

Article

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

Dan R. Lipsa<sup>1,\*</sup>, Richard C. Roberts, Robert S. Laramée<sup>1,\*</sup>

<sup>1</sup>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*

---

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

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

---

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

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

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

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

- 30 1. Access to simulation data is difficult and requires domain-specific knowledge. Parsing and special  
31 processing are required to access the entire simulation data. Important bubble attributes are not  
32 provided by the simulation but inferred using domain specific-knowledge.
- 33 2. It is challenging to visualize general foam behavior. While bubble-scale simulation makes it  
34 possible to investigate the influences that material properties have on general foam behavior, it  
35 makes it difficult to visualize the general behavior that is of primary interest. Simulation data is  
36 complex (unstructured grid with polygonal cells) and time-dependent, with large fluctuations in  
37 the values of the parameters determined by changes in the topology of the soap film network.
- 38 3. Triggers to various foam behaviors are difficult to infer. Multiple attributes have to be examined  
39 and foam properties have to be taken into account. Topological changes (T1s), in which bubbles  
40 swap neighbors, have to be considered.
- 41 4. Foam scientists work with dozens of simulations with a wide range of simulation parameters.  
42 Examples include foam container properties (such as shape and roughness), foam attributes (such  
43 as bubble size and distribution, liquid fraction and surface tension) or the properties of objects  
44 interacting with foam (such as shape, size and position). The large number of existing simulations  
45 and the variety of simulation parameters makes it difficult to manage simulation data. The  
46 possibility to compare related datasets results in a better understanding of various foam behaviors,  
47 however existing tools do not facilitate that.

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

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

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

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

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

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

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

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

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

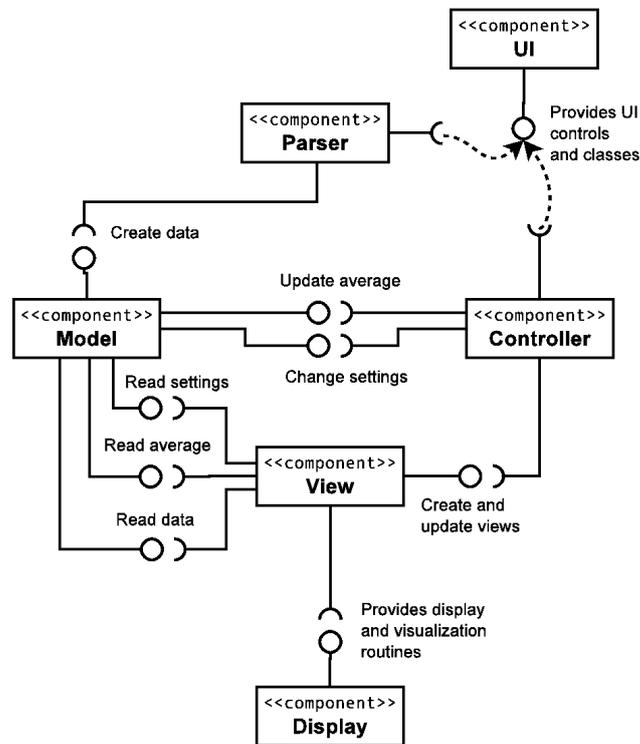
99 Foam scientists wish to understand what triggers certain behavior in foam simulations (challenge  
100 three). Foam behavior is studied by either examining different attributes that influence it or by comparing  
101 simulations (challenge four) where the behavior is varied by modifying simulation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

103 To present our solutions for visualization of foam simulation data, we use three simulation groups  
104 containing related simulations: the falling discs and the falling ellipse (2D), constriction (2D), and  
105 the falling disc (2D) and the falling sphere (3D). The falling-objects simulation group contains the  
106 *falling-ellipse* and the *falling-discs* simulations (Fig. 6). The *falling-discs* simulates two discs falling  
107 through a monodisperse (bubbles having equal volume) foam under gravity. It contains 330 time steps  
108 and simulates 2200 bubbles. The two discs are initially side-by-side and in close proximity. As they  
109 fall, they interact with the foam and each other by rotating towards a stable orientation in which the  
110 line that connects their centers is parallel to gravity. The *falling-ellipse* simulates an ellipse falling  
111 through a monodisperse foam under gravity. This dataset contains 540 time steps and simulates 600  
112 bubbles. The major axis of the ellipse is initially horizontal. As the ellipse falls, it rotates toward a  
113 stable orientation in which its major axis is parallel to gravity. The constriction dataset contains two  
114 simulations, one with a *square-constriction* and one with a *rounded-constriction* (Fig. 8). They simulate  
115 a 2D polydisperse (bubbles with different volumes) foam flowing through a constricted channel, with  
116 725 bubbles and 1000 time steps. The radius of the curvature of the rounded corners of the constriction  
117 is five times smaller for the square-constriction compared with the rounded-constriction. The falling disc  
118 (2D) / sphere (3D) simulate a disc/sphere falling through a monodisperse (bubbles having equal volume)  
119 foam under gravity. In 2D we have 254 time steps and 1500 bubbles. In 3D we have 208 time steps  
120 and 144 bubbles. Note that the number of bubbles that scientists are able to simulate in 3D is severely  
121 restricted by the duration of the computation time.

122 *3.1. Overview* In this section we present the structural relationships between FoamVis' main components  
123 (Fig. 1). For this purpose we use a UML 2 components diagram [1]. Briefly, a component represented  
124 in our diagram as a rectangle, is a design unit that is typically implemented using a replaceable module.  
125 A component may provide one or more public interfaces, represented with a complete circle at their  
126 end (lollipop symbol). Provided interfaces represent services that the component provides to its clients.  
127 Similarly, a component may require services from other components. These services are formalized  
128 through the required interfaces, represented with a half circle at their end (socket symbol).

129 FoamVis starts by executing the **Parser** component. This component, uses services from the **UI**  
130 component to allow the user to specify the simulations to be analyzed and additional information about  
131 the simulations. This is done either through the command line or through graphical user interface. Then,  
132 the **Parser** parses the specified simulation files, creates an in-memory representation of the simulation  
133 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.



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

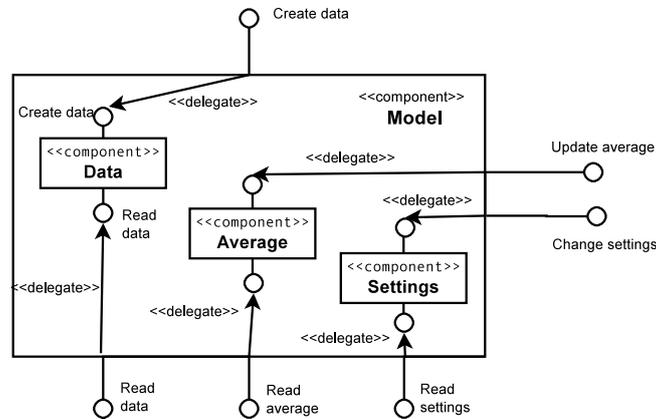
141 The **Controller** manages the interaction between a user, the **Model** (that stores data and program  
 142 state) and the views that show foam simulation data.

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

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

149 Each of these logical components contains several implementation files which in turn contain one  
 150 or several related C++ classes. Logical components (modules), their implementation files and classes,  
 151 and groups of member functions (member groups) are also documented [9] using Doxygen [22]. We are  
 152 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.



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

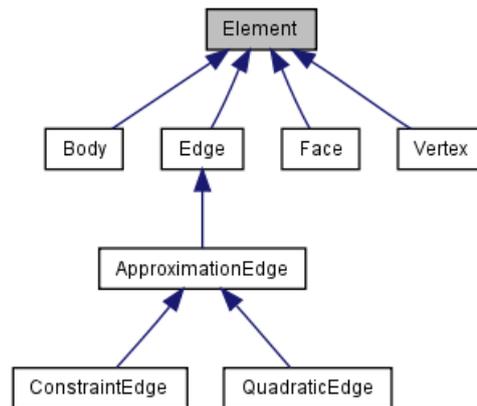
158 The **Parser** parses Surface Evolver `.dmp` files and calls the **Data** component to build a memory  
 159 representation of the simulation data. It contains the following files:

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

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

- 172 • `MainWindow` – Stores the `OpenGL`, `Vtk` and, `Histogram` widgets, implements the `Interface` and  
 173 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.



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

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

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

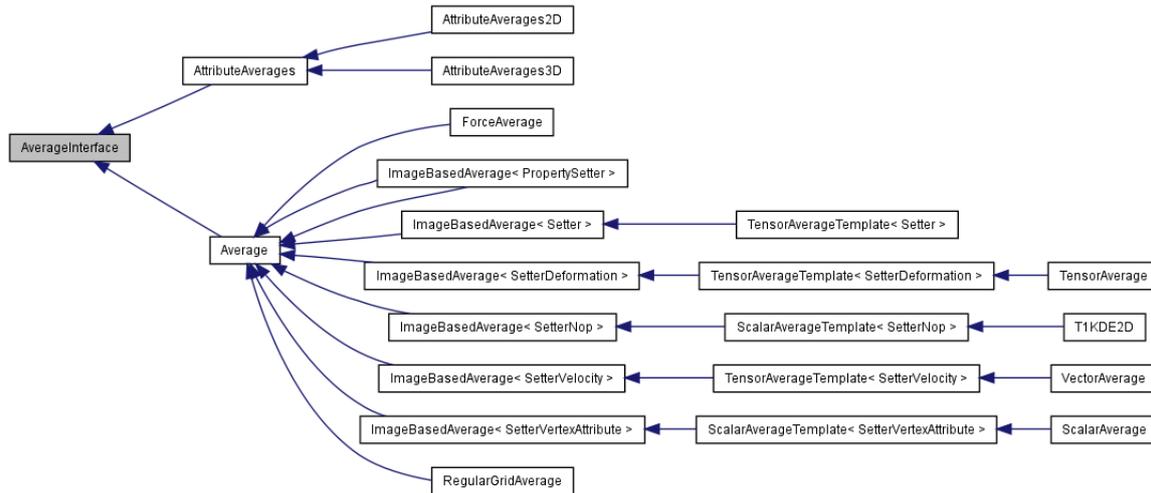
202 The **Settings** component stores and provides access to program state. This component is composed  
203 from the the following files:

- 204 • **BodySelector** – Functors that specify selected bubbles.
- 205 • **Settings** – Settings that apply to all views.
- 206 • **ViewSettings** – Settings that apply to one view.

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

- 208 • **AttributeAverages** – Computes the average for several attributes in a view. Base class for  
209 **AttributeAverages2D** and **AttributeAverages3D** (Fig. 4).
- 210 • **AttributeAverages2D** – Computes the average for several attributes in a 2D view. Casts the  
211 computed averages to the proper 2D types (Fig. 4).
- 212 • **AttributeAverages3D** – Computes the average for several attributes in a 3D view. Casts the  
213 computed averages to the proper 3D types (Fig. 4).
- 214 • **Average** – Computes a time-average of a foam attribute. Base class for 2D and 3D time-average  
215 computation classes (Fig. 4).
- 216 • **AverageInterface** – Interface for computing a time-average of a simulation attribute (Fig. 4).
- 217 • **AverageShaders** – Shaders used for computing a pixel-based time-average of attributes.
- 218 • **ForceAverage** – Time-average for forces acting on objects interacting with foam (Fig. 4).
- 219 • **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).



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

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

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

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

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

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

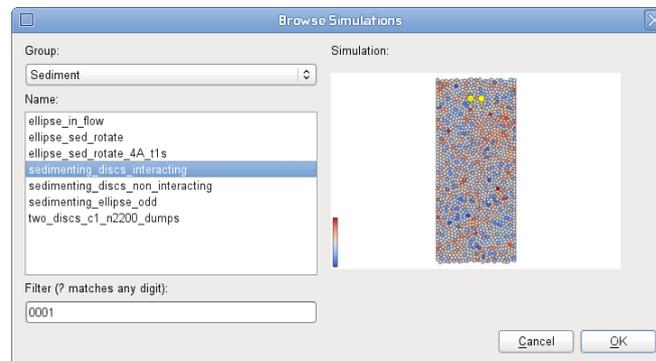
243 *3.2. Parsing and Data Processing* Foam simulation data consists of a list of SE output files, one per  
244 time step. A file stores the entire configuration of the simulated foam at a particular time step. For  
245 maximum generality and flexibility, we parse SE files directly instead of using derived files created  
246 by foam scientists. This allows our application to work with any simulation created using SE and at  
247 the same time it gives us access to the entire duration and state of the simulation. Parsing is done  
248 using `flex` [5] and `bison` [6] tools using `EvolverData.l` lexical analyzer and `EvolverData.y`  
249 grammar. Parsing is run by `Simulation::ParseDMPs` which parses the simulation files, stores the  
250 simulation data in memory and performs the additional processing required.

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

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

265 The design of the data structures for storing bubbles and their topology is object-oriented. We have  
266 an object that stores an instance of each bubble. The `Bubble` contains a list of edges. These are the  
267 shared edges between neighboring bubbles. The `Bubble` object also stores a pointer to its neighboring  
268 bubbles. Technically speaking this information could be considered redundant since bubbles share edges,  
269 however, it does accelerate computation. Another option is to have each edge contain a list of pointers to  
270 its bubble objects. This is an important consideration for neighbor searching. During the foam evolution,  
271 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.

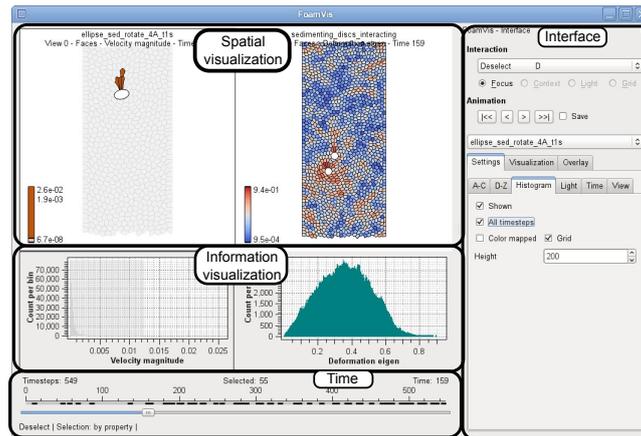


272 **3.3. Interface** Each dataset consists of a list of data files stored in a folder. The only information about  
 273 the simulation available without parsing the simulation files is the name of the folder. While this often  
 274 encodes important parameters of the simulation, their meaning may be cryptic and only known to the  
 275 scientist that created the simulation. Additionally there is an increasing number of parameters providing  
 276 additional information about the simulation which is not encoded in the simulation files. To address these  
 277 issues we create a simulations database and a browsing interface. The simulations database records for  
 278 each simulation three pieces of information: a simulation name - usually this is the name of the folder  
 279 that stores the simulation files; a list of labels, each label is used to group simulations based on specific  
 280 criteria; and simulation specific visualization parameters. The database is stored as a `.ini` file and is  
 281 created by the user from a template. The **UI**, `Options` file contains classes that read options either from  
 282 the command line or from an `.ini` file.

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

290 FoamVis' main window (Fig. 6) contains three panels that are used for both visualization and user  
 291 interaction (Spatial and Information Visualization and Time), and one panel (Interface) that allows the  
 292 user to specify desired visualizations and visualization parameters. The spatial visualization panel  
 293 shows multiple views with each view showing a different visualization, a visualization of a different  
 294 simulation attribute or a visualization of a different simulation. The information visualization panel  
 295 shows histograms for simulation scalars shown in the spatial visualization panel. The time panel shows  
 296 the current time step and marks time steps resulting from selections on scalar values. The main window  
 297 of the application is implemented in **Controller**, `MainWindow`. This class implements the Interface  
 298 panel and handles user notifications resulted from user interactions with the panel or with simulation data.  
 299 The Spatial Visualization panel is implemented by the **View** component. In this component `WidgetGl`  
 300 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.

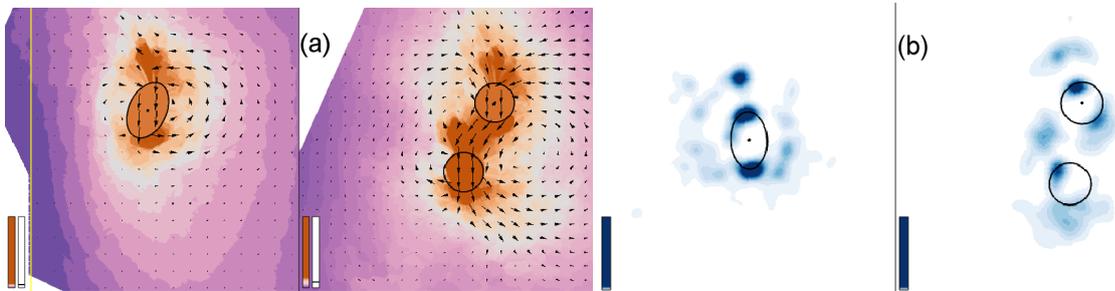


301 displays 3D attribute time-average and T1s KDE visualizations. The decision to use VTK [8] rather  
 302 than plain OpenGL [18] for some of the 3D visualizations was based on the desire to speed-up the  
 303 development of the application. We believe this was a sound decision which, besides speeding-up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

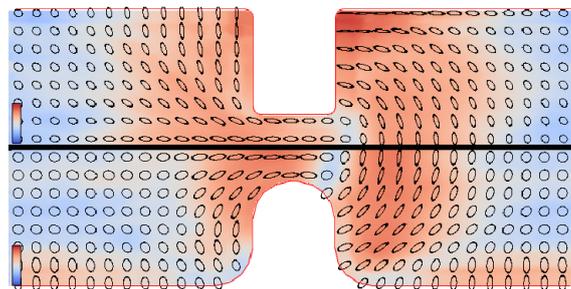
305 **3.4. Simulation attributes** Scalar bubble attributes include velocity along principal axes, velocity  
 306 magnitude, edges per face, deformation, pressure, volume and growth rate. Scalar bubble attributes  
 307 are visualized using color mapping. The user can change the color palette and change the range  
 308 of scalar values mapped to color through clamping (Sec. 3.10). Fig. 7-a and Fig. 8, Fig. 6 show  
 309 examples of scalar attributes visualized through color mapping. While domain experts are mostly  
 310 interested in bubble attributes, in SE attributes can be attached to a body (bubble), face, edge or vertex.  
 311 Information about predefined attributes that can be attached one of these elements is stored in **Parser**,  
 312 `AttributesInfoElements` class. New attributes can be defined in a `.dmp` file. The **Parser** calls  
 313 `DataFoam::AddAttributeInfo` to register a new attribute. The **Parser**, `EvolverData.y` parses  
 314 a list of attributes on the following grammar rules: `xxx_attribute_list` where `xxx` is vertex, edge,  
 315 face or body. It creates a list of `NameSemanticValue` objects and it passes them to the `Foam` object  
 316 for storage.

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



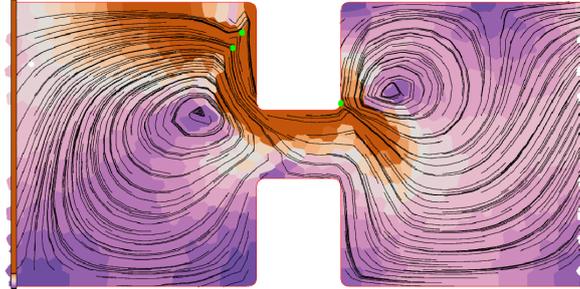
**Figure 8.** Rounding the corners of the constriction results in reduced elastic deformation of the foam (top versus bottom). In both simulations, there is an area where bubbles are not deformed just downstream from the constriction. We show the square (top) and rounded (bottom) constriction simulations. 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].



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

332 When foam is subjected to stress, bubbles deform (elastic deformation) and move past each other  
 333 (plastic deformation). Domain experts are interested in the distribution of the plasticity which is  
 334 indicated by the location of topological changes. A topological change is a neighbor swap between  
 335 four neighboring bubbles. In a stable configuration, bubble edges meet 3-way at  $120^\circ$  angles. As foam is  
 336 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.



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

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

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

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

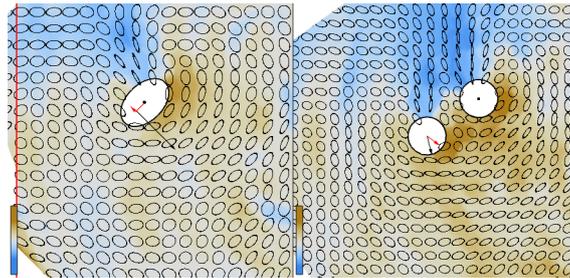
353 The forces and torques acting on objects are read from the simulation files, from variable  
 354 names passed as parameters either from command line or from the `.ini` file. Variable names  
 355 that store forces and torques are passed as parameters in `Data`, `ForceNamesOneObject` class  
 356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

359 **3.5. Bubble paths** Visualization of bubble paths provides information about the trajectory of individual  
 360 bubbles in the simulation. The paths are a useful way to compare simulation with experiment. They  
 361 also provide insight into the overall behavior of the foam. A bubble path is determined by connecting

---

<sup>1</sup> 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  $45^\circ$  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.



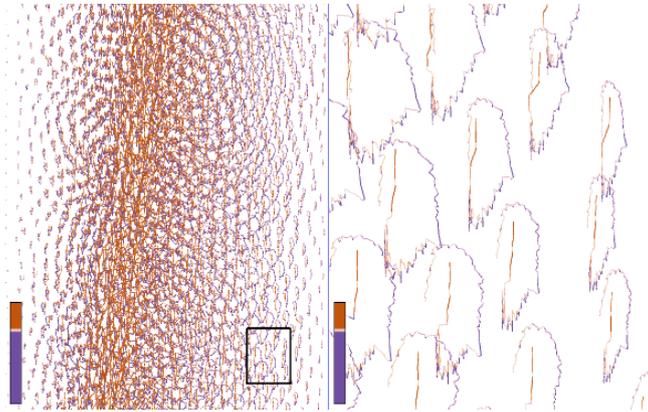
362 the center of bubbles with the same ID in consecutive time steps. Fig. 11 shows a pattern of bubbles  
 363 traversing loops revealed by a bubble paths visualization.

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

367 **3.6. Time-Average of a Simulation Attribute** Bubble-scale simulations can be too detailed for observing  
 368 general foam behavior and topological changes generate large fluctuations in attribute values that hide the  
 369 overall trends. A good way to smooth out these variations is to calculate the average of the simulation  
 370 attributes over all time steps, or over a time window before the current time step. This visualization  
 371 reveals global trends in the data because large fluctuations caused by topological changes are eradicated.  
 372 This results in only small variations between averaged successive time steps. The time window is a  
 373 parameter set by the user. We compute the average for the entire simulation (Fig. 8) if there are no  
 374 dynamic objects interacting with the foam. In this case, at a high level of detail, there is no difference  
 375 between different time steps in the simulations. For simulations that include dynamic objects interacting  
 376 with the foam (Fig. 7, 10) a smaller time window is appropriate, as objects may traverse transient states  
 377 that have to be analyzed independently.

378 Time-averages of several 2D foam simulation attributes are stored in `AttributeAverages2D`  
 379 and are referenced from `WidgetGl`. A pixel-based time-average of one simulation attribute is  
 380 computed by `ScalarAverage`, `VectorAverage` and `TensorAverage` for scalars, vectors  
 381 and tensors. Most of this computation is done in the base class `ImageBaseAverage`  
 382 (Fig. 4). Displaying an average of attributes is done by the graphics card fragment shader using  
 383 `AverageInterface::AverageRotateAndDisplay`<sup>1</sup> overwritten for each attribute type. For  
 384 3D simulations, unstructured grid data is converted to regular grid data and is cached in files inside  
 385 `.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.



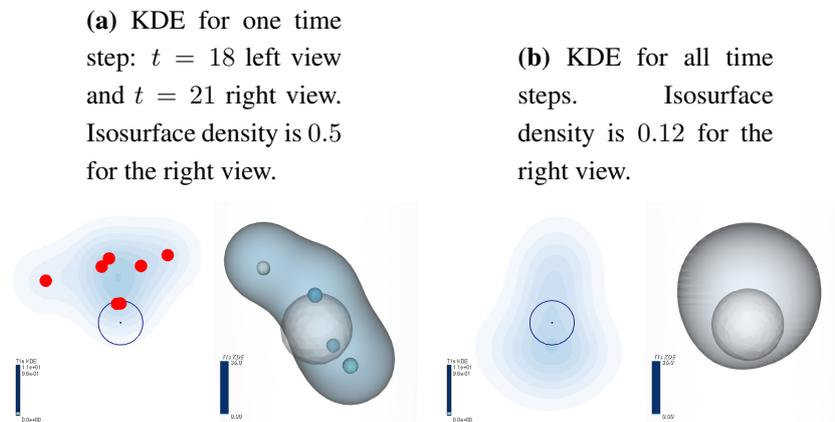
386 parsing `Simulation::Preprocess`. Time-averages of several 3D foam simulation attributes are  
 387 stored in `AttributeAverages3D` and are referenced from `WidgetVtk`. A time-average for  
 388 one attribute, for all types of attributes, is computed by `RegularGridAverage` and displayed  
 389 using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

391 *3.7. Topological changes kernel density estimate (KDE)* Topological changes, in which bubbles change  
 392 neighbors, indicate plasticity in a foam. Domain experts expect that their distribution will be an important  
 393 tool for validating simulations. Simply rendering the position of each topological change suffers from  
 394 over-plotting, so it may paint a misleading picture of the real distribution. We compute (see Lipsa et  
 395 al. [13] for details) a KDE for topological changes (Fig. 7, Fig. 12). While traditional histograms  
 396 show similar information and are straightforward to implement they have drawbacks which may prove  
 397 important depending on the context. Drawbacks of histograms include the discretization of data into bins  
 398 which may introduce aliasing effects and the fact that the appearance of the histogram may depend on  
 399 the choice of origin for the histogram bins [4,19]. Kernel-based methods for computing the probability  
 400 density estimate eliminate these drawbacks.

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

406 *3.8. Histograms* We provide both a histogram of bubble attribute values over one time step and  
 407 over all time steps. To facilitate data analysis, our histogram is configurable. The user can  
 408 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.



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

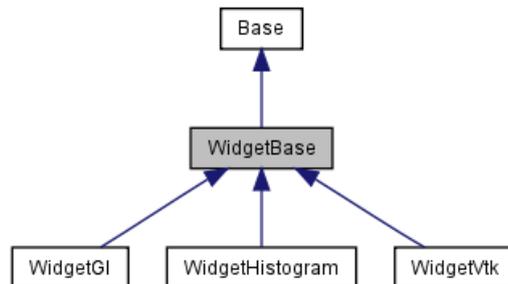
414 **3.9. Multiple linked-views** Foam scientists wish to understand what triggers certain behavior in

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

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

430 The *two halves* option facilitates visual comparison of two related foam simulations (Fig. 8). It  
 431 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.



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

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

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

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

465 **3.10. Interaction** Interaction with the data is an essential feature of our application.

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

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

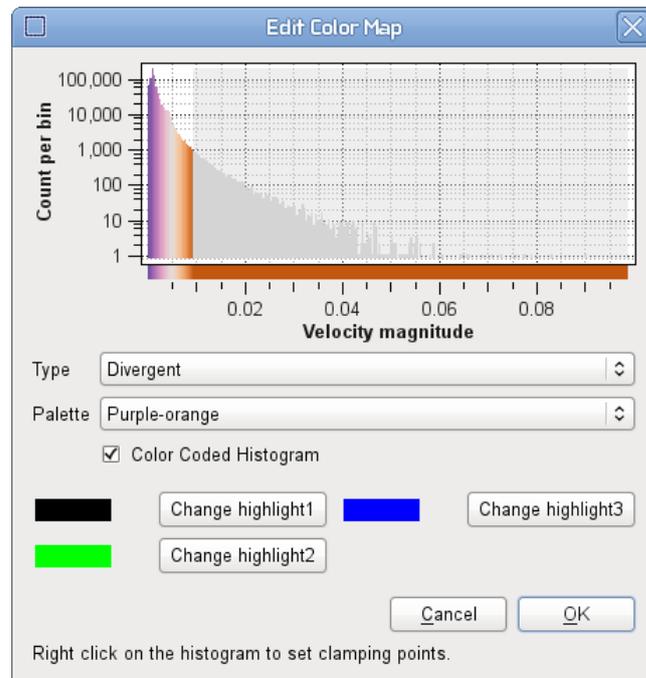
482 Selected bubbles or center paths constitute the focus of our visualization, and the rest of the bubbles  
483 or center paths provide the context [7]. The context of the visualization is displayed using user-specified  
484 semi-transparency, or it can be hidden altogether.

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

491 **4. Conclusions and Future Work** We present challenges faced by domain scientists and describe  
492 `FoamVis`, a software application designed to address some of these challenges. We describe its main  
493 implementation components and their interactions, and present `FoamVis`' main features together with  
494 implementation notes that describe how and where these features are implemented.

495 We see many directions for future work. We would like to add more algorithms for visualization of  
496 3D foam simulations, enable comparison between foam simulation and experiments and support analysis  
497 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.



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

501 **The authors declare no conflict of interest**

## 502 References

- 503 1. D. Bell. UML basics: The component diagram, 2004. Online document, accessed 25 June 2013,  
504 <http://www.ibm.com/developerworks/rational/library/dec04/bell/index.html>.
- 505 2. K. Brakke. The Surface Evolver. *Experimental Mathematics*, 1(2):141–165, 1992.
- 506 3. C. A. Brewer. ColorBrewer. Online document, <http://www.ColorBrewer.org>, accessed 3 March.  
507 2012.
- 508 4. O. Daae Lampe and H. Hauser. Interactive Visualization of Streaming Data with Kernel Density  
509 Estimation. In *Pacific Visualization Symposium (PacificVis)*, pages 171–178. IEEE, 2011.
- 510 5. flex - The Fast Lexical Analyzer. Online document, <http://flex.sourceforge.net/>, accessed 29 Nov.  
511 2010.
- 512 6. Bison - GNU Parser Generator. Online document, <http://www.gnu.org/software/bison/>, accessed 29  
513 Nov. 2010.

- 514 7. H. Hauser. Generalizing Focus+context Visualization. *Scientific visualization: The visual extraction*  
515 *of knowledge from data*, pages 305–327, 2006.
- 516 8. K. Inc. *The VTK User's Guide Version 5 (Paperback)*. Kitware Inc., 2006.
- 517 9. D. Lipsa. FoamVis, 2013. Online document, accessed 25 June 2013, <http://cs.swan.ac.uk/~csbob/research/foamVis/design/html/>.
- 518
- 519 10. D. R. Lipşa, R. S. Laramée, S. Cox, and I. T. Davies. Visualizing 3D Time-Dependent Foam  
520 Simulation Data. In *Lecture Notes in Computer Science, International Symposium on Visual*  
521 *Computing (ISVC)*, Rethymnon, Crete, Greece, July 2013.
- 522 11. D. R. Lipşa, R. S. Laramée, S. J. Cox, and I. T. Davies. FoamVis: Visualization of 2D Foam  
523 Simulation Data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2096–2105,  
524 Oct. 2011.
- 525 12. D. R. Lipşa, R. S. Laramée, S. J. Cox, and I. T. Davies. A Visualization Tool For Foam Research.  
526 In *NAFEMS World Congress (NWC) Conference Proceedings*, page 141, Salzburg, Austria, June  
527 2013.
- 528 13. D. R. Lipşa, R. S. Laramée, S. J. Cox, and I. T. Davies. Comparative Visualization and Analysis of  
529 Time-Dependent, 2D Foam Simulation Data. Technical report, Swansea University, 2013.
- 530 14. Microsoft. Model-View-Controller, 2013. Online document, accessed 25 June 2013, [http://msdn.](http://msdn.microsoft.com/en-us/library/ff649643.aspx)  
531 [microsoft.com/en-us/library/ff649643.aspx](http://msdn.microsoft.com/en-us/library/ff649643.aspx).
- 532 15. R. Prud'homme and G. Warr. *Foams in Mineral Flotation and Separation Processes*, pages  
533 511–554. Volume 57 of prud1996foams [16], 1996.
- 534 16. R. K. Prud'Homme and S. A. Khan. *Foams: theory, measurements, and applications*, volume 57.  
535 CRC Press LLC, 1996.
- 536 17. W. Rossen. *Foams in Enhanced Oil Recovery*, pages 413–464. Volume 57 of prud1996foams [16],  
537 1996.
- 538 18. D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide, Fifth Edition*. Addison  
539 Wesley, 2006.
- 540 19. B. Silverman. *Density Estimation for Statistics and Data Analysis*, volume 26 of *Monographs on*  
541 *Statistics and Applied Probability*. Chapman & Hall/CRC, 1986.
- 542 20. The Surface Evolver, Jan. 2008. Online document, accessed 29 Nov. 2010, [http://www.susqu.edu/](http://www.susqu.edu/brakke/evolver/html/evolver.htm)  
543 [brakke/evolver/html/evolver.htm](http://www.susqu.edu/brakke/evolver/html/evolver.htm).
- 544 21. Surface Evolver Workshop, Apr. 2004. Online document, accessed 1 Dec. 2010, [http://www.susqu.](http://www.susqu.edu/brakke/evolver/workshop/workshop.htm)  
545 [edu/brakke/evolver/workshop/workshop.htm](http://www.susqu.edu/brakke/evolver/workshop/workshop.htm).
- 546 22. D. van Heesch. Doxygen, 2013. Online document, accessed 26 June 2013, [http://www.stack.nl/](http://www.stack.nl/~dimitri/doxygen/)  
547 [~dimitri/doxygen/](http://www.stack.nl/~dimitri/doxygen/).
- 548 23. M. Ward, G. Grinstein, and D. Keim. *Interactive Data Visualization. Foundations, Techniques, and*  
549 *Applications*, chapter 10, pages 315–334. A K Peters, Ltd., Natick, Massachusetts, 2010.
- 550 24. D. Weaire and S. Hutzler. *The Physics of Foams*. Oxford University Press, Oxford, 1999.