Submitted to *Computers*. Pages 1 - 21.

OPEN ACCESS

Article

FoamVis, A Visualization System for Foam Research: Design and Implementation

Dan R. Lipşa[1,*], Richard C. Roberts, Robert S. Laramee [1,*]

[1]Visual and Interactive Computing Group, Department of Computer Science, Swansea University, Swansea, UK

* Authors to whom correspondence should be addressed; {d.lipsa, r.s.laramee}@swansea.ac.uk

Version February 24, 2015 submitted to Computers. *Typeset by LaTeX using class file mdpi.cls*

Abstract: Liquid foams are used in areas such as mineral separation, oil recovery, food and beverage production, sanitation and fire fighting. To improve the quality of products and efficiency of processes in these areas, foam scientists wish to understand and control foam behavior. To this end, foam scientists have used foam simulations to model foam behavior, however, analyzing these simulations presents difficult challenges. We describe the main foam research challenges and present the design of FoamVis, the only existing visualization, exploration and analysis application created to address them. We describe FoamVis' main features together with relevant design and implementation notes. Our goal is to provide a global overview and individual features implementation details that would allow a visualization scientist to extend the FoamVis system with new algorithms and adapt it to new requirements. The result is a detailed presentation of the software that is not provided in previous visualization research papers.

Keywords: FoamVis; Surface Evolver; bubble-scale foam simulation; time-dependent visualization

1. Introduction and Requirements

Liquid foams have important practical applications in areas such as oil extraction, mineral separation, food and beverage production, cleaning and fire safety [24]. In oil extraction, foam is pushed through porous rock to displace oil [17]. Domain experts desire to understand how the constricted geometry of the rock affects the flow of foam. Foam is used in mineral

separation [15] in a process where ground ore is treated with foam. The efficiency of the separation between mineral and rock depends on how objects with different properties interact with foam.

Liquid foam behavior is not yet well understood. Scientists try to determine foam behavior from measurable properties such as bubble size and distribution, liquid fraction, and surface tension. One way to study this dependence is to simulate foams at the bubble-scale, which makes it possible to model foam properties and see their influence on general foam behavior. However, it also poses challenges for visualizing and inferring generic foam response. Foam is simulated at a small scale, where each bubble is modeled individually, yet the goal is to determine behavior at a large scale, where foam can be described as a continuous medium.

Surface Evolver (SE) [2] is the *de facto* standard for simulating foams at the bubble-scale. SE foam simulations pose specific challenges:

1. Access to simulation data is difficult and requires domain-specific knowledge. Parsing and special processing are required to access the entire simulation data. Important bubble attributes are not provided by the simulation but inferred using domain specific-knowledge.

2. It is challenging to visualize general foam behavior. While bubble-scale simulation makes it possible to investigate the influences that material properties have on general foam behavior, it makes it difficult to visualize the general behavior that is of primary interest. Simulation data is complex (unstructured grid with polygonal cells) and time-dependent, with large fluctuations in the values of the parameters determined by changes in the topology of the soap film network.

3. Triggers to various foam behaviors are difficult to infer. Multiple attributes have to be examined and foam properties have to be taken into account. Topological changes (T1s), in which bubbles swap neighbors, have to be considered.

4. Foam scientists work with dozens of simulations with a wide range of simulation parameters. Examples include foam container properties (such as shape and roughness), foam attributes (such as bubble size and distribution, liquid fraction and surface tension) or the properties of objects interacting with foam (such as shape, size and position). The large number of existing simulations and the variety of simulation parameters makes it difficult to manage simulation data. The possibility to compare related datasets results in a better understanding of various foam behaviors, however existing tools do not facilitate that.

These challenges make it difficult to use a general-purpose visualization tool for foam research. Domain experts' visualizations only partially address these challenges. They may require intervention in the simulation code and potentially recomputing the simulations for summarizing and saving the relevant data. Their standard visualizations do not have the ability to explore and analyze the data and do not facilitate comparison of datasets. They do not have the high level of detail and speed that is achieved using graphics hardware. We address shortcomings of existing visualizations used by domain experts and we provide visualizations to address foam research challenges. To the best of our knowledge, no previous visualization software exists for foam simulations modeled with SE. FoamVis [10,11,13] fills this void by providing a comprehensive solution which facilitates advanced examination, visualization,

analysis and comparison of foam simulation data. This paper presents design and implementation details required for understanding the software not found in previous literature.

The design and implementation of the software is not featured in the previous literature which focuses on visualization. We present a software-centric view of FoamVis which is essential for future developers wishing to implement or extend this framework.

The rest of this paper is organized as follows: We describe how our design choices meet foam research challenges and provide an overview of the implementation in Sec. 3.1. In the next sections we present design and implementation details for parsing and processing, interface, visualizations (simulation attributes, bubble paths, time-average, topological changes kernel density estimate (KDE) and, histograms), multiple linked-views, and user interaction. We end with conclusions and future work (Sec. 4).

2. Related Work

In this work we aim to provide a global overview and individual features implementation details for FoamVis, a visualization tool for foam research. Our description is based on previous visualization literature, the source code documentation [9] and the source code itself. In a previous publication [11] we describe the foam research application area and introduce FoamVis, a novel application that provides various techniques for visualization, exploration and analysis of time-dependent 2D foam simulation data. We show new features in foam simulation data and new insights into foam behavior discovered using our application. Features described include: color-mapping of scalar attributes, display of topological changes, visualization of bubble paths, multiple-linked views and histograms.

Next, we describe extensions [13] to FoamVis that allow comparison of related simulations and enhance its analysis capabilities. Comparative visualization features include: the two halves view, linked time with event synchronization, the reflection feature, force difference and torque visualizations. Additional visualization and analysis features include: deformation tensor computation and visualization using ellipses, time-average computation for vector and tensor simulation attributes, velocity vector visualizations using glyphs and streamlines, average around moving objects and, topological changes kernel density estimate visualization.

Solutions to visualize and analyze 3D foam simulations are described in a third [10] paper. Three-dimensional visualization include color-mapping of scalar attributes, location and type for topological changes, visualization of velocity vectors using glyphs, average of scalar and vector simulation attributes and topological changes kernel density estimate. A description of FoamVis from a user's perspective is also presented [12]. Again, none of the previous literature provides guidance on how the implement the features.

3. Design and Implementation

Our visualization solutions are driven by the foam research and visualization challenges listed in Sec. 1. Surface Evolver output files are parsed and processed to access the complete data generated by the simulation. Our application works with any SE simulation and no changes to the simulation output are necessary to accommodate the application. This processing addresses challenge one.

We visualize important simulation attributes (Sec. 3.4) which include bubble scalar measures, bubble velocity (a vector), bubble deformation (a tensor), location of topological changes and forces acting on objects in foam. Overall foam behavior is analyzed using the average feature (Sec. 3.6), kernel density estimate for topological changes (Sec. 3.7) and bubble paths (Sec. 3.4). This addresses challenge two.

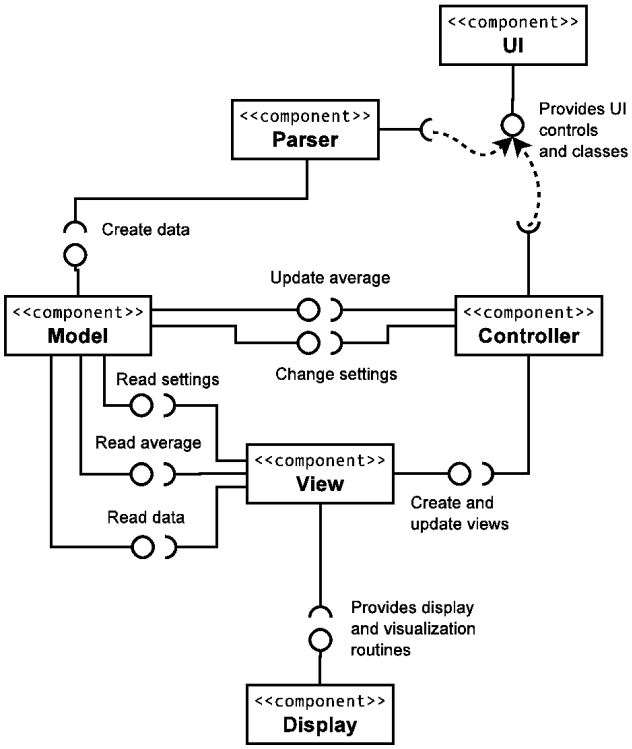
Foam scientists wish to understand what triggers certain behavior in foam simulations (challenge three). Foam behavior is studied by either examining different attributes that influence it or by comparing simulations (challenge four) where the behavior is varied by modifying simulation parameters. Both these requirements are addressed using multiple linked-views (Section 3.9).

To present our solutions for visualization of foam simulation data, we use three simulation groups containing related simulations: the falling discs and the falling ellipse (2D), constriction (2D), and the falling disc (2D) and the falling sphere (3D). The falling-objects simulation group contains the *falling-ellipse* and the *falling-discs* simulations (Fig. 6). The *falling-discs* simulates two discs falling through a monodisperse (bubbles having equal volume) foam under gravity. It contains 330 time steps and simulates 2200 bubbles. The two discs are initially side-by-side and in close proximity. As they fall, they interact with the foam and each other by rotating towards a stable orientation in which the line that connects their centers is parallel to gravity. The *falling-ellipse* simulates an ellipse falling through a monodisperse foam under gravity. This dataset contains 540 time steps and simulates 600 bubbles. The major axis of the ellipse is initially horizontal. As the ellipse falls, it rotates toward a stable orientation in which its major axis is parallel to gravity. The constriction dataset contains two simulations, one with a *square-constriction* and one with a *rounded-constriction* (Fig. 8). They simulate a 2D polydisperse (bubbles with different volumes) foam flowing through a constricted channel, with 725 bubbles and 1000 time steps. The radius of the curvature of the rounded corners of the constriction is five times smaller for the square-constriction compared with the rounded-constriction. The falling disc (2D) / sphere (3D) simulate a disc/sphere falling through a monodisperse (bubbles having equal volume) foam under gravity. In 2D we have 254 time steps and 1500 bubbles. In 3D we have 208 time steps and 144 bubbles. Note that the number of bubbles that scientists are able to simulate in 3D is severely restricted by the duration of the computation time.

3.1. Overview In this section we present the structural relationships between FoamVis' main components (Fig. 1). For this purpose we use a UML 2 components diagram [1]. Briefly, a component represented in our diagram as a rectangle, is a design unit that is typically implemented using a replaceable module. A component may provide one or more public interfaces, represented with a complete circle at their end (lollipop symbol). Provided interfaces represent services that the component provides to its clients. Similarly, a component may require services from other components. These services are formalized through the required interfaces, represented with a half circle at their end (socket symbol).

FoamVis starts by executing the **Parser** component. This component, uses services from the **UI** component to allow the user to specify the simulations to be analyzed and additional information about the simulations. This is done either through the command line or through graphical user interface. Then, the **Parser** parses the specified simulation files, creates an in-memory representation of the simulation data and yields the execution to the Controller module.

Figure 1. FoamVis UML Component Diagram. The **Parser** parses simulation data and stores it in memory. FoamVis uses the Model-View-Controller design pattern to separate the data and program state (**Model**), the presentation (**View**), and the interaction with the user (**Controller**) in three different components. The **UI** provides user interface controls and classes and the **Display** provides display and visualization algorithms.



The main logic of the program uses the Model-View-Controller design pattern [14]. This pattern separates the data and program state (**Model**), the presentation (**View**), and the interaction with the user (**Controller**) in three different components. This architecture has two main benefits. First, because views are separated from data, several views of the same data can be displayed in the same time. Second, because the **Model** does not depend on the **View** or **Controller** components, changing the user interface or adding new views generally do not affect the **Model**. This results in a more modular and maintainable code and in quicker development cycle.

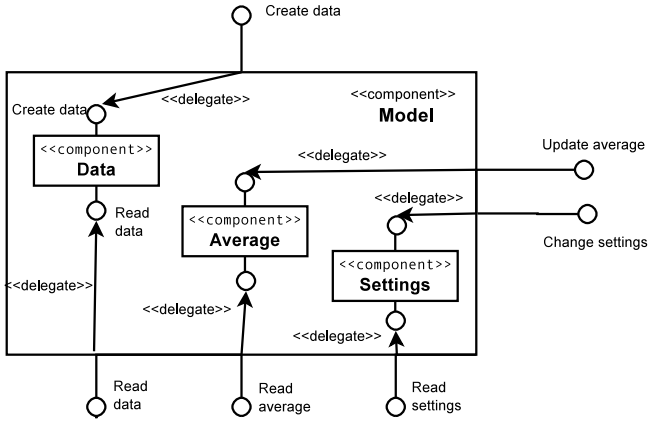
The **Controller** manages the interaction between a user, the **Model** (that stores data and program state) and the views that show foam simulation data.

The **Model** component (Fig. 2) is composed of three sub-components: **Data** which provides interfaces to create the in-memory representation of the foam simulation data and to read that data; **Settings** which stores the program state; and **Average** which stores and provides interfaces to compute and read time-averages of simulation attributes.

The **View** component provides visualizations for 2D and 3D foam simulation data as well as histograms for scalar attributes.

Each of these logical components contains several implementation files which in turn contain one or several related C++ classes. Logical components (modules), their implementation files and classes, and groups of member functions (member groups) are also documented [9] using Doxygen [22]. We are going to refer to the doxygen documentation as we describe the main features of the program and present

Figure 2. The **Model** Component. This component is responsible for storing data and program state. It is composed from three subcomponents: **Data** which stores simulation data, **Average** which stores derived time-average of simulation attributes, and **Settings** which stores program state.



implementation notes for those features. Here we provide a brief summary of the main components of the program: `Parser`, `Model`, `View` and `Controller`. For brevity we omit the `Display` and `UI` components. We include the name and a brief description for each file part of a component. We use a file name without an extension, to refer to both the interface (`.h`) and the implementation (`.cpp`) files with that name.

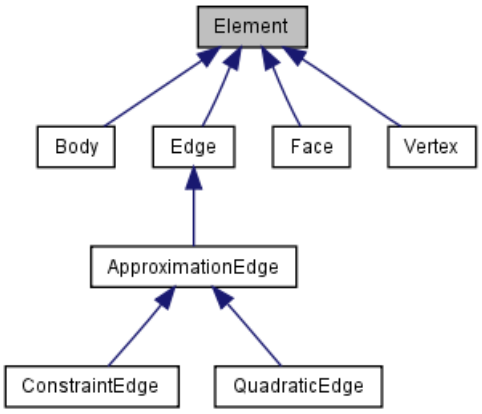
The **Parser** parses Surface Evolver `.dmp` files and calls the **Data** component to build a memory representation of the simulation data. It contains the following files:

- AttributeCreator – Create attributes which can be attached to vertices, edges, faces, and bodies.
- AttributeInfo – Information about attributes for vertices, edges, faces, and bodies.
- EvolverData.l – Lexical analyzer for parsing a `.dmp` file produced by Surface Evolver.
- EvolverData.y – Grammar for parsing a `.dmp` file produced by Surface Evolver.
- ExpressionTree – Nodes used in an expression tree built by the parser.
- main.cpp – Drives parsing of SE .dmp files and creates FoamVis main objects.
- NameSemanticValue – Tuple (name, type, value) used for a vertex, edge, face, and body attribute.
- ParsingData – Stores data used during the parsing such as identifiers, variables and functions.
- ParsingDriver – Drives parsing and scanning.
- ParsingEnums – Enumerations used for parsing.

The **Controller** manages the interaction between a user, the **Model** (that stores data and program state) and the views that show foam simulation data. This component contains one implementation file:

- MainWindow – Stores the OpenGL, Vtk and, Histogram widgets, implements the Interface and manages the interaction between a user, the **Model** and the views.

Figure 3. `Element` class Inheritance Graph. This class stores a vector of attributes that can be attached to bodies (bubbles), faces, edges, and vertices. This diagram also shows the three types of edges represented in FoamVis: regular (`Edge`) edges that have a begin and an end vertex, quadratic edges (`QuadraticEdge`) that have an additional middle vertex, and constraint edges (`ConstraintEdge`) that are described using a begin vertex, an end vertex and a curve $f(x, y, z)$ on which the edge lies.



The **Data** component creates, processes and stores foam simulation data. It contains the following implementation files:

- AdjacentBody – Keeps track of all bodies a face is part of.
- AdjacentOrientedFace – Keeps track of all faces an edge is part of.
- ApproximationEdge – Curved edge approximated with a sequence of points (Fig. 3).
- Attribute – Attribute that can be attached to vertices, edges, faces and bodies.
- Body – A bubble.
- BodyAlongTime – A bubble path.
- ConstraintEdge – Edge on a constraint approximated with a sequence of points (Fig. 3).
- DataProperties – Basic properties of the simulation data such as dimensions and if edges are quadratic or not.
- Edge – Part of a bubble face, stores a begin and an end vertex (Fig. 3).
- Element – Base class for Vertex, Edge, Face and Body. Stores a vector of attribute (Fig. 3)
- Face – A bubble is represented as a list of faces, a face is an oriented list of edges.
- Foam – Stores information about a time step in a foam simulation.
- ForceOneObject – Forces and torque acting on one object.
- ObjectPosition – Stores an object interacting with foam position and rotation

- OOBox – An oblique bounding box used for storing a torus original domain.
- OrientedEdge – An oriented edge. Allows using an Edge in direct or reversed order.
- OrientedElement – Base class for OrientedFace and OrientedEdge. Allows using a Face or Edge in direct or reversed order.
- OrientedFace – An oriented face. Allows using a Face in direct or reversed order.
- ProcessBodyTorus – Processing done to "unwrap" bodies in torus model.
- QuadraticEdge – Quadratic edge approximated with a sequence of points (Fig. 3).
- Simulation – A time-dependent foam simulation.
- T1 – A topological change.
- Vertex – Element used to specify edges. An edge has at least two vertices, begin and end. A quadratic edge has a middle vertex as well.

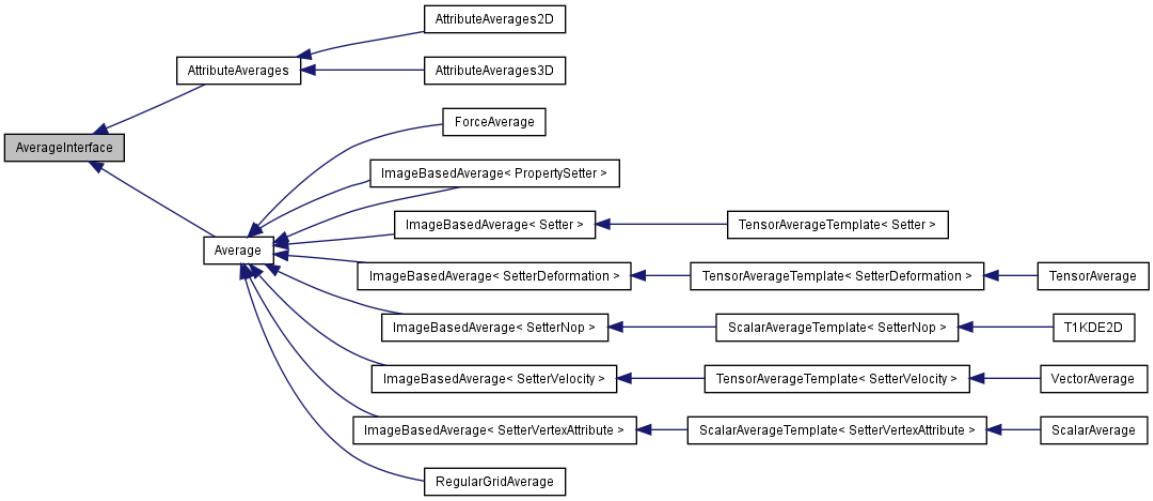
The **Settings** component stores and provides access to program state. This component is composed from the the following files:

- BodySelector – Functors that specify selected bubbles.
- Settings – Settings that apply to all views.
- ViewSettings – Settings that apply to one view.

The **Average** component computes time-average of simulation attributes. It contains:

- AttributeAverages – Computes the average for several attributes in a view. Base class for AttributeAverages2D and AttributeAverages3D (Fig. 4).
- AttributeAverages2D – Computes the average for several attributes in a 2D view. Casts the computed averages to the proper 2D types (Fig. 4).
- AttributeAverages3D – Computes the average for several attributes in a 3D view. Casts the computed averages to the proper 3D types (Fig. 4).
- Average – Computes a time-average of a foam attribute. Base class for 2D and 3D time-average computation classes (Fig. 4).
- AverageInterface – Interface for computing a time-average of a simulation attribute (Fig. 4).
- AverageShaders – Shaders used for computing a pixel-based time-average of attributes.
- ForceAverage – Time-average for forces acting on objects interacting with foam (Fig. 4).
- ImageBasedAverage – Calculates a pixel-based time-average of 2D foam using shaders (Fig. 4).

Figure 4. `AverageInterface` Inheritance Graph. This class provides the interface for updating an average of attributes. `AttributeAverages` stores an average of many simulation attributes. `Average` provides common computation for averages of forces (`ForceAverage`), 2D simulation attributes (`ImageBasedAverage`) and 3D simulation attributes (`RegularGridAverage`). Two-dimensional averages include scalars (`ScalarAverage`), vectors (`VectorAverage`), tensors (`TensorAverage`) and kernel density estimates (`T1KDE2D`).



- PropertySetter – Sends an attribute value to the graphics card (Fig. 4).
- RegularGridAverage – Time-average for a 3D regular grid (Fig. 4).
- ScalarAverage – Computes 2D scalar average (Fig. 4).
- T1KDE2D – Calculates T1s KDE for a 2D simulation (Fig. 4).
- TensorAverage – Computes a pixel-based time-average of vector and tensor attributes (Fig. 4).
- VectorAverage – Computes a pixel-based time-average of vector attributes (Fig. 4).
- VectorOperation – Math operations for vtkImageData, used for 3D average computation.

The **Model** component consists of the **Data**, **Settings** and **Average** components. It also includes two additional files:

- Base – Simulation data, derived data and, program status.
- DerivedData – Data derived from simulation data such as caches and averages.

The **View** component contains the views for displaying data. It contains the following implementation files:

- AttributeHistogram – A GUI histogram of a scalar attribute useful for one time step and all time steps.
- FoamvisInteractorStyle – Interactor that enables FoamVis style interaction in a VTK[8] view.

- Histogram – A histogram GUI that allows selection of bins.
- HistogramItem – Implementation of a GUI histogram, modified from Qwt[3].
- WidgetBase – Base class for all views: WidgetGl, WidgetVtk, WidgetHistogram.
- WidgetGl – View that displays 2D (and some 3D) foam visualizations using OpenGL.
- WidgetHistogram – View for displaying histograms of scalar values.
- WidgetSave – Widget that knows how to save its display as a JPG file.
- WidgetVtk – View that displays 3D foam visualizations using VTK.

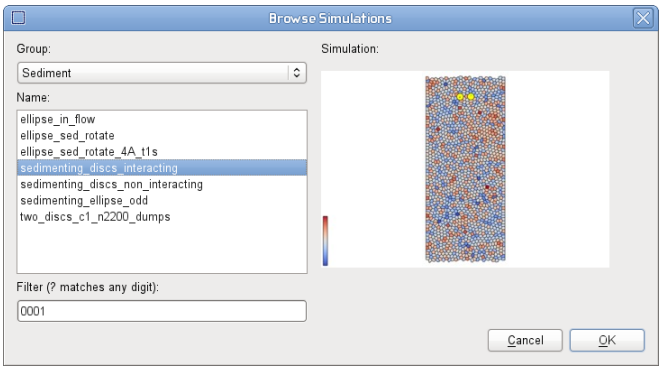
3.2. Parsing and Data Processing Foam simulation data consists of a list of SE output files, one per time step. A file stores the entire configuration of the simulated foam at a particular time step. For maximum generality and flexibility, we parse SE files directly instead of using derived files created by foam scientists. This allows our application to work with any simulation created using SE and at the same time it gives us access to the entire duration and state of the simulation. Parsing is done using `flex` [5] and `bison` [6] tools using `EvolverData.l` lexical analyzer and `EvolverData.y` grammar. Parsing is run by `Simulation::ParseDMPs` which parses the simulation files, stores the simulation data in memory and performs the additional processing required.

Our tool can read the following optional data that is saved by the simulation code: a list of T1s and the network and pressure forces that act on a body (Sec. 3.4).

After parsing foam simulation data and creating the corresponding data structures, we perform additional data processing (`Foam::Preprocess` and `Simulation::Preprocess`). First we compact each list of geometric elements as there can be numbering gaps in the list specified in a SE file (`Foam::compact`). Then, if the foam described in the SE file contains periodic boundary conditions (PBC) [20,21] we unwrap the geometric elements so that we can display the foam (`Foam::unwrap`). Additional processing include calculating each bubble's center of mass (`Foam::calculateBodyCenters`), bounding box and the bounding box of the foam at each time step (`Foam::CalculateBoundingBox` and overall (`Simulation::CalculateBoundingBox`, and calculating statistical quantities such as histogram, minima and maxima for values of attributes (`Simulation::calculateStatistics`). For 3D foam simulations unstructured simulation data is converted to a regular grid and it is cached in files on disk (Sec. 3.6).

The design of the data structures for storing bubbles and their topology is object-oriented. We have an object that stores an instance of each bubble. The Bubble contains a list of edges. These are the shared edges between neighboring bubbles. The Bubble object also stores a pointer to its neighboring bubbles. Technically speaking this information could be considered redundant since bubbles share edges, however, it does accelerate computation. Another option is to have each edge contain a list of pointers to its bubble objects. This is an important consideration for neighbor searching. During the foam evolution, the foam topology must be updated for every T1 event.

Figure 5. The `BrowseSimulations` dialog which allows the user to view related simulations and select simulations of interest for individual analysis or comparison.

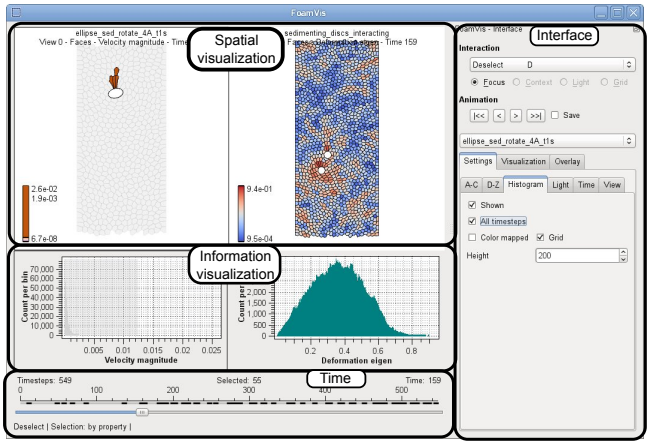


3.3. Interface Each dataset consists of a list of data files stored in a folder. The only information about the simulation available without parsing the simulation files is the name of the folder. While this often encodes important parameters of the simulation, their meaning may be cryptic and only known to the scientist that created the simulation. Additionally there is an increasing number of parameters providing additional information about the simulation which is not encoded in the simulation files. To address these issues we create a simulations database and a browsing interface. The simulations database records for each simulation three pieces of information: a simulation name - usually this is the name of the folder that stores the simulation files; a list of labels, each label is used to group simulations based on specific criteria; and simulation specific visualization parameters. The database is stored as a `.ini` file and is created by the user from a template. The **UI**, `Options` file contains classes that read options either from the command line or from an `.ini` file.

The browsing feature (Fig. 5) presents all grouping labels from the simulation database in a list. When a user selects a label, a list with all simulations tagged by that label is presented. When a user selects a simulation name, a picture of the first time step in the simulation is displayed. The image is saved beforehand so no parsing of simulation files is required. This allows a user to explore existing simulations based on similarity criteria encoded in labels and visually select simulations of interest for individual analysis or comparison. The browsing dialog is implemented by **`UI`**, `BrowseSimulations` class.

FoamVis' main window (Fig. 6) contains three panels that are used for both visualization and user interaction (Spatial and Information Visualization and Time), and one panel (Interface) that allows the user to specify desired visualizations and visualization parameters. The spatial visualization panel shows multiple views with each view showing a different visualization, a visualization of a different simulation attribute or a visualization of a different simulation. The information visualization panel shows histograms for simulation scalars shown in the spatial visualization panel. The time panel shows the current time step and marks time steps resulting from selections on scalar values. The main window of the application is implemented in **Controller**, `MainWindow`. This class implements the Interface panel and handles user notifications resulted from user interactions with the panel or with simulation data. The Spatial Visualization panel is implemented by the **View** component. In this component `WidgetGl` class displays 2D visualizations and 3D attribute and bubble paths visualizations; `WidgetVtk` class

Figure 6. FoamVis' `MainWindow` showing the spatial visualization, information visualization, time and interface panels. The spatial visualization panel shows two views: bubble velocity magnitude for a falling ellipse simulation and bubble deformation a falling discs simulation. A selection of velocity magnitude values is performed on the histogram showing this scalar and it is reflected in the spatial visualization and time panels. We can observe in the time panel that only 55 time steps out of 549 contain high velocity bubbles and in the spatial visualization panel we see those bubbles color mapped, the rest of the bubbles are rendered in gray as context information.



displays 3D attribute time-average and T1s KDE visualizations. The decision to use VTK [8] rather than plain OpenGL [18] for some of the 3D visualizations was based on the desire to speed-up the development of the application. We believe this was a sound decision which, besides speeding-up development, opened-up a wide range of visualization algorithms for adoption into FoamVis.

3.4. Simulation attributes Scalar bubble attributes include velocity along principal axes, velocity magnitude, edges per face, deformation, pressure, volume and growth rate. Scalar bubble attributes are visualized using color mapping. The user can change the color palette and change the range of scalar values mapped to color through clamping (Sec. 3.10). Fig. 7-a and Fig. 8, Fig. 6 show examples of scalar attributes visualized through color mapping. While domain experts are mostly interested in bubble attributes, in SE attributes can be attached to a body (bubble), face, edge or vertex. Information about predefined attributes that can be attached one of these elements is stored in **Parser**, `AttributesInfoElements` class. New attributes can be defined in a `.dmp` file. The **Parser** calls **Data**,`Foam::AddAttributeInfo` to register a new attribute. The **Parser**,`EvolverData.y` parses a list of attributes on the following grammar rules: `xxx_attribute_list` where `xxx` is vertex, edge, face or body. It creates a list of `NameSemanticValue` objects and it passes them to the `Foam` object for storage.

Bubble velocity, defined as the motion of the center of mass, provides information about foam flow. We visualize bubble velocities using glyphs (2D and 3D) (Fig. 7-a) and streamlines (2D only) (Fig. 9). We compute the velocity attribute in a processing step after parsing (`Simulation::Preprocess`) in `Simulation::calculateVelocity`. Velocity glyphs are visualized using the **Display** module, `DisplayBodyFunctors` file, `DisplayBodyVelocity` class.

Figure 7. Are the falling discs behaving like the falling ellipse? (a) The foam between the discs moves at high velocity with the discs. Velocity is displayed using glyphs and velocity magnitude is also color-mapped. (b) Few topological changes occur between the discs, so the foam behaves like an elastic solid there. Topological changes over time visualized using KDE [13].

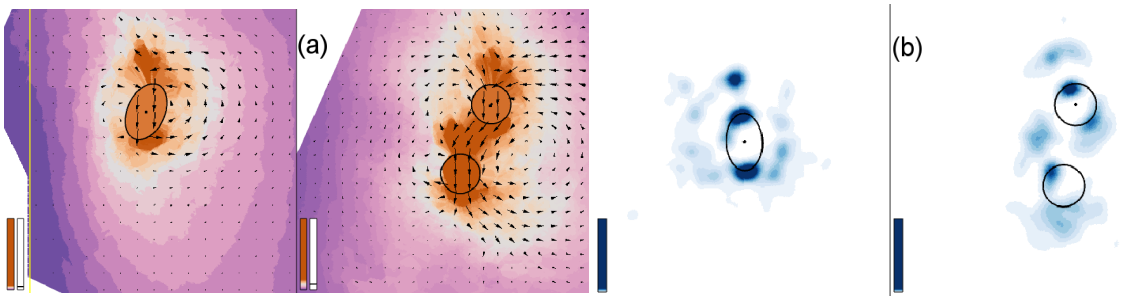
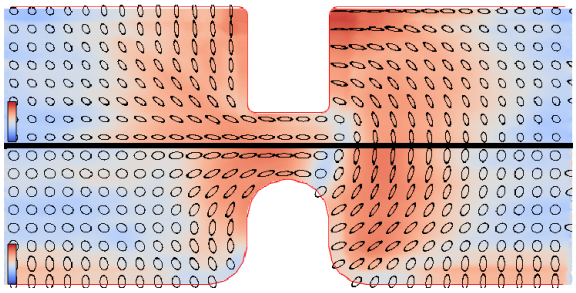


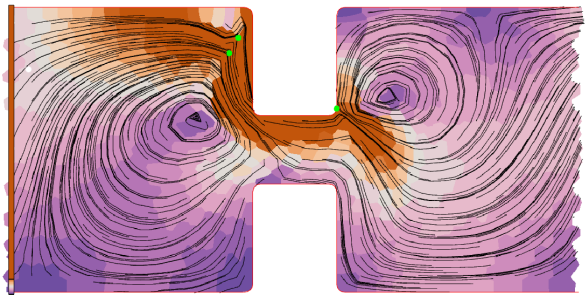
Figure 8. Rounding the corners of the constriction results in reduced elastic deformation of the foam (top versus bottom). In both simulations, there is an area where bubbles are not deformed just downstream from the constriction. We show the square (top) and rounded (bottom) constriction simulations. Deformation magnitude and direction is displayed with ellipses, deformation magnitude is also color-mapped. An average over the entire duration of the simulations is displayed [13].



Bubble deformation magnitude and *direction* are important bubble attributes, because they facilitate validation of simulations and provide information about the force acting on a dynamic object in foam. While visual inspection of individual bubbles provides information about foam deformation, this information is not quantified and, more importantly, cannot be averaged to obtain the general foam behavior. To address these issues, we define a bubble deformation measure [13] expressed as a tensor. The deformation tensor is visualized using glyphs as shown in Fig. 8. We compute a deformation scalar and tensor measures in a processing step after parsing: `Foam::CalculateDeformationSimple` and `Foam::CalculateDeformationTensor`. Two-dimensional deformation glyphs are visualized using the **Display** module, `DisplayBodyFunctors` file, `DisplayBodyDeformation` class.

When foam is subjected to stress, bubbles deform (elastic deformation) and move past each other (plastic deformation). Domain experts are interested in the distribution of the plasticity which is indicated by the location of topological changes. A topological change is a neighbor swap between four neighboring bubbles. In a stable configuration, bubble edges meet 3-way at 120° angles. As foam is sheared, bubbles move into an unstable configuration, in which edges meet 4-way, then quickly shift into

Figure 9. Topological changes are associated with strong circulation; topological changes shown with green dots, velocity field shown with streamlines, $t = 412$. Velocity is shown with streamlines, and velocity magnitude is color-mapped.



a stable configuration. Topological changes for the current time step or for all time steps are visualized with glyphs (or spheres) of configurable color and size showing the location of the topological change (Fig. 9). Topological changes are parsed either from a separate file or from variables inside the `.dmp` file by the overloaded function `Simulation::ParseT1s`. They are stored in the `Simulation` object.

The forces and the torque acting on objects are computed by the simulation code and stored in the simulation data. Each force acting on an object is represented with an arrow that starts in the center of the object and with length proportional to the magnitude of the force.

For the falling discs simulation, the interplay of the network and pressure forces rotate one disc around the other. We provide a user option that displays the difference between the forces acting on the leading disc and forces acting on the trailing disc. This difference allows us to better analyze the causes of the rotation as there is a direct correspondence between the forces displayed on the screen and the movement of the disc (Fig. 10 right).

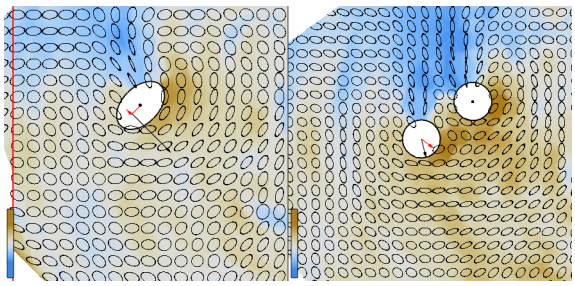
The torque τ rotating an object around its center is displayed as a force F acting off-center on the object $\tau = r \times F$, where r is the displacement vector from the center of the object to the point at which the force is applied. The distance $|r|$ is a user-defined parameter, FoamVis calculates the appropriate value of F to keep the torque constant (Fig. 10 left).

The forces and torques acting on objects are read from the simulation files, from variable names passed as parameters either from command line or from the `.ini` file. Variable names that store forces and torques are passed as parameters in **Data**, `ForceNamesOneObject` class while the forces and torques are stored in **Data**, `ForceOneObject` in the `Foam` object. Forces are displayed using `ForceAverage::DisplayOneTimeStep`[1] for OpenGL views or using `PipelineAverage3D::createObjectActor` for VTK views.

3.5. Bubble paths Visualization of bubble paths provides information about the trajectory of individual bubbles in the simulation. The paths are a useful way to compare simulation with experiment. They also provide insight into the overall behavior of the foam. A bubble path is determined by connecting

[1] This function would better fit in the **Display** module, as this would separate the data from its display. We plan to address this issue in future work.

Figure 10. Falling-ellipse versus falling-discs. The *linked time with event synchronization* feature [13] is used to synchronize the rotation of the ellipse and the two discs such that they reach an orientation of $45°$ in the same time. Attributes (pressure, deformation and forces) are averaged over 52 time steps for the ellipse simulation (resulting in an average over 15 time steps for the two disc simulation). Pressure is color-mapped, deformation is shown using ellipses. The force *difference* between the leading disc and the trailing disc and the torque on the ellipse is indicated. The network force and torque are indicated with a black arrow and the pressure force and torque are indicated with a red arrow.



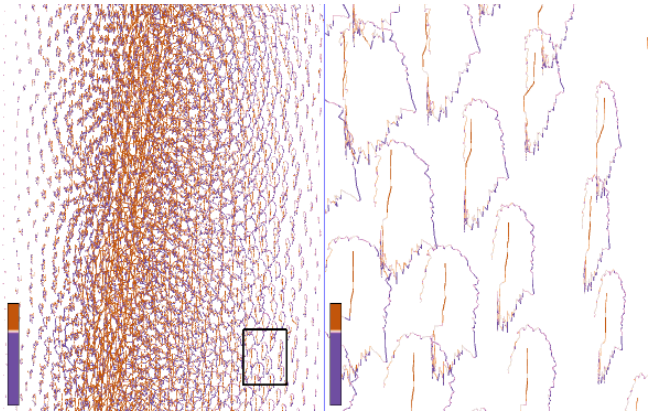
the center of bubbles with the same ID in consecutive time steps. Fig. 11 shows a pattern of bubbles traversing loops revealed by a bubble paths visualization.

Bubble paths are stored in a processing step after parsing by calling `Simulation::CacheBodiesAlongTime` and they are displayed in **Display**, `DisplayBubblePaths`.

3.6. Time-Average of a Simulation Attribute Bubble-scale simulations can be too detailed for observing general foam behavior and topological changes generate large fluctuations in attribute values that hide the overall trends. A good way to smooth out these variations is to calculate the average of the simulation attributes over all time steps, or over a time window before the current time step. This visualization reveals global trends in the data because large fluctuations caused by topological changes are eradicated. This results in only small variations between averaged successive time steps. The time window is a parameter set by the user. We compute the average for the entire simulation (Fig. 8) if there are no dynamic objects interacting with the foam. In this case, at a high level of detail, there is no difference between different time steps in the simulations. For simulations that include dynamic objects interacting with the foam (Fig. 7, 10) a smaller time window is appropriate, as objects may traverse transient states that have to be analyzed independently.

Time-averages of several 2D foam simulation attributes are stored in `AttributeAverages2D` and are referenced from `WidgetGl`. A pixel-based time-average of one simulation attribute is computed by `ScalarAverage`, `VectorAverage` and `TensorAverage` for scalars, vectors and tensors. Most of this computation is done in the base class `ImageBaseAverage` (Fig. 4). Displaying an average of attributes is done by the graphics card fragment shader using `AverageInterface::AverageRotateAndDisplay`[1] overwritten for each attribute type. For 3D simulations, unstructured grid data is converted to regular grid data and is cached in files inside `.foamvis` folder using `Foam::SaveRegularGrid`. This is done in the processing step after

Figure 11. Pattern of bubbles traversing loops visualized using bubble paths in the falling discs simulation. The bubbles paths are color-mapped to velocity along Y, with orange indicating descent and purple indicating ascent. The left image shows the bubble paths over the entire simulation. The red area shows the paths of the two discs. The black rectangle shows the region that is magnified in the right image.



parsing `Simulation::Preprocess`. Time-averages of several 3D foam simulation attributes are stored in `AttributeAverages3D` and are referenced from `WidgetVtk`. A time-average for one attribute, for all types of attributes, is computed by `RegularGridAverage` and displayed using a VTK pipeline created by `PipelineAverage3D::createScalarAverageActor` and `PipelineAverage3D::createVelocityGlyphActor` for scalars and respective vectors.

3.7. Topological changes kernel density estimate (KDE) Topological changes, in which bubbles change neighbors, indicate plasticity in a foam. Domain experts expect that their distribution will be an important tool for validating simulations. Simply rendering the position of each topological change suffers from over-plotting, so it may paint a misleading picture of the real distribution. We compute (see Lipsa et al. [13] for details) a KDE for topological changes (Fig. 7, Fig. 12). While traditional histograms show similar information and are straightforward to implement they have drawbacks which may prove important depending on the context. Drawbacks of histograms include the discretization of data into bins which may introduce aliasing effects and the fact that the appearance of the histogram may depend on the choice of origin for the histogram bins [4,19]. Kernel-based methods for computing the probability density estimate eliminate these drawbacks.

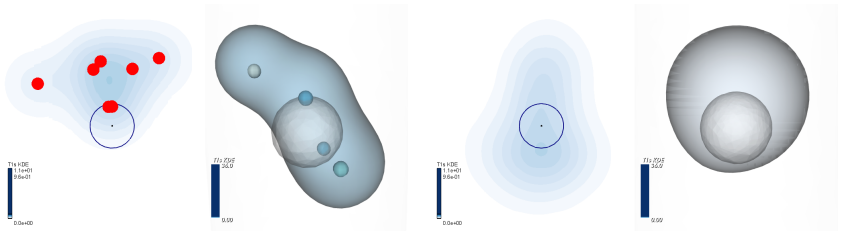
T1s KDE is computed using the average framework (Fig. 4) (`T1KDE2D` or `RegularGridAverage` classes for 2D or 3D foam simulation). For 2D simulations, for each topological change in a time step, a Gaussian is added to the average using `T1KDE2D::writeStepValues`. For 3D simulations, a Gaussian determined by a topological change in a time step is returned by `Simulation::GetT1KDE`. This Gaussian added to the current average in `RegularGridAverage::OpStep`.

3.8. Histograms We provide both a histogram of bubble attribute values over one time step and over all time steps. To facilitate data analysis, our histogram is configurable. The user can choose a maximum height, logarithmic or linear height scale and uni-color or color-coded display

Figure 12. KDE *around* the falling disc versus falling sphere simulations. The maximum values in the color bar represent the maximum number of topological changes in a time step. KDE for all time steps (b) shows that, for 3D, topological changes on top of the sphere dominate the final result. This is caused by topological changes in the same area being triggered repeatedly in the simulation code, feature discovered using our visualization.

(a) KDE for one time step: $t = 18$ left view and $t = 21$ right view. Isosurface density is 0.5 for the right view.

(b) KDE for all time steps. Isosurface density is 0.12 for the right view.



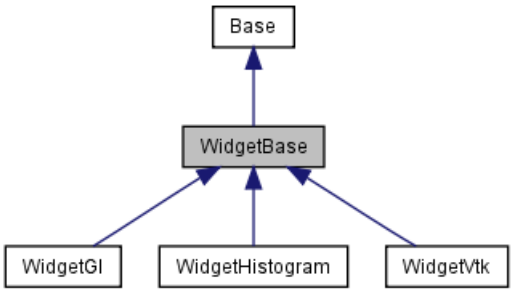
using `HistogramSettings` dialog. Histograms are also used in selection and filtering of data based on attribute value and in color-map clamping used for selecting features of interest in the data. These interactions are described in detail in Sec. 3.10. Histograms are displayed by **View**, `WidgetHistogram`. Histograms notify the **Controller** when scalar selection has changed using `WidgetHistogram::SelectionChanged`.

3.9. Multiple linked-views Foam scientists wish to understand what triggers certain behavior in foam simulations. Foam behavior is determined by many simulations attributes so the ability to see different attributes at the same time and to understand how different attributes relate to one another is very important. At the same time, to understand the influence that simulation parameters have on foam behavior, foam scientists would like to analyze and compare related simulations. Both these requirements are addressed using multiple linked views. We provide up to four different views. For maximum flexibility, each view can depict a different simulation attribute, a different visualization or even a different simulation. Each view uses its own color-bar and can show the navigation context. Each of the three widgets used to show data (`WidgetGl`, `WidgetVtk` and `WidgetHistogram`) can display up to four views. These three classes are derived from `WidgetBase` which provides view related functionality; WidgetBase is derived from `Base` which provides access to data and program status (Fig. 13).

To set up optimal views to analyze data, users can copy viewing transformations (WidgetXXX::CopyTransformFromSlot where XXX is Gl or Vtk) and color mapping between views depicting the same attribute (`MainWindow::CopyColorMapXXX` where XXX is Scalar or Velocity).

The *two halves* option facilitates visual comparison of two related foam simulations (Fig. 8). It visualizes related simulations that are assumed to be symmetric with respect to one of the main axes.

Figure 13. `WidgetBase` Inheritance Graph. This class provides functionality common to all views. It inherits from `Base` which stores simulation data and program status. `WidgetGl` displays views rendered with OpenGl, `WidgetVtk` displays views rendered with VTK, and `WidgetHistogram` displays histograms.



While the same information can be gathered by examining the two simulations in different views, the *two halves* view may facilitate analysis as images to be compared are closer together and it is useful for presentation as it saves space. This type of visualization was previously performed manually by domain experts. This option is only available for 2D simulations in `WidgetGl`. It is set using `Settings::SetTwoHalvesView`.

We provide three *connection operations* [23] between views: one linked-selection connection and two linked-time connections. The linked-selection connection works by showing data selected in one view in other views. This is used to see, for instance, the elongation of high pressure bubbles or both pressure and elongation for bubbles involved in a topological change. This connection works by copying the selection in one view in any other view using `ViewSettings::CopySelection`.

The first linked-time connection works by having each view linked to the same time step, as foam scientists want to analyze several attributes at the same time to understand foam behavior influenced by those attributes. The linked-time connection is set to independent time or linked time using `Settings::SetTimeLinkage`. The second linked-time connection, linked-time with event synchronization, is described next. In simulations that involve dynamic objects interacting with foam, we may want a similar event in both simulations to be visualized at the same time so that behavior up to that event can be compared and analyzed together. When comparing the falling discs with the falling ellipse simulations, the ellipse and the discs start in similar configurations. The main axis of the ellipse and the line connecting the center of the two discs are horizontal. We want the ellipse and the discs to reach intermediate configurations and the stable configuration at the same time. These configurations are defined in terms of the angle that the major axis of the ellipse and the line connecting the centers of the two discs make with gravity. For instance, an angle for the intermediate configuration could be $45°$ while the angle for the stable configuration is $0°$. A new event for the current view and current time is added using `Settings::AddLinkedTimeEvent`. All views that use linked-time with event synchronization have to have the same number of events. This technique splits simulation times in intervals - an interval before each event and an interval after the last event. For each interval before an event, one simulation will run at its normal speed (the simulation with the longest interval as returned by `Settings::GetLinkedTimeMaxInterval`), all other simulations will be "slowed down" using `Settings::GetLinkedTimeStretch`. Simulations will run at normal speed for the

time-interval after the last event. Using this approach, related events occur at the same *linked time* in all simulations, facilitating their comparison as well as the comparison of their temporal context. Fig. 7, 10 use linked-time with event synchronization feature. The complete interface for using the linked-time with event synchronization is in class `Settings`, member group Time and LinkedTime.

3.10. Interaction Interaction with the data is an essential feature of our application.

Navigation is used to select a subset of the data to be viewed, the direction of view, and the level of detail [23]. We provide the following navigation operations: rotation around a bounding box center for specifying the direction of view, and translation and scaling for specifying the subset of data and the level of detail. Navigation operations are implemented in the `WidgetGl` views in `mousePressEvent` and `mouseMoveEvent`. These operations are provided by the VTK library in the `WidgetVtk` views. A navigation context (Fig. 11 left) ensures that the user always knows its location and orientation during exploration of the data. Focus and context related settings are in `ViewSettings`, `Context view` member group.

We can **select and/or filter** bubbles and center paths based on three distinct criteria: based on bubble IDs (`WidgetGl::SelectBodiesByIds`), to enable data to be related to the simulation files and for debugging purposes; based on location of bubbles (`WidgetGl::mousePressEvent` and `WidgetGl::mouseMoveEvent`), to analyze interesting features at certain locations in the data; and based on an interval of attribute values specified using the histogram tool (Fig. 6) (The histogram sends `WidgetHistogram::selectionChanged` signal which is handled in `MainWindow::SelectionChangedFromHistogram`). A composite selection can be specified using both location and attribute values.

Selected bubbles or center paths constitute the focus of our visualization, and the rest of the bubbles or center paths provide the context [7]. The context of the visualization is displayed using user-specified semi-transparency, or it can be hidden altogether.

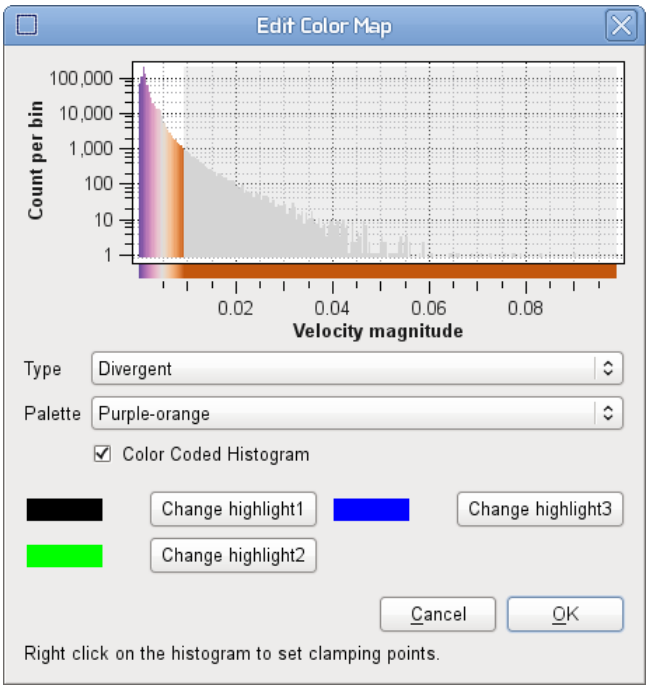
Encoding operations are variations of graphical entities used in a visualization that emphasize features of interest [23]. We provide encoding operations to change the color map used, to specify the range of values used in the color map and to adjust the opacity of the visualization context. Selection of the interval used in color-mapping is guided by the histogram tool (Fig. 14) (the implementation is in `EditColorMap`). This provides essential information for selecting an interval that reveals features of interest.

4. Conclusions and Future Work

We present challenges faced by domain scientists and describe FoamVis, a software application designed to address some of these challenges. We describe its main implementation components and their interactions, and present FoamVis' main features together with implementation notes that describe how and where these features are implemented.

We see many directions for future work. We would like to add more algorithms for visualization of 3D foam simulations, enable comparison between foam simulation and experiments and support analysis of other kinds of Surface Evolver simulations.

Figure 14. Color-map clamping guided by the histogram tool (`EditColorMap` class). This is a histogram of the constriction simulation which uses a logarithmic height scale. The histogram is clamped at high values. The dialog also allows the user to choose a different color palette and to change highlight colors used for vector and tensor glyphs and forces acting on objects.



Acknowledgments This research was supported in part by the Research Institute of Visual Computing (rivic.org) Wales. We thank Ken Brakke for answering our many questions about the Surface Evolver.

The authors declare no conflict of interest

References

1. D. Bell. UML basics: The component diagram, 2004. Online document, accessed 25 June 2013, http://www.ibm.com/developerworks/rational/library/dec04/bell/index.html.
2. K. Brakke. The Surface Evolver. *Experimental Mathematics*, 1(2):141–165, 1992.
3. C. A. Brewer. ColorBrewer. Online document, http://www.ColorBrewer.org, accessed 3 March. 2012.
4. O. Daae Lampe and H. Hauser. Interactive Visualization of Streaming Data with Kernel Density Estimation. In *Pacific Visualization Symposium (PacificVis)*, pages 171–178. IEEE, 2011.
5. flex - The Fast Lexical Analyzer. Online document, http://flex.sourceforge.net/, accessed 29 Nov. 2010.
6. Bison - GNU Parser Generator. Online document, http://www.gnu.org/software/bison/, accessed 29 Nov. 2010.

7. H. Hauser. Generalizing Focus+context Visualization. *Scientific visualization: The visual extraction of knowledge from data*, pages 305–327, 2006.
8. K. Inc. *The VTK User's Guide Version 5 (Paperback)*. Kitware Inc., 2006.
9. D. Lipsa. FoamVis, 2013. Online document, accessed 25 June 2013, http:http://cs.swan.ac.uk/~csbob/research/foamVis/design/html/.
10. D. R. Lipşa, R. S. Laramee, S. Cox, and I. T. Davies. Visualizing 3D Time-Dependent Foam Simulation Data. In *Lecture Notes in Computer Science, International Symposium on Visual Computing (ISVC)*, Rethymnon, Crete, Greece, July 2013.
11. D. R. Lipşa, R. S. Laramee, S. J. Cox, and I. T. Davies. FoamVis: Visualization of 2D Foam Simulation Data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2096–2105, Oct. 2011.
12. D. R. Lipşa, R. S. Laramee, S. J. Cox, and I. T. Davies. A Visualization Tool For Foam Research. In *NAFEMS World Congress (NWC) Conference Proceedings*, page 141, Salzburg, Austria, June 2013.
13. D. R. Lipşa, R. S. Laramee, S. J. Cox, and I. T. Davies. Comparative Visualization and Analysis of Time-Dependent, 2D Foam Simulation Data. Technical report, Swansea University, 2013.
14. Microsoft. Model-View-Controller, 2013. Online document, accessed 25 June 2013, http://msdn.microsoft.com/en-us/library/ff649643.aspx.
15. R. Prud'homme and G. Warr. *Foams in Mineral Flotation and Separation Processes*, pages 511–554. Volume 57 of prud1996foams [16], 1996.
16. R. K. Prud'Homme and S. A. Khan. *Foams: theory, measurements, and applications*, volume 57. CRC PressI Llc, 1996.
17. W. Rossen. *Foams in Enhanced Oil Recovery*, pages 413–464. Volume 57 of prud1996foams [16], 1996.
18. D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide, Fifth Edition*. Addison Wesley, 2006.
19. B. Silverman. *Density Estimation for Statistics and Data Analysis*, volume 26 of *Monographs on Statistics and Applied Probability*. Chapman & Hall/CRC, 1986.
20. The Surface Evolver, Jan. 2008. Online document, accessed 29 Nov. 2010, http://www.susqu.edu/brakke/evolver/html/evolver.htm.
21. Surface Evolver Workshop, Apr. 2004. Online document, accessed 1 Dec. 2010, http://www.susqu.edu/brakke/evolver/workshop/workshop.htm.
22. D. van Heesch. Doxygen, 2013. Online document, accessed 26 June 2013, http://www.stack.nl/~dimitri/doxygen/.
23. M. Ward, G. Grinstein, and D. Keim. *Interactive Data Visualization. Foundations, Techniques, and Applications*, chapter 10, pages 315–334. A K Peters, Ltd., Natick, Massachussetts, 2010.
24. D. Weaire and S. Hutzler. *The Physics of Foams*. Oxford University Press, Oxford, 1999.