Advances in Engineering Software 97 (2016) 72–84

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Framework and algorithms for illustrative visualizations of time-varying flows on unstructured meshes

Alexander S. Rattner[a], Donna Post Guillen[b], Alark Joshi[c], Srinivas Garimella[a,*]

[a] Sustainable Thermal Systems Laboratory, George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA
[b] Advanced Process and Decision Systems Department, Idaho National Lab, Idaho Falls, ID 83401, USA
[c] Computer Science Department, College of Arts and Sciences, University of San Francisco, San Francisco, CA 94117, USA

ARTICLE INFO

Article history:
Received 15 June 2015
Revised 29 December 2015
Accepted 15 February 2016

Keywords:
Flow visualization
Illustrative visualization
Feature detection and tracking
Unstructured meshes
Surface rendering
Two-phase flow

ABSTRACT

Photo- and physically realistic techniques are often insufficient for visualization of fluid flow simulations, especially for 3D and time-varying studies. Substantial research effort has been dedicated to the development of non-photorealistic and illustration-inspired visualization techniques for compact and intuitive presentation of such complex datasets. However, a great deal of work has been reproduced in this field, as many research groups have developed specialized visualization software. Additionally, interoperability between illustrative visualization software is limited due to diverse processing and rendering architectures employed in different studies. In this investigation, a framework for illustrative visualization is proposed, and implemented in MarmotViz, a ParaView plug-in, enabling its use on a variety of computing platforms with various data file formats and mesh geometries. Region-of-interest identification and feature-tracking algorithms incorporated into this tool are described. Implementations of multiple illustrative effect algorithms are also presented to demonstrate the use and flexibility of this framework. By providing an integrated framework for illustrative visualization of CFD data, MarmotViz can serve as a valuable asset for the interpretation of simulations of ever-growing scale.

1. Introduction

1.1. Limitations of photo- and physically realistic visualization techniques

The challenge of presenting intelligible and useful visualizations of data has attracted substantial attention due to the growing prevalence of large-scale computational simulations across most fields in science and engineering [1–3]. Such simulations are tackling increasingly ambitious problems that require unsteady modeling in complex 3D geometries or consider inherently unsteady 3D physics such as *direct numerical simulations* of turbulence in fluid mechanics, and necessitate considerations of multiple physical and temporal scales.

Various *physically accurate* or *photo-realistic* visualization techniques have been employed in the investigation and presentation of such simulation data. Vector plots and streamlines/surfaces can indicate the trajectory and magnitude of vector fields (typically velocity). However, these techniques have limited ability to present 3D vector data, because data are projected to a 2D view plane in most display systems [4], and dense lines/arrows can introduce visual clutter. In isosurface rendering, the 3D dataset is reduced to a set of surfaces that lie along some threshold value of a scalar field. This approach can help observers identify regions of interest but may obscure internal features enclosed by isosurfaces. Semi-transparent isosurfaces and cutaway views can help present such features, but application of such corrections can be a user-intensive process. Volume rendering simulates the passage of light through a 3D dataset that is assigned varying color, absorptivity, and transmissivity properties. These optical material properties are specified through a *transfer function*, which may depend on dataset fields. The flexibility of volume rendering enables users to explore and highlight diverse characteristics of interest, but manual tuning of transfer functions can be tedious.

Time-varying simulations present the additional visualization challenge of indicating histories and development of data in a simulation. Animations offer an intuitive approach to presenting time-varying phenomena, when supported by the presentation platform. However, the value of animations may be limited for complex simulations, because research on human vision indicates that viewers have difficulty simultaneously tracking many moving phenomena [5]. Image series are amenable to more presentation environments

* Corresponding author. Tel.: +1 404 894 7479; fax: +1 404 894 8496.
E-mail address: sgarimella@gatech.edu (S. Garimella).

http://dx.doi.org/10.1016/j.advengsoft.2016.02.004
0965-9978/

Nomenclature

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
sc	bounding contour around a feature
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})
v,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})

than animations, but can take longer to interpret [6]. Additionally, individual image size is often limited in a time series, reducing the clarity of the overall visualization.

1.2. Illustrative visualization techniques

The field of *illustrative visualization* has primarily been inspired by historical work in scientific illustration. Born et al. [7] highlighted drawings by Dallman [8] that qualitatively demonstrate key properties of vortical flow using techniques such as half-toning to indicate interior surfaces and dashed lines for hidden sections (Fig. 1a). Similarly, Correa et al. [9] discuss work in medical and anatomical illustrations that reveal internal details clearly by deforming bodies, removing extraneous structures, and coloring tissues in vivid fashion [10, Fig. 1b]. Born et al. [7] summarized the objectives of illustrative visualization as: "simplification of complex contexts, concentration on relevant features, and neglect of details that obstruct understanding." Effective illustrative visualizations should present representations that facilitate this *qualitative partitioning and interpretation*. Similarly, analyses of unsteady phenomena often 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

Considering the goals and desirable features discussed in Section 1.2, a framework for illustrative visualization of time-varying CFD simulation data on unstructured meshes is presented here. In this framework, source simulations are assumed to record data fields (such as pressure, velocity, etc.) for each point or cell in mesh geometries at discrete *time steps*. Following the formulation of Joshi and Rheingans [11], the illustrative visualization process is divided into three stages for each 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, etc.
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.

This illustrative visualization process is distinct from conventional approaches in that it enables application of visualization techniques to specific features, rather than the entire data domain or geometrically defined subdomains. Thus, the resulting visualizations can highlight key elements of simulation data, and better capture the intuitions of subject matter experts. Reviews of research conducted in these three tasks for illustrative visualization are presented in the following sections.

1.4. Prior work in ROI identification in simulation datasets

Monga et al. [12] classified ROI identification algorithms into two groups: those that group "homogeneous" cells and those that identify boundaries around regions of interest. Monga et al. found that it was difficult to identify suitable homogeneity conditions for their target applications in medical imaging; therefore, they pursued boundary detection methods. Their study presented formulations for first order (gradient based) and second order (Laplacian based) boundary detection algorithms. They discussed challenges and resolutions in setting threshold gradient values and boundary closing on rectilinear meshes.

Banks and Singer [13] investigated the problem of identifying vortices in flow simulations. Given the specialized problem domain, physical insights could be used as the basis for ROI identification techniques. Banks and Singer developed a seed-and-growth algorithm in which vortex cores are iteratively adjusted to lie along minimum pressure curves, effectively being transported by centripetal forces. Such physically inspired approaches to ROI identification can yield accurate and "sensible" results, but are often application specific.

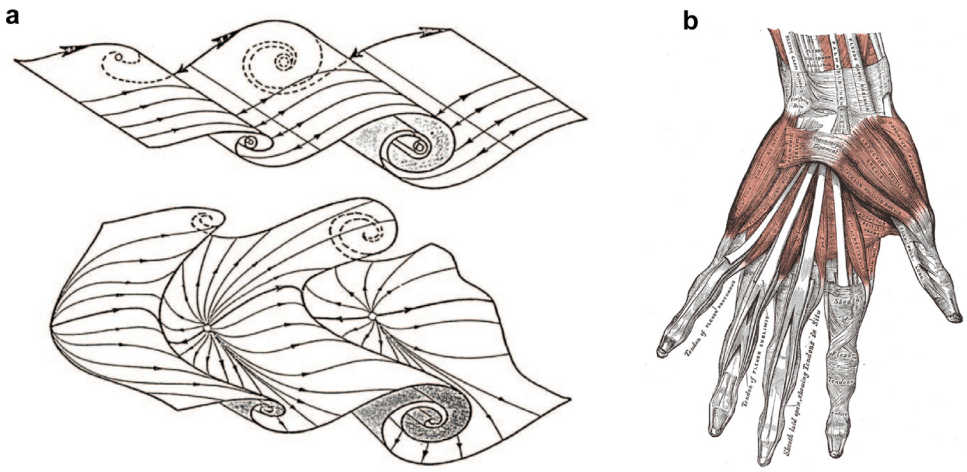

Fig. 1. Examples of scientific illustrations: (a) Illustration of vortical flow employing half-toning and hidden sections [8]. (b) Anatomical illustration demonstrating removed extraneous structures and vivid coloring of tissues [10].

Silver and Wang [14] 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 geometry specific (for connectivity based approaches). In summary, a variety of ROI identification algorithms have been proposed, from generic techniques (*e.g.*, gradient based) to specialized physics-based approaches.

1.5. Prior work in feature matching and tracking in simulation datasets

The feature-matching or *correspondence* problem has also received significant attention in the computer vision community for image processing applications. Reinders et al. [15] investigated pixel- and feature-based approaches. Pixel-based methods measure correspondence of individual cells between nearby ROIs in two dataset time steps. Feature-based approaches operate at a higher level of abstraction, and could range from methods that compare bulk attributes (*e.g.*, 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 specific information about feature history.

Meyer–Spradow et al. [4] developed a cell-based matching technique. In their approach, each cell and its surrounding neighbors in one frame are matched to most similarly valued cells in a second frame. Their approach can track the motion of individual cells in a feature, but is somewhat limited as presented, because it only supports datasets with single features on uniform structured rectilinear grids.

Kalivas and Sawchuk [16] developed a feature-based matching algorithm that measured the difference between 3D regions in a relatively sparse fashion. In their approach, regions are projected into a 2D plane and matched based on minimization of best-fit affine transforms.

Silver and Wang [17] presented a feature-based algorithm for datasets on regular meshes. Their matching algorithm assumes that subsequent time steps are sufficiently close; therefore, ROIs corresponding to the same feature at different times overlap in space, significantly reducing the computational cost of searches. ROIs from two time steps are matched if their relative volume of intersection exceeds some threshold, and pairs with the greatest volume of intersection are matched in the case of multiple overlaps. Silver and Wang [18] published a follow-on study in which they generalized their algorithm to unstructured static mesh geometries and achieved a 20× speedup by using refined data structures to compare features.

Reinders et al. [15] developed a matching algorithm that represented individual ROIs in terms of a few bulk 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 subsequent matching of less-similar ones.

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.

1.6. Prior work in illustrative visualization effects

A wide variety of illustrative visualization effects has 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 [19] 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, they presented vortices (using stream-ribbons), shock-lines, and skin-friction lines.

Post et al. [20] 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, such as replacing complex features with best-fit ellipsoids.

Silver and Wang [14] proposed a number of illustrative techniques for time-varying data for surface- and volumetric-rendering paradigms. In enhanced surface visualization features are assigned distinct surface colors to assist in tracking over multiple time steps. They also suggested rules for coloring features generated from the coalescence or breakup of other features (*e.g.*, red and blue features could combine into a purple feature). Similarly, in enhanced volume rendering, matched features (perhaps from the same bifurcated source feature) can be assigned specific transfer functions for volumetric rendering. In feature isolation, features of

Table 1
Summary of illustrative visualization techniques from literature, categorized by application.

Feature emphasis	Iconic representation [20]	Silhouette enhancement [7,21]
	Enhanced surface visualization [14]	Viewpoint dependent distortion [24]
	Feature coloring [14]	Mixed fields for boundary enhancement [27]
	Feature isolation [14]	2D transfer functions [27]
	Enhanced volume rendering [14]	Segment-aligned cuts [9]
Geometry clarification	Iconic representation [20]	Viewpoint dependent distortion [24]
	Silhouette enhancement [7,21]	Volume splitting and leafer deformation [24]
	Surface shading [21,22]	Geometry deformation [9]
	Distance color blending [21,22]	Surface halftones and hatching [7]
Dynamics cues	Temporal enhancement [22]	Opacity modulation [11,23]
	Speedlines [11]	2D transfer functions [27]
	Strobe silhouettes [11]	Storyboard presentation of data [28]
Other effects	Overlay of multiple fields [19]	Schlieren and shadowgraph techniques [27]
	Mixed rendering techniques [22]	

interest can be emphasized by reducing the color saturation or opacity of nearby features.

Ebert and Rheingans [21] 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 dataset, thus reducing visualization flexibility and possibly increasing computational cost. In their silhouette enhancement technique, the opacity of cells around the silhouette of a feature is increased to assist in distinction between features (also implemented by Stompel et al. [22] and Born et al. [7]). Similarly, in feature halos the region around features, parallel to the view plane, is assigned increased opacity. These halos provide depth cues to the viewer, as features partially obscured by halos are easily recognized to be in the background. In surface shading, sections of feature surfaces facing away from the view direction are assigned desaturated and darkened colors. This provides additional cues about the shape and orientation of features.

Stompel et al. [22] presented a number of illustrative feature enhancement techniques for both surface and volume rendering. In their depth enhancement approach, cells closer to the view plane are rendered with warmer colors to provide additional depth cues to the viewer. In temporal enhancement, cells or surface segments with high temporal derivatives can be rendered in warm colors to indicate regions of rapid change. 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.

Joshi and Rheingans presented a number of visualization techniques geared toward time-varying datasets – particularly those with moving features. In their speedlines technique, superimposed lines and streaks are rendered to indicate feature motion and history. With opacity modulation, instances of features from previous time steps are presented at corresponding locations with increasing transparency and blurriness. This permits the user to observe the history of a feature in a single image. This technique was adapted by Hsu et al. [23]. With strobe silhouettes [11], the receding sections of feature boundaries are offset along the path of motion with reduced thickness and detail.

Viola and Gröller [24] surveyed multiple illustrative visualization techniques. Their study included importance-driven feature enhancement in which 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. In viewpoint-dependent distortion, geometry is distorted to increase the relative size of features of interest, and occluding features may be moved aside [25]. In volume splitting and leafer deformation, outer layers of nested features are split and shifted to permit simultaneous visualization of inner and outer regions [26].

Svakhine et al. [27] investigated a number of illustrative visualization techniques geared toward the specialized problem of visualizing CFD data. They demonstrated the use of Mixed data fields for boundary enhancement. In an example rendering, boundary color was determined from temperature data, and opacity from velocity data. Using two-dimensional transfer functions, "soft" tangent-lines can be generated automatically in a feature by using a periodic function to modulate cell opacity. Svakhine et al. also developed computational techniques to simulate schlieren and shadowgraph "photographic" techniques.

Correa et al. [9] 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. Correa et al. also highlighted the value of using feature data for generating cut-away views of 3D volumetric data without impinging on key geometry.

Representative illustrative visualization techniques are summarized and categorized by application in Table 1. Readers are also directed to a review of illustrative visualization by Rautek et al. [1] that discusses the history of the field and perspectives on its future.

1.7. 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), and cannot be directly extended to applications on unstructured and dynamic meshes, which are frequently employed in CFD. 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 CFD simulations on general unstructured 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 [29] 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. Additionally, ParaView was developed using the Visualization Toolkit (VTK) [30] 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 pre-processing 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 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 processes:

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

A schematic of the MarmotViz program structure is presented in Fig. 2.

The following sections provide detailed descriptions of the subsystems and algorithms developed in this study for ROI identification and assessment (Section 2) and feature matching and tracking (Section 3). Section 4 presents the illustrative effects subsystem and the developed effects, which include: feature coloring and selective visibility, feature smoothing, tube outlines, feature halos, speedlines, and strobe silhouettes.

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

The algorithms developed here are demonstrated using data from a two-phase flow simulation of a bubble-column – a container of liquid with multiple gas injection ports at its base (Fig. 3). In particular, the fluid-phase field will be considered because it yields intuitive *features* for the following discussion, such as bubbles, droplets, and fluid layers.

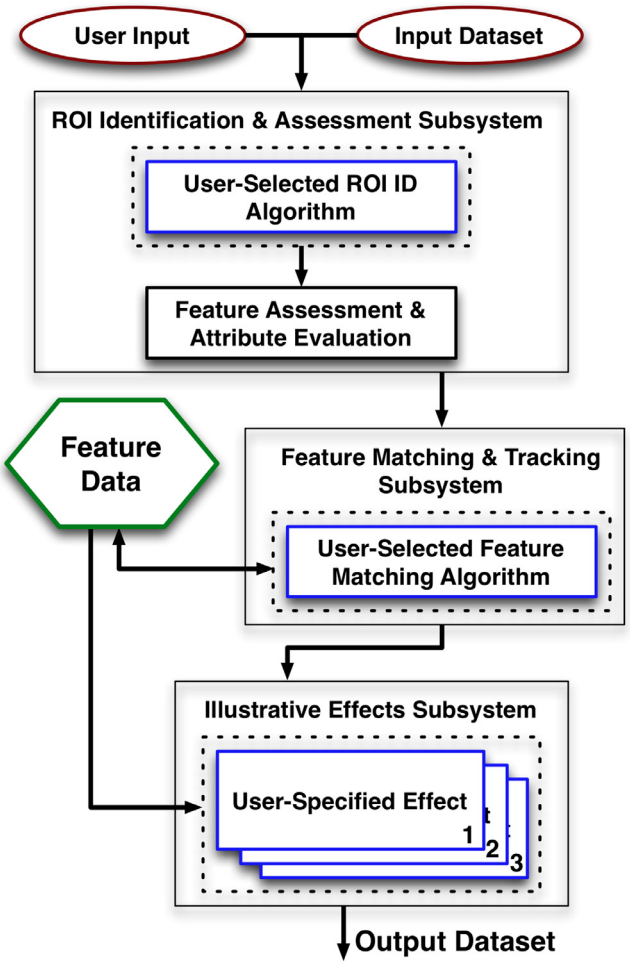

Fig. 2. Schematic of the MarmotViz plug-in structure.

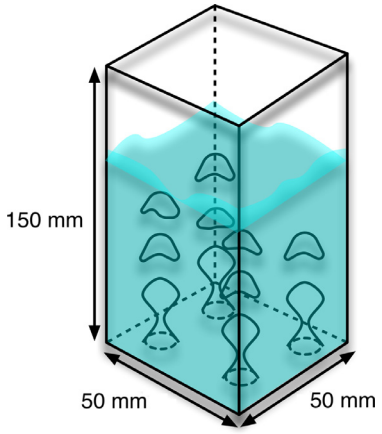

Fig. 3. Schematic of small-scale bubble column simulation employed to demonstrate illustrative visualization effects.

2. Region-of-interest identification and assessment

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 (Section 2.3), 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 Fig. 4.

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. [12]. This routine begins by evaluating the gradient magnitude of a user-specified scalar field such as phase fraction, pressure, or stream function. 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 non-boundary cells that share faces (Fig. 5). This connectivity information can be computationally expensive to

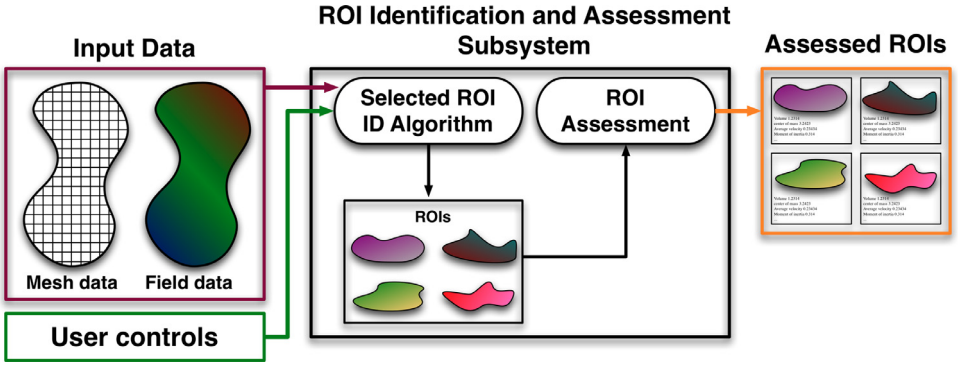

Fig. 4. Schematic of the ROI identification and assessment subsystem.

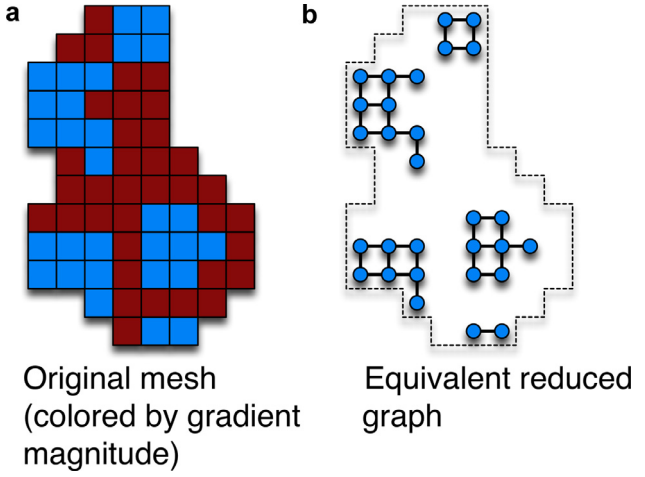

Fig. 5. Transformation from mesh with high gradient-magnitude cells in red (a) to graph with connections between low-gradient magnitude cell face-neighbors (b). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

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 ROI cell count to skip small spurious regions. A second, simple threshold value-based ROI identification algorithm is also implemented.

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.

Because 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. ROI cells are thus, first, subdivided into tetrahedra using built-in VTK routines. Total region-of-interest (k) volumes (V_k) can be computed as sums of tetrahedral volumes (j) composing cells (i) (Eq. (1)).

$$V_k = \sum_{i}^{cells} \sum_{j}^{tets} v_{i,j} \tag{1}$$

ROI centroids ($\vec{C}_k$) and volume-averaged velocities ($\vec{U}_k$) (using cell velocity data, $\vec{u}_i$) are computed similarly using Eqs. (2) and (3).

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

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

Additionally, cell velocity data ($\vec{u}_i$) are used to evaluate ROI volumetric angular momentum ($\vec{L}_k$), products of inertia tensor ($\underline{\underline{J}}_k$), and average angular velocity ($\vec{\Omega}$) using derivations by Essén [31] (Eqs. (4)–(6). Here, coordinates ($\vec{r} = (x, y, z)$) are relative to the ROI centroid.

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

$$\underline{\underline{J}}_k = \sum_{i}^{cells} v_i \begin{bmatrix} y_i^2 + z_i^2 & -x_i y_i & -x_i z_i \\ -x_i y_i & x_i^2 + z_i^2 & -y_i z_i \\ -x_i z_i & -y_i z_i & x_i^2 + y_i^2 \end{bmatrix} \tag{5}$$

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

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.

3. Feature matching and tracking

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 for unmatched ROIs. A representative adaptive volume based feature matching algorithm is presented in Section 3.2. A schematic of the feature matching and tracking subsystem is presented in Fig. 6.

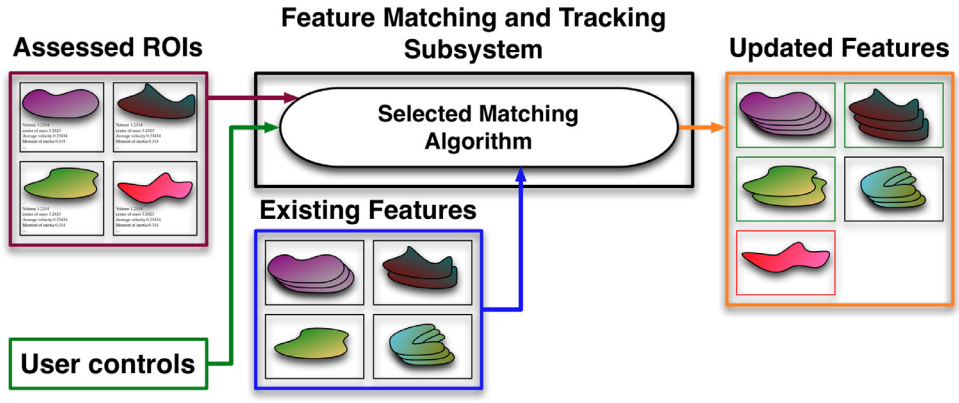

Fig. 6. Schematic of the feature matching and tracking subsystem.

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. To support large simulation datasets, 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 developed by Krafft et al. [32].

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. An expected centroid at t' is evaluated using the average feature velocity ($\vec{C}_k = \vec{C}_k + \vec{U}_k(t' - t)$). 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 [17], because features are not required to overlap spatially. Correspondence between the current region at time t and the feature at t' is evaluated using a volume-based heuristic (α) (Eq. (7)).

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

This value of the matching parameter, α, ranges from 0 when one of the volumes is zero and the other is not, 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 ($\eta \leq \theta \cdot \eta$). 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: $\eta = 0.9$ (first pass value), $\theta = 0.8$, 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 – thus, ROIs are not matched with feature instances that differ in volume by more than 50%.

One of the key advantages of this algorithm is that it dynamically adjusts the matching criterion during operation, avoiding high sensitivity to a single static parameter as in the approach of Silver and Wang [17]. In future revisions, additional matching parameters (such as products of inertia or additional data fields) and feature data from multiple time steps could be considered, as in the algorithm of Reinders et al. [15].

4. Illustrative visualization effects

4.1. Feature coloring and selective visibility

Feature coloring and selective visibility are presented first, because they are independent of the main illustrative effect subsystem that supports application of general effects (Section 4.2).

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 column dataset (Fig. 7). The feature smoothing illustrative effect (Section 4.3) is applied to features in Fig. 7 for aesthetic purposes.

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 [24]. Fig. 8 demonstrates the application of this effect to the bubble column test case to conceal obscuring background features.

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 Fig. 9.

Formulations for five illustrative effets are presented in the following sections to demonstrate the 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 feature visualizations – especially in cases like the example dataset used here, where features (bubbles,

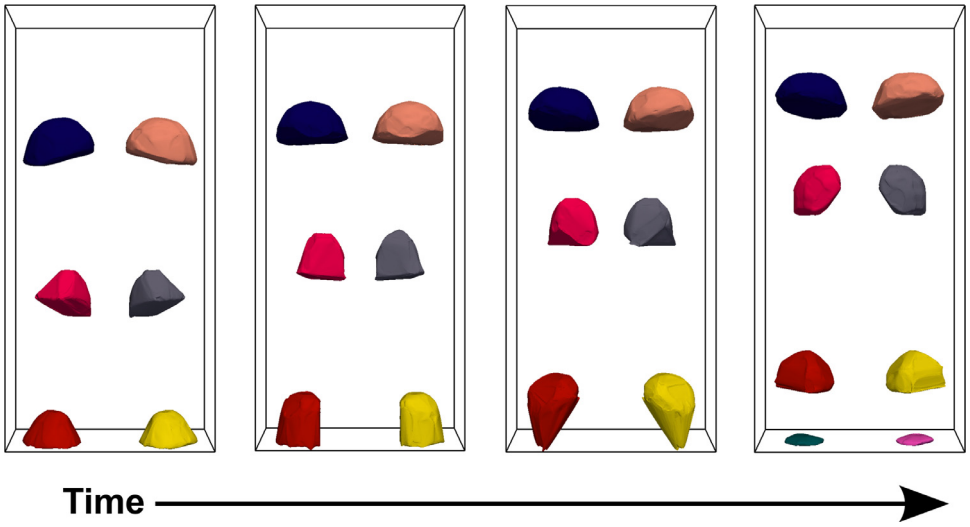

Fig. 7. Feature coloring illustrative effect applied to bubble-column dataset, assigning unique colors to features assists in interpretation of time series or animations.

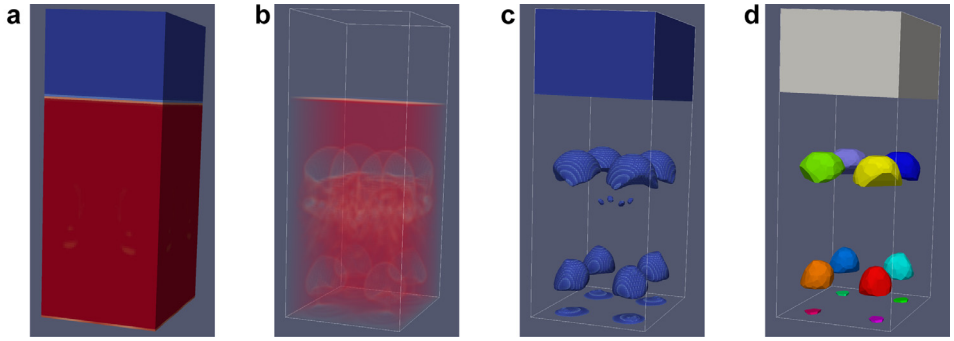

Fig. 8. Application of feature coloring to bubble column dataset: (a) Original dataset with surface rendering. (b) Volume rendering reveals internal features, but with limited clarity. (c) Isosurface rendering can extract certain features of interest, but overlapping features may be difficult to distinguish. (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.

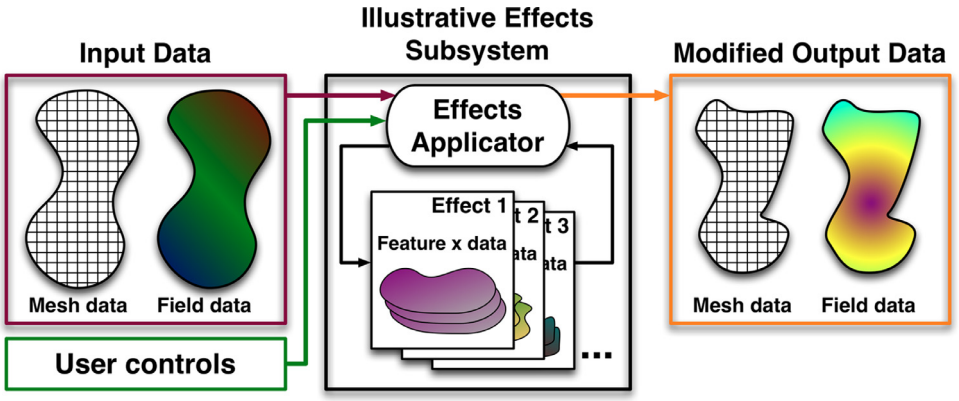

Fig. 9. Schematic of the illustrative effects subsystem. Mesh and field data and user controls are received as input, individual effects are applied sequentially, and the modified mesh and field data are returned as output.

droplets) are physically smooth. Additionally, rendering costs can 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 meshing filter (`vtkDelaunay3D`) 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 (`vtkDataSetSurfaceFilter`), 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. A second, feature smoothing algorithm is also simplemented based on the sphere shrink-wrap approach of Doria [25]. Feature smoothing is employed in conjunction with other illustrative effects in Figs. 7, 8d, 12, 14, 16, and 18. A demonstration of the feature smoothing effect is presented in Fig. 10.

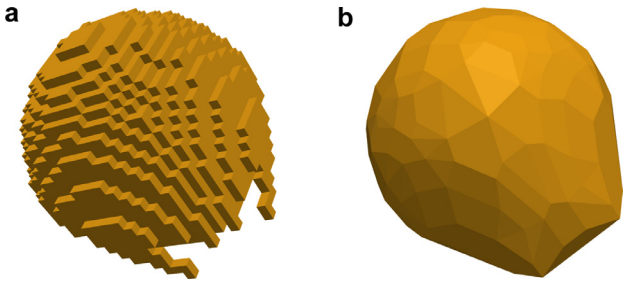

Fig. 10. Features extracted from the mesh geometry may appear jagged or blocky (a), the feature smoothing effect (b) produces more aesthetically pleasing, and often, more physically realistic visualizations, while reducing the number of exterior surfaces that must be rendered.

4.4. Tube outline effect

The tube outline effect developed in this study is motivated by silhouette enhancement techniques discussed in the literature [21, 22]. 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 silhouette enhancement techniques directly to 2D rendered images of datasets [7, 23, 33]. Due to the pipelined nature of ParaView, effects developed in this study are limited to operating on the 3D 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 (silhouette contour) is then obtained in a subroutine.

The silhouette contour determination subroutine begins by constructing three uniform square grids (100×100 in this implementation): `CoveredCells`, `VisitedCells`, and `HeightMap` (Fig. 11a). Next, all exterior face-points and edges on the selected feature are retrieved (previously identified during the ROI assessment process, Fig. 11b). For convenience, a homogeneous transform is applied to these points so that the projection plane normal ($\hat{n}$) is aligned to the z-axis. The algorithm then iterates through all exterior feature edges, and marks intersected grid cells in `CoveredCells` (Fig. 11c). 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 (Fig. 11d). 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 is completed when a previously visited cell is reached – typically the seed point, but potentially another cell for cases with 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` (Fig. 11e). This contour is then transformed back into the global coordinate system (Fig. 11f).

The obtained silhouette 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 (`vtkTubeFilter`) is employed to extrude the contour lines into cylindrical tube sections. The tube contour is then added to the unstructured mesh and colored with a user-defined intensity value. 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 Fig. 12 for the bubble column test case.

4.5. Feature halos effect

The feature halos illustrative effect developed here is motivated by previous work from Ebert and Rheingans [21]. 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 2D surface elements rather than by

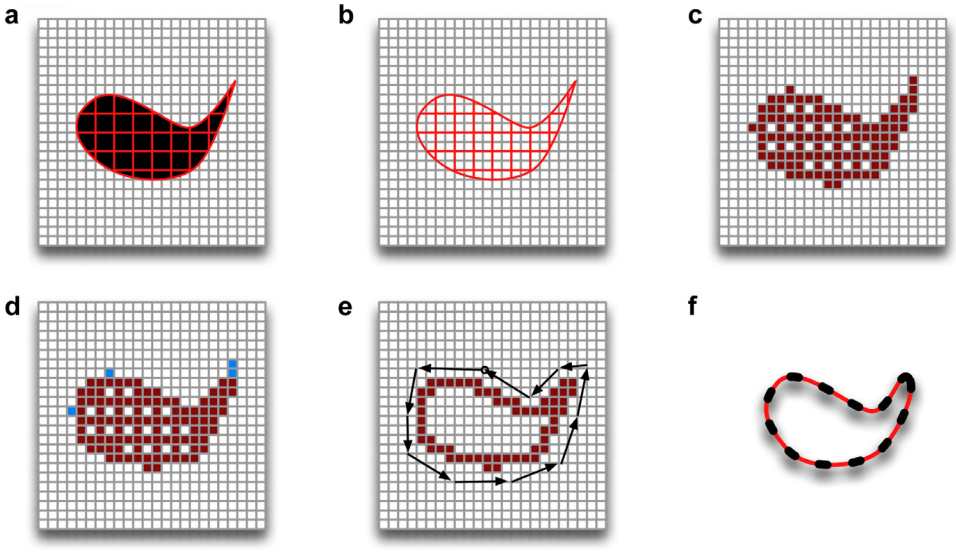

Fig. 11. The silhouette contour identification process: (a) Working grids are constructed. (b) Exterior edges are extracted. (c) Grid cells that intersect edges are marked (and height map is produced). (d) Weakly connected marked cells are removed. (e) The grid is traversed in a counter-clockwise fashion to produce a silhouette contour. (f) The contour is transformed back to the global coordinate system.

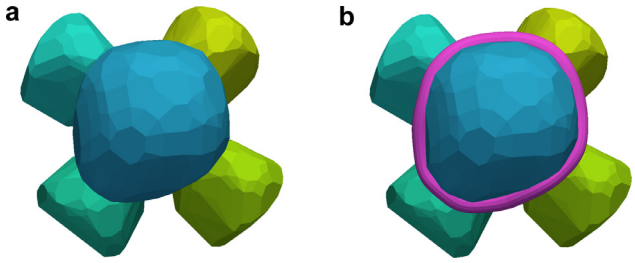

Fig. 12. In cases where features may be difficult to distinguish (a), the tube outline effect (b) can clarify features of interest.

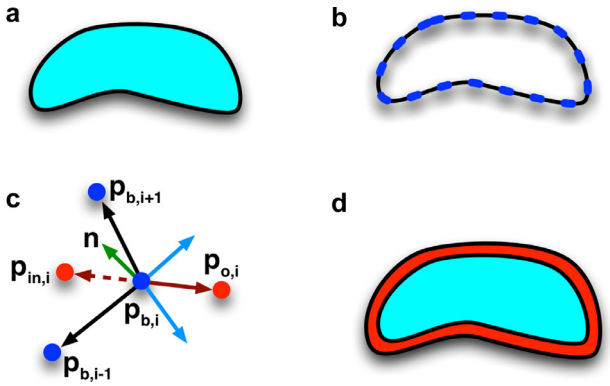

Fig. 13. Halo effect process: (a) The feature geometry is retrieved. (b) A silhouette contour is identified and smoothed. (c) The halo contour is produced by offsetting each point ($\vec{p}_{h,i}$) along a vector (dark red) to ($\vec{p}_{o,i}$) that is the average of cross products (blue) between silhouette contour segments (black) and the view-plane normal (green), an inset contour (points $\vec{p}_{in,i}$) is similarly produced. (d) The feature and halo are displayed. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

modulating the opacity of existing cells in a volumetric rendering paradigm.

The feature halos effect begins by generating a silhouette contour around a feature, reusing the routine described in Section 4.2. The contour is then smoothed using a DCT filter, as in the tube outline effect. The silhouette contour (b) is then offset (o) by thickness h_o using averaged cross products between the view-plane normal ($\hat{n}$) and vectors along the segments connected to each point ($\vec{p}_{b,i}$) on the contour (Eq. (8)). This process requires normalization of a number of intermediate vectors, which is indicated with the **N** operator in Eq. (8) for brevity (*i.e.*, $\mathbf{N}(\vec{v}) = \vec{v}/\|\vec{v}\|$).

$$\vec{p}_{o,i} = \vec{p}_{b,i} + h_o \mathbf{N}\{\mathbf{N}[\mathbf{N}(\vec{p}_{b,i} - \vec{p}_{b,i-1}) \times \hat{n}] + \mathbf{N}[\mathbf{N}(\vec{p}_{b,i+1} - \vec{p}_{b,i}) \times \hat{n}]\} \quad (8)$$

A schematic of the contour offsetting process is presented in Fig. 13.

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. 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 Fig. 14.

4.6. Speedlines effect

The speedlines effect developed in this study was motivated by prior work from Joshi and Rheingans [11] and Meyer-Spradow et al. [4]. This effect adds trailing line-type elements to a feature to indicate its path of motion and history.

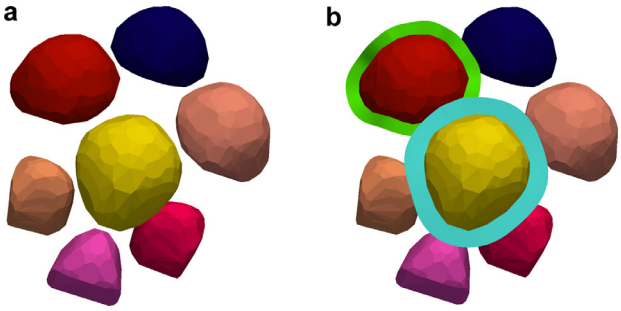

Fig. 14. In cases where feature positions are ambiguous (for example, is the teal feature on the left side of (a) a small feature in the foreground or a large feature in the background?), feature halos (b) can provide additional depth cues.

As in previously discussed effects, the speedline algorithm begins by generating a silhouette 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: Eq. (9).

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

The trailing portion of the silhouette 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 (Eq. (10)).

$$\vec{u}_i^* = \vec{U}_k + \vec{\Omega}_k \times (\vec{p}_i - \vec{c}_k) \quad (10)$$

Speedlines are then produced using a built-in VTK cone generator (`vtkConeSource`) with base centers inset by the base radius toward the feature centroid, preventing overhang past the feature boundary. Central axes for the cones are oriented in a direction opposite to the estimated velocities ($\vec{u}_i^*$). Cone lengths are set as $\delta|\vec{u}_i^*|$, where δ is a user-defined time difference. User controls are provided for speedline color/intensity, speedline count, cone base radius, δ, and the number of cone facets. The speedlines algorithm is summarized graphically in Fig. 15, and a demonstration of this effect is presented in Fig. 16.

4.7. Strobe silhouettes effect

The strobe silhouette illustrative effect developed in this study is motivated by earlier work from Joshi and Rheingans [11]. In this effect, the receding portion of a feature silhouette is offset multiple times, indicating the history of feature motion.

The strobe silhouette algorithm begins by identifying and smoothing a silhouette 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 $-\delta\vec{U}_k$. For certain features, portions of this offset curve may overlap with the feature, and should be trimmed. First, the rear-most point on the silhouette contour with respect to the direction of motion ($\vec{p}_{back}$) is identified by minimizing the following distance function (Eq. (11))

$$d = (\vec{p}_i - \vec{C}_k) \cdot \vec{U}_k \quad (11)$$

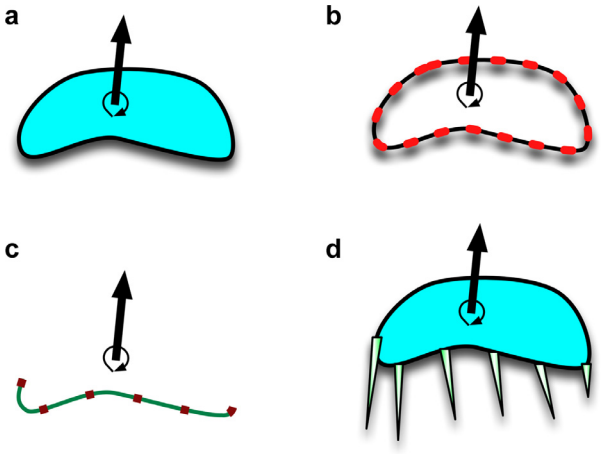

Fig. 15. Speedlines effect process: (a) The feature geometry and motion data are retrieved. (b) A silhouette contour is identified and smoothed. (c) The receding portion of the contour is extracted and speedline seed points are evenly distributed. (d) Speedline cones are constructed on inset seed points using instantaneous velocity and angular velocity data.

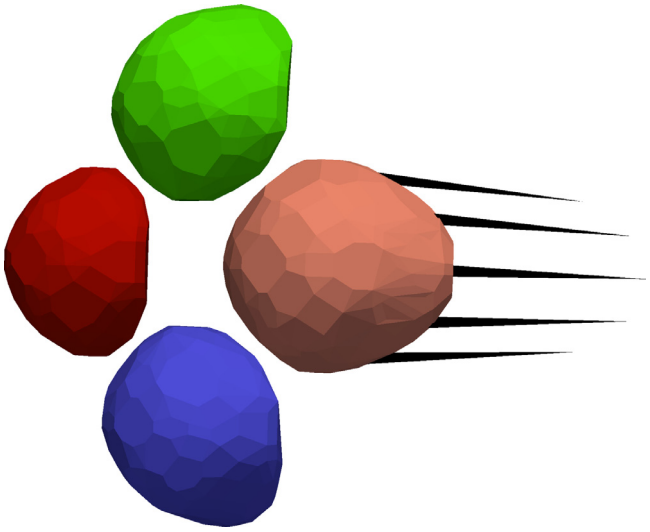

Fig. 16. Demonstration of the speedlines illustrative effect, which provides an intuitive indication of feature motion.

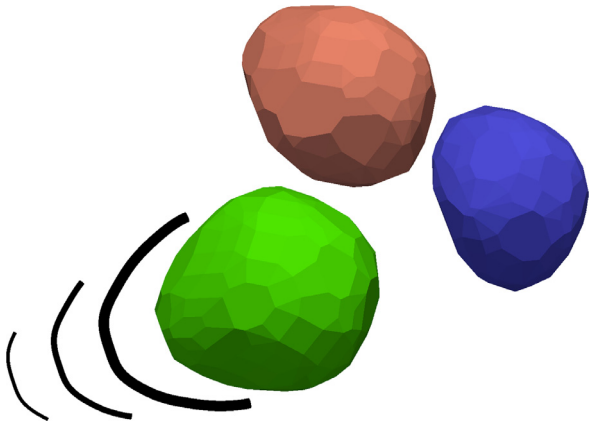

Fig. 18. Demonstration of the strobe silhouettes effect, which indicates the motion history of the receding portion of a feature.

Then, all points in the offset curve with values of d greater than that of $\bar{p}_{\text{back}}$ are removed. The resulting curve is then 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 points from the curve ends (shortening the curve), smoothing the resulting curve using a Gaussian filter, and generating tubes with reduced diameters. A summary of the strobe silhouette algorithm is presented graphically in Fig. 17. Users can control the silhouette color/intensity, initial tube diameter, number of silhouettes, relative reduction in curve length and tube diameter between consecutive silhouettes, δ, number of facets per tube, and the DCT contour smoothing process. A demonstration of this effect is presented in Fig. 18 for the bubble column dataset.

5. Discussion

Illustrative visualization can serve as a valuable asset in the exploration and presentation of CFD simulation data. The ability to autonomously track, selectively visualize, and highlight particular features can aid in the understanding of evolution of phenomena such as bubbly flows and turbulence structures wherein multitudes of similar features are present. Feature halos and similar techniques can also provide depth cues when 3D simulation datasets are rendered in 2D media. Illustrative effects, such as speedlines

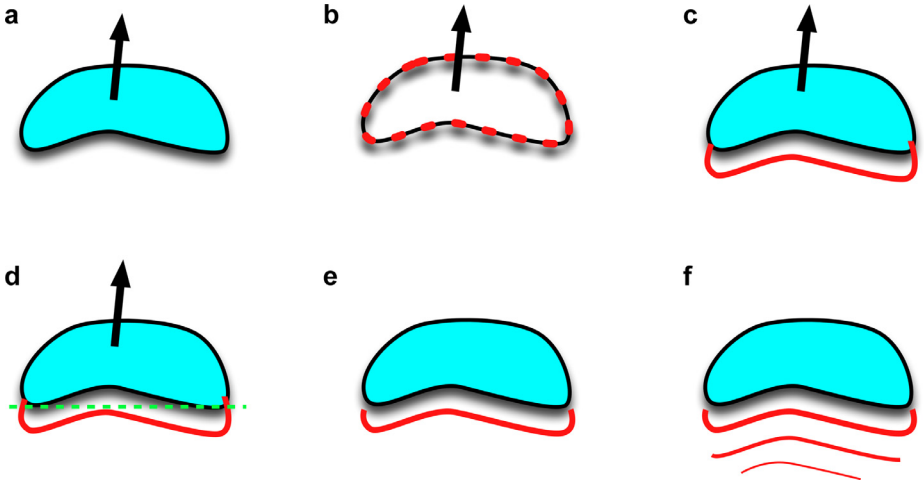

Fig. 17. Strobe silhouette effect process: (a) The feature geometry and velocity data are retrieved. (b) A silhouette contour is identified and smoothed. (c) The receding portion of the contour is extracted. (d) The overlapping portion of the silhouette is excised. (e) The offset silhouette is produced. (f) Steps (c)–(e) are repeated for each silhouette with reduction and simplification stages.

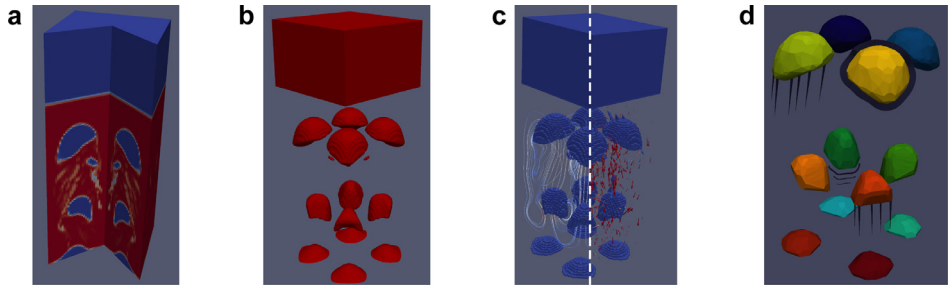

Fig. 19. Conventional visualization techniques such as cut-planes (a) and isosurfaces rendering (b) rely on manual user input. Streamlines and vector plots (c) may significantly increase visual clutter. The illustrative visualization approach (d) enables autonomous feature identification, selective visualization, and compact representation of dynamics.

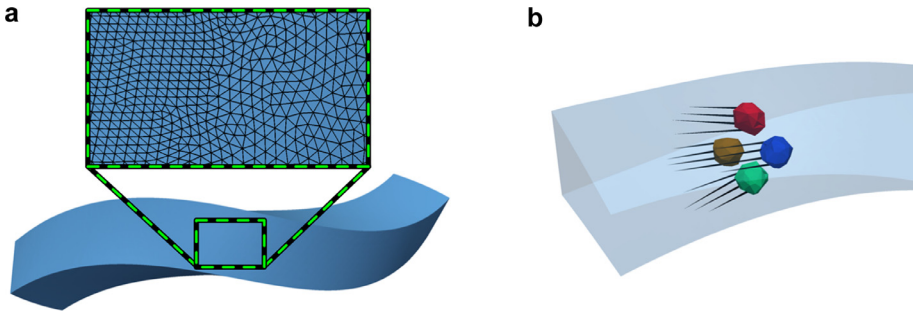

Fig. 20. (a) MarmotViz applied to simulation of bubbly flow in a twisted channel with an unstructured mesh (detail view), (b) resulting visualization of bubbles with speedlines.

and strobe silhouettes, can provide intuitive representations of dynamics with minimal visual clutter in generated renderings. Comparative presentations of the entire bubble column dataset conventional and the developed illustrative visualization techniques are presented in Fig. 19.

The key feature of MarmotViz is its flexibility in supporting general ROI identification algorithms, feature matching and tracking algorithms, and illustrative visualization effects. MarmotViz supports input data on both structured meshes (*e.g.*, the bubble column test case) and unstructured meshes (shown for bubbly flow in a twisted channel, Fig. 20). This 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 silhouette 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, few ROI identification and feature matching algorithms are provided in the present implementation. In the future, we plan to incorporate additional and more sophisticated algorithms for these subsystems. Furthermore, future implementations of MarmotViz could also benefit from an event detection stage after feature matching is performed, as proposed by Reinders et al. [15].

Additionally, while the pipelined nature of ParaView facilitates interaction with other visualization filters and tools, it can be somewhat limiting because the illustrative techniques developed here cannot operate on pre-rendered 2D images. As a further consequence of this architecture, the MarmotViz filter must be manually triggered to update when the view position or orientation changes. Also, certain illustrative techniques that require more flexible display capabilities, such as the storyboard approach presented by Lu and Shen [28], may be infeasible in the current implementation.

6. Conclusions

In this study, a generalized framework was developed for illustrative visualization of time varying CFD 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 enables the exploration and presentation of CFD data in an intuitive fashion, and can serve as a valuable resource for interpreting simulations of ever-growing scale.

Role of funding source

The funding sources: Krell Institute and U.S. Department of Energy Idaho Operations Office did not participate in the design and execution of this study or in the preparation of this manuscript.

Acknowledgments

The authors wish to 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).

References

[1] Rautek P, Bruckner S, Gröller E, Viola I. Illustrative visualization: new technology or useless tautology? SIGGRAPH Comput Graph 2008;42:1–8.
[2] Johnson C, Moorhead R, Munzner T, Pfister H, Rheingans P, Yoo TS. NIH/NSF visualization research challenges. CA: Los Alamitos; 2006.
[3] Johnson C. Top scientific visualization research problems. IEEE Comput Graph Appl 2004;24:13–17.
[4] Meyer-Spradow J, Ropinski T, Vahrenhold J, Hinrichs K. Illustrating dynamics of time-varying volume datasets in static images. of vision. Model Vis 2006:333–40.
[5] Pylyshyn ZW. Seeing and visualizing: it's not what you think. MIT Press; 2003.
[6] Joshi A, Rheingans P. Evaluation of illustration-inspired techniques for time-varying data visualization. Comput Graph Forum 2008;27:999–1006.
[7] Born S, Wiebel A, Friedrich J, Scheuermann G, Bartz D. Illustrative stream surfaces. IEEE Trans Vis Comput Graph 2010;16:1329–38.
[8] Dallmann U. Topological structures of three dimensional flow separations: deutsches zentrum fur luft- und raumfahrt; 1983.
[9] Correa CD, Silver D, Chen M. Feature aligned volume manipulation for illustration and visualization. IEEE Trans Vis Comput Graph 2006;12:1069–76.
[10] Gray H, Lewis WH. Anatomy of the human body. Philadelphia, PA: Lea & Febiger; 1918.
[11] Joshi A, Rheingans P. Illustration-inspired techniques for visualizing time-varying data. In: Proceedings of the IEEE visualization conference; 2005. p. 679–86.
[12] Monga O, Deriche R, Rocchisani J-M. 3D edge detection using recursive filtering: application to scanner images. CVGIP - Image Und 1991;53:76–87.
[13] Banks DC, Singer BA. A predictor-corrector technique for visualizing unsteady flow. IEEE Trans Vis Comput Graph 1995;1:151–63.
[14] Silver D, Wang X. Visualizing evolving scalar phenomena. Future Gener Comput Systs 1999;15:99–108.
[15] Reinders F, Post FH, Spoelder HJW. Visualization of time-dependent data with feature tracking and event detection. Vis Comput 2001;17:55–71.
[16] Kalivas DS, Sawchuk AA. A region matching motion estimation algorithm. CVGIP - Image Und 1991;54:275–88.
[17] Silver D, Wang X. Volume tracking. Proceedings of the IEEE visualization conference 1996:157–64.
[18] Silver D, Wang X. Tracking scalar features in unstructured data sets. In: Proceedings of the IEEE visualization conference; 1998. p. 79–86.
[19] Pagendarm H-G, Walter B. Feature detection from vector quantities in a numerically simulated hypersonic flow field in combination with experimental flow visualization. In: Proceedings of the IEEE visualization conference; 1994. p. 117–23.
[20] Post FH, Post FJ, Walsum TV, Silver D. Iconic techniques for feature visualization. In: Proceedings of the IEEE visualization conference; 1995. p. 288–95.
[21] Ebert D, Rheingans P. Volume illustration: nonphotorealistic rendering of volume models. IEEE Trans Vis Comput Graph 2001;7:253–64.
[22] Stompel A, Lum EB, Kwan-Liu M. Visualization of multidimensional, multivariate volume data using hardware-accelerated non-photorealistic rendering techniques. In: Proceedings of the IEEE pacific conference on computer graphics and applications; 2002. p. 394–402.
[23] Hsu W-H, Mei J, Correa CD, Ma K-L. Depicting time evolving flow with illustrative visualization techniques. In: Huang F, Wang R-C, editors. Arts and technology. Springer Berlin Heidelberg; 2010. p. 136–47.
[24] Viola I, Gröller ME. Smart visibility in visualization. Comput Aesthet Graph Vis Imag 2005:209–16.
[25] Doria D. ConvexHullShrinkWrap, http://www.vtk.org/Wiki/VTK/Examples/Cxx/PolyData/ConvexHullShrinkWrap; 2010 [accessed 12.28.2015].
[26] McGuffin M, Tancau M, Balakrishnan L. Using deformations for browsing volumetric data. In: Proceedings of the IEEE visualization conference; 2003. p. 401–8.
[27] Svakhine NA, Jang Y, Ebert D, Gaither K. Illustration and photography inspired visualization of flows and volumes. In: Proceedings of the IEEE visualization conference; 2005. p. 687–94.
[28] Lu A, Shen H-W. Interactive Storyboard for Overall Time-Varying Data Visualization. In: Proceedings of the IEEE pacific visualization conference; 2008. p. 143–50.
[29] Henderson A. the paraview guide. A parallel visualization application. 4th ed. Kitware, Inc.; 2012.
[30] Kitware, Inc.. The VTK user's guide. 11 ed. Kitware, Inc.; 2010.
[31] Essén H. Average angular velocity. Eur J Phys 1993;14:201–5.
[32] Krafft M.F., Harris P., Bougerel S. libkdtree++ 0.7.0. 2008.
[33] Burns M, Klawe J, Rusinkiewicz S, Finkelstein A, DeCarlo D. Line drawings from volume data. ACM Trans Graph 2005;24:512–18.