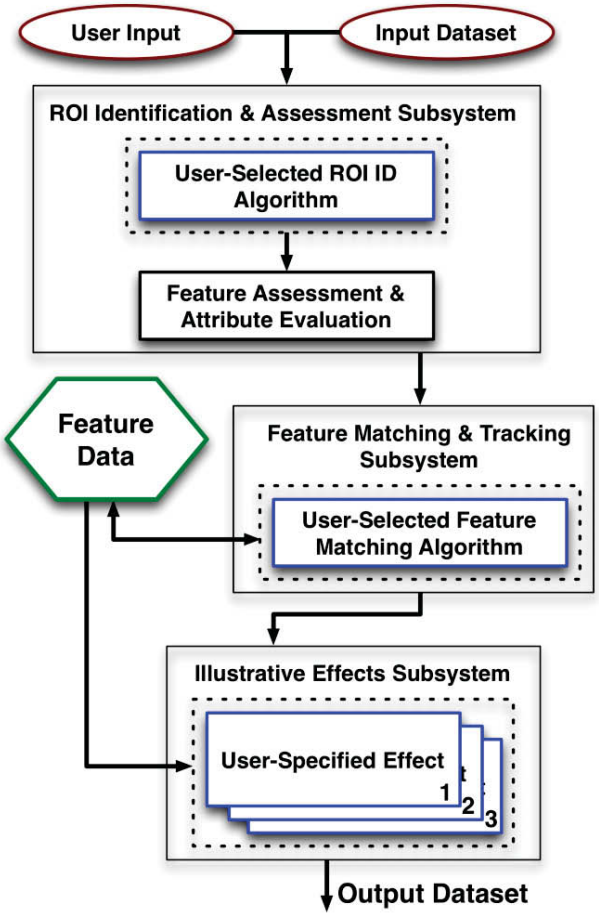

Figure 11. Schematic of the MarmotViz plug-in structure.

The Section 2 provides a detailed description of the subsystems and algorithms developed in this study for ROI identification and assessment, Section 3 describes feature matching and

tracking, and Section 4 presents the illustrative effects subsystem and developed effects, which include:

- **Feature coloring and selective visibility** – Features are colored by feature number and can be selectively hidden
- **Feature smoothing** – Smooth bounding surfaces are generated for features to improve aesthetics and accelerate rendering
- **Tube outlines** – Similar to the boundary enhancement technique implemented by Stompel et al. [10]
- **Feature halos** – Motivated by the halo effect described by Ebert and Rheingans [24]
- **Speedlines** – Based on the effect described by Joshi and Rheingans [3]
- **Strobe silhouettes** – Also similar to the effect presented by Joshi and Rheingans [3].

All implemented ROI identification, feature matching, and illustrative effects algorithms are formulated to support general unstructured meshes.

For the remainder of this document, described algorithms will be demonstrated using data from a two-phase flow simulation of a bubble-column—a container of liquid with multiple low-density fluid injection ports in its base as shown in Figure 12. In particular, the fluid-phase field will be considered since it yields intuitive *features* for the follow-on discussion such as bubbles, droplets, and fluid layers. The simulation dataset includes information for 540,000 cells at 45 time-steps.

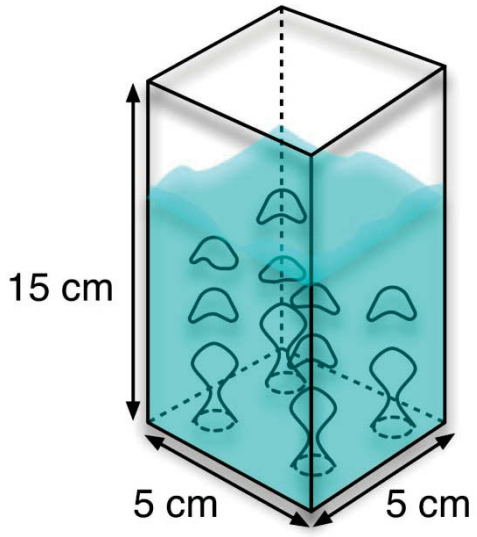

Figure 12. Schematic of small-scale bubble column simulation employed to demonstrate illustrative visualization effects.

2. REGION-OF-INTEREST IDENTIFICATION AND ASSESSMENT SUBSYSTEM

2.1 Subsystem Overview

The ROI identification and assessment subsystem receives the input mesh geometry and data fields. It first applies a user-selected ROI identification algorithm to the dataset, which can be controlled with user-specified parameters. A gradient-based ROI identification algorithm was developed in this effort to demonstrate the use and flexibility of this subsystem (Section 2.2). These ROIs are then assessed to evaluate useful attributes (see Section 3.3), including: volumes, centroids, and average velocities. These properties are employed in the feature matching and tracking subsystem and in various illustrative effects. A schematic of the ROI identification and assessment subsystem is presented in Figure 13.

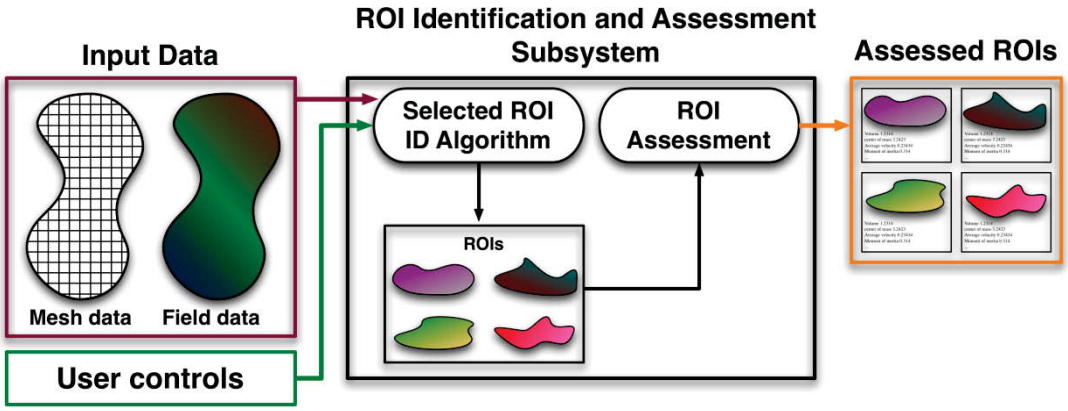

Figure 13. Schematic of the ROI identification and assessment subsystem.

2.2 ROI Identification Process

The demonstrative ROI identification algorithm developed for the MarmotViz visualization framework employs a boundary detection scheme to partition the geometry, as discussed by Monga et al. [17]. This routine begins by evaluating the gradient magnitude of a user-specified input field. Boundary mesh cells are identified as those with gradient magnitude of the selected input field above a user-defined threshold. The mesh geometry is then converted into a graph with each node representing a mesh cell, and edges connecting nonboundary cells that share faces as shown in Figure 14. This connectivity information can be computationally expensive to evaluate, especially for unstructured meshes, and is therefore cached for reuse after the first evaluation (for static input meshes). Contiguous regions in the graph are identified using a breadth first search. Following this approach, all contiguous mesh regions separated by high gradient cells are considered to be ROIs, potentially including large background regions with little variation. However, such regions can easily be excluded from visualizations over multiple time-steps once they are classified as persisting features. Additionally, this implementation allows users to specify a minimum cell count to skip small spurious regions.

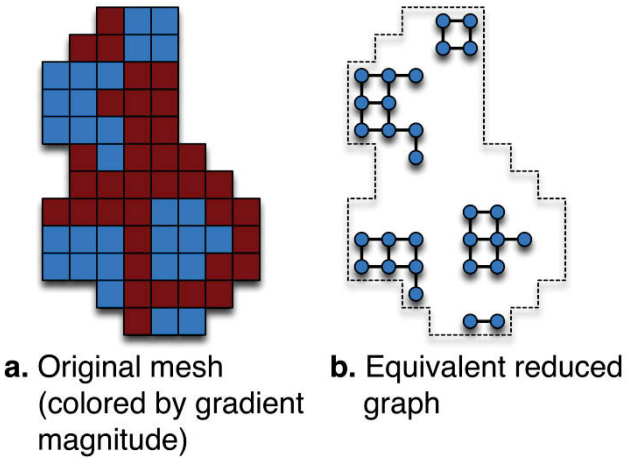

Figure 14. Transformation from mesh with (**a**) high gradient-magnitude cells in red to (**b**) graph with connections between low-gradient magnitude cell face-neighbors.

2.3 ROI Assessment Process

Once ROIs are identified in a time-step they are assessed to yield attributes useful for feature matching and illustrative visualization, including: total volume, centroid, average velocity, products of inertia, and average angular velocity. Also, exterior cells and faces are identified for use in illustrative effect routines, which often operate only on feature boundaries.

Since the MarmotViz utility is developed to support general unstructured meshes, potentially composed of various cell geometries, individual cell properties, such as volume (v_i) and centroid ($\vec{c}_i$), cannot be calculated directly. Instead ROI cells are subdivided into tetrahedra using built-in VTK routines. Thus, total region-of-interest (k) volumes (V_k) can be computed as a sum of tetrahedral volumes (j) composing cells (i) as

$$V_k = \sum_{i}^{\text{cells}} \sum_{j}^{\text{tets.}} v_{ij} \quad . \tag{1}$$

ROI centroids ($\vec{C}_k$) and volume-averaged velocities ($\vec{U}_k$) (when cell velocity data, $\vec{u}_i$, are available) are computed similarly using

$$\vec{C}_k = \sum_{i}^{\text{cells}} \vec{c}_i v_i \Big/ V_k \tag{2}$$

$$\vec{U}_k = \sum_{i}^{\text{cells}} \vec{c}_i u_i \Big/ V_k \quad . \tag{3}$$

Additionally, when cell velocity data ($\vec{u}_i$) are available, the volumetric angular momentum ($\vec{L}_k$), products of inertia tensor ($\underline{\underline{J}}_k$), and average angular velocity ($\vec{\Omega}$) are evaluated using derivations by Essén [28] in Eq. 4–6 where, coordinates ($\vec{r} = (x, y, z)$) are relative to the ROI centroid:

$$\vec{L}_k = \sum_i^{\text{cells}} v_i \left(\vec{r}_i \times \vec{u}_i \right) \tag{4}$$

$$\underline{\underline{J}}_k = \sum_i^{\text{cells}} v_i \begin{bmatrix} y_k^2 + z_k^2 & -x_i y_i & -x_i z_i \\ -x_i y_i & x_k^2 + z_k^2 & -y_i z_i \\ -x_i z_i & -y_i z_i & x_k^2 + y_k^2 \end{bmatrix} \tag{5}$$

$$\vec{\Omega}_k = \underline{\underline{J}}_k^{-1} \vec{L}_k \ . \tag{6}$$

Exterior cells in a ROI are found by scanning for cells that do not have neighbor cells internal to the region on all faces. The identified exterior cell faces are stored compactly as ordered point lists for later use in illustrative effect algorithms. A summary of the gradient-based ROI identification algorithm and this assessment process are presented in Algorithm 1.

Algorithm 1. Gradient-based ROI identification and assessment:

1. Compute gradient for each cell i in mesh
2. Transform mesh into a graph g, with connections between face-sharing cells with gradient magnitude below threshold: T_g
3. Perform breadth first search to identify contiguous regions-of-interest in g
4. Remove regions-of-interest with cell count below threshold T_c
5. **FOR EACH** region-of-interest k
    1. Compute total volume: V_k
    2. Iterate through cells to compute: centroid $\vec{C}_k$, mean velocity $\vec{U}_k$, angular momentum $\vec{L}_k$, and products of intertia $\underline{\underline{J}}_k$
    3. Evaluate average angular velocity: $\vec{\Omega}_k = \underline{\underline{J}}_k^{-1} \vec{L}_k$
    4. Construct lists of boundary cells and exterior faces
6. **LOOP**

3. FEATURE MATCHING AND TRACKING SUBSYSTEM STAGE

3.1 Subsystem Overview

The feature matching and tracking subsystem is applied once for each newly visited simulation time-step. The subsystem receives information about ROIs identified at the present time-step and known features from other time-steps. It applies a user-selected feature-matching algorithm to assign ROIs to existing features and define new features as appropriate. An adaptive volume based algorithm was developed to demonstrate the use this subsystem (Section 3.2). A schematic of the feature matching and tracking subsystem is presented in Figure 15.

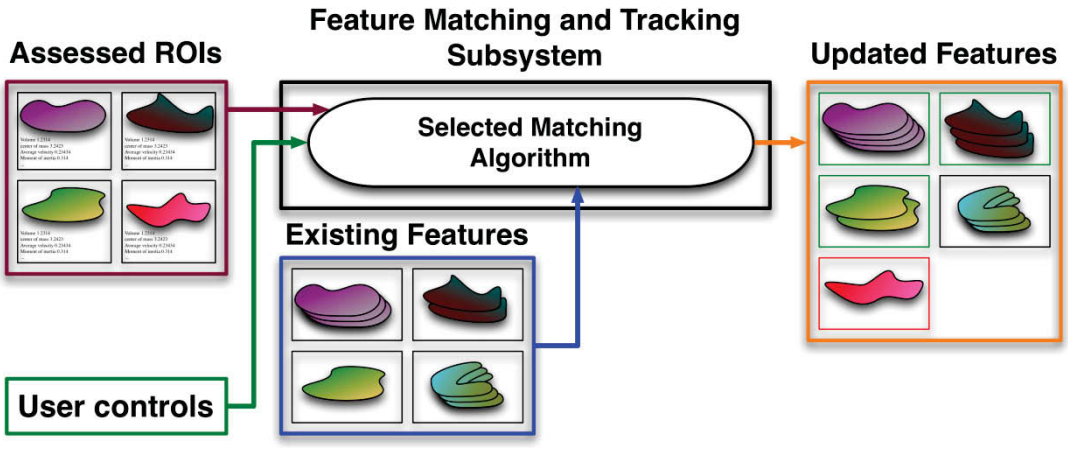

Figure 15. Schematic of the feature matching and tracking assessment subsystem.

3.2 Adaptive Volume-Based, Feature-Matching Algorithm

A variety of feature-matching strategies ranging from computationally expensive pixel-based to fast bulk attribute-based schemes were identified in the literature. In order to support large simulation geometries, a bulk attribute-based scheme was selected for this effort. The developed matching algorithm begins by seeking feature data from the temporally closest evaluated time-step (t') to the current time-step (t). All extant features at time-step t' are stored in an octree indexed by centroids, which permits efficient comparison of large sets of regions-of-interest. The octree implementation employed in this study was implemented by Krafft et al. [29].

The actual matching process between ROIs at time t and features at t' proceeds in a cyclic fashion, looping through a queue of unmatched regions. First, the ROI at the head of the queue is dequeued, and its centroid is obtained. The octree is then searched to identify the feature with the closest centroid at t'. This approach permits matching with greater time differences between data frames than that of Silver and Wang [22], since features are not required to overlap spatially. Additionally, when velocity data are available (as in many fluid mechanics simulations), an improved, expected region centroid is used to search the octree ($\vec{C}_k^{'} = \vec{C}_k + \vec{U}_k\left(t'-t\right)$). Correspondence between the current region at time t and the feature at t' is evaluated using a volume-based heuristic (α).

$$\alpha = 1 - \frac{\left|V - V'\right|}{\max\left(V, V'\right)} \,. \tag{7}$$

This value of the matching parameter, α ranges from 0 when the compared region and feature are totally dissimilar in volume, to unity when they have equal volumes. If α is greater than some similarity criterion η, the region is matched to the feature, and the feature is removed from the octree. Otherwise, the unmatched region is enqueued. Every time the loop completes a pass through all unmatched features, the similarity criterion is relaxed by a user-specified factor ($\eta \leftarrow \theta \cdot \eta$). This process continues until all regions at *t* are matched to features at *t'* (or vice versa), or the similarity parameter becomes smaller than some user-specified hard limit, ω. Typical values of these parameters are: $\eta_0 = 0.9$, $\alpha = 0.8$, and $\omega = 0.5$. These values specify that the initial matching criterion requires volume differences of less than 10% (1 - 0.9), the criterion relaxes to 80% at each pass, and matching ceases when η falls below 0.5. So ROIs are not matched with feature instances that differ in volume by more than 50%. A summary of this feature-matching algorithm is presented in Algorithm 2.

Algorithm 2. Summary of feature-matching algorithm:

1. For dataset at time *t*, find data from closest previously evaluated time *t'*
2. Construct octree *ot* of features from *t'* indexed by centroids $\vec{C}_k$
3. Insert regions of interest r_i into queue *q*
4. Initialize η
5. **WHILE** *ot* not empty, *q* not empty, $\eta \geq \omega$ **DO**
    1. $n \leftarrow$ number of unmatched regions
    2. **FOR** i = 1..*n*
        1. Dequeue head region *r* from *q*
        2. Find closest expected feature in *ot* at *t'*: *f'*
        3. Evaluate $\alpha \leftarrow \alpha(V, V')$
        4. **IF** $\alpha \geq \eta$ **THEN**
            1. Assign *r* to feature *f*
            2. Remove *f* from *ot*
        5. **ELSE**
            1. Enqueue *r* in *q*
        6. **END IF**
    3. **LOOP**
    4. $\eta \leftarrow \theta \cdot \eta$
6. **LOOP**
7. Construct new features for remaining regions-of-interest *r* in *q*

One of the key advantages of this algorithm is that it dynamically adjusts the matching criterion during operation, so tuning of a static parameter is not needed as in the approach of Silver and Wang [22]. In future revisions, additional matching parameters (such as products of inertia or additional data fields) and feature data from multiple time-steps could also be considered, as in the algorithm of Reinders et al. [20].

4. ILLUSTRATIVE EFFECTS STAGE

4.1 Feature Coloring and Selective Visibility

The feature coloring and selective visibility illustrative effects are presented here, before discussion of the general effect framework, since they are independent of the main illustrative effect subsystem.

After completing feature matching, the MarmotViz filter generates an output field in which mesh cells are colored by feature number. By viewing renderings of this output field, users can easily track distinct features in animations or image series of time varying simulation data. This effect can be observed in the image-series of rising bubbles from the example bubble reactor dataset as shown in Figure 16. The feature smoothing illustrative effect (Section 4.3) is applied to features in Figure 16 for aesthetic purposes.

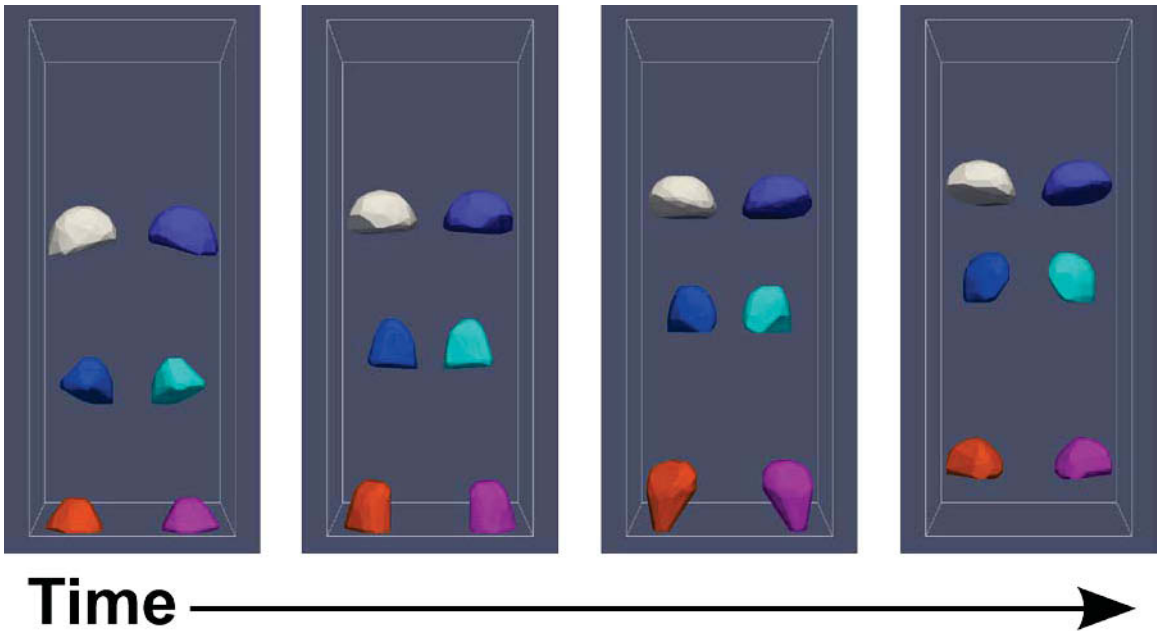

Figure 16. Feature coloring illustrative effect applied to bubble-reactor dataset, assigning unique colors to features assists in interpretation of time series or animations.

Additionally, the MarmotViz GUI provides user controls for selectively disabling features and removing them from the output dataset. The visualizations achieved from this process yield *importance-driven feature enhancement*, as proposed by Viola and Gröller [1]. Figure 17 demonstrates the application of this effect to the bubble column test case to conceal obscuring background features.

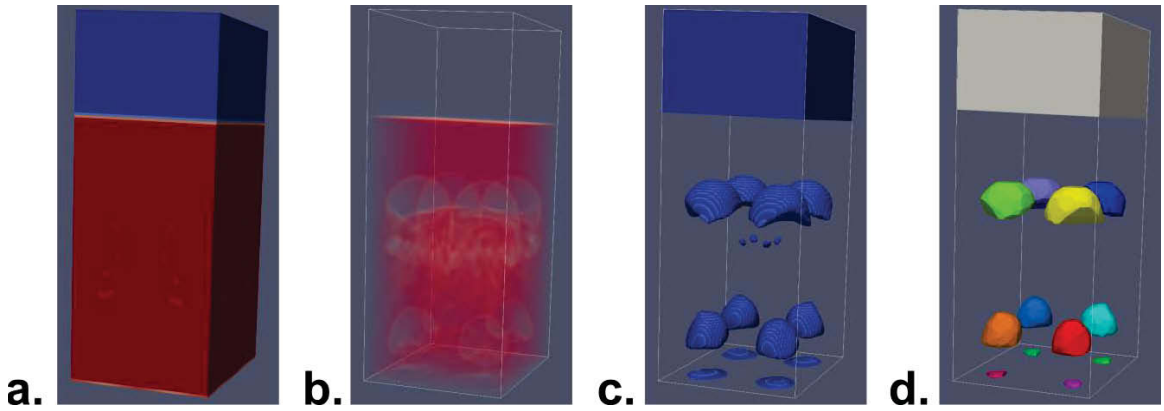

Figure 17. Application of feature coloring to bubble column dataset: **a**. original data set with surface rendering, **b.** volume rendering reveals internal features, but with limited clarity, **c.** isosurface rendering can extract certain features of interest, but overlapping features may be difficult to distinguish, **d.** selective visualization can be used in conjunction with feature coloring to isolate features of interest (background and small features removed) and facilitate feature identification.

4.2 Overview of Illustrative Effects Subsystem

The developed framework also supports application of general illustrative visualization effects to features. In the current implementation, illustrative effect objects receive data from a particular feature, and apply illustrative effects by altering output mesh geometry and field data. A higher-level illustrative effect applicator object manages the creation, modification, and deletion of illustrative effects, and applies activated effects to the output dataset during each update of the MarmotViz filter. A schematic of the illustrative effect subsystem is presented in Figure 18.

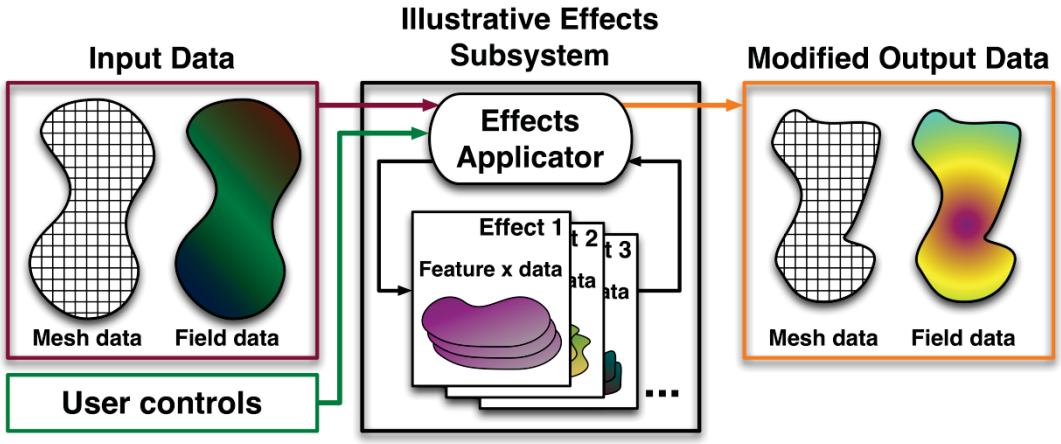

Figure 18. Schematic of the illustrative effects subsystem. Mesh and field data and user controls are received as input, individual effects are applied sequentially, and the modified mesh and field data are returned as output.

Formulations for five illustrative effects are presented in the following sections to demonstrate the use and utility (and limitations) of the framework developed in this study.

4.3 Feature Smoothing Effect

ROIs identified using the gradient-based approach developed in this effort may appear blocky or jagged, depending on the underlying mesh geometry. It is often desirable to present smoothed representations of these features visualizations, especially in cases like the example dataset used here where features (bubbles, droplets) are physically smooth. Rendering costs can also be reduced by using a simplified bounding surface instead of a blocky set of cells with more exterior faces.

The smoothing effect developed here begins by extracting all exterior points from a feature at a given time-step. These points are passed to the built-in VTK Delaunay 3-D meshing filter to generate a locally-convex representation of the feature comprised of tetrahedra. The exterior surfaces of this representation are then extracted using another built-in VTK filter, and added to the output mesh with the same data field values as the original feature. The generated smoothed surfaces are cached for reuse if time-steps are revisited. Feature smoothing is employed in conjunction with other illustrative effects in Figures 16, 17d, 21, 23, 25, and 27. A demonstration of the feature smoothing effect is presented in Figure 19.

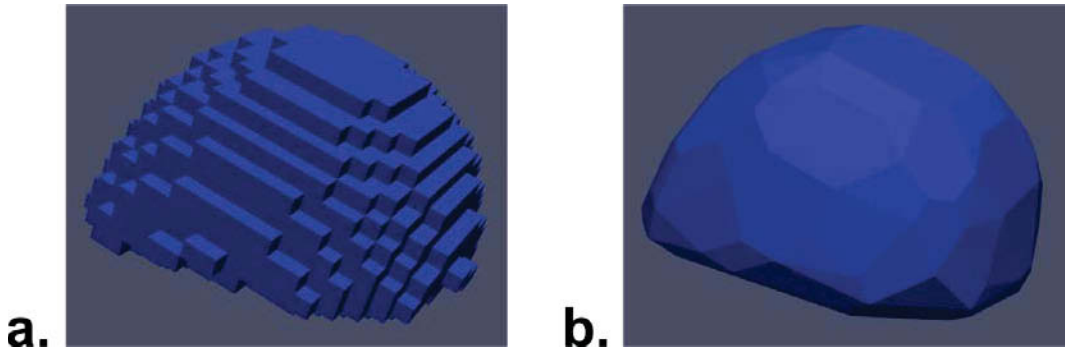

Figure 19. Features extracted from the mesh geometry may appear jagged or blocky (**a**); the feature smoothing effect (**b**) produces more aesthetically pleasing and, often, more physically-realistic visualizations, while reducing the number of exterior surfaces that must be rendered.

4.4 Tube Outline Effect

The tube outline effect developed in this study is motivated by boundary enhancement techniques discussed in the literature [10,24]. These techniques attempt to increase the clarity of bounding edges and contours around features, relative to the view direction, assisting in the identification of and discrimination between features. The effect developed here constructs a tubular contour around features to clarify borders.

Many related efforts in the literature apply boundary enhancement techniques directly to 2-D, prerendered images of datasets [8,13,30]. Because of the pipelined nature of ParaView, effects developed in this study are limited to operating on the 3-D field data and meshes. As such, the first step of the tube outline effect is to determine the projection of the selected feature to the view plane. First, the effect retrieves the camera projection plane vector ($\hat{n}$) from ParaView. The bounding contour around the feature relative to this view direction is then obtained in a subroutine.

The bounding contour determination subroutine begins by retrieving all exterior face-points on the selected feature (previously identified during the ROI assessment process). For convenience, a homogeneous transform is applied to these points so that the projection plane normal ($\hat{n}$) is aligned to the *z*-axis. Three uniform square grids (100 × 100 in this implementation) are constructed: `CoveredCells`, `VisitedCells`, and `HeightMap`. The algorithm then iterates through all exterior feature edges and marks intersected grid cells in `CoveredCells`. During this process, cells in

`HeightMap` are assigned the highest *z*-value of intersecting edges—essentially, the position of edges closest to the view plane. Once this process completes, any weakly connected cells (marked cells with only one marked face-neighbor) in `CoveredCells` are unmarked. The resulting grid yields a connected, closed representation of the projection of the feature into the view plane. An exterior marked cell in `CoveredCells` is then selected as a seed point, and the grid is traversed in a counter-clockwise fashion between face-neighboring marked cells. Each newly visited marked cell in `CoveredCells` is also marked in `VisitedCells`. The contour identification process completes when a previously visited cell is reached—typically the seed point, but potentially another cell for unusual features. A closed contour approximating the outline of the feature from the view direction is obtained from the marked cells in `VisitedCells` and corresponding *z*-values in `HeightMap`. This contour is then transformed back into the global coordinate system. A summary of this bounding contour determination algorithm is presented in Algorithm 3, and a graphical representation is presented in Figure 20.

The obtained bounding contour is then smoothed using a discrete cosine transform (DCT) filter to remove a user-specified portion of high frequency components resulting from the mapping of the feature geometry to a rectilinear grid. A built-in VTK filter is employed to extrude the contour lines into cylindrical tube sections. The tube contour is then added to the unstructured mesh and colored using a user-defined intensity value. A summary of the entire tube outline illustrative effect process is provided in Algorithm 4. User controls are provided for tube thickness, color value/intensity, number of sides per cylinder segment, and the DCT contour smoothing process. A demonstration of this illustrative effect is presented in Figure 21 for the bubble column test case.

Algorithm 3. Boundary contour construction:

1. Transform all exterior face points $\vec{p}_i$ in feature *f* to local coordinates with $\hat{k} \square \hat{n}$
2. Construct grids: `CoveredCells`, `VisitedCells`, `HeightMap`
3. **FOR EACH** exterior face edge *e*
    1. **MARK** overlapping cells in `CoveredCells`
    2. Assign greatest *z*-values of overlapping cells in `HeightMap`
4. **LOOP**
5. **WHILE** `CoveredCells` contains weakly connected cells **DO**
    1. **FOR EACH** cell *c* in `CoveredCells`
        1. **UNMARK** *c* **IF** weakly connected
    2. **LOOP**
6. **LOOP**
7. Initialize boundary contour *bc*
8. Select exterior marked seed cell *c* in `CoveredCells`
9. **WHILE** *c* not marked in `VisitedCells` **DO**
    1. Add *c* (and *z*-value from `HeightMap`) to *bc*
    2. Mark *c* in `VisitedCells`
    3. Traverse counter-clockwise around *c* to find next marked cell (*c*) in `CoveredCells`
10. **END**
11. Transform *bc* to global coordinate system

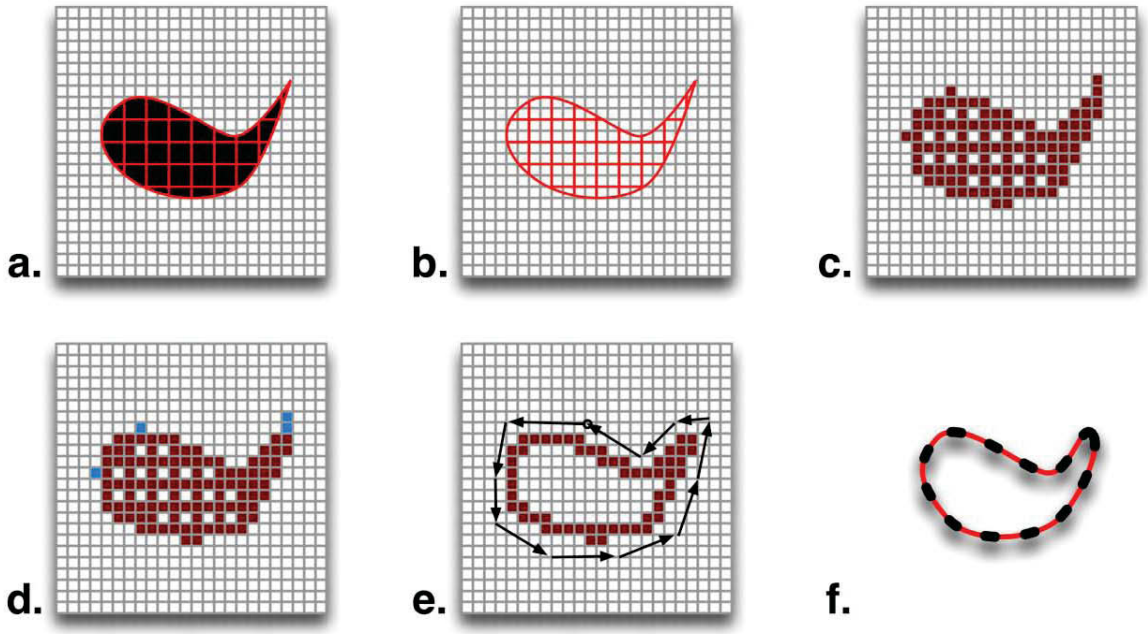

Figure 20. The contour identification process: **a.** working grids are constructed, **b.** exterior edges are extracted, **c.** grid cells that intersect edges are marked (and height map is produced), **d.** weakly connected marked cells are removed, **e.** the grid is traversed in a counter-clockwise fashion to produce a bounding contour, **f.** the contour is transformed back to the global coordinate system.

Algorithm 4. Tube outline illustrative effect:

1. For feature *f*, obtain bounding contour *bc* using Algorithm 3
2. Apply DCT smoothing to *bc*
3. Extrude tubes from *bc* using built-in VTK filter with user-defined parameters
4. Add tube outline to output mesh geometry and color using user-specified value

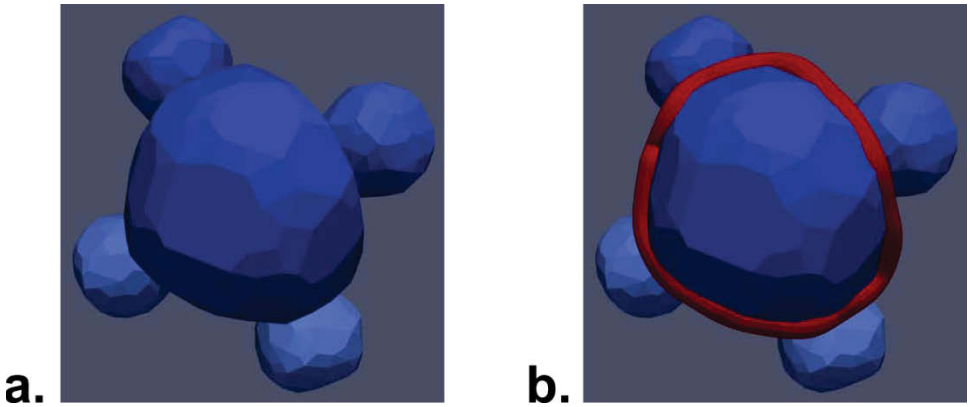

Figure 21. In cases where features may be difficult to distinguish (**a**), the tube outline effect (**b**) can clarify features of interest.

4.5 Feature Halos Effect

The feature halos illustrative effect developed here is motivated by previous work from Ebert and Rheingans [24]. In this effect, ribbon-like halos are added around selected features to clarify boundaries and provide additional depth cues. The formulation developed here differs from that of Ebert and Rheingans in that halos are constructed by adding 2-D surface elements rather than by modulating the opacity of existing cells in a volumetric rendering paradigm.

The feature halos effect begins by obtaining a boundary contour around a feature relative to the view direction, reusing the routine described in Section 4.2. The contour is then smoothed using a DCT filter, as in the tube outline effect. The bounding contour (b) is then offset (o) by thickness h_o using averaged cross products between segments connected to each point ($\vec{p}_{b,i}$) around the contour and view-plane normal ($\hat{n}$) by

$$\vec{p}_{o,i} = \vec{p}_{b,i} + h\left[\hat{n} \times \frac{\left(\vec{p}_{b,i-1} - \vec{p}_{b,i}\right) + \left(\vec{p}_{b,i+1} - \vec{p}_{b,i}\right)}{2}\right]. \tag{8}$$

A schematic of the contour offsetting process is presented in Figure 22.

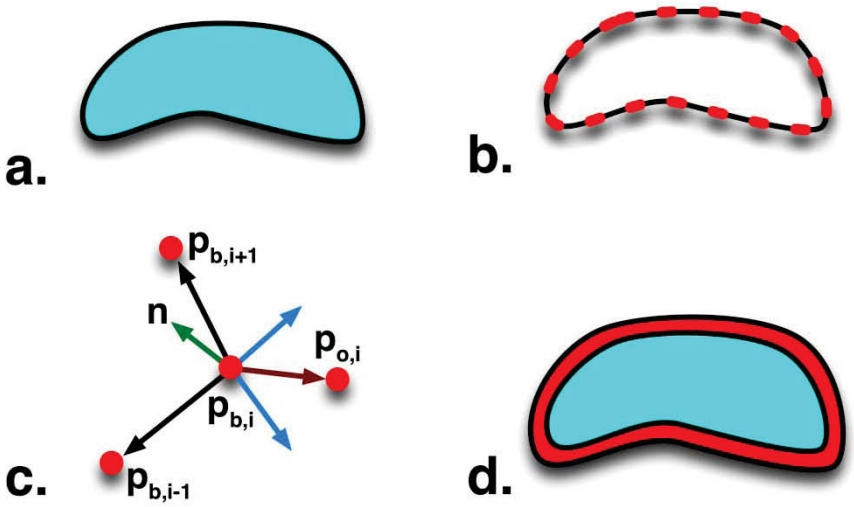

Figure 22. Halo effect process: **a.** The feature geometry is retrieved, **b.** a bounding contour is identified and smoothed, **c.** the halo contour is produced by offsetting each point ($\vec{p}_{b,i}$) along a vector (dark red) to ($\vec{p}_{o,i}$) that is the average of cross products (blue) between bounding contour segments (black) and the view-plane normal (green), and **d.** the feature and halo are displayed.

An inset contour is similarly constructed to fill in any gaps between the outer halo and the feature. Triangular surface elements are then generated to fill the space between contours, producing the halo. A summary of the feature halo illustrative effect process is provided in Algorithm 5. User controls are provided for halo offset/inset thickness (h_o , h_{in}), color value/intensity, and the DCT contour smoothing process. A demonstration of this illustrative effect is presented in Figure 23.

Algorithm 5. Feature halo construction:

1. For feature *f*, obtain bounding contour *bc* using Algorithm 3
2. Apply DCT smoothing to *bc*
3. Initialize offset and inset contours (bc_o, bc_i)
4. **FOR EACH** point $\vec{p}_i$ in *bc*
    1. Apply Eqn. 8 to determine $\vec{p}_{o,i}$ (and equivalent $\vec{p}_{i,i}$)
5. **LOOP**
6. Construct triangles between three contours ($\vec{p}_i$, $\vec{p}_{i,i}$, $\vec{p}_{o,i}$)
7. Add triangles to output mesh geometry and color using user-specified value

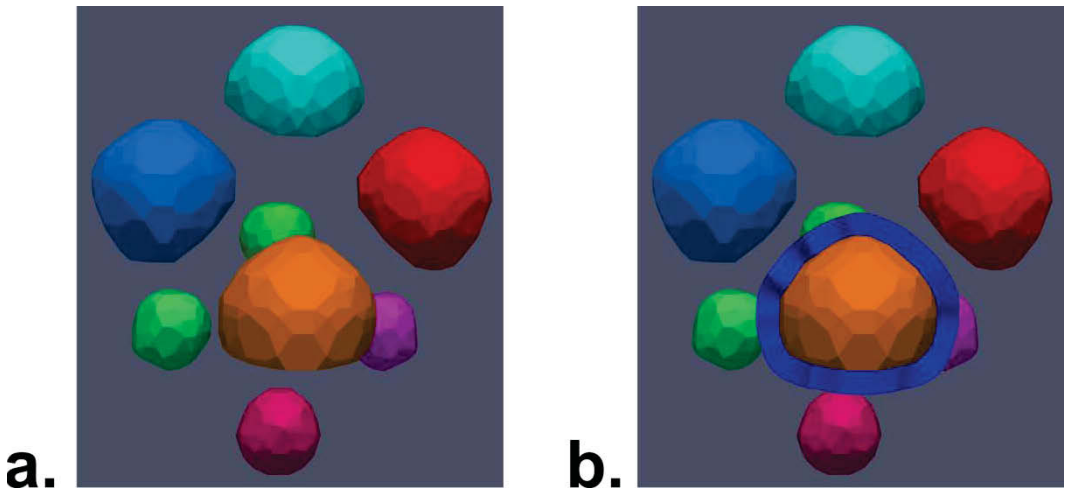

Figure 23. In cases where feature positions are ambiguous (for example, is the green feature on the left side of **(a.)** a small feature in the foreground or a large feature in the background?), feature halos **(b.)** can provide additional depth cues.

4.6 Speedlines Effect

The speedlines effect developed in this study was motivated by prior work from Joshi and Rheingans [3] and Meyer-Spradow et al. [14]. This effect adds trailing line-type elements to a feature to indicate its path of motion and history. Visualizations produced by Joshi and Rheingans took advantage of known histories of individual points in features to draw accurate speedline paths. In the present study, this effect is simplified by only using instantaneous average feature velocity and angular velocity data.

As in previously discussed effects, the speedline algorithm begins by generating a boundary contour for the selected feature, which is smoothed using a Gaussian filter. In general, speedlines are only drawn from the trailing portion of features, so the rear portion of the contour must be extracted. This is achieved by finding the furthest contour points on each side of an axis intersecting the feature centroid ($\vec{C}_k$) parallel to the average feature velocity ($\vec{U}_k$). These points ($\vec{p}_l$, $\vec{p}_r$) are obtained by scanning the contour points for the two that minimize and maximize the distance function by

$$d = \left(\vec{p}_i - \vec{C}_k\right) \cdot \left(\hat{n} \times \vec{U}_k\right). \tag{9}$$

The trailing portion of the bounding contour is defined by these two end-points. Speedline seed points are then evenly distributed along this portion of the contour. Local approximate velocities for each of these seed-points ($\vec{u}_i^*$) are evaluated using average feature velocity and angular velocity in

$$\vec{u}_i^* = \vec{U}_k + \vec{\Omega}_k \times \left(\vec{p}_i - \vec{C}_k \right). \tag{10}$$

Speedlines are then produced using a built-in VTK cone generator with these seed points as base centers, and central axes oriented opposite of estimated velocities ($\vec{u}_i^*$). Cone lengths are set as $\delta \left| \vec{u}_i^* \right|$, where δ is a user defined time-difference. A summary of the speedline effect process is provided in Algorithm 6 and presented graphically in Figure 24. User controls are provided for speedline color/intensity, speedline count, cone base radius, δ, and the number of cone facets. A demonstration of this illustrative effect is presented in Figure 25.

Algorithm 6. Speedlines construction:

1. For feature *f*, obtain bounding contour *bc* using Algorithm 3
2. Apply Gaussian smoothing to *bc*
3. Initialize: $d_{min} \leftarrow \infty$, $d_{max} \leftarrow -\infty$
4. **FOR EACH** point $\vec{p}_i$ in *bc*
    1. Apply Eqn. 9 to determine d_i
    2. **IF** $d_i < d_{min}$ **THEN**
        1. $d_{min} \leftarrow d_i$, $i_{min} \leftarrow d_i$
    3. **ELSE IF** $d_i > d_{max}$ **THEN**
        1. $d_{max} \leftarrow d_i$, $i_{max} \leftarrow d_i$
    4. **END IF**
5. **LOOP**
6. Reduce *bc* to the points between i_{min} and i_{max}
7. Distribute user-specified number of points evenly along *bc*
8. Construct cones with built-in VTK function from seed points using Eqn. 10 and user-specified parameters
9. Add cones to output mesh geometry and color using user-specified value

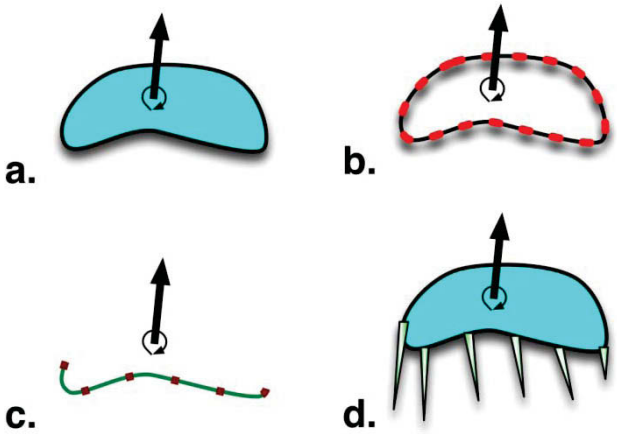

Figure 24. Speedlines effect process: **a.** the feature geometry and motion data are retrieved, **b.** a bounding contour is identified and smoothed, **c.** the receding portion of the contour is extracted and speedline seed points are evenly distributed, **d.** speedline cones are constructed using instantaneous velocity and angular velocity data.

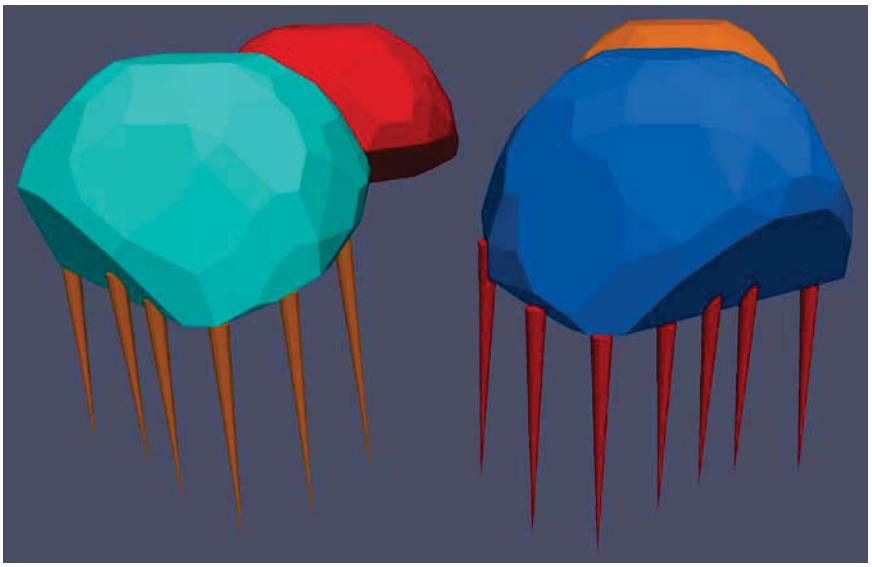

Figure 25. Demonstration of the speedlines illustrative effect, which provides an intuitive indication of feature motion.

4.7 Strobe Silhouettes Effect

The strobe silhouette illustrative effect developed in this study is motivated by earlier work from Joshi and Rheingans [3]. In this effect, the receding portion of a feature boundary is offset multiple times, indicating the history of feature motion. When data is available for the history of specific points of a feature, strobe silhouettes can also present the deformation of a feature over time. In this implementation, as in the speedlines algorithm, a simple approach is developed that only uses instantaneous average feature velocity.

The strobe silhouette algorithm begins by identifying and smoothing a boundary contour around a given feature using a DCT filter. The receding section of the contour is then extracted, as in the speedlines algorithm. The remaining curve (*s*) is then offset away from the direction of motion by the vector $-\delta\vec{U}_{\mathrm{k}}$. For certain features, portions of this offset curve may overlap with the feature, and should be trimmed. First, the rear-most point on the bounding contour with respect to the direction of motion ($\vec{p}_{\mathrm{back}}$) is identified by minimizing the distance function in

$$d = \left(\vec{p}_{\mathrm{i}} - \vec{C}_{\mathrm{k}}\right) \cdot \vec{U}_{\mathrm{k}} . \tag{11}$$

All points in the offset curve with values of *d* greater than that of $\vec{p}_{\mathrm{back}}$ are then removed. The resulting curve is extruded into cylinders as in the tube outline algorithm, added to the mesh, and colored. Subsequent strobe silhouettes are produced by repeatedly offsetting the curve, removing a fraction of outer curve points, smoothing the resulting curve using a Gaussian filter, and generating tubes with reduced diameters. A summary of the strobe silhouette algorithm is provided in Algorithm 7 and presented graphically in Figure 26.

Users can control the silhouette color/intensity, initial tube diameter, number of silhouettes, reduction factor (for curve point count and tube diameter), δ, number of facets per tube, and the DCT contour smoothing process. A demonstration of this effect is presented in Figure 27 for the bubble column dataset.

Algorithm 7. Strobe silhouettes construction:

1. For feature *f*, obtain bounding contour *bc* using Algorithm 3
2. Apply DCT smoothing to *bc*
3. Reduce *bc* to the receding portion (*s*) using Algorithm 6, steps 4-6
4. Offset *s* by $-\delta\vec{U}_k$
5. Scan points $\vec{p}_i$ in *bc* for the minimum value of *d* (Eqn. 11): d_{min}
6. Remove points i from s with $d_i > d_{min}$
7. **FOR** user-specified number of strobe silhouettes
    1. Offset cylinders from *s* with built-in VTK filter and user-specified parameters
    2. Add cylinders to output mesh geometry and color using user-specified value
    3. Offset *s* by $-\delta\vec{U}_k$
    4. Remove user specified portion of end points from *s*
    5. Apply Gaussian smoothing to s
8. **LOOP**

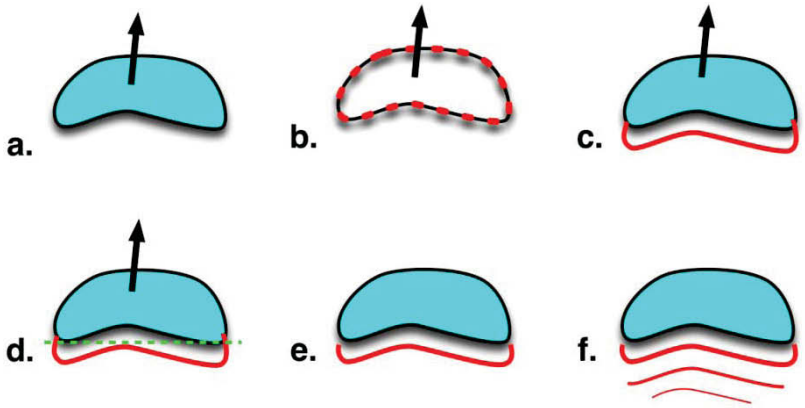

Figure 26. Strobe silhouette effect process: **a.** the feature geometry and velocity data are retrieved, **b.** a bounding contour is identified and smoothed, **c.** the receding portion of the contour is extracted, **d.** the overlapping portion of the silhouette is excised, **e.** the offset silhouette is produced, **f.** steps **c**-**e** are repeated for each silhouette with reduction and simplification stages.

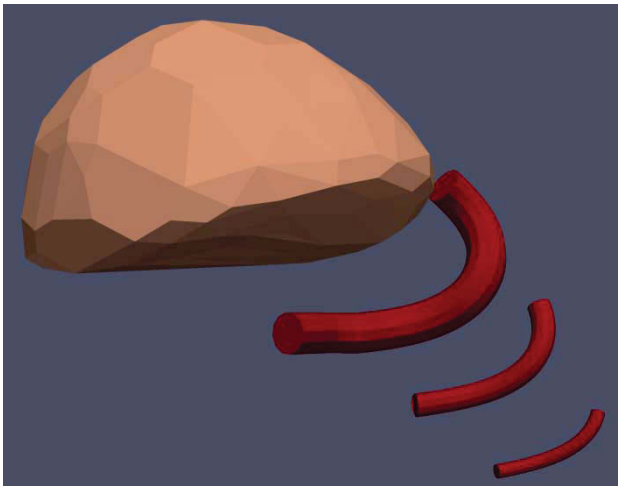

Figure 27. Demonstration of the strobe silhouettes effect, which indicates the motion history of the receding portion of a feature.

5. DISCUSSION

The key advantage of the MarmotViz tool developed in this effort is its flexibility in supporting general ROI identification algorithms, feature matching and tracking algorithms, and illustrative visualization effects. This effect framework permits modification of the underlying mesh geometry in addition to intensities and opacities of output data fields. As such, the current implementation can serve as a basis for experimentation and development of novel algorithms for illustrative visualization. Also, by using the MarmotViz platform, developers can reuse functionality such as the boundary contour detection routine employed in the example effects discussed in this study. Furthermore, by operating in the ParaView environment, developers and users can harness built-in filters and visualization tools and access datasets in a wide variety of file formats and geometries.

While MarmotViz includes a variety of illustrative effects, only single ROI identification and feature matching algorithms are provided in the present implementation. Future plans are to incorporate additional and more sophisticated algorithms for these subsystems; future implementations of MarmotViz could also benefit from an event detection stage after feature matching is performed (Reinders et al. [20]).

While the pipelined nature of ParaView facilitates interaction with other visualization filters and tools, it can be somewhat limiting because developed illustrative techniques cannot operate on prerendered 2-D images. As a further consequence of this architecture, the MarmotViz filter must be manually triggered to update when the view position or orientation changes. Certain illustrative techniques that require more flexible display capabilities, such as the storyboard approach presented by Lu and Shen [6], may also be infeasible in the current implementation.

6. CONCLUSIONS

A generalized framework was developed for illustrative visualization of time varying datasets on unstructured meshes. The framework was implemented in MarmotViz, a ParaView plug-in, and supports the use of generalized ROI identification algorithms, feature matching and tracking algorithms, and illustrative effects. This implementation incorporated a gradient-based ROI identification routine and a novel, adaptive, volume-based feature matching algorithm. A number of illustrative visualization effects were implemented to demonstrate the use and utility of this framework, including: feature coloring, selective visibility, feature smoothing, tube outlines, feature halos, speedlines, and strobe silhouettes.

The current implementation of the MarmotViz plug-in is somewhat limited as it only incorporates basic algorithms for region-of-interest identification and feature matching. Future versions will include additional and more sophisticated algorithms for these subsystems. Additional illustrative techniques will also be incorporated in future releases. An event detection subsystem may improve the functionality of MarmotViz, and enable more sophisticated and expressive illustrative effects. Overall, MarmotViz offers flexible illustrative visualization capabilities to a wide audience and can serve as a springboard for future research and development in the field.

7. REFERENCES

[1] I. Viola and M. E. Gröller, *Smart Visibility in Visualization*, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2005.

[2] U. Dallman, *Topological Structures of Three Dimensional Flow Separations: Deutsches Zentrum fur Luft- und Raumfahrt,* 1983.

[3] A. Joshi and P. Rheingans, "Illustration-inspired techniques for visualizing time-varying data." IEEE Visualization Conference 2005. p. 679-86.

[4] H-G. Pagendarm and B. Walter, "Feature detection from vector quantities in a numerically simulated hypersonic flow field in combination with experimental flow visualization." *IEEE Visualization Conference*, 1994, p. 117–23.

[5] C. D. Correa, D. Silver, and M. Chen, "Feature aligned volume manipulation for illustration and visualization," *IEEE Transactions on Visualization and Computer Graphics*, 2006, 12:1069-76.

[6] A. Lu and H-W. Shen, "Interactive Storyboard for Overall Time-Varying Data Visualization," *IEEE Pacific Visualization Symposium*, 2008, pp. 143–50.

[7] Nucleus_Medical_Media, "Wrist Exposure," FZ00009, *Nucleus Medical Media*.

[8] S. Born, A. Wiebel, J. Friedrich, G. Scheuermann, D. Bartz, "Illustrative Stream Surfaces," *IEEE Transactions on Visualization and Computer Graphics*, 2010, Vol. 16, pp. 1329–38.

[9] M. McGuffin, M. Tancau, and L. Balakrishnan, "Using deformations for browsing volumetric data," *IEEE Visualization Conference*, 2003, pp. 401–408.

[10] A. Stompel, E. B. Lum, M. Kwan-Liu, "Visualization of multidimensional, multivariate volume data using hardware-accelerated non-photorealistic rendering techniques," *IEEE Pacific Conference on Computer Graphics and Applications*, 2002, pp. 394–402.

[11] F. H. Post, F. J. Post, T. V. Walsum, D. Silver, "Iconic Techniques for Feature Visualization," *IEEE Visualization Conference*, 1995, p. 288.

[12] N. A. Svakhine, Y. Jang, D. Ebert, and K. Gaither, "Illustration and photography inspired visualization of flows and volumes," *IEEE Conference on Visualization*, 2005, pp. 687–694.

[13] W-H. Hsu, J. Mei, C. D. Correa, and K. L. Ma, "Depicting Time Evolving Flow with Illustrative Visualization Techniques," Huang F, Wang R-C, editors, *Arts and Technology: Springer Berlin Heidelberg*, 2010. pp. 136–147.

[14] J. Meyer-Spradow, T. Ropinski, J. Vahrenhold, and K. Hinrichs, "Illustrating Dynamics of Time-Varying Volume Datasets in Static Images," *Proceedings of Vision, Modeling and Visualization*, 2006, pp. 333–340.

[15] Z. W. Pylyshyn, *Seeing and Visualizing: It's Not What You Think*, MIT Press, 2003.

[16] A. Joshi and P. Rheingans, "Evaluation of illustration-inspired techniques for time-varying data visualization," *Computer Graphics Forum*, 2008, Vol. 27, pp. 999–1006.

[17] O. Monga, R. Deriche, and J-M. Rocchisani, "3D edge detection using recursive filtering: Application to scanner images," *CVGIP: Image Understanding*, 1991, Vol. 53, pp. 76–87.

[18] D. C. Banks and B. A. Singer, "A predictor-corrector technique for visualizing unsteady flow," *IEEE Transactions on Visualization and Computer Graphics*, 1995, Vol.1, pp. 151–163.

[19] D. Silver and X. Wang, "Visualizing evolving scalar phenomena," *Future Generation Computer Systems*, 1999, Vol. 15, pp. 99–108.

[20] F. Reinders, F. H. Post FH, and H. J. W. Spoelder, "Visualization of time-dependent data with feature tracking and event detection," *The Visual Computer*, 2001, Vol. 17, pp. 55–71.

[21] D. V. Kalivas and A. A. Sawchuk, "A region matching motion estimation algorithm," *CVGIP: Image Understanding*, 1991, Vol. 54, pp. 275–288.

[22] D. Silver and X Wang, "Volume tracking," *IEEE Visualization Conference*, 1996, pp. 157–164.

[23] D. Silver and X. Wang, "Tracking scalar features in unstructured data sets," *IEEE Visualization Conference*, 1998, pp. 79–86.

[24] D. Ebert and P. Rheingans, "Volume illustration: nonphotorealistic rendering of volume models," *IEEE Transactions on Visualization and Computer Graphics*, 2001, Vol. 7, pp. 253–264.

[25] P. Rautek, S. Bruckner, E. Gröller, and I. Viola, "Illustrative visualization: new technology or useless tautology?" *SIGGRAPH Computer Graphics*, 2008, Vol. 42, pp. 1–8.

[26] A. Henderson, "The ParaView Guide, A Parallel Visualization Application," 4th ed, Kitware, Inc.; 2012.

[27] *The VTK User's Guide*. 11th ed: Kitware, Inc.; 2010.

[28] H. Essén, "Average angular velocity," *European Journal of Physics*, 1993 Vol. 14, pp. 201–205.

[29] M. F. Krafft, P. Harris, S. Bougerel. libkdtree++ 0.7.0. 2008.

[30] M. Burns, J. Klawe, S. Rusinkiewicz, A. Finkelstein, and D. DeCarlo, "Line drawings from volume data," *ACM Transactions on Graphics*, 2005 Vol. 24, pp. 508–512.