BROWN UNIVERSITY
Department of Computer Science
Master's Project
CS-94-M9

"An Annotation System for 3D Fluid Flow Visualization"

by
Maria M. Loughlin

An Annotation System for 3D Fluid Flow Visualization

Maria M. Loughlin

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University

May 1994

This thesis by Maria M. Loughlin

is accepted in its present form by the Department of

Computer Science as satisfying the

thesis requirement for the degree of Master Of Science

Date 2/18/94 .... [illegible]

John F. Hughes

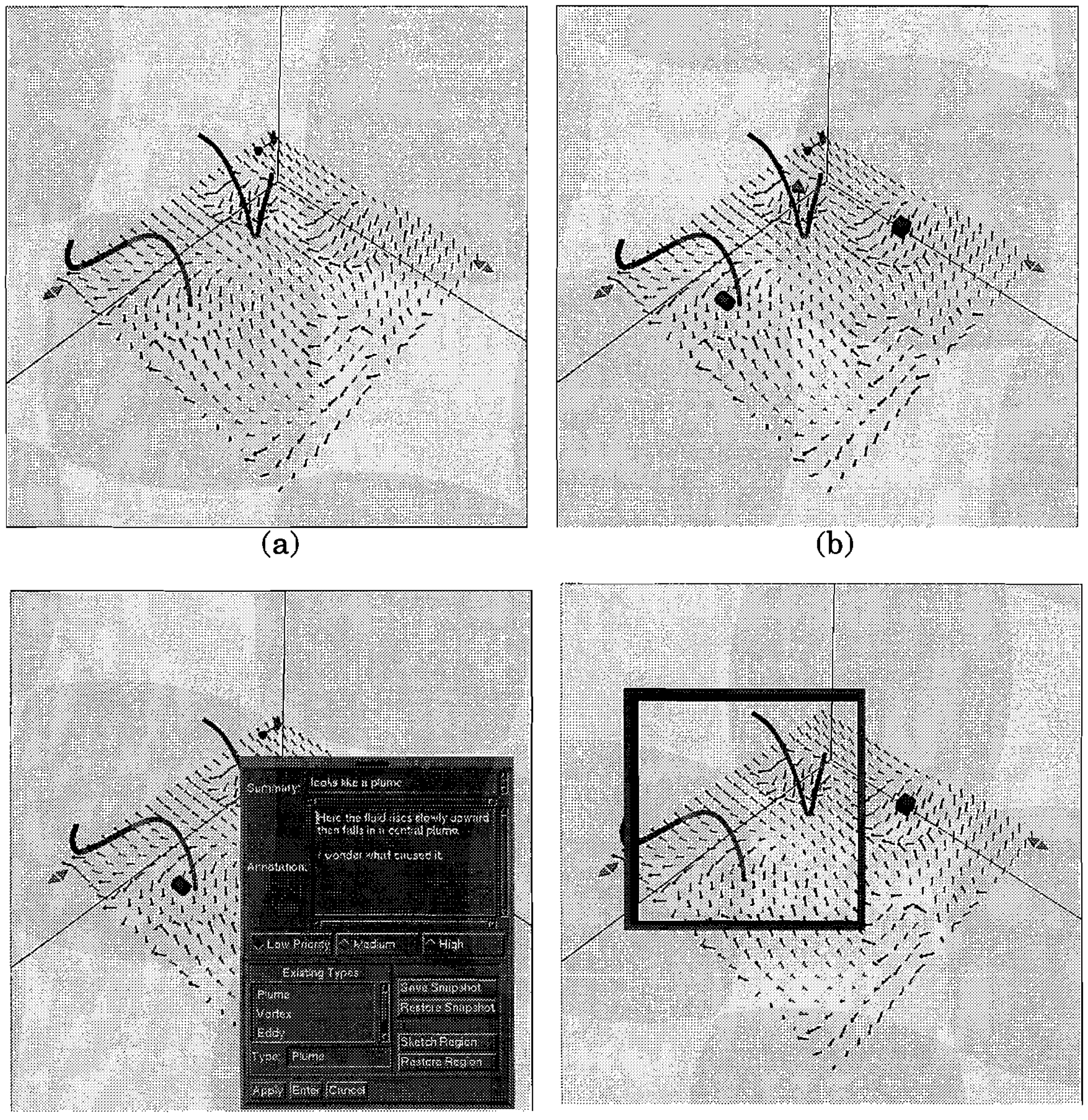
Approved by the Graduate Council

Date.................. ....

1.0 Introduction

This thesis presents an annotation system to support scientific data analysis. Data analysis can be defined as the process of distilling potentially large amounts of measured or calculated data into simple observations or parameters that characterize the phenomenon under study. One of the key activities in data analysis is recording results and histories of analysis sessions. However current interfaces for data analysis emphasize scientific visualization, focusing on the rendering and playback of images, and provide little or no annotation support. We describe a means to integrate annotation to the framework of scientific visualization tools.

Our annotation system allows users to record information in the data visualization itself. Annotation markers are placed in the visualization, and annotation information is associated with the markers. This allows contextual information storage and retrieval, and facilitates information sharing in collaborative environments. Thus the annotation system becomes a form of communication as well as a history of the data analysis session. Annotation markers also aid analysts in navigating through the data space, by providing landmarks at interesting positions. Figure 1 shows screen snapshots from the visualization and annotation system. The project has been implemented for three dimensional (3D) Computational Fluid Dynamics (CFD) applications. However, the techniques can be applied to visualization systems in any discipline, such as medical, geological, and business process visualization. The design can also be extended to 3D stereo and virtual-reality environments.



Summary: looks like a plume

Annotation: Here the fluid rises slowly upward then falls in a central plume.

I wonder what caused it.

Type: Plume

FIGURE 1. The visualization and annotation system (a) hedgehog and streamlines in a 3D fluid flow, (b) annotation markers at areas of high velocity, (c) annotation information-entry panel, (d) Magic Lens^{TM} hiding annotation markers.

Section 2 discusses the need for annotation in data analysis, and reviews previous approaches to annotation. Section 3 describes the design issues considered during the development of our annotation system. Section 4 details the implementation of an annotation system within a 3D modeling and animation system. Section 5 discusses possible future work, and section 6 gives some concluding remarks.

2.0 Background

2.1 The Need for Annotation in Data Analysis

Annotation was identified as a key component of the data analysis process in a study by Springmeyer *et al.* [Spring92]. The study, performed with ten analysts over a period of many months, tried to characterize the data analysis process and to consider how technology can be used to support it more effectively. The techniques of contextual inquiry and interaction analysis were used to observe scientists analyzing their own data. The study decomposed the scientific data analysis process into two primary activities: *investigation*, or exploring the data to extract information or confirm results, and *integration of insight*, or assimilating the resulting knowledge. Integration of insight was seen to involve organizing information and expressing the ideas generated.

Scientists recorded notes in some form, and inspected previous notes, in <u>every</u> observation session. Recording media included notebooks, scratch paper, and post-it notes. Two distinct types of annotation were observed:

- *recording*, or preserving contextual information throughout an investigation and

- *describing* or capturing conclusions of the analysis sessions.

Thus annotations were used both for organization and as records for later reference.These annotation operations were not directly supported by the visualization tool in any case, except in the form of file-naming conventions and printouts of isolated figures and lists of numbers. The authors conclude that expressing ideas is an important, but often overlooked, aspect of scientific data analysis. The report recommends that scientific data analysis tools assist analysts in maintaining a record of analysis sessions, and in annotating results of different stages of a study.

2.2 Annotation Support in Existing Systems

Scientific visualization tools typically provide little or no support for information annotation. Other tools developed outside the scientific data analysis domain, allow annotation in different ways. In this section, we review annotation support in scientific visualization and other environments.

2.2.1 Scientific Visualization Systems

Scientific visualization systems usually provide a level of annotation support that is helpful for making presentations from visualized data. Automatic Visualization System (AVS) facilitates attachment of labels to an image [Up89]. AVS also allows recording of a sequence of interactions with the visualization. Flow Analysis Software Toolkit (FAST), a software environment for visualizing scientific data, allows users to add textual titles to scenes and animations. Additionally, during a FAST session, all user interactions with the tool are recorded in a script, which the user may choose to save and replay [Ban90]. This support does not facilitate the *recording* and *describing* operations observed by Springmeyer *et al.*

2.2.2 Document-publication and Communication Systems

Annotations of various sorts have been integrated in applications outside the scientific visualization domain. MacDraw 1.1, a 2D paint program, introduced a notes feature, which allows insertion of annotations using the post-it metaphor. Solutions International's SuperGlueII [Thom89], also uses the electronic equivalent of post-its in their GlueNotes feature. GlueNotes allows addition of textual and image-based annotations to documents. SuperGlueII also supports printing annotated documents. This is handled by printing a miniature view of the document in which each note's position is marked and assigned a number. The notes themselves, identified by number, are printed below this thumbnail view.

The Media View system [Phil91], developed in the Next Step environment, provides a further level of annotation support. This tool extends the paradigm of the traditional document to electronic documents that can include text, line art, images, sound, video sequences, and computer animations. The post-it metaphor can be applied to all media components. Media View documents can also include a data set, so that users can perform and view a simulation wholly within Media View. There is also a facility for visualizing data sets produced on another computer. The authors of this tool state that Media View does not compete with systems such as AVS or APE for scientific visualization. Rather it complements those systems by providing a vehicle for sharing, archiving, and further exploring the visualizations they produce.

Document annotation is used as a key means of communication in the Wang Laboratories multimedia communication system, Freestyle [Francik91]. Freestyle's multimedia messages are based on images, including screen snapshots and hand-drawn sketches. Users can annotate images with synchronized pointing, drawing, writing and speaking. Graphic buttons above the image are used to play back the synchronized voice, hand-drawn and typed messages. Synchronization of input modalities is one of the key features of this tool, as it allows messages to contain information about the process by which they were created. The stylus cursor's movement which generated the annotation is played back with the annotation, allowing users to point while talking, and so refer to "this" object with coordinated hand and voice references. Francik *et al.* observed the use of Freestyle (and its prototypes) in groups with targeted applications, such as law, finance, insurance, health, travel, entertainment, commerce, engineering, and utilities. Experience showed that much of its success relied on the seamless integration of annotation capabilities and regular applications. Users could capture the details of what they were working on, add comments or questions, and send the message without greatly interrupting the flow of work.

Verlinden *et al.* [Ver93] developed an annotation system to explore communication in Virtual Reality (VR) environments. In general, communication in virtual reality systems is restricted, as the user must interrupt the simulation to take notes or get some extra information about features of the environment. Verlinden's system overcomes this problem by embedding verbal annotations in the VR space. Users can read, write and communicate using these annotations. The annotations are represented as visual 3D markers, which can be attached to objects or locations in space. When the user activates a marker, using a ray intersection technique, the verbal message stored with that marker is sounded. Annotations are created by moving to the required position and then pressing a mouse button to create a marker and make a recording. The annotation system was tested in a prototype VR tourist guide to locations in the city of Atlanta. Based on the success of this prototype, the authors feel that the addition of verbal communication opens up a range of new uses for virtual environments.

3.0 Design Issues

Before embarking on an annotation system implementation, we consider the design issues of such a system.

3.1 How to Integrate Annotation in a Visualization System?

An annotation system for the data analysis process must fit naturally within a visualization system. Within the framework of the visualization system, the annotation system must be available at all times to support both the "describing" and "recording" operations observed by Springmeyer *et al.*

One of the first issues is the placement and storage of annotations. Where should annotations be stored? Traditionally, annotations to scientific visualizations are recorded on paper or in electronic files, and both the dataset and the files are labelled to mark their association. Thus the analyst assumes the cognitive load of associating annotations with locations and features in the visualization. However, the visualization space of many scientific visualization tools is three dimensional, and thus provides a 3D context in which annotations can be placed. Recording annotations in this space provides a strong integration of annotation and visualization. It also capitalizes on human's spatial senses by facilitating the retrieval of information based on its spatial location in the visualization. Thus the cognitive load required to associate annotations with features is reduced to a perceptual level.

The decision to insert annotations in the visualization space creates an immediate conflict between the annotation and visualization data analysis functions. Both compete for screen territory. We do not wish to impose any restrictions on the amount of information that can be recorded. However, since the information is contained in the data itself, we do not wish annotations to obscure data. In fact, visual clutter is already a problem in many scientific data visualizations. An annotation system must be designed so that it does not aggravate this problem. Our approach is to associate each annotation with an *annotation marker*. The marker is a small geometric object, positioned in the data space by the user. The geometry and graphic attributes of the marker are chosen so that they are easily distinguished from existing visualization tools. By clicking on the marker, a user can expand the annotation to read or add to the annotation's content. Separation of the annotation's content from the annotation marker in this way allows direct insertion of arbitrarily large annotations.

3.2 What Kind of Information to Store in an Annotation?

Annotations must be powerful enough to capture information considered important by the data analyst. This prompted an investigation of the types of information that are manipulated by data analysts. Tanimoto [Tan90] distinguishes between *data*, *information* and *knowledge.* Data consists of raw figures and measurements, which do not necessarily answer the questions that users may have. Information is more refined, and may be the result of processing crude data, or answering specific questions posed by users. Knowledge is a refined type of information. It can be considered as "information in context", that is, information organized so that it can be readily applied to solving problems, perception, and learning. Bertin [Bert81] classifies the levels of information in a similar way. He considers information as a relationship which can exist between elements, subsets or sets. The broader the relationship, the higher the level of information. We assume that an annotation system should be able to store information at each of these levels - scientists need to record both the data values at probe points in the data set, and a higher level qualitative analysis of these figures. We need to consider how each of these levels of information can be represented.

We also need to consider whether an annotation system should be customized for the application at hand. Some aspects of an annotation, such as date of creation and author, are likely to be relevant to all applications. It is possible, however, that the real power of an annotation system is revealed only when it is customized. Springmeyer *et al.* support this idea, stating that "a designer can apply knowledge of how domain activities are actually practiced to improve the effectiveness and usability of software tools to support data analysis." Syntactically, this means that an annotation system should be able to interact with scientists in familiar terms. Semantically, the characteristics of the domain information should be easily captured by the annotations. For example, if the information of a particular application is time-varying, the annotation system should provide time-varying annotations that can track the features being described.

Many modalities (textual, graphical, image, video, audio, sensory), are available for information capture in an annotation system. Two dimensional text, graphics and images, are the standard annotation modalities. Aural annotation is also an effective candidate. Chalfonte, in an experiment on the use of annotation for collaborative document authoring, found aural annotations a richer and more effective medium for high-level communication [Chalf91]. Freestyle showed that coordinating hand/cursor movements with textual and aural annotations adds a further advantage. Virtual reality environments may have annotation needs different from those of traditional desktop environments.

3.3 How to Interact with an Annotation System?

The graphical user interface of an annotation system is necessarily a mixture of 2D and 3D techniques -- 2D interaction methods with 2D metaphors such as sketchpad/paper, and 3D interaction techniques with objects in the 3D world of the data visualization. Annotations that interface with non-graphical modalities, such as audio or tactile interaction, require other interface types. Some of the principles of user interface design are independent of the dimensionality and modality of the application space. For example, a good user interface allows its users to work with minimal conscious attention to their tools. A direct manipulation interface, that is, an interface in which the objects that can be operated on are represented physically, helps achieve this goal [Fol92]. Similarly, it is important for a user interface to provide feedback on the status of user-computer interactions, the current settings of domain variables etc. For example, in fluid flow visualization, a user navigating through a data set needs feedback on his or her location in the visualization space.

A user interface must be designed to suit the diverse community of its users. The functionality should be easy-to-use, so that novice users will quickly be able to use the system, and yet flexible, so that advanced users may perform complicated tasks. This is especially important in an annotation system, which must be simple and unobtrusive enough to be adopted by data analysts (whose primary interest is gaining information from a dataset), yet must be powerful enough to capture all that an analyst considers important in an annotation. An annotation system must also be flexible enough to support the different styles in which its users analyze data. While scanning a large dataset, users may want to position markers at many interesting locations, as "placeholders", without recording annotation content. However, users who are focused on one area may prefer to perform the instantiation, positioning, and recording of an annotation at the same time.

There are design issues specific to 3D graphical user interfaces [Conn92]. First, they must deal with the complexity introduced by 3D viewing projections, visibility determination, etc. Second, the degrees of freedom in the 3D world are not easily specified with common interface hardware. Substantial manual dexterity may be required to perform 3D interaction tasks. Third, the interface can easily obscure itself. The use of widgets (encapsulations of 3D geometry behavior used to control or display information about application objects) allows a higher bandwidth between the application and the interface. Some of the guidelines for successful 3D widget design [Snib92] are self-disclosure, implicit versus explicit control of parameters, constraint on the degrees of freedom where appropriate, and design for the intended use.

4.0 Implementation

This section describes the annotation system which was implemented. We begin by setting a context for the implemented system with a description of fluid flow visualizations and the software development environment. Then we discuss the main components of the annotation system - the annotation markers, support for information capture, and interaction techniques.

4.1 Fluid Flow Visualizations

Computational fluid dynamics (CFD) involves the use of high speed computers to simulate the characteristics of flow physics. Computed flow data is typically stored as a 3D grid of vector and scalar values (e.g., velocity, temperature, and vorticity values), which are static in a steady flow, and change over time in an unsteady flow. CFD visualization tools allow a scientist to examine the characteristics of the data in 3D computer images. Interaction with the visual representation is essential in the exploratory process of data comprehension and analysis. The goals of the interaction can be described hierarchically as *feature identification*, *scanning*, and *probing* [Haim91]. Feature identification techniques help locate flow features over the entire domain, and give the scientist a feel for the position of interesting parts of the flow volume. An example of this type of technique is a vector hedgehog, a three-dimensional array of velocity vectors, that may be thresholded to display velocity vectors in areas of high velocity. Scanning techniques are used to interactively search the domain, by varying one or more parameters, through space or through scalar and vector field values. Many scanning techniques are well established. These include cutting planes, planar surfaces which slice the computational domain and show scalar field value at each grid point of the plane, and iso-surfaces, which are three dimensional surfaces of a constant scalar value. Probing techniques are localized visualization tools, typically used in the final step of investigating a flow feature, to gather quantitative information. Examples of probing tools include streamlines and particle paths, which show the path in which a particle would flow if positioned in a steady or unsteady fluid flow.

Brown University has developed a flow visualization tool for researching new modes of interaction with flow visualization tools. The annotation system was developed in the framework of this flow visualization system. This provided a context for the annotation effort, and a testbed for techniques to integrate visualization and annotation functionality.

4.2 The Development Environment

The annotation system was developed using FLESH, an object oriented animation and modeling scripting language [Mey93], and C++. In the FLESH programming language, scenes are described as collections of "objects". Objects belong to object classes, which dictate their behaviors. Some object classes have a geometric representation, others, such as the camera class, do not. As in traditional object-oriented systems, objects have methods associated with them. In FLESH, objects may interact with other objects through the use of dependencies. The FLESH language is interpreted by the UGA system through its graphical modeler/animator, Trim [Hub91].

The annotation system is defined by the FLESH object classes it uses. These include geometric objects such as annotation markers, 3D regions, region vertices, lenses, and non-geometric objects, such as a holders for collections of annotations and an annotation filter. Some of these FLESH classes have corresponding C++ classes, in which data is stored, and compute-intensive operations performed. This allowed us to benefit from the power of an interpreted interactive prototyping modeling system, and the efficiency of a compiled language. The software was developed on Sun Sparcstation 10 and Hewlett Packard 9000-735 workstations.

4.3 Annotation Markers

Annotations are represented in the 3D space of the flow visualization as small geometric objects, known as annotation markers. Each marker is associated with an annotation recording, which the user can edit at any time. In this section, I describe the annotation markers in terms of their geometry and behavior.

When an annotation marker is created, it is respresented as a small sphere. Feedback from data analysts indicated that this icon's lack of self-disclosure hindered their acceptance of the annotation system. Thus, the geometry of a marker is now designed to give some visual feedback on the content of the annotation. In the context of fluid flow visualizations, the user can define annotation keywords (such as plume, vortex, eddy, bifurcation), and associate a geometry with each keyword. When the user assigns a keyword to the annotation, the marker takes the associated shape. It is likely that other mappings between graphical attributes of markers and annotation content would also be useful. For example, the color saturation of a marker could depend on the age or priority of the annotation. Annotations are organized hierarchically in containing objects known as *annotators*. The size and color of all markers of an annotator can be changed, to highlight the fact that they belong to separate hierarchies. If many scientists work collaboartively on a data set, for example, each scientist can define a unique color and size for her markers.

Since the function of a marker is simply to identify points of interest in the visualization, its behavior is quite simple. A marker is created when the user presses the annotation push-button. It appears at the point at which the current camera is focused. If the camera is focused at or near the feature of interest, this greatly simplifies the user‘s task of positioning the marker. Users can translate and rotate markers with simple mouse movements. The user can also project "interactive shadows" of the marker on the planes defined by the principal axes [Hern92]. Each shadow is constrained to move in the plane in which it lies. If a user moves a shadow, the marker moves in a parallel plane. This constrained translation helps in precisely positioning a marker. Markers can be highlighted in response to a filter request. In the current system, the color of a marker changes to a luminescent yellow when highlighted. This simple approach seems adequate. However, the user may change this highlight behavior, by, for example, having highlighted markers flash between alternating colors.

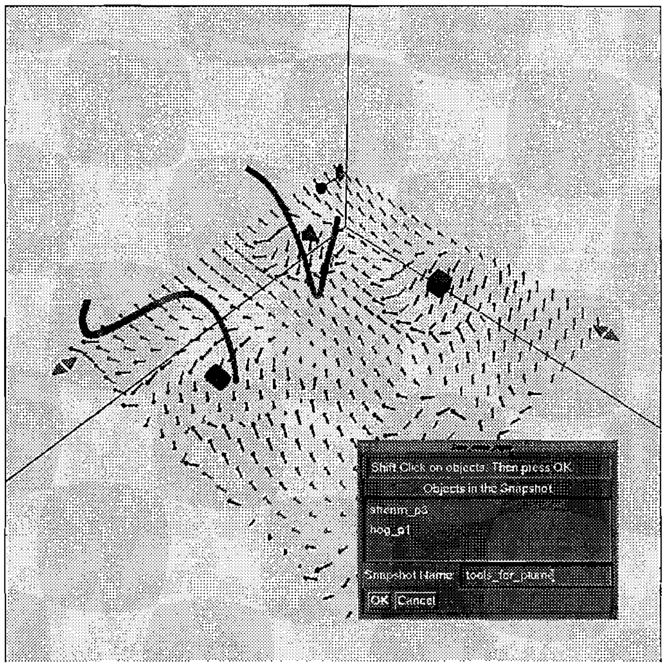
Since the features of unsteady fluid flows move over time, a user would like the annotation describing a particular feature to follow the feature's movement in the visualization. The current annotation system provides partial support for this by allowing the user to specify the position of an annotation at any number of points in time. The annotation markers then linearly interpolate between the specified positions as time runs forward.

4.4 Knowledge Stored

Our first prototype annotation system allowed storage of keyword, textual summary and description, author, and date information. Some of this information (author and date) are captured implicitly when the annotation is created. The rest must be explicitly added after the author has "opened" the annotation by clicking on it. This data entry is performed via a 2D Motif-based panel of buttons and text widgets. In an effort to customize the annotation system to fluid flow visualizations, we consulted with fluid flow analysts during the course of the project, to gain insight into the types of information that should be captured in this context.

4.4.1 Saving Visualization Snapshots

One of the key additions to the annotation system results from the interactive nature of flow data analysis. As described earlier, a scientist must insert flow visualization tools in the flow space to "see" the underlying data. Much time is spent determining which tools most effectively highlight a feature, and positioning and orienting both the tools and camera to best show off the feature being described. Springmeyer *et al.* observed this activity of the data analysis process, and described it as *orientating* the data, or altering a representation to gain perspective. To support this activity, our concept of an annotation was expanded to include "snapshots" of the flow tools which display a feature. To take a snapshot, the annotator simply clicks on flow tools relevant to the feature being described. Any number of snapshots can be stored with an annotation. When an annotation is restored at a later time, the analyst is presented with a list of all saved snapshots, and can simply restore each snapshot to see how the annotated feature is displayed by flow visualization tools. Figure 2 shows a user saving a snapshot of visualization tools with the annotation system.



stremm_p3
hog_p1
Snapshot Name: tools_for_plume

FIGURE 2. Saving a snapshot of visualization tools

4.4.2 Describing 3D Volumes

It also became obvious that annotation markers, which are appropriate for locating point features in a visualization, are not suited to marking larger-scale features of a fluid flow. Fluid flows contain volume features, such as vortices (mass of flow with a whirling or circular motion), plumes (mass of rapidly descending flow) and eddys (current of flow running contrary to the main current). Annotators may want to associate an annotation with a region of the visualization space, rather than a single point in the space. To achieve this, the user needs some way to sketch the 3D volume in the visualization space. The volume-sketching method must be intuitive, so that flow analysts (who may not be interested in becoming artistic volume sculptors!), can easily describe the volume. Also, the resolution of the volumes sketched need not be too precise. A sketched volume need only be as precise as the grid on which the flow field is defined, as it is not meaningful to refer to volumes at any greater resolution.

3D volume sketching is difficult in the traditional graphics workstation environment for a number of reasons. The most fundamental problem is that traditional 2D input devices are not expressive enough to sketch out a 3D volume easily. During the course of this project, we experimented with two approaches to 3D volume sketching.

The first approach uses a 3D input device to control a 3D paintbrush widget, and uses the voxel representation of the flow volume. In these respects, it resembles the free-form sculpting system developed by Galyean and Hughes [Gal92]. When in paint mode, the cursor of the 3D mouse is depicted as a 3D paintbrush. A boolean value "on" or "off" is associated with all voxels of the visualization. At the start of a volume-sketching operation, all voxels are set "off." Subsequently, all voxels touched by the paintbrush are turned "on." The resulting volume is the union of all the "on" voxels. The paintbrush itself is a user-controllable 3D widget. When the user wishes to change the paintbrush parameters, he or she displays the paintbrush affordances. These include extent handles for each of the three principal axes of the brush. The brush is resized (at voxel resolution) by dragging on the extent handles. Similarly, the brush can be alternated between additive and subtractive (erasing) modes via a button on the widget.

This 3D paint implementation proved very difficult to use. It is not easy to sweep out a desired volume with the brush, without constantly changing the view perspective. The 3D volumes created tend to be irregular, with large indentations and holes at voxels missed by the 3D paintbrush. Perhaps this implementation coupled with some "intelligent" space-filling algorithm would be more useful.

Our second attempt at 3D volume sketching is easier to use, but more restrictive in the volumes that can be defined. In this implementation, the user positions "pegs" that define the extreme vertices of the region to be drawn. The pegs are created and moved within the visualization in a way similar to the creation and translation of annotation markers. When the user is done positioning pegs, the system can draw the convex hull of the pegs. Vertices can be added, deleted and moved, and the volume redrawn, until the user is happy that the volume is accurate. The quickhull algorithm, as implemented at the Geometry Center, University of Minnesota [Barb93], is used to compute the 3D convex hull of the pegs. The convex hull code generates the voronoi triangulation of the facets, which can then be drawn in outline or transparent mode. This implementation provides a simple means to draw 3D regions. However, since it uses the convex hull of the pegs, certain shapes, such as a 3D "L" shape, can not be sketched. Figure 3 shows a 3D region being sketched by the convex hull method.

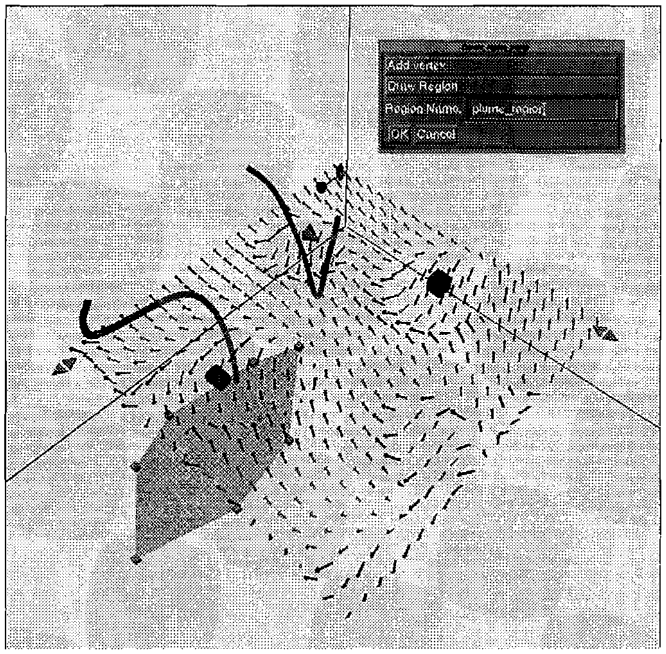


FIGURE 3. Sketching a 3D region by the convex hull method

4.5 Retrieving the Annotations

Effective information retrieval and communication requires that a user can easily identify annotations relating to a specific topic, by a specific author, etc. The annotation system facilitates such data filtering in two ways.

Firstly, a traditional database filter is provided. The user can specify data selection criteria (such as the annotation creation date, author, or keyword), via a 2D Motif panel. Annotations that satisfy the search criteria are highlighted.

A second filter uses a Magic Lens^{TM} as described by Bier *et al.* [Bier93]. A Magic Lens is a rectangular frame, placed in front of the visualization, which defines both a screen region and an operation. The geometry of the lens is a 2D frame. It is constrained to move and scale only in a plane between the viewer and the film plane, parallel to the film plane. The lens performs some operation on the data behind it. Traditional lenses provide pixel-based operations, such as magnification. Magic lenses can access application-specific data structures, and so can perform qualitatively different operations. Four functions are defined for the lens in the annotation system. The first sets the color of all FLESH objects, except annotation markers, to gray. This facilitates finding markers in a cluttered scene. The second lens operator shows only the annotations that satisfy the criteria specified in the Motif-based database filter. The third lens function undraws all annotation markers behind the lens. Finally, the default function hides all annotation markers and all interaction handles on the visualization tools behind the lens. This is useful when a scientist wishes to remove all clutter and focus on the visualization data only. Many other interesting lens functions could be defined. One such function could be to remove all fluid flow tools except those in the user-sketched volume behind the lens.

We believe that the magic lens alleviates the problem of visualization and annotation functions sharing the same screen space. Using the lens, a scientist can tightly integrate the two functions when appropriate. When she wishes to focus exclusively on either visualization or annotation, however, the clutter introduced by the other component can be hidden.

5.0 Future Work

This thesis has laid down the foundations for an annotation system for fluid flow data analysis. The work can be expanded in many ways.

First, the existing system can be further customized for fluid flow applications. The facility for taking snapshots of visualization tools could be extended to record view parameters. Then, snapshots stored in annotations could automatically be viewed from the same camera angle and with the same magnification as when the snapshot was taken. Flow visualization tools could also be used when specifying positions and volumes in the visualization space. For example, users may want to attach annotation markers or region vertices to a tool such as a streamline, and constrain movement of the object along (or perpendicular to) the streamline. Furthermore, a set of streamlines could directly specify a volume. Annotations could also become more active in the data investigation process. For example, annotation markers could be used as seed points for automatic flow feature-characterization code. The output of the feature characterization code (i.e., specifications of the feature found), could then be added to the annotation content. The current system's support for time-varying annotations could also be improved. In an unsteady fluid flow, an annotation describing a flow feature should follow the movement of the feature through time. Currently, an annotator can specify the position of the annotation marker at any point in time. The process could be enhanced by setting the location of the annotation marker to depend on the feature's position, as found by feature-characterization code. Formal user-studies could be run to see how data analysts interact with the system. This may lead to design revisions and extensions.

Second, the techniques could be extended to other application areas and visualization environments. Examination of annotation requirements of other applications would allow one to determine the generic needs of an annotation system, and the trends in annotation customization required for different disciplines. This would allow a broader annotation system design, to which new capabilities could easily be added. Springmeyer *et al.* noted that data analysts tend to keep records of their interactions with visualization systems. Perhaps the annotation system can help in recording and examining these interactions. Also, much of the work of data analysts revolves around comparing different visualizations. As implemented, the annotation system is not equipped to fit in the context of more than one data set. It is also unclear how the information in annotations can be accessed, other than through the visualization system. For example, we have not defined a means to print out the visualization and annotation.

We would also like to experiment with annotation in virtual reality environments. User studies should be performed to determine which annotation modalities are appropriate in this space. If textual annotations are used, one would have to determine where to place the text: floating in space near the marker, or on 2D panels which exist in the virtual space, or perhaps in some other place. New interaction mechanisms for annotation markers and filters need to be developed. The magic lens concept, for example, would need to be redefined, since virtual reality environments don't include the concept of a film plane in which the lens can lie.

6.0 Conclusions

Since the purpose of analysis is insight rather than pretty pictures, it is clear that data analysis tools should support expression and communication of the knowledge gained. This thesis presents an approach to annotation support, which is customized for analysis of fluid flow data. The integration of raw scientific data and higher-level information may change how analysts record and communicate information. Our notion of a publication as a textual document, with embedded images and data graphs, may be replaced in data-intensive domains by a data-set with embedded annotations.

7.0 Acknowledgments

Thanks to my advisor, John Hughes, for much inspiration and guidance during this project. Thanks also to the members of the Brown graphics group, especially Bob Zeleznik, Ken Herndon, Dan Robbins, Nate Huang and Tim Miller, for their help in implementing this project. Thanks are also due to Steve Bryson (NASA Ames), Paul Fischer, Fred Bisshopp and Sam Fulcomer (Brown Applied Mathematics Dept., Brown), and Lee Silverman (Planetary Geology Dept., Brown) who gave valuable insight into the requirements of an fluid flow analysis systems. I would also like to acknowledge the financial support provided by Digital Equipment Corporation's Graduate Engineering Education Program which made my studies at Brown possible. Finally, thanks to Marty for his support at all stages of this masters program - thanks for the times you calmed me down when I was getting panicked, and for the times you got me going when I got lazy!

8.0 Appendix: Implementation Details

8.1 Development Environment

The annotation system is written in Brown University's UGA graphical environment. UGA is a modeling and animation system that supports the construction and use of timeparameterized objects. UGA clients can access the UGA database at many levels. Using an interpreted scripting language, FLESH, a user can add new objects and change existing UGA objects. UGA libraries can also be accessed from procedural languages through a set of C/C++ routines which form the database interface. The annotation system is written in both FLESH and C++. The source code at Brown resides in $UGA_ROOT/script/ann/src.

8.2 Architecture

The system is designed using the object-oriented paradigm. In this section, I describe the main object classes. Almost all classes are instantiated as both FLESH classes (where geometry and graphical behavior are described) and C++ classes (where data-traversal and compute-intensive operations are performed.) User-interaction is controlled at the FLESH level. The FLESH script contains the address of the top-level C++ object, so that a FLESH object can call a method of a C++ object at any time. Parameter and result passing between FLESH and C++ is achieved using UGA's VALU data-management package, which deals with data in syntactic rather than semantic terms.

The annotation system is organized hierarchically -- a top-level object contains global information about the current annotation session. This object contains a list of all active annotators, or sets of annotations. Each annotator contains a list of its component annotation. This containment structure is shown in Figure 4.

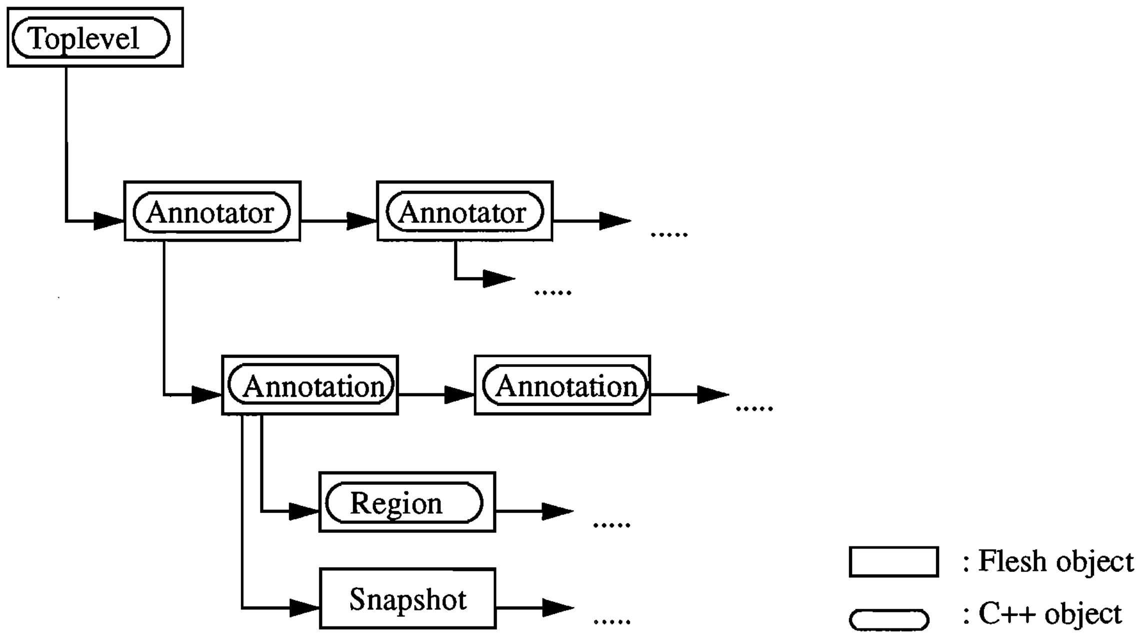


FIGURE 4. Containment within object classes

The toplevel object is instantiated in both FLESH and C++. The FLESH object holds generic information about the annotation system, such as the currently selected annotator and annotation and the mappings between annotation keyword and geometry. In addition, the FLESH and C++ toplevel objects contain a list of all annotators that have been added in the current visualization session, or imported from a previous session.

Each annotator manages a set of annotations. The user controls how the annotations are organized into annotators. We envision that users may want to group annotations according to the feature being described, or the author. Each annotator also has a text field, which can be used to describe the grouping.

An annotation associates information with a graphical marker. The FLESH annotation object controls the geometry and behavior of the visible annotation marker. The corresponding C++ object contains the annotation's content - date, author, description etc. If the user has stored one or more snapshots with the annotation, these are stored with the FLESH annotation object. Similarly, any regions associated with the annotation are stored in the C++ and FLESH objects.

8.3 Motif Interface

UGA provides an interface to simple Motif widgets such as scroll bars, buttons, and text windows. This interface, contained in the IBOX package, allows us to define a Motif interface as UGA objects in the FLESH language. A callback function, written in FLESH, can be associated with each widget. This allowed rapid prototyping of the 2D Motif interface. Figure 5 shows a simple panel of Motif widgets created with the IBOX package.

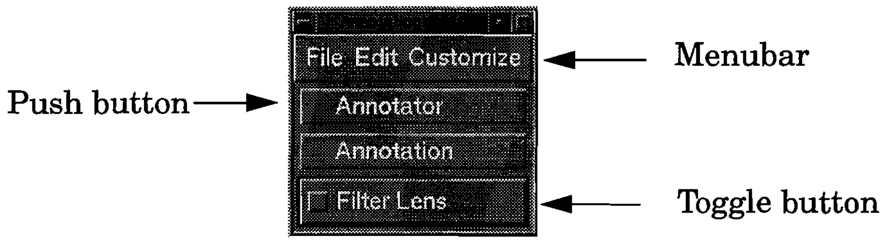


FIGURE 5. Sample IBOX Motif panel

8.4 Other Interesting Modules

8.4.1 The Save and Restore Mechanism

An annotation system must provide a way to save and restore annotations. We made this save/restore mechanism as generic as possible, so that the same scheme can be used to save snapshots of visualization tools. We achieved this by defining a prototype saver, or *saver_p*, object in FLESH. One of the methods defined for a saver_p, the *Save* method, takes a list of objects to save, and adds to its own definition a field per object. The field for a specific object records the value of key attributes of that object. All objects can define a *save_list* with the names of the key attributes that should be saved. If an object does not have a custom save_list, a default list of attributes -- translation, rotation, scale, and color -- is used. The saver_p also has a *Restore* method. This method recreates each of its saved objects, and restores each object to its original state, based on the values in the save_list.

A saver_p can store any collection of objects. When a tool snapshot is associated with an annotation, a saver_p is created to store the tools. This saver_p is then added to the annotation's list of snapshots. When a user saves an annotator, a saver_p is created to contain the save_list of each annotation in the annotator. Then this saver_p and the saver_p's of the snapshots associated with these annotations are written to a text file using the UGA FILEwrite command. When the annotator is imported at a later time, the saver_p's are parsed, and the restore method applied to each to recreate the annotations and any tool snapshots that were saved.

The information stored in C++ data structures must be saved and restored in a different way. This is achieved by simply looping through all annotations and writing their content to a text file. When the annotations are imported, a C++ annotator and annotation is created for each saved object, and the saved text written into the new structures.

8.4.2 Magic Lens

The geometry of the magic lens is implemented as the CSG (Constructive Solid Geometry) difference of two cubes, scaled such that they are very shallow relative to their height and width. The lens's position is initialized to a plane in front of the film plane. From there, it is constrained to translate only in the plane in which it lies. In the current implementation, the lens cannot be rotated.

When the lens is active in the visualization, we perform two actions on each camera update. First, we determine what objects in the 3D scene are in the frustum behind the lens. This is achieved by finding the projection of each object on the camera's film plane and determining which of the projected objects lies within the projection of the lens on the film plane. Second, we apply the current lens callback function, which is stored as a field of the lens object, to the enclosed objects. Typical callback functions are hiding annotation markers, hiding widget handles, and highlighting selected annotation markers.

8.4.3 3D Regions

The three-dimensional sketched regions are formed as OD objects. OD is a library of drawing primitives within UGA that provides architecture-independent access to hardware-accelerated 3D graphics. When the vertices of a region are positioned by the user, FLESH passes the list of vertices to a C++ routine. The routine then uses the qhull algorithm to determine the vertices in the convex hull. An ODobject is then formed from a triangulation of the resulting faces. All visible regions are added to the current viewer's list of displayed objects.

9.0 Bibliography

[Ban90] Gordon V. Bancroft, Fergus J. Merritt, Todd C. Plessel, Paul G. Kelaita. R. Kevin McCabe, Al Globus, *FAST: A Multi-Processed Environment for Visualization of Computational Fluid Dynamics*, Proceedings of the First IEEE Conference on Visualization 1990, pp. 14-27.

[Barb93] C. Bradford Barber, David P. Dobkin, Hannu Huhdanpaa, *The Quickhull Algorithm for Convex Hull*, Geometry Center Technical Report GCG53, U. Minnesota, July 1993.

[Bert81] Jacques Bertin, *Graphics and Graphic Information-Processing*, Walter de Gruyter & Co.

[Bier93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, Tony D. DeRose, *Toolglass and Magic Lenses: The See-Through Interface*, Proceedings of the ACM Computer Graphics Conference, 1993

[Chalf91] Barbara L. Chalfonte, Robert S. Fish, Robert E. Kraut, *Expressive Richness: A Comparison of Speech and Text as Media for Revision*, Proceedings of the ACM Computer Human Interaction Conference 1991, pp. 21-26.

[Conn92] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, Andries van Dam, *Three-Dimensional Widgets,* Proceedings of the 1992 Symposium on Interactive 3D graphics, pp. 183-188.

[Fol92] James Foley, Andries van Dam, Steven Feiner, John Hughes, *Computer Graphics Principles and Practice*, Addison Wesley 1992.

[Francik91] Ellen Francik, Susan Ehrlich Rudman, Donna Cooper, Stephen Levine, *Putting Innovation to Work: Adoption Strategies for Multimedia Communication Systems*, Communications of the ACM, Dec 1991, Vol. 34, No. 12, pp. 53-63.

[Gal91] Tinsley A. Galyean, John F. Hughes, *Sculpting: An Interactive Volumetric Modeling Technique,* Proceedings of the ACM Siggraph 1991, pp. 267-274.

[Haim91] Robert Haimes and Dave Darmofal, *Visualization in Computational Fluid Dynamics: A Case Study,* Proceedings of the Second IEEE Conference on Visualization 1991, pp. 392-397.

[Hern92] Kenneth P. Herndon, *Interactive Shadows,* 1992 UIST Proceedings, November 1992, pp 1-6.

[Hub91] Philip M. Hubbard, Matthias M. Wloka, Robert C. Zeleznik, *UGA: A Unified Graphics Architecture*, Technical Report CS-91-30, Department of Computer Science, Brown University, Providence, RI 1991.

[Mey93] Tom Meyer and Nate Huang, *Programming in FLESH*, Computer Science Department, Brown University, April 1993.

[Phil91] Richard L. Phillips, *An Interpersonal Multimedia Visualization System*, IEEE Computer Graphics and Applications, May 1991, pp. 20-27.

[Snib92] Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, D. Brookshire Conner and Andries van Dam, *Using Deformations to Explore 3D Widget Design,* Proceedings of the ACM SIGgraph 1992, pp. 351-352.

[Spring92] Rebecca R. Springmeyer, M. M. Blattner, N. L Max, *A Characterization of the Scientific Data Analysis Process,* Proceedings of the Second IEEE Conference on Visualization 1991, pp. 235-242.

[Thom89] Tom Thompson, *Save and Annotate Your Mac Output*, BTYE, September 1989, pp. 82-84.

[Tan90] Steven L. Tanimoto, *The Elements of Artificial Intelligence*, Computer Science Press 1990.

[Up89] C. Upson *et al.*, *The Application Visualization System: A Computational Environment for Scientific Visualization*, IEEE Computer Graphics and Applications, Volume 9, Number 4, July 1989, pp. 60-69.

[Ver93] Jouke C. Verlinden, Jay David Bolter, Charles van der Mast, *Voice Annotation: Adding Verbal Information to Virtual Environments*, Proceedings of the European Simulation Symposium 1993.