Evaluating the Flash Platform for Web-based Collaborative Data Visualization

BY

ALESSANDRO FEBRETTI
B.S.

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate College of the
University of Illinois at Chicago, 2008

Chicago, Illinois

PREFACE

This thesis will present and discuss the use of **Adobe Flash** for the implementation of collaborative visualization applications over the web. The work will be driven by the analysis of typical requirements for this kind of applications, depending on their usage context: in particular, the work will address requirements of visualization applications targeted at *public users*, rather than scientists or other professionals with specific knowledge about visualization systems. Public users dispose of little or no training on the application, need simplified access to data which is usually of limited complexity, and may prefer an application that, while being functional, also offers a pleasant *interactive experience*. The Flash technology may bring several advantages to the developement of such a system, compared to other available alternatives.

The work will first define a novel and simple conceptual visualization framework: the framework will be based on the standard dataflow visualization model, expanding it to add support for interaction and collaborative work. The framework design will be kept as simple as possible to support easy and fast developement of web applications based on it. A prototype implementation of the framework will be built using the Flash technology, and a proof-of-concept visualization application will be developed on top of the framework, to test its capabilities. The final evaluation of the technological choice and of the developed system will consider both the success of the application in respecting the specified requirements, and the developement effort needed to implement the overall system.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This chapter will give a brief introduction to information visualization. Focus will be given in particular to collaborative visualization, and to web-based visualization systems.

The flash technology will be presented as a possible foundation for web-based collaborative visualization applications. The chapter will close with an outline of the rest of this thesis.

1.1 Data Visualization

In (17) the author defines visualization in the following way:

> *"Visualization is the process of exploring, transforming, and viewing data as images (or other sensory forms) to gain understanding and insight into the data."*

Data visualization techniques are a fundamental tool for analysis and understanding of complex data. Data visualization is an important area of data analysis, where the data collected is summarized and presented in visual form to aid in decision-making and in grasping the minute details and relationships of data sets. The visual elaboration of information is a more natural way to explore data and discover its features. Also, data acquired by satellites, generated by supercomputer simulations or logged for documenting the transactions of the stock market can lead to datasets in the Terabyte range: therefore, one has to consider that datasets generated by the analysis of real world phenomena or by their simulation are becoming so big that it is infeasible to analyze the raw numerical data directly. Visualization techniques

have been applied in a variety of sectors: Data visualization is nowadays commonly applied in physics, medicine, metereology and economics, to cite a few (3; 13; 7; 8).

1.2 Introduction to visualization techniques

Information visualization is usually strictly associated to computer graphics. In its earlier development, computer graphics has been often described as "a solution looking for a problem": It sure has found a suitable problem in the area of data visualization. This may lead to the wrong assumption that data visualization has been developed after the advent of computer graphics. Actually, information visualization has a much longer history. Past visualization approaches were mainly based on human analysis of data, rather than automated elaboration. Common examples are the representation of mathematical functions for qualitative analysis, or the creation of graphic depictions of data coming from the observation of natural or historic phenomena.

The complexity of data that can be managed by these human-based methods is of course very limited. Today visualization is almost always a result of computer elaboration: this has clear practical advantages, and in some cases, due to source data complexity, represents the only feasible solution.

1.2.1 Showing high dimensional data

In (17) another important characteristic of data subject to visualization is underlined:

> *"The dimensionality of data is three dimensions or greater. Many well known methods are available for data of two dimensions or less; visualization serves best when applied to data of higher dimensions."*

Due to the frequent high-dimensionality of data, it is natural to consider visualizing it using more that two dimensions: tridimensional data visualization usually makes data exploration more intuitive, and allows to identify features of data that would otherwise go unnoticed. Tridimensional visualization can be exploited mainly in two ways:

- *visualizing a 3D representation of a real 3D feature:* the three dimensions of the visual representation directly map to the real world 3d dimensions of the feature or phenomenon being analyzed;
- *using 3D to represent heterogeneous dimensions:* one or more of the visualization dimension are used to represent a *transformed* spatial dimension, or even a non-spatial one, like time.

Animating the visualized data (that is, adding a time domain to the visualization) is another way to increase the dimensionality of the representation.

1.2.2 Interactivity

Another important feature of computer-based visualization is interactivity: the user is not just a spectator in the process of visualization, but takes an active role inside it, influencing how data is created, transformed and viewed. Interactivity also allows the user to dynamically focus on interesting features that may emerge from the visualization.

1.2.3 Collaboration

Collaboration represents a fundamental enhancement of visualization techniques: many scientific discoveries are typically made by interdisciplinary teams, and not by isolated scientists. Frequently, these teams need visualization systems that allow the different members,

who are often located at geographically different sites, to jointly investigate the results of a simulation or of an experiment, and to share their knowledge and their experience (4). While all of the participating users have to share a common perception of the data, their different fields of expertise may make it necessary to customize their view of different properties of the visualization and/or customize their interfaces.

1.3 Limitations of existing visualization systems

Even though advantages of visualization systems are clear, there are also issues that may limit their power. Some of the problems that will be underlined in this section were less evident in the past for several reasons:

- visualized datasets were smaller and less complex than the ones available today;
- people using visualization systems were also the developers of the system itself. As such, they had specific and relevant expertise on the system they were using; this situation is less frequent today, and users may have no computer science background or program development skills.

In the optimal case, visualization systems should require no other knowledge out of the domain of the information being visualized. Also, they should be flexible enough to let the user construct custom visualizations, depending on his needs. This often represents a difficult task to achieve: a tradeoff between ease of use and expressive power of a visualization system is a pretty common occurrence. A lot of general purpose systems rely on some computer science skills (or even programming ability) on the user's side. Domain-specific visualization

technologies are in general easier to use, but extending the capabilities of these applications beyond their initial domain may require a major development effort.

The presence of collaboration poses another challenge to visualization system development. Cooperative visualization is often desirable, but developing and implementing an effective collaboration metaphor can be difficult. Some of the most important features required in a collaboration system are (12; 4):

- *Multiple independent participants:* each participant should run a separate instance of a visualization system, with freedom to collaborate as much as they please within a larger collaborative environment. The design should allow an arbitrary number of participants who should be able to join and leave as the session progresses.
- *Dynamic collaboration:* the nature of the collaboration should evolve, not be predetermined. Decisions to join a collaboration, and in what way to collaborate, should be allowed as the visualization analysis proceeds. Indeed, the nature of collaboration can be driven by the visualization.
- *Shared control:* any participant should be able to control any parameter of the visualization of any collaborator. This allows scenarios where different participants are allocated control of different parameters in a pipeline, according to their individual skills and expertise.
- *Workspace Awareness:* users should perceive and understand the activity of the other persons within the shared workspace. Workspace awareness is much harder to maintain in a collaborative environment with remote participants than in a face-to-face situation, and

it is often difficult to determine who else is in the workspace, what the other participants are working on, and what they are doing. Awareness models help in understanding what are the features of communication that enhance workspace awareness (2).

From the technical point of view, collaboration can be limited by the size of involved datasets and by the speed of the network that connects users. maintaining synchronized views of the data among a set of cooperative users requires a good planning and optimization of data exchanges, or relies on strong assumptions over the collaboration model (26; 1).

1.3.1 Scientific visualization and the Web

The diffusion of local and wide area networks offers new perspectives for collaborative work. Internet connectivity is constantly increasing: more and more people use the web as a source of heterogeneous information, or as a way to get in contact with other people. Web applications, globally accessible by users through a simple browser, represent a powerful and immediate way to retrieve, add and manipulate information, as well as to interact with remote users. It has been underlined how visualization systems and techniques can have an extremely high power in communicating and understanding information. The internet is today a huge source of scenarios where the application of these techniques can really add value to the presentation of data to end users.

As underlined in (28), creating a modern web-based visualization system poses some additional challenges. Classic visualization systems, targeted to specific user communities or to expert users, had *functionality* as their main objective. Initial user training is an accepted and frequent practice in this scenario. Through the internet, it is possible to target visualiza-

tion systems to larger communities of users that have no detailed domain knowledge or lack application-specific training. In this case it would be desirable to keep the system immediately accessible even to people who do not have deep knowledge of it: *application usability* is of extreme relevance in this case. Also, over the web people are becoming more and more expecting in terms of overall experience created by applications or web sites they access and use. In this regard, offering the causal user an application that is functional in terms of services it offers, but visually poor or with unappealing interactive features, may leave users unsatisfied. The *creation of rich and graphically pleasing experiences* related to visualization represents an additional task for people who design and create visualization systems for the web. Current visualization technologies offer little or no support to the design of such systems, since they are targeted more at functional efficiency. It is therefore important to choose a technology that allows the development of web-based visualization applications with richer interfaces, while *keeping the development effort as low as possible*.

1.4 Purpose of this work

This work will describe and evaluate the implementation of a visualization system based on Adobe Flash, (33), a very common technology used for the development of rich web applications, animated graphics and other interactive and non interactive content for the web. Flash has been in use since 1997, and its evolution has led it to a state where it may be considered a possible foundation technology for collaborative data visualization. Compared with other web technologies, Flash offers several significant advantages in terms of ease of developement, interface customization and diffusion. On the other side, a major drawback of this

platform is represented by the lack of hardware 3D rendering support, which severely limits the complexity of information that can be visualized at interactive (or nearly interactive) frame rates. This thesis will attempt to understand if the limits of Flash are overcome by the advantages it offers to the developement a web-based visualization application. This evaluation will be based on the following executive methodology:

1. identify the most important requirements for a web-based visualization system. These requirements will be as general as possible, i.e. they won't be related to the domain of any specific visualization application; also, the requirements will consider the needs and preferences of both end users and developers of the system.
2. define a conceptual framework for web-based collaborative visualization: this framework should focus on the identified requirements, while remaining simple enough to favor an easy and fast developement process;
3. implement a framework prototype, based on the flash technology;
4. build a proof-of-concept visualization application on top of the framework;
5. evaluate the framework and the application with respect to the identified requirements.

1.4.1 Thesis Outline

The thesis is structured as follows:

- **Chapter 2** will give a theoretical presentation of the visualization process, and will review some of the visualization systems available today.

- **Chapter 3** will focus on requirements for a modern web-based visualization system. Several platforms will be analysed as possible foundations of such a system. The choice of flash among other platforms will be justified in more detail.
- **Chapter 4** will describe a simple conceptual framework for collaborative data visualization over the web.
- **Chapter 5** will describe the architecture of a flash implementation of the framework presented in previous chapter, called *QbViz*.
- **Chapter 6** will introduce *Hydroviz*, a proof-of-concept application developed on top of the QbViz framework. Hydroviz purpose is the visualization of salinity and currents information for the Corpus Christi Bay area, Texas.
- In **Chapter 7** an evaluation of the overall work will be reported. The implemented system will be judged against the requirements discussed in Chapter 3. The evaluation will underline the advantages given by the choice of Flash to the developed application, and to the developement process itself. The practical limits of the technology will be underlined as well, describing areas of the application that were most penalized by those limits. Some final considerations and possible future developements will then be presented.

CHAPTER 2

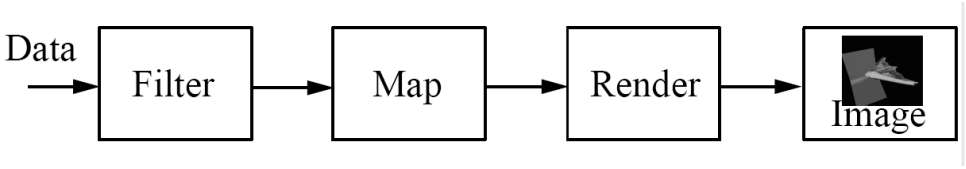
STATE OF THE ART

This chapter will provide a theoretical presentation of data visualization models. The standard dataflow model will be described, along with extensions that add interactivity, distributed computing and collaboration to the model.

A survey of several visualization technologies will then be presented. Visualization systems will be classified according to their most distinctive characteristics, like applicative domain or collaboration support.

2.1 Data Visualization Theory

In this section some details about theoretical visualization models will be discussed. Theoretical visualization models define a conceptual structure of entities and operations that are involved in the process that transforms source data into a visual representation of its content. These models can define just simple operation sequences, or they can be structured to consider more complex factors like user interaction or collaboration. Several models have been proposed to describe a generic data visualization process (22; 23; 25): One of the most common reference models is the one proposed in (21) by Haber and McNabb.



1. The Haber and McNabb dataflow model diagram, showing the sequence of steps involved in the visualization process.

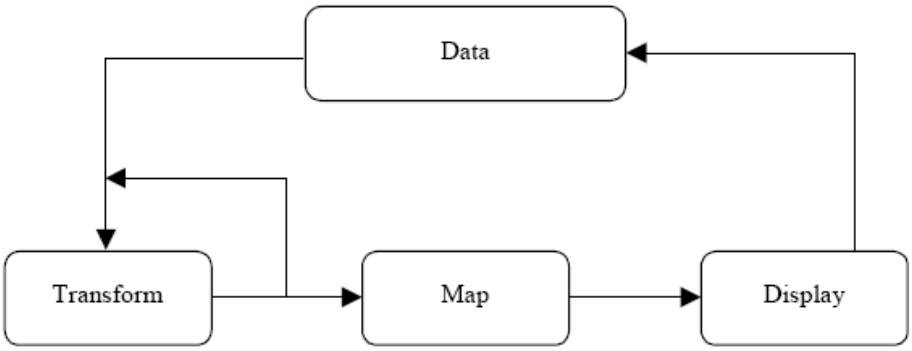
2.1.1 The Dataflow model

The Haber - McNabb model (Figure 1) is based on a three phase pipeline. Data entering the pipeline is filtered, transformed and output as a visual representation of the originating information. The steps of the Haber and McNabb dataflow are:

1. **Data Filtering:** Original dataset is filtered to isolate the region of interest. In this phase numerical data can also be transformed, elaborated or enriched to make it more significative for subsequent elaboration.

2. **Mapping:** Data coming from the filter phase is transformed into an enriched geometrical representation of its content. For instance, a scalar field dataset can be transformed into a 3D grid, in which node heights represent the magnitude of the scalar values.

3. **Rendering:** The geometrical representation of data is transformed into a 2D image. The output of this phase depends on both the input from the mapping phase and the characteristics of the viewpoint used to observe the data. The rendering phase, usually applies standard image synthesis techniques like those described in (50).

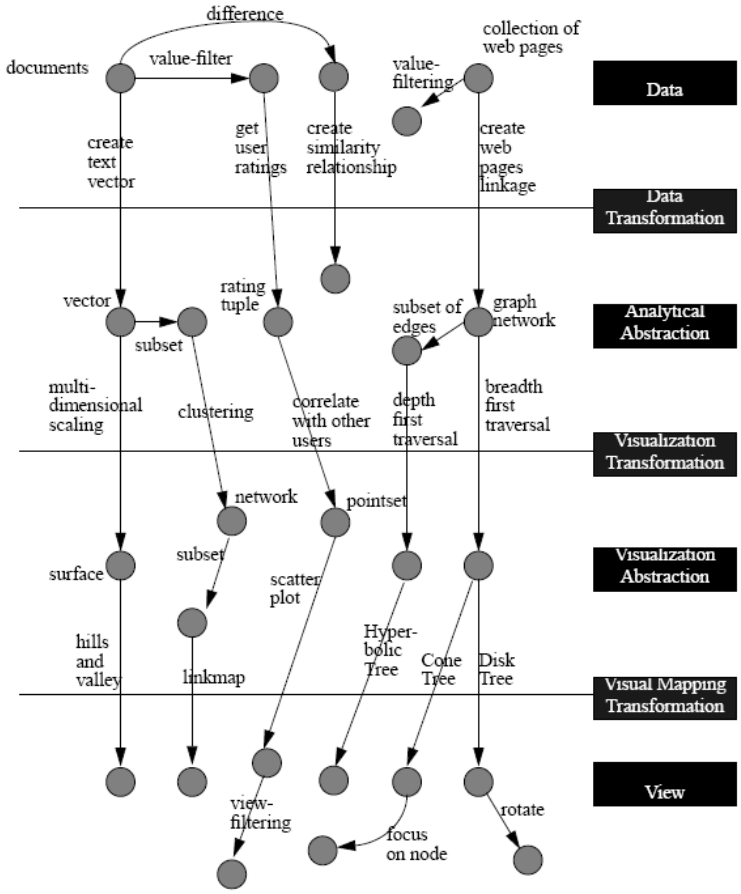


2. The interactive dataflow model.

2.1.2 Interactive visualization

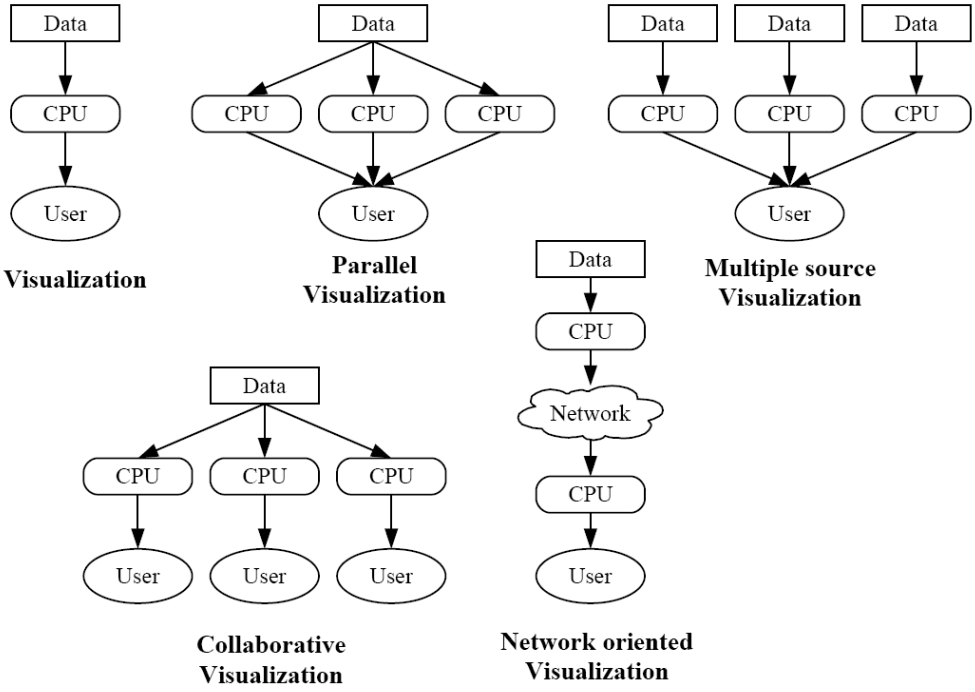
The three phase model is simple and well suited to describe the kernel of the visualization process. Yet, it does not take into account the interaction of the user with the visualization. Also, collaboration of multiple users over the same visualization pipeline is not considered. A basic extension of the model that keeps interaction into account is presented in figure 2: the user analyzes the image generated by the visualization pipeline, and then modifies some parameters of the pipeline itself to generate a new significative output. The repetition of this task (image generation, user feedback on the pipeline, new visual output) is the basis of interactivity in visualization systems.

The presented interaction model, based on simple feedback, is very general but does not describe the fine details of how interaction can work. More refined models can be defined: (25) for instance, defines an *operator framework* that classifies all possible interactive operations supported by a visualization system. These operations can be applied during various stages of the



3. An example of operator flow. Operators are associated to the visualization stage in which they apply, and are connected to operators that may be executed afterwards.

visualization pipeline: the execution of an operation in a specific stage in turn allows the execution of subsequent operations in the same stage or in following ones. This model is therefore able to represent also the *flow* of operations that lead from data to visualization. Figure 3 (25) shows an example of operation flow.



4. Several possible configurations of a distributed visualization pipeline.

2.1.3 Distributed visualization

The increasing trend in size of datasets subject to visualization has already been discussed in chapter 1. It is clear that elaboration and visualization operations for such datasets are resource-consuming tasks. In many situations the data is too big to be processed and displayed by a single computer. That is why, to perform fast interactive rendering of large datasets, distributed and parallel processing are a common practices (26). The visualization pipeline presented in section 2.1.1 can be easily split into separate execution units that can be organized in different configurations (figure 4).

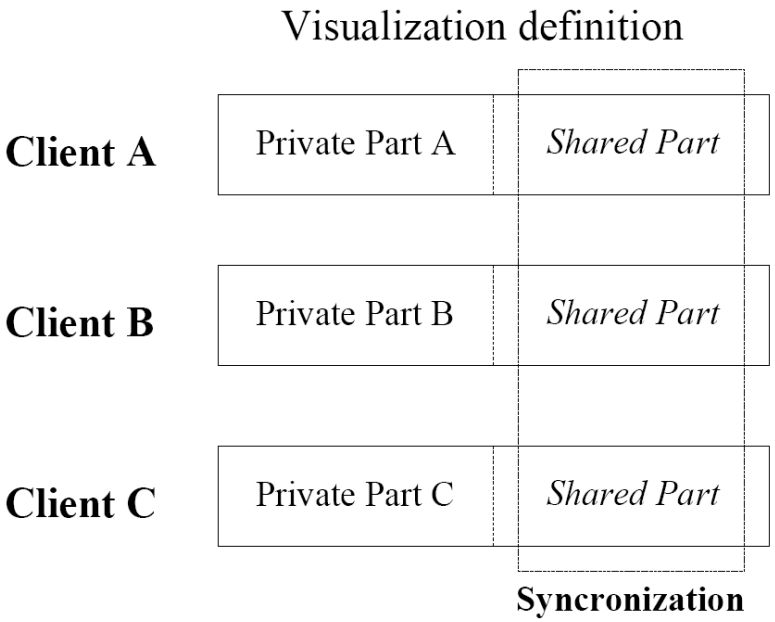
Another situation in which the visualization pipeline gets distributed among different units arises in the case of online-visualization. In the case of online visualization, the user endpoint of the pipeline may be in a remote location with respect to the data store, the simulation services or the data analysis facilities. In the extreme situation, all phases of the visualization pipeline may be implemented by remote online services that can be linked dynamically.

Distributed configurations may lead to advantages in the overall system flexibility, but also carry important concerns when one takes interactivity into account. The presence of interaction requires the constant exchange of data between the user and the pipeline services. Limited bandwidth and high latencies in the network links between these entities could lead to increased system response time, which in turn may adversely affect interactivity.

2.1.4 Collaborative distributed visualization models

The important role of collaborative features in visualization systems has already been discussed in section 1.2.3. In its most general definition, a collaborative visualization system allows multiple users to view and manipulate some form of common representation of the data. Each user view of the data can be divided into two separate components (figure 5):

- the **private component** is local to each user. Modifications to the private part of a visualization do not get broadcast to other users. Each user has his own private component configuration;
- the **public component** is the part of the view that gets shared among all the users. Data and visualization entities inside the public part of the view are the ones that are actually involved in collaborative interaction.



5. Public and private view components in a collaborative distributed visualization system.

As an example of private and public view components, a 3d methereological visualization system can place in the shared part all the *content* of the visualization (i.e. the region of interest, the cloud layers, streamlines etc.), and leave the *viewpoint* (that is, the position from which data is observed) private to each user. In this way, users cooperate in defining the content of the visualization, but each one can look at the data from his own position.

Collaborative visualization systems are, in a sense, an extension of distributed visualization systems. All the considerations and limitations that were made for distributed systems in section 2.1.3 still apply in this scenario.

Collaborative systems should be subject to additional considerations: in this case there are multiple user endpoints, and so there are multiple potential interaction sources within the

system. This characteristic has the immediate effect of increasing the need of remote data exchanges between the entities of the system (whether they are user endpoints or data processing services). As in the case of distributed single-user visualization, collaborative visualization models have to take into account the limits that those data exchanges impose on the consistency and synchronization of all the user visualizations.

2.2 An overview of Visualization Systems

This section will present a review of some of the existing visualization tools and technologies available today. Providing an exhaustive classification of all existing visualization systems is a practically unfeasible task. Still, it is significant to identify some classification criteria that are particularly useful to differentiate systems. These criteria should also help in choosing several systems that can then be presented in order to give a good overview of the visualization system universe.

2.2.1 Classifying Visualization Systems

Several different classification methodologies can be applied to visualization systems. In (20) the author divides visualization systems in three categories (special purpose applications, modular environments, and libraries with toolkits), based mainly on the level of abstraction those technologies have with respect to a specific data visualization domain. Those considerations will be applied also in this work, but classification will refine it and take into account a major number of factors.

The categorizations used to distinguish among visualization systems will be the following ones.

- **Toolkit vs. application**: a toolkit is a lower level technology than an application. Applications are ready to use out-of-the-box, while toolkits come in the form of programming interfaces that can be used to *create* applications. Using a toolkit requires a lot more expertise than using an application, even if toolkits allow the user to create exactly the kind of data manipulation and visualization system they need.
- For systems classified as applications, there is a further distinction between **modular environments** and **domain-specific applications**: domain specific applications are dedicated to a particular type of visualization problem. This kind of application has usually been built in order to satisfy the classical needs of a given user community. Modular environments provide a number of simple data processing, manipulation and visualization modules, that can be assembled by the user to create the required visualization system. When needed, the capabilities of the system can be expanded by programming new modules. Modular visualization environments are simpler to use that a toolkit, even if the creation of visualization networks requires some expertise, and have a similar expressive power.
- The presence of **Collaborative visualization support**.
- A **standalone or web-based interface**: web-based applications are accessible using a standard browser and some additional technology, like java, flash, VRML and so on (as described in chapter 3). The advantage of web-based applications versus stand alone ones is that their diffusion and access is much simpler. Generally, the users won't have to download and install anything on their machine: they will just connect to a specific

System Name	Class	Domain specific	Collaborative	Web based
VTK	Toolkit	-	-	-
IRIS Explorer	Application	-	-	-
Mathlab	Mixed	-	-	-
CSpray	Application	?	YES	-
COVISA	Application	-	YES	-
WBCIVT	Application	?	YES	YES
JMOL	Application	YES	-	YES
Web-IRIS	Application	-	-	YES

I

CATEGORIZATION OF PRESENTED SYSTEMS.

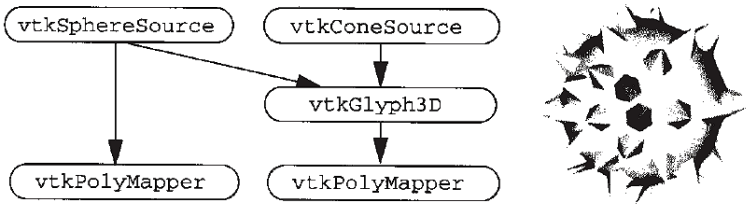
web address. This is especially important when developing visualization applications targeted to *occasional* users, that may feel undesirable to install a new application on their machine, rather than simply navigating a web site.

2.2.2 Summary of presented systems

All of the characteristics discussed above are independent from the others: each one therefore represents a distinct categorization dimension for the presented systems. Table I shows a summary of all the visualization systems that will be described in following sections. In the table, features marked with "?" are difficult to classify, or may be only partially present. The corresponding system's section will give detailed information about these features.

2.2.3 VTK

The Visualization ToolKit (VTK), (19)(18), is an open source software system for 3D computer graphics, image processing, and visualization. VTK consists of a C++ class library, and several interpreted interface layers including Tcl/Tk, Java, and Python. The model supported

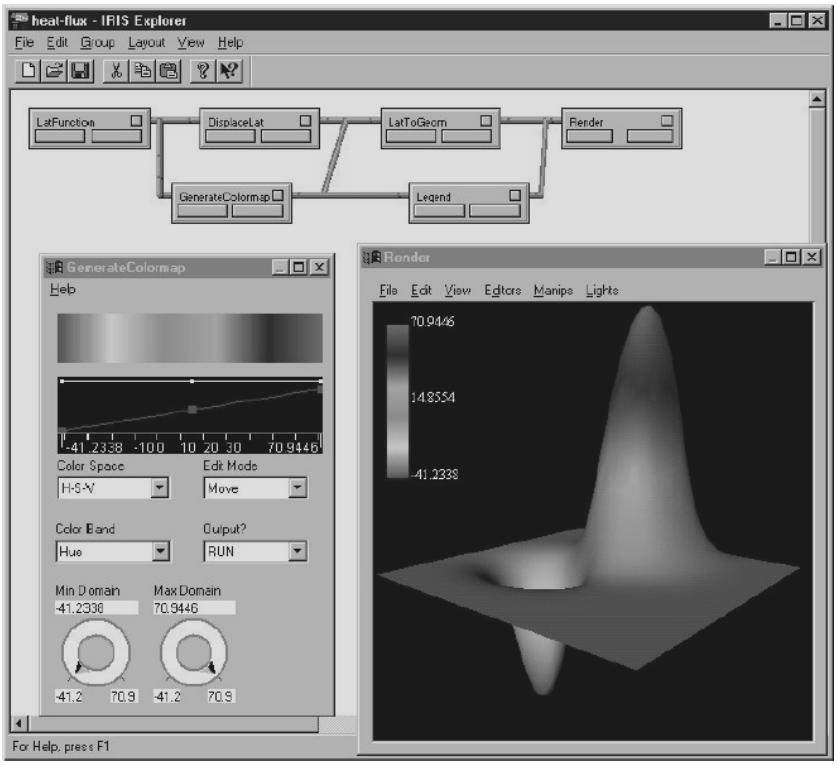



```
sphere = new vtkSphereSource(8);
sphereMapper = new vtkPolyMapper;
  sphereMapper->SetInput(sphere->GetOutput());
sphereActor = new vtkActor;
  sphereActor->SetMapper(sphereMapper);
cone = new vtkConeSource(6);
glyph = new vtkGlyph3D;
  glyph->SetInput(sphere->GetOutput());
  glyph->SetSource(cone->GetOutput());
  glyph->UseNormal();
  glyph->ScaleByVector();
  glyph->SetScaleFactor(0.25);
spikeMapper = new vtkPolyMapper;
  spikeMapper->SetInput(glyph->GetOutput());
spikeActor = new vtkActor;
  spikeActor->SetMapper(spikeMapper);
```

6. VTK visualization network diagram, code and resulting visualization.

by this library is the *visualization network* (see Figure 6, (20)). The pipeline consists of objects to represent data (data objects) and objects to operate on data (process objects). Programmatically linking instances of these objects together leads to the desired visualization. VTK can be used to build powerful standalone visualization applications, but it cannot directly be used to develop web-based applications. It would be possible to develop a browser plugin that exposes VTK functionality, but this would require significant developement effort.

VTK also offers no native support for collaborative features: even though network functionality can be implemented inside a VTK application using third party solutions, the integration between VTK and the communication layer is entirely up to the user.



7. Screenshot of an IRIS explorer session, showing a visualization network and the generated result inside the render window.

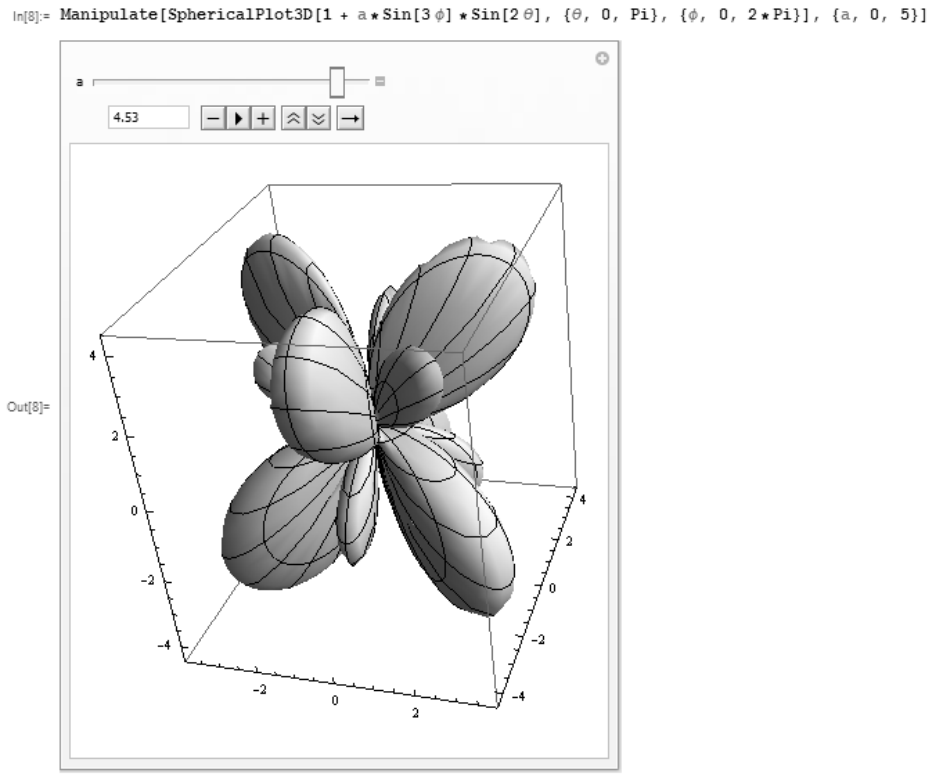
2.2.4 IRIS Explorer

IRIS Explorer, (31), can be seen as the classic example of a Modular Visualization Environment (MVE). In a modular visualization environment, users interactively create their visualization in the form of a network of modules. Editing of the network is performed via a point-and-click interface, where modules are selected from a library, dragged onto the map editor and connected together. The behavior of each module is usually controlled by some set of parameters, and the user can interact with these while the application is running via a standard set of widgets. Other systems similar to IRIS are AVS/Express (29) and Data Explorer

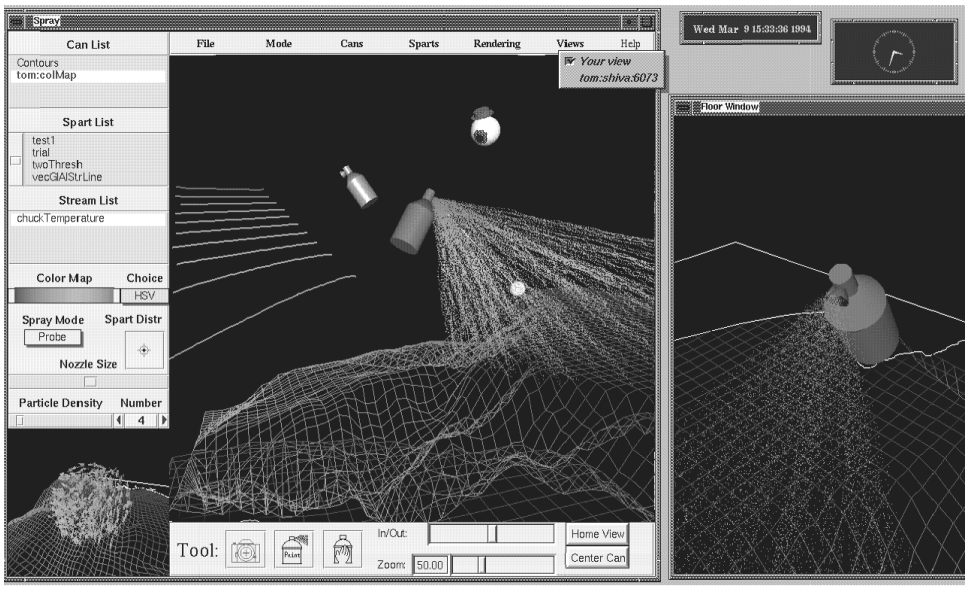
(30). IRIS can support collaborative work through the use of special modules designed for this purpose: the work described in section 2.2.7 is an example of this approach. Since collaboration is not integrated inside IRIS at a lower level, this kind of solution may result less intuitive than other ones, expecially during collaborative session setup. IRIS is a standalone application and cannot be used to build visualizations over the web. Some work has been carried on to interface IRIS to a web-based application interface (section 2.2.8, but actual implementations severely limit the amount of accessible IRIS functionalities.

2.2.5 Mathematica

Mathematica, (32), is a computation and visualization environment, and used mainly in scientific and mathematical fields. Tools like Mathematica, or Matlab (49) are half-way between MVEs and visualization toolkits: they usually require some programming effort using their own special-purpose language, but they also simplify immediate visualization and manipulation of graphic results. As figure 8 shows, simple language constructs allow the creation of an interactive visual representation of the generated result. Mathematica also offers a web-accessible version of its services, even if it is limited to server-side generation of visualizations that are then sent to the browser as images: real time data manipulation is not possible. Also, this kind of applications does not offer any kind of collaboration support, and the integation of collaborative features inside them may result particularly difficult: the reason is that these application, despite being able to perform pretty complex visualization tasks, do not implement any specific visualization metaphor: adding collaboration to such a generic environment is far



8. A sample Mathematica visualization, generated through a single line of code.



9. CSpray visualization workspace. User controls viewpoint, can position, orientation and particle spraying.

from being easy, and the implemented solution may result less intuitive that the ones designed for systems where the visualization methaphor is clear and structured.

2.2.6 CSpray

CSpray (6) has been developed at University of California, Santa Cruz. *CSpray* stands for Collaborative Spray rendering and is an extension of *Spray*, (15)(16), a visualization environment based on the spray rendering metaphor: in the Spray environment, the data are represented using the metaphor of a spray acting on a portion of space. It sprays sparts (smart particles) that react with the data they encounter according to a behavior described by a program written in a simple language that the sparts interpret. The flexibility is thus provided by the possibility of implementing various functions in the sparts (figure 9). What is particularly

interesting in this attempt is the fact that the techniques of surface, volume and how visualization are generalized and that spray rendering allows selective progressive refinement, i.e. the ability for a user to concentrate on the phenomena occurring in a precise region of interest.

Collaboration in the Spray environment is quite intuitive: every user has his own point of view to the data, and can manipulate a spray can: additionally, users can see data from the viewpoints of other participants, or take control of their spray cans when allowed to do so. Workspace awareness is maintained through the use of *Eyecons* (eye icons), that represent the position and looking direction of other users inside the workspace.

One of the main limitations of CSparts is represented by the spray rendering concept itself. Some types of visualization are difficult to manage through this kind of metaphor: the visualized data should be significantly representable by "clouds" of simple graphic primitives represented by the smart particles. Several visualization techniques, like texture-based visualization, are not practical in a spray rendering environment. Another limitation of the actual CSpray implementation is the lack of support for sparse source datasets: the data managed by the spray environment must be sampled on a regular grid. In many scenarios (like the one presented in Chapter 6) this would require a resampling of the source dataset.

As for now, no web-based implementation of spray rendering has been developed.

2.2.7 COVISA

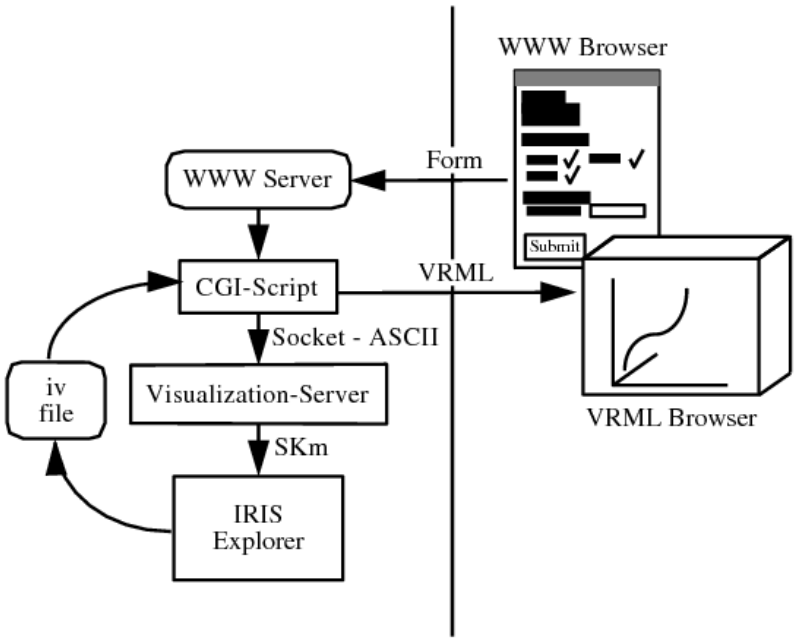
The Cooperative working in Visualization and Scientific Analysis (COVISA) project, (4) (5) (14) is developed at the University of Leeds. Its aim is to expand the capability of single user modular visualization environments, adding collaborative support. The current implementa-

tion targets the IRIS explorer platform, and is based on a set of custom modules: "these new modules, when wired up in the same manner as the standard modules, can pass either data or parameters out of or into a pipeline. These extra modules then form a collaborative toolkit for the visualizer to build shared pipelines in any of the forms representable by the model. This approach is feasible in all of the MVEs though the implementation details will be different.". The COVISA implementation for the IRIS framework is considered complete: COVISA modules are now part of the standard module library distributed with IRIS explorer.

Altough COVISA works in integrating collaboration inside a modular visualization environment, it requires a significant amount of work from the users, expecially during collaborative session setup: the users have to build up two symmetric visualization networks on their respective environments, and add the special-purpose collaboration modules at the right places. COVISA offers no support for these operations. Also, COVISA does not manage collaborative changes in the network topology.

2.2.8 Web-IRIS

The web-IRIS prototype system aims at allowing access to the visualization services offered by IRIS explorer through a normal browser. The system, whose structure is illustrated in figure 10 is similar, in functionality and limitations, to the approach used by the web-accessible version of Mathematica presented at the end of section 2.2.5. Through the web interface users can just manipulate some parameters of the visualization network, and receive an updated version of the corresponding visualization. Changes in the actual network structure are not supported. An interesting variation of this system, compared to the one used for Mathematica,

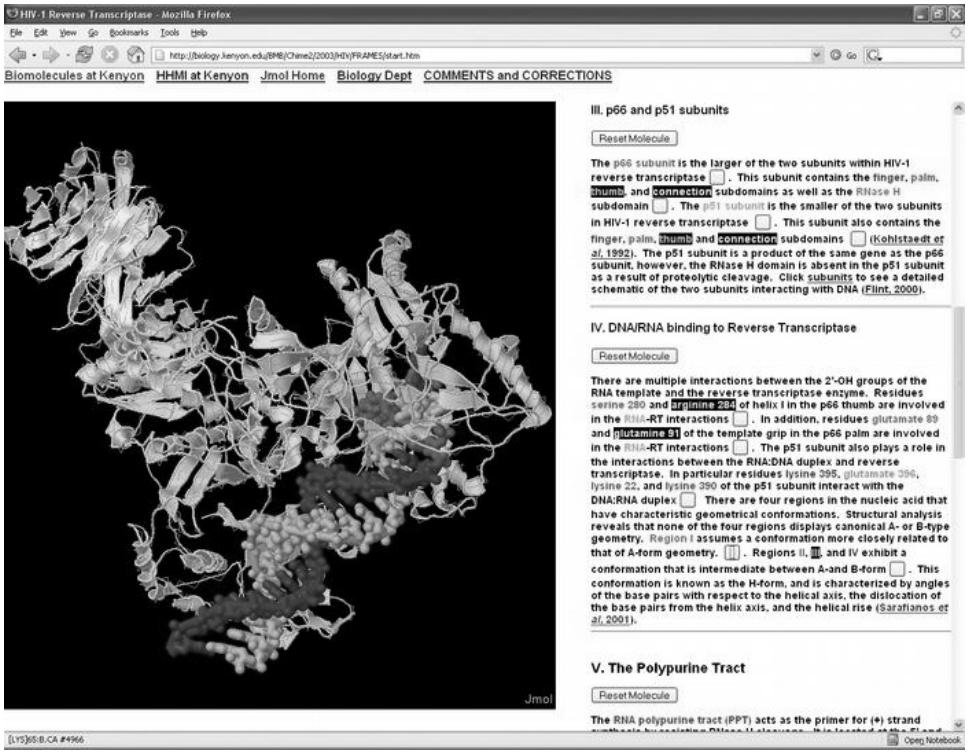


10. Making IRIS functionalities accessible from a web browser.

is that it generates visualization results as VRML files, instead of still images: this allows end users to "explore" the 3D visualization inside their browsers. Details about the VRML technology are discussed in section 3.2.2. The developement of this prototype system does seems to be no longer active.

2.2.9 Web-based collaborative visualization tool.

This experimental tool (24) has been developed from departments of Informatics and Electrical engineering of Universidade Federal do Pará. It allows the collaborative visualization of 3D data inside a normal browser. Two main client side technologies are used: Java and VRML. The java applet handles both user interaction and the generation of new VRML visualizations,



11. A web page integrating a JMOL interactive view.

depending both on local and remote manipulation. The application interface is split up into two separate views, one local only and the other shared. The two views are independent one from the other, meaning there is no way to easily share data among them, i.e. building up a view which is partially private and partially shared and manipulated by all the users. There is also limited information about the kind of data supported by this application, and about how data is exchanged and accessed during collaborative sessions. Excluding a simple graph visualization application, no significative test case has been presented for this tool yet.

2.2.10 Jmol

Jmol is a molecule viewer for use in chemistry and biochemistry. Jmol has been developed in java (see section 3.2.1) and is available as both a standalone application and an applet that can be integrated into web pages, as figure 11 shows. Jmol is able to display molecular structures in a variety of ways. For example, molecules can be displayed as "ball and stick" models, "space filling models", etc. Jmol does not support any collaborative feature out of the box, even though it has been shown how it is possible to integrate it inside the standalone application version (1). Jmol is a good example of a "turnkey" application: it targets a specific and limited applicative domain, but it offers extremely effective functionality for that domain. It is really difficult for any generic visualization system to challenge this kind of special purpose applications. The disadvantage of special purpose systems from a developer point of view, is that they require significantly more work than any visualization application built on top of a generic system: in some cases the benefit of fast developement times may overcome the possible potential of a special purpose application developed from scratch. Generic systems my be the best choice when needing to build a working prototype of the application in limited times.

2.3 Summary

This chapter introduced a critical review of several available visualization systems: these systems, even if representing a small fraction of visualization applications developed so far, have been chosen to give a significative overall picture of the field. This choice was driven by the classification presented in section 2.2.1, and it was made to cover up as many significant feature configurations as possible.

From the presented reviews one significant thing emerged: some standalone systems are trying to offer a web-based interface to their functionality. This is an additional evidence of the fact that deploying visualization applications over the web is considered more and more important. It has also be noted how the process of exposing functionality of existing application was not completely effective: often web interfaces to standalone applications offered a much limited subset of the original system's functionalities. Due to implementation constraints, these interfaces can also be less intutive, or offer limited interaction when compared to their standalone counterparts.

On the other side, it was difficult to find completely web-based applications which offered the same visualization and collaboration features available in standalone versions. This is partly caused by to the fact that developing a web-based visualization application is usually more difficult, due to the additional constraints associated with the developement of such systems (as explained in section 1.3.1).

The choice of the correct technology for developing visualization applications over the web is therefore of extreme importance. This choice should be driven by the requirements that the system needs to satisfy, which in turn depend on the applicative scenario. In chapter 1 it has been underlined how this work specifically addresses the needs of common users, who potentially don't have any prior knowledge of the application, need simplified access to data which is usually of limited complexity, and may prefer an application that also offers a pleasant interactive experience.

Next chapter will describe the identified applicative requirements in more detail, and it will derive a set of technological constraints to guide the choice of a foundation technology for web-based visualization system developement.

CHAPTER 3

TECHNOLOGIES FOR WEB-BASED VISUALIZATION

This chapter presents a critical review of several technologies that can be used to develop a collaborative visualization application over the web. The technologies will be evaluated through a set of requirements that emerge from the desired features of the application. These requirements will justify the choice of Flash as a *possible* foundation technology for web-based visualization systems.

3.1 General requirements for a data visualization web-application

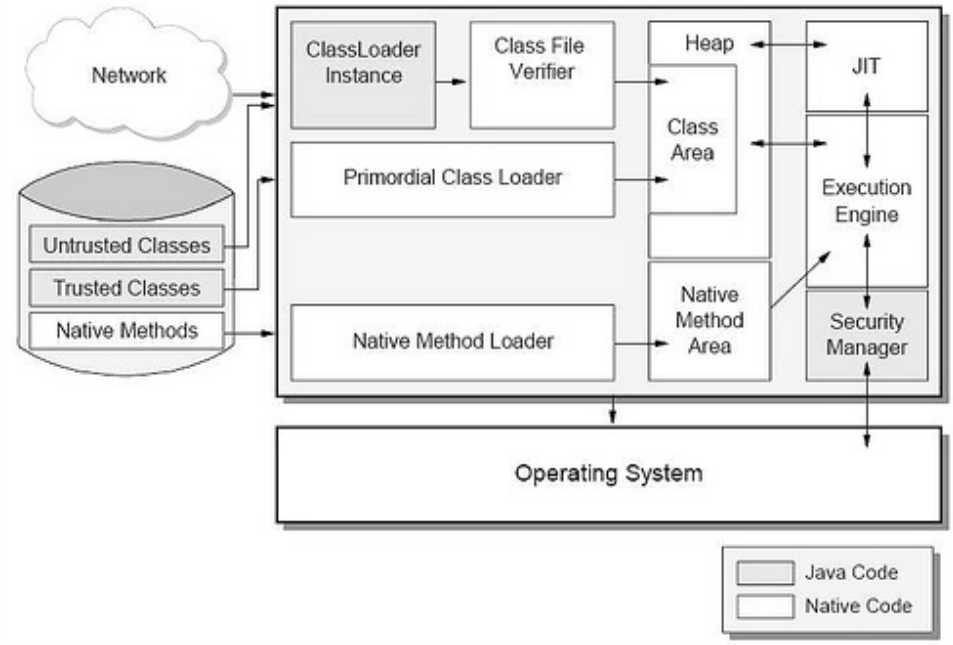
An partial overview of requirements for a web-based visualization application targeted at common users have already been presented in section 1.3.1. Here the most important of these requirements are listed and explained more formally:

- **Simple access:** the application should avoid the need for the user to install custom or third party browser extensions. Widespread and trusted technologies should be used whenever possible.
- **Ease of use:** the application should offer an easy to use interface, one that an untrained user can learn by himself with little or no assistance.
- **Collaboration:** the application should offer some form of collaborative visualization support, either synchronous or asynchronous.

- **3D visualization:** Rendering and manipulation of mildly complex 3D visualizations should be supported.
- **Enjoyment:** the application should be functional, but also offer a rich *interaction experience* to users.

These requirements pose some constraints in the choice of technologies that one can choose as a foundation for a web-based visualization system:

- **Diffusion:** the chosen technology should have a huge installed base. For users not already reached by the technology, the download and installation of the required browser plugins should be a fast and easy operation. In the case of technologies which get regularly updated, plugin updates should guarantee backwards compatibility, to avoid versioning issues between the technology available on the end user machine, and the one that was used to develop the application. Also, the update system should be simple and automated, to simplify the access to the right version of the technology for users that require it.
- **Easy development:** implementing a visualization system using the chosen technology should be as straightforward as possible. The used programming language should be high-level enough to allow fast development times (in the order of some man-months).
- **Support for interface design:** the technology should offer some way to easily implement a graphically rich user interface. The interface should also have an easily customizable *look and feel*.



12. Architecture of the Java Virtual Machine execution environment.

- **Advanced communication layer:** several communication and data access interfaces should be made available by the technology (i.e. web-service access, custom HTTP messaging, socket support, remote procedure call, ...)
- **3D graphics support:** The technology should offer some 3D rendering capability, either directly or through additional APIs.

The next section will present several technologies that may be considered as development platforms for a web-based visualization system.

3.2 Web-based technologies for visualization

3.2.1 Java

Java is a widespread high-level language, used for both standalone and online application development. Java programs are compiled into a machine independent binary format and then executed by a Java virtual machine (48) (figure 12). Executing code on a virtual machine is much more secure, since code is run in an environment where strict security checks can be applied. This is a fundamental requirement when it comes to executing code that has been downloaded from remote and potentially unsecure sources. Java web applications come in the form of applets that can be downloaded and run inside the client browser.

Exploiting java as a base technology for a web-based data visualization application is possible through the use of third party libraries that offer general purpose 3d rendering support. One such library is JOGL, (41). JOGL exposes OpenGL (42) functionality inside a Java applet. It is possible to develop a Java / JOGL data visualization applet that will be accessible through a normal Java-enabled browser. Requiring the users to have a java virtual machine installation to be able to run java applets is not a particularly limiting constraint, since the JVM installed base is pretty large (see figure 13). The JOGL library is not part of the standard JVM installation, so most users will have to install it. The download and installation is automatic when one starts a JOGL-enabled applet, even if some minimal interaction with the user is needed (the user has to accept a certificate prior to installation). The Java / JOGL development technology is a pretty powerful choice, but has some disadvantages too: JOGL is basically a wrapper that makes OpenGL functionality accessible from a Java applet. In this sense, it does not offer any

higher level functionality than OpenGL itself. It has already been underlined how developing a complete visualization system using such low level technologies, although possible, may require a considerably longer development time and effort compared to other choices. Also, Java applications can integrate users interface features through the SWIG (54) and AWT (55) technologies but they do not easily support complex graphic animations and effects on visual components, like other technologies can do (see sections 3.2.5 and 3.2.3).

3.2.2 VRML

The Virtual Reality Modeling Language (VRML) is a standard file format for representing 3-dimensional interactive vector graphics, designed particularly with the World Wide Web in mind. After installing a VRML plugin, users can download and explore 3D environments inside their browser. The environment is described through a simple description language that defines the virtual world geometry.

VRML environments also support user interaction: URLs can be associated with graphical components so that a web browser might fetch a web-page or a new VRML file from the Internet when the user clicks on the specific graphical component. Animations, sounds, lighting, and other aspects of the virtual world can interact with the user or may be triggered by external events such as timers. It is also possible to add program code (e.g., written in Java or JavaScript, (43), a simple script-like java dialect) to a VRML file.

An XML-based markup language with the same applicative purpose as VRML is X3D (47). X3D is considered the natural evolution of VRML, and it features several extensions to the latter technology, like humanoid animation and NURBS.

As already underlined, browsers need a VRML (or X3D) plugin installed, to be able to show VRML content. The diffusion of VRML viewers is pretty low compared to other technologies presented in this chapter. The need to install a custom plugin is alleviated by the fact that viewers are usually pretty small (about 2MBs), allowing fast download and installation times.

A major drawback of VRML is that it is just a 3D world definition language: it cannot be used to support client computation or to manage user interfaces directly, requiring instead the integration with other technologies.

3.2.3 Shockwave

The shockwave technology (35) was initially developed to create interactive content for presentations, simple games and multimedia CD-ROMS. Shockwave evolved over the years, adding support for hardware - accelerated 3D and for online content delivery to browsers equipped with a shockwave plugin. Shockwave applications can be developed inside a proprietary authoring environment. Application logic can be written in lingo (46) or Javascript.

Developing complex, data-intensive and collaborative applications using shockwave can be a pretty complex task, due to the lack of advanced data access and communication libraries. The authoring environment is also pretty spartan and less reliable compared to other alternatives.

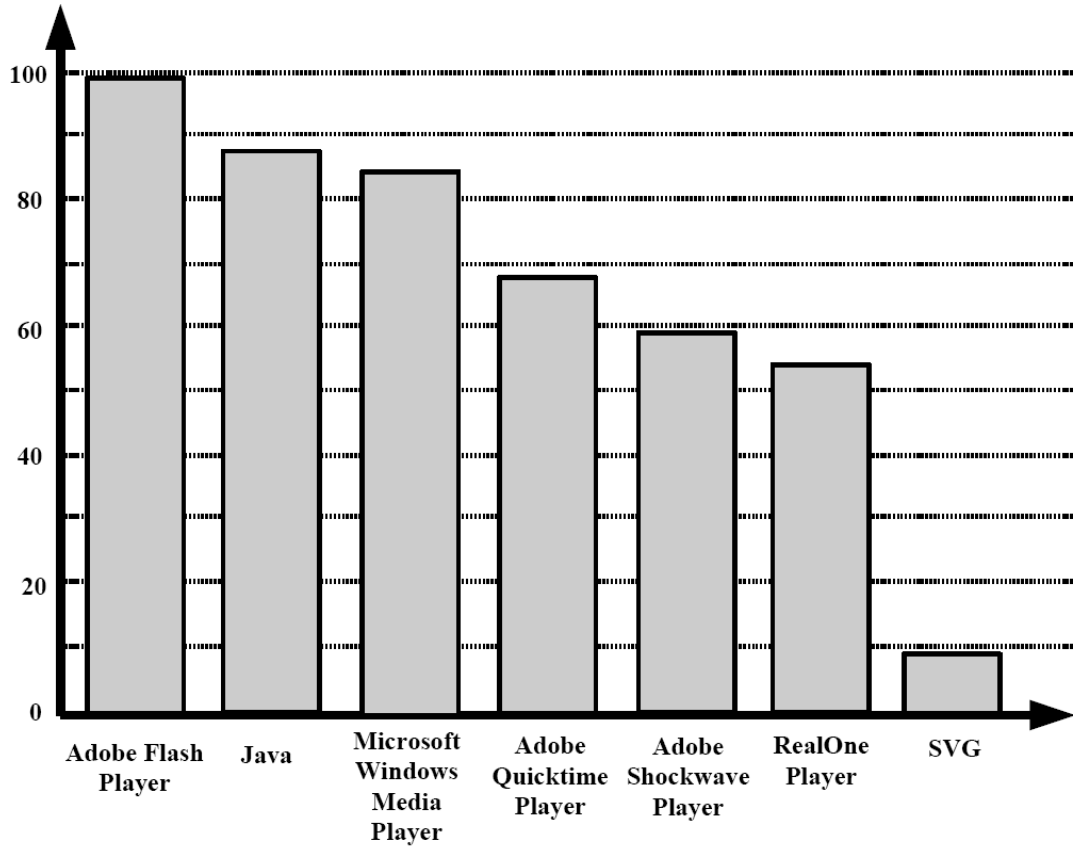
Future support for this technology is not guaranteed, since much of the capabilities of Shockwave can now be replicated using Flash (section 3.2.5), which is currently developed and distributed by the same company.

3.2.4 Proprietary technologies

Proprietary technologies like VirTools (45) and Hypercosm (44) may allow for simpler development of web-based data visualization applications, since they are oriented to that exact purpose while other generic technologies are not. Development with this technologies usually requires the purchase of proprietary authoring tools. The quality of these tools should be thoroughly analyzed to see if development requirements are respected. Also, the use of proprietary technologies carries a significant disadvantage: the installed base for custom proprietary plugins is extremely low. Developing a web application supported by such a technology will require the user to install a custom third party plugin in almost every case: while a casual user may decide to download and install a well-known technology (like Java or shockwave) he may be dubious on downloading an unknown (and potentially malicious) plugin on his machine.

3.2.5 Flash

The Flash technology, (33), has been developed with the explicit target of delivering interactive graphic content on flash-enabled browsers. Flash visual content (which is mainly composed of vector graphics) is displayed on end user machines using the vector renderer embedded in the flash plugin. Application logic, compiled into bytecode, is run on a simple virtual machine. The code for a flash application is written in a simple scripting language, called ActionScript (34). Actionscript underwent a notable evolution over different versions of flash. In its current version, Actionscript can be used as a full-fledged object oriented lan-



13. Penetration of most diffuse browser plugins

guage (like java), but still retains some scripting-language features that allow faster application prototyping and development.

Starting from version 9, Flash integrates the Flex framework, (36), that greatly enhances the development of rich and interactive visual interfaces. Interface components and layout can be defined using MXML, an XML-based user interface modeling language. MXML markup can embed actionscript code or access code from external source files, to achieve a good separation between application view and logic.

Even though flash applications can be created inside a proprietary authoring environment, it is possible to develop a complete application using a standard text editor to write Actionscript and MXML code, and compile it to its executable form (a .swf file) using the free compiler offered by Adobe.

Developing data-driven applications in flash is straightforward. Access to data sources and services can be achieved in a variety of ways: flash remoting, web service access, http requests or raw socket connections are some of the available options.

As figure 13 shows, flash installed base is huge. In most cases, content developed in flash will be seamlessly displayed and run on the client browser without requiring any user interaction.

3.2.5.1 Vector rendering

The main disadvantage of current Flash implementations is their lack of hardware - accelerated 3D support. All of the graphic content of a flash application is rendered using a software rasterizer. The software rasterizer is extremely optimized, and can event take advantage of multi-core processing capabilities of the machine when available. Still, its performance is easily overcome even by medium level 3d hardware renderers.

The choice of a software vector renderer was originally made to guarantee that delivered content would look the same on the widest possible range of machines and platforms (even those with no 3d acceleration available). Taking into account all the possible levels of support of accelerated 3d graphics would have been far more complex than implementing a generic

software rasterizer. Also, in its original incarnation flash was not thought to deliver highly complex graphics.

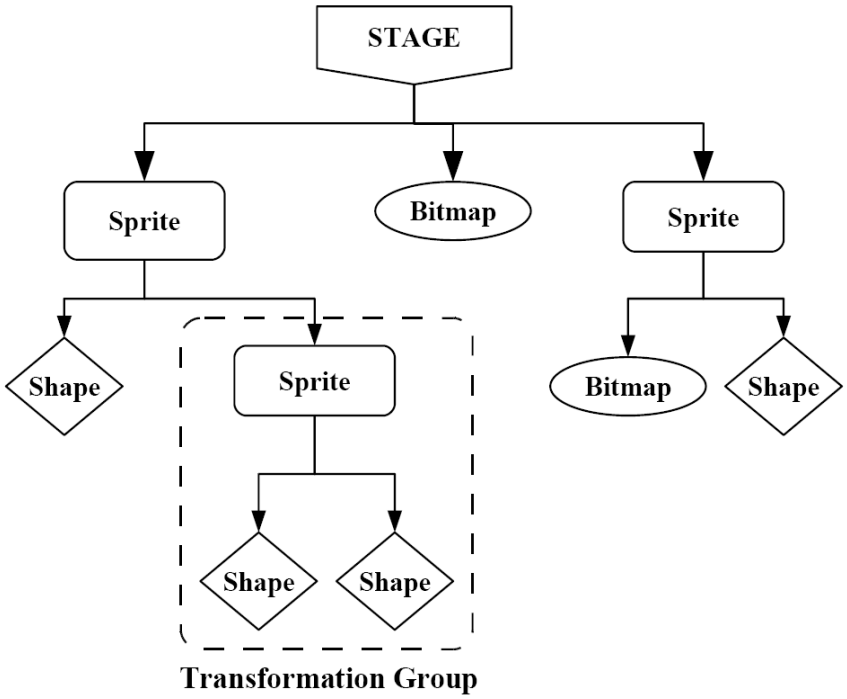
The situation is now beginning to change. Flash applications are becoming more demanding in graphic capabilities: this already led to the introduction of hardware level support for certain graphic operations like full-screen scaling.

3.2.6 SilverLight

A technology similar in purpose and mechanics to flash is SilverLight, (38),developed by Microsoft. Silverlight can be programmed using the C# language, an object-oriented language similar to Java. C# can be used also to develop the server side components of a web application, simplifying communication among the client and server, and allowing to recycle parts of the code. Actionscript does not allow so, since it is designed to be just a client-side development language. This is a significant development advantage over flash, and may play an important role when one has to choose between Flash and Silverlight as his target platform. As for now though, SilverLight is a young and developing technology with respect to Flash. SilverLight is still in its beta stage, and its installed base is extremely low. At present, these considerations make flash a better choice over SilverLight if one were to choose among them, even if SilverLight development, diffusion and acceptance might lead to a different consideration in the future.

3.3 Choosing a Technology

Considering requirements and constraints recalled in section 3.1, the choice of flash to develop a visualization application over the web appears reasonable. It has been underlined



14. The flash display tree. The tree is traversed depth-first during rendering.

how the main limitation of the flash technology right now is its lack of hardware-accelerated 3d support. This is indeed a major disadvantage in the choice of flash: yet, application interactivity and real-time responsiveness can be preserved through the use simplified rendering techniques when manipulating 3D content.

3.4 Visualizing 3D data over flash

When presenting the technology, it has been underlined how Flash implements a vector renderer that basically supports the drawing of 2d shapes (lines, curves, circles, rectangles and so on) and raster images. Multiple shapes and images can be grouped together into more complex visual objects, called *sprites*. Sprites can also contain other sprites: this ends up form-

15. Diagram of the papervision rendering pipeline

ing a *visualization hierarchy*, with the *stage* (that is, the total available visual space) as its root (figure 14. The hierarchy allows to consistently apply 2d transformations to complex objects. The hierarchy also determines the order in which objects are drawn (during the draw phase, the hierarchy is traversed depth-first).

Flash has no concept of 3D space. 3D primitives and transformations have to be mapped to 2D prior to drawing. Some actionscript libraries have been written to perform this task. Currently, the most advanced implementation of a 3D manager for flash is offered by the Papervision library (52). Figure 15 illustrates the main function of papervision: converting 3D primitives to a hierarchy of 2D sprites that can be managed by the flash rendered and represents the original 3D scene. This conversion involves three main operations:

- Converting 3D coordinates of objects into 2D screen coordinates. This is done using standard world, view and projection matrix transformations.
- Sorting object depending on their distance from the user viewpoint, so that near objects are drawn after far-away objects, occluding them properly (painter's algorithm, (56)). This is far from being an optimal solution. Modern hardware renderers use more efficient

techniques that avoid the need to sort objects, like Z-Buffering, (57). Unfortunately this cannot be done in flash, and sorting is the only way to achieve correct object occlusion.

- Drawing sorted objects using 2D primitives. This operation is possible at this point, since object coordinates have been converted to 2D screen space in the first step.

Papervision also supports other features, like lighting calculation and postprocessing effects. A detailed description of this techniques goes beyond the purpose of this section.

3.5 Summary

This chapter discussed several criteria used to understand whether a web developement technology can be considered suitable for implementing a web-based collaborative visualization application. The choice of flash among other presented technologies was partially justified by these criteria, even though limitations associated with this choice have been underlined as well. The major drawback of Flash is software rasterization, which represents a severe constraint on the complexity of visualized data. The idea supported in this work is that, for applications requiring 3d visualizations with limited complexity, the penalty represented by software rendering may be overcome by all the other advantages offered by the Flash technology. Next chapters will describe the design and implementation of a visualization framework based on Flash. The framework will be used to build an application that will test Flash visualization capabilities on real data.

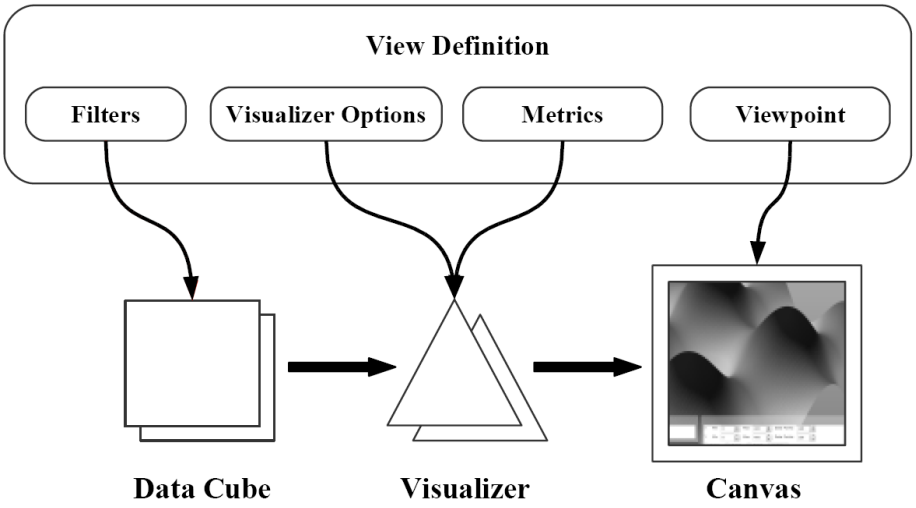
CHAPTER 4

THE PROPOSED FRAMEWORK

This chapter describes the conceptual structure of a collaborative visualization framework, designed to allow for an easy and lightweight implementation of both the framework and the client applications. The framework main entities roughly correspond to the phases of the Haber - McNabb visualization model described in chapter 2. Additionally, framework entities are adapted to allow for efficient collaboration support.

4.1 Visualization Pipeline

The visualization pipeline describes the transformation steps that bring from one or more sets of numerical data to a visualization of their content. The dataflow model described in 2.1.1 is the best known example of a visualization pipeline. Figure 16 illustrates the main components of the visualization pipeline defined by the presented framework. Three components of the pipeline (Data Cubes, Visualizers, and the Canvas) have functionalities that can respectively be mapped to the three phases of the dataflow model (Filter, Map, and View). In this regard, the conceptual framework described in this chapter does not define a novel visualization methaphor: instead it expands or refines several concepts of a standard model to integrate interactivity and collaboration support while keeping the entire design as simple to implement as possible. Next sections will present in detail each framework element.



16. The main elements of the framework visualization pipeline.

4.1.0.1 The View

The central element of the visualization framework is the **View**. A View is simply a visual representation derived from one or more sources of data. Three main elements are involved in the process of creating a View: **DataCubes**, **Visualizers**, and the **Canvas**.

4.1.1 Data Cubes

DataCubes locally store the actual data that will be used to generate the visualization. DataCubes are obtained trough queries to **Data Services**. Data Services mediate the access to remote data sources, and fill the local DataCubes with the requested data. Queries are essentially composed by a set of *filters* that determine the properties of desired data (ie space boundaries, time reference, detail levels and so on).

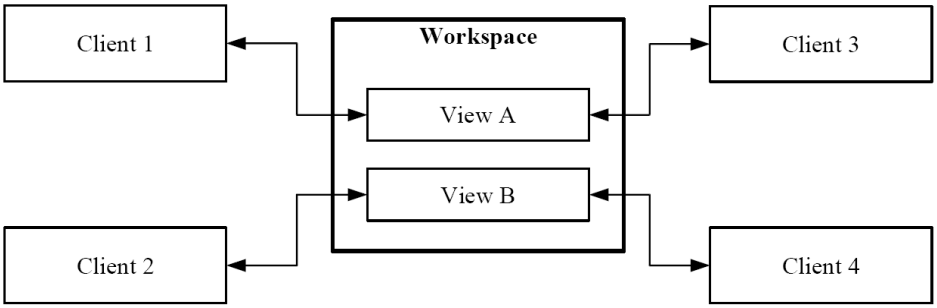
4.1.1.1 Visualizers

Visualizers perform the mapping process between numerical data and its visual representation. Different visualizer implementations generate different visualizations of the same DataCube. Visualizers can implement generic visualization metaphors (ie the mapping of a scalar field to a 3d heightmap), or define application-specific visualization techniques. A visualizer may access three input sources:

- A **DataCube** containing the data used to generate the visualization.
- A set of Visualizer-specific options that control the way the visualization is generated (ie color gradients, opacities, tessellation levels, ...)
- A set of *metrics* used to convert numeric units (meters, seconds, ...) used in source data to nondimensional 3d units used by the visualization. Unlike visualizer-specific options, metrics are shared by all the visualizers which are part of the same view. This greatly helps in the creation of consistent views, and simplifies dimension scaling operations.

4.1.2 The Canvas

The **Canvas** represents the viewport in which the View will actually be rendered. Viewpoint options determine the characteristics of the virtual camera used to observe the view, like position, zoom, rotation etc.



17. Two clients working on different views managed by the same workspace service.

4.1.3 The View Definition

Filters, Visualizer Options, Metrics and the Viewpoint all together form a View Definition. A View Definition is a simple data structure that completely defines how a View should be created. Given a view definition, an application based on the framework should be able to:

- connect to the requested Data Services (if available);
- obtain correctly filtered data cubes;
- instantiate needed visualizers, attach them to related data cubes and set up their parameters;
- setup the canvas viewpoint.

At the end of this process a visualization that exactly corresponds to the view definition can be rendered.

4.2 Collaborative visualization

A view definition can be implemented by a relatively lightweight data structure. Definitions can then be easily exchanged between remote users, carrying around information on how to build visualizations from client to client. The controlled exchange of view definitions is actually the basis of cooperative services offered by the framework.

4.2.1 The Workspace Service

The core of cooperative functionality is the *Workspace Service*. A workspace service acts mainly as a repository of views. Users can publish their views on a workspace or connect to views published by others to participate in a cooperative session. It is worth noticing that users connected to the same workspace may be participating to different cooperative sessions. As an example in figure 17 clients 1 and 3 are cooperating on view A while clients 2 and 4 are working on view B. Being connected to the same workspace though, all the clients will be able to see newly published views by other users, connect to any of the published views and communicate with other users.

4.2.2 Update policies

As underlined above, cooperation is achieved mainly by sending around view definition updates. The cooperation metaphor (ie the way in which users will actually collaborate in a cooperative session) can be described by two main factors:

- Which components of a view definition are part of update messages (a subset of filters, visualizer options, metrics, viewpoint definition);

- What is the filter policy used by single client in sending and receiving those updates.

The conceptual framework currently defines two different kinds of update message:

- *Camera updates* carry only the viewpoint component of a view definition;
- *View updates* carry a complete view definition.

The possible filter policies applied to update messages may be:

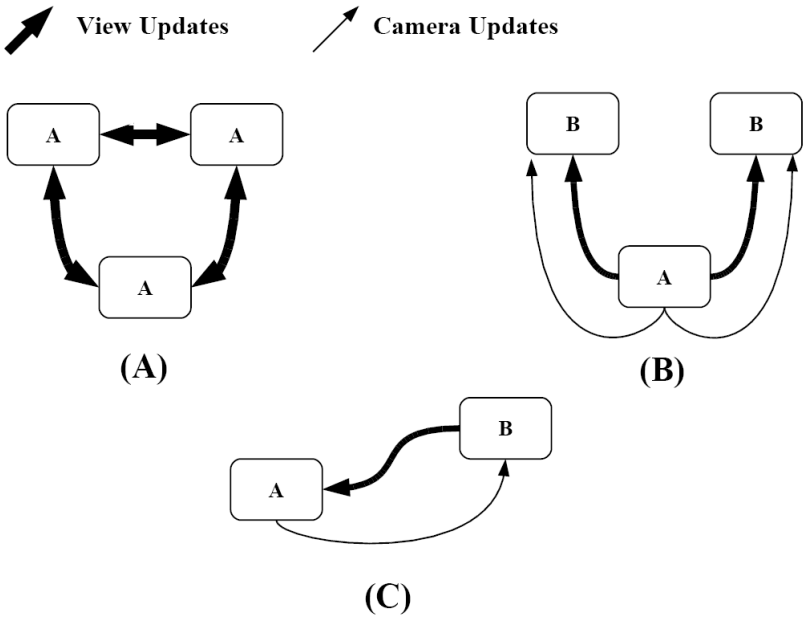
- *input:* a client will accept updates from other clients;
- *output:* a client will broadcast local updates to other clients;
- *input-output:* both of the above.

The definition of a filter policy for each class of update, possibly in different ways for each participating client, determines the applied cooperation metaphor.

4.2.3 Cooperative configurations

Figure 18 illustrates three of the possible collaborative configurations supported by the collaboration model defined in section 4.2:

- in figure 18A three clients cooperate in defining the contents of the view, but each client has his own independent viewpoint;
- in figure 18B a single client determines both view content and the viewpoint, and sends updates to two clients which act as passive viewers;
- in figure 18C shows an exotic, yet possible, cooperative configuration: a client chooses the viewpoint while the other one determines view contents.



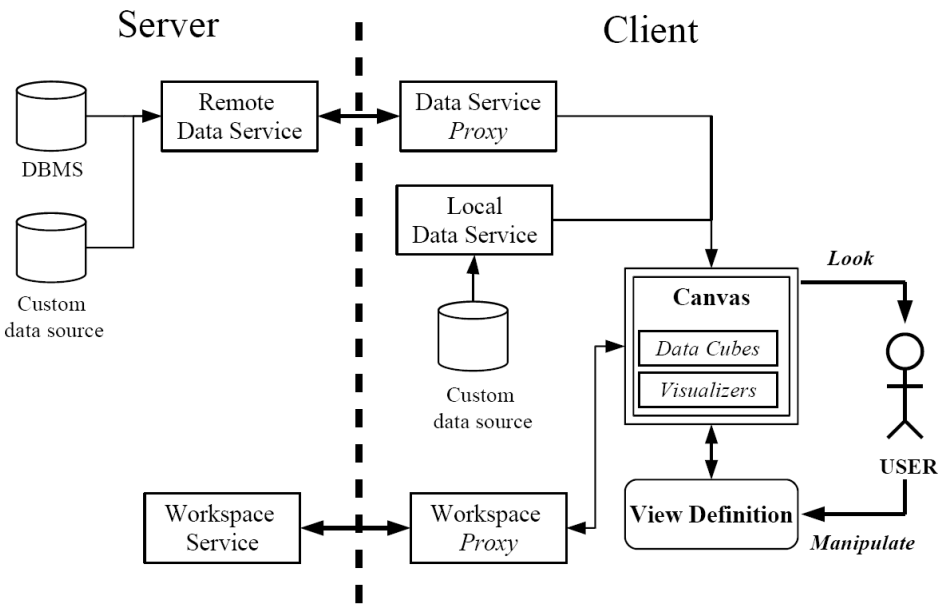
18. Examples of collaborative configurations supported by the framework.

4.2.4 Overall application structure

Figure 19 illustrates the architecture of a complete visualization application based on the presented framework. The user manipulates the view definition, and looks at the manipulation result on the canvas connected to the view. The canvas contains the data cubes and visualizers used to generate the visualization, makes requests to data services and possibly communicates to the remote workspace to manage a collaborative session.

4.3 Summary

This chapter defined a conceptual framework for collaborative visualization. The framework was not designed to introduce any new visualization methaphor: its core concepts, the Data Cubes, the Visualizers and the Canvas are based on the same concepts which can be



19. Overall structure of a visualization application.

found in the standard dataflow model. The framework defines instead a new concept that adds both interactivity and collaboration to the dataflow model: the View Definition. The view definition is a complete and lightweight descriptor of the contents of a view. Each element of a view definition is a parameter of one of the core elements of the visualization pipeline. The exchange of view definitions is the basis of collaborative support. View definition shared by clients through Workspace Services, and view definition updates are sent and received depending on a set of *filter policies* defined locally by each client: these filter policies define the collaboration rules used by the set of clients that are working on the same view.

Section 3.1 defined some constraints for technologies that could be considered for web-based visualization application developement: One of these constraints was ease of develope-

ment. This constraint depends not only on the technological choice, but also on the *design* of the system that is going to be implemented. This is the reason why the number of entities and entity interactions introduced by the conceptual framework has been kept as low as possible: the purpose is to simplify the implementation of the framework itself, independently from the technology chosen to implement it.

CHAPTER 5

FRAMEWORK IMPLEMENTATION

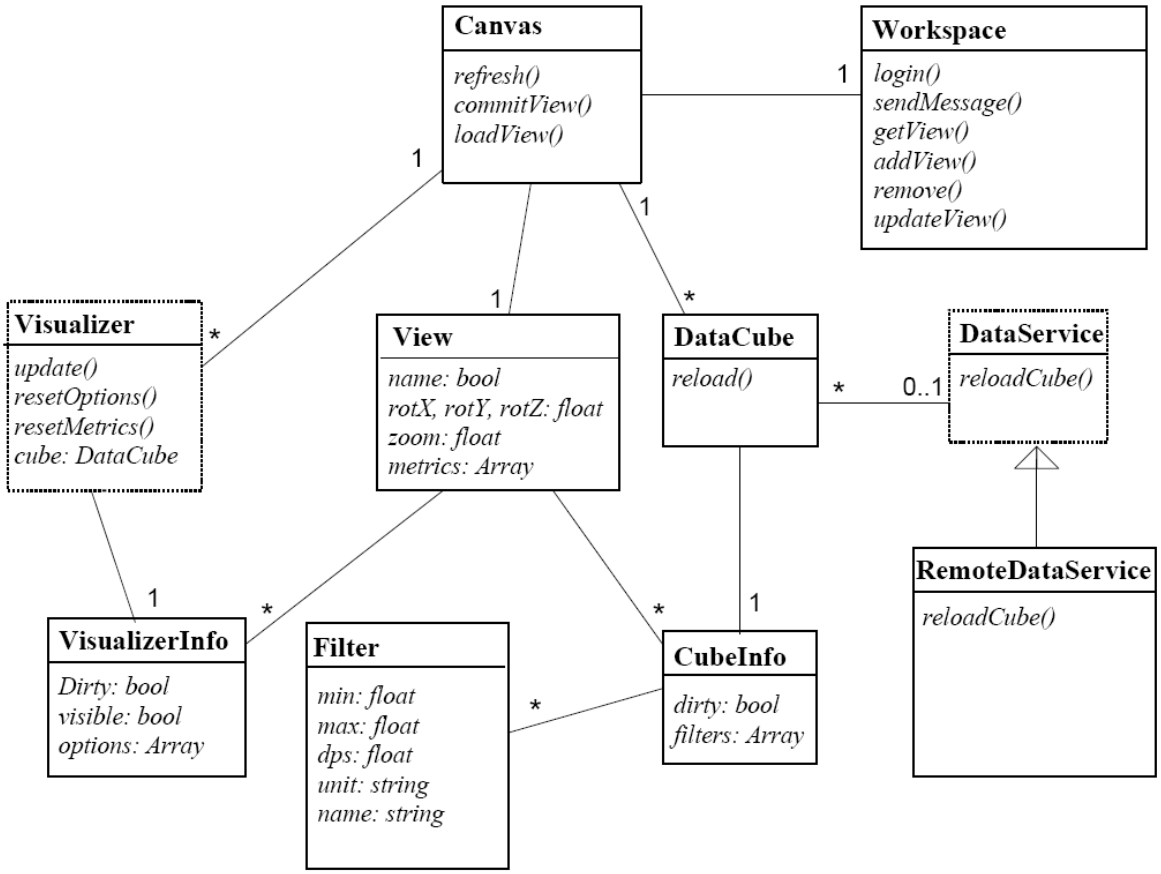
This chapter will describe the implementative details of *QbViz*, a prototype flash-based visualization library. The QbViz client API has been implemented as a set of ActionScript classes that directly map to the conceptual entities described in chapter 4. The server backend is currently implemented in C#, although other language choices are possible for alternative implementations.

5.1 Client structure

The UML diagram of the principal framework client classes is shown in figure 20. The following sections will present these classes in detail.

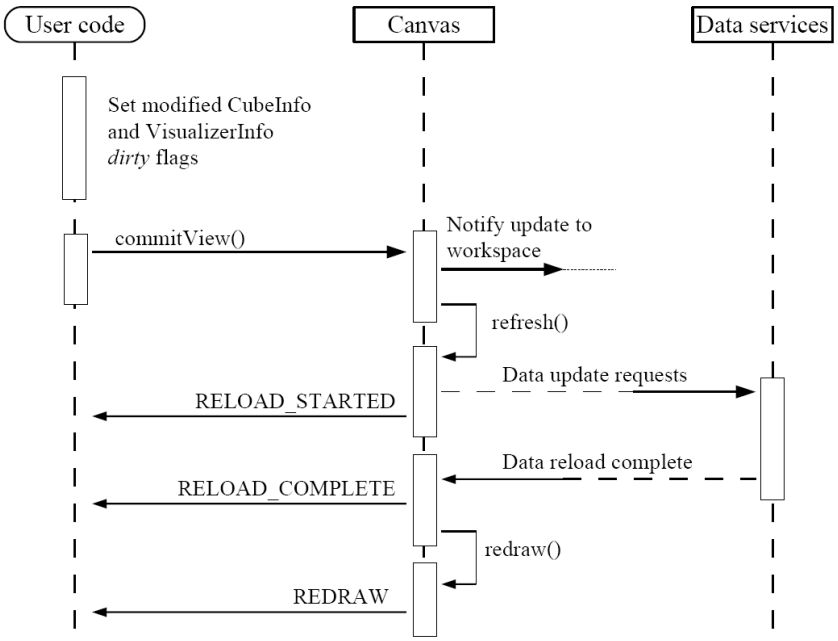
5.1.1 The Canvas Class

The `Canvas` class implements the drawing surface on which visualizations are rendered. The class is implemented as standard Flex user interface component: this allows to easily use it inside a Flex application layout, like one would do with any other visual component (i.e. buttons, text areas and so on). `Canvas` internally manages all the viewpoint manipulation logic, which supports mouse-controlled viewpoint rotation zooming and translation. Viewpoint manipulation can be disabled, if one wants to control the viewpoint directly inside the application logic.



20. UML digram of client-side QbViz classes.

A `Canvas` instance has associated a `View` object, which represents the definition of the currently rendered view. A Canvas can optionally be associated with a `Workspace` object, to enable View sharing and collaboration between remote users connected to a common Workspace service.



21. Sequence diagram of a canvas update process.

The most important methods of the `Canvas` class are:

- `Refresh`, which performs a local refresh of the visualization. The contents of the canvas will be updated to reflect the possibly changed view definition inside the `View` object associated with the canvas;
- `CommitView`, which performs both a local refresh, and sends a view update to all the other clients connected to the same view;
- `LoadView`, which loads a view from the workspace service, and attaches it to the canvas. The LoadView method is commonly used to initiate a collaborative session on the client.

When a user wants to interact with a view and see the outcome of the manipulation on the canvas it is attached to, he has to perform the following operations (see Figure 21):

1. the user modifies some properties of the `View` definition (i.e. cube filters, visualizer options, metrics, ...);
2. the user marks modified cubes or visualizers as *dirty*, through the **VisualizerInfo** and `CubeInfo` dirty property. This allows the canvas to quickly identify elements that need an update, without the need to perform complex traversal and comparisons with the previous view structure;
3. the `Canvas.commitView` method is called. The `Canvas.refresh` can be called instead of **commitView** if the user explicitly wants to avoid broadcasting view updates to other users;
4. user code can optionally listen on the canvas for `RELOAD_STARTED`, `RELOAD_COMPLETE` and `REDRAW` events to monitor the ongoing refresh process. These events respectively signal the begin of data request for the new visualization, the completion of data request and the end of canvas redrawing.

5.1.2 The Visualizer Class

`Visualizer` is the base class for visualizer implementations. A `Visualizer` instance is always associated with a `VisualizerInfo` object, that describes the properties of the visualization. the `VisualizerInfo` object is a complete descriptor of the visualization, meaning it stores all the information needed to instantiate and initialize a visualizer. Classes that derive

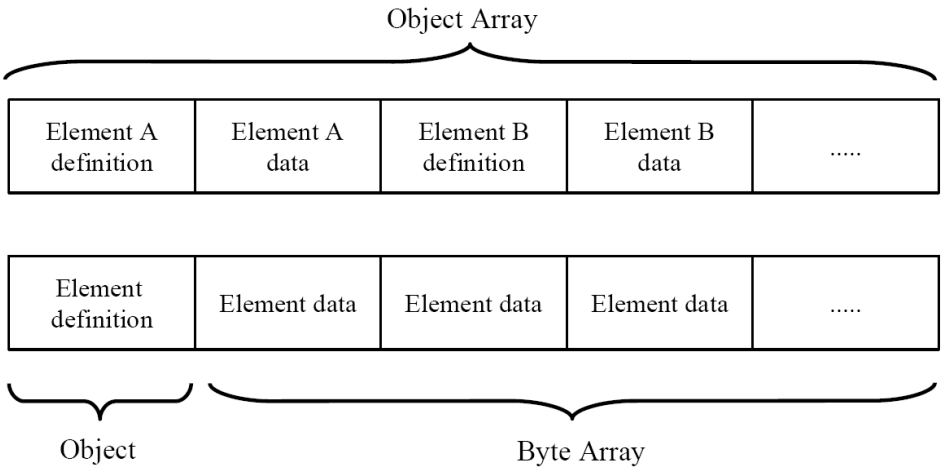
from `Visualizer` must at least reimplement the `update` method, that should completely regenerate the geometry (or other representation elements) managed by the visualizer. This method will be called whenever a new version of the visualized data has been retrieved. Derived classes can optionally override the `resetMetrics` and `resetOptions` methods, that are called respectively when view metrics or visualizer options have changed, without requiring a full data reload. Redefining these methods allows a visualizer to avoid useless (and potentially time demanding) full visualization updates.

5.1.3 The DataCube Class

A `DataCube` instance represents a generic set of data used for visualization purposes. A `DataCube` instance is associated with a `DataService` object, which actually performs the data request and retrieval operations. A `DataCube` also contains a `CubeInfo` object that describes the properties of the data cube. This is the same exact decoupling of *descriptive* and *functional* parts of the object that has been applied to the `Visualizer` class. As explained in section 4.2 this separation allows to easily build a complete and compact view definition, made only of the descriptive components of the view.

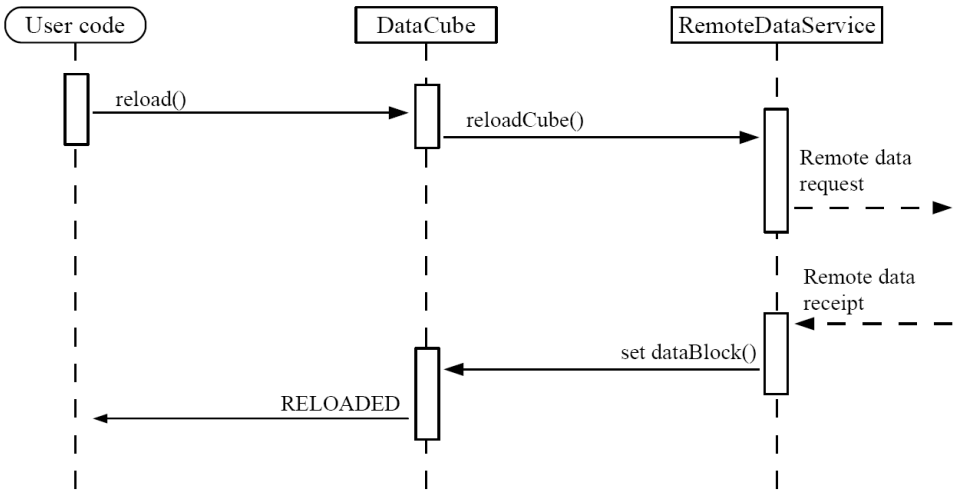
5.1.3.1 Data format

Data stored inside data cubes is conceptually represented by an array of data elements: each element can be a scalar, a vector or some other structured type depending on the format of the source dataset. A problem with this type of data representation arises in the case of remote data requests: executing these request can be quite inefficient depending on the applied *serialization format*. A serialization format describes the way in which structured objects are



22. Data layouts of standard and optimized data serialization formats.

converted into a linear representation of their content, for storage or transmission purposes. It has been underlined that the current QbViz implementation uses flash remoting as its main communication technology. Flash remoting employs a data serialization format called AMF3, (37). When an object array is serialized in a data stream using the AMF3 format each data element carries a significant overhead associated with its *type definition* (figure 22a). The type definition describes the structure of the serialized object, like the name of fields, the type of serialized values and so on. This overhead can be extremely relevant, up to 30% of stream length in experimental tests using the application prototype presented in chapter 6. Its worth noticing that the need for a type definition of each object inside a serialized array has sense only when handling arrays made of eterogeneous elements. In the case of data cubes though, the data stream are always made up of arrays of elements having the *same structure* (i.e. the elements are all scalars, or vectors, or other omogeneous datatypes). This allows to use a much



23. Sequence diagram of a Data Cube reloading process.

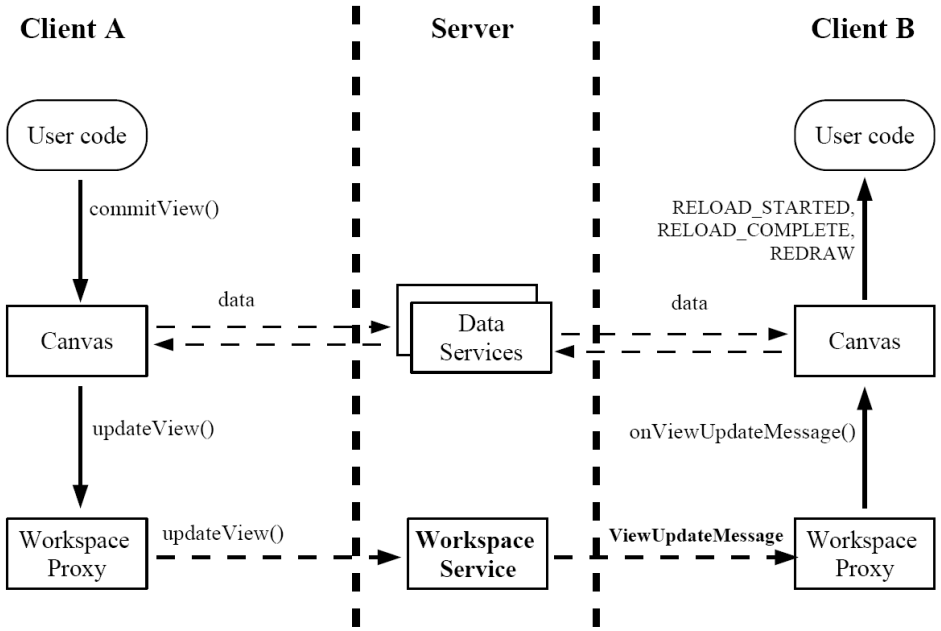
more compact serialized representation of data: The entire element sequence is serialized as an array of bytes (which requires a negligible overhead in the AMF3 format), and a single object containing the type description sent together with the data (figure 22b).

5.1.4 The DataService Class

`DataService` is the base class for objects that expose dataset access functionalities inside QbViz. A `DataService` instance can create `DataCube` objects and fill them with data matching the filters found inside the cube definition.

A `DataService` derived class must at least implement the `reload` method, which manages the update of a data cube.

The only data service implementation currently offered by QbViz is `RemoteDataService`, which offers access to a remote data source, exposed through QbViz server components, using

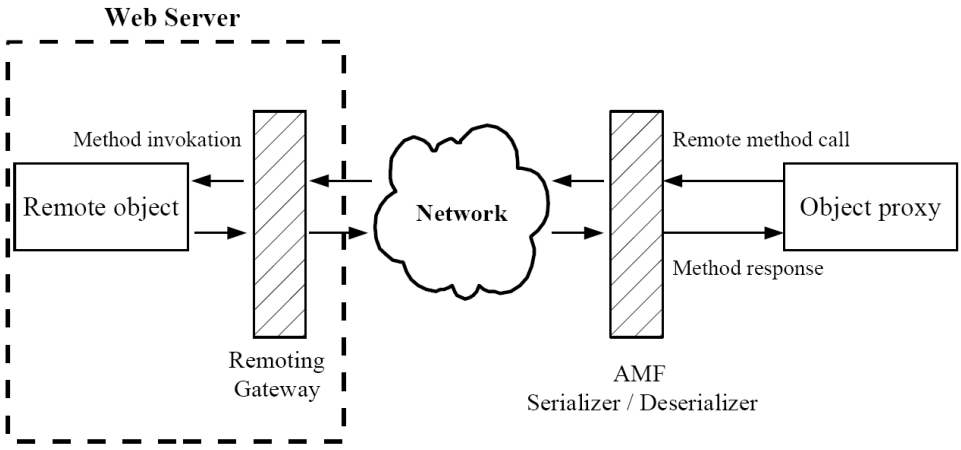


24. Diagram showing how view updates are communicated between two clients.

a flash remoting communication channel. A sample cube reload sequence is shown in figure 23

5.1.5 The Workspace Class

The `Workspace` class is used to manage all the collaborative features of a QbViz application. Upon creation, a workspace connects to a remote, user-specified *workspace service*. All clients connected to the same workspace service are logically part of the same collaboration session: they can publish views and manipulate them cooperatively. The easiest way to exploit collaborative services is to associate a workspace with a canvas instance, using the `Canvas.workspace` property. When the canvas is showing a shared view, it will automatically use workspace services to send and receive view updates from other users (figure 24).



25. Remote method invocation through a flash remoting gateway.

User code can also directly access some other workspace features through its methods:

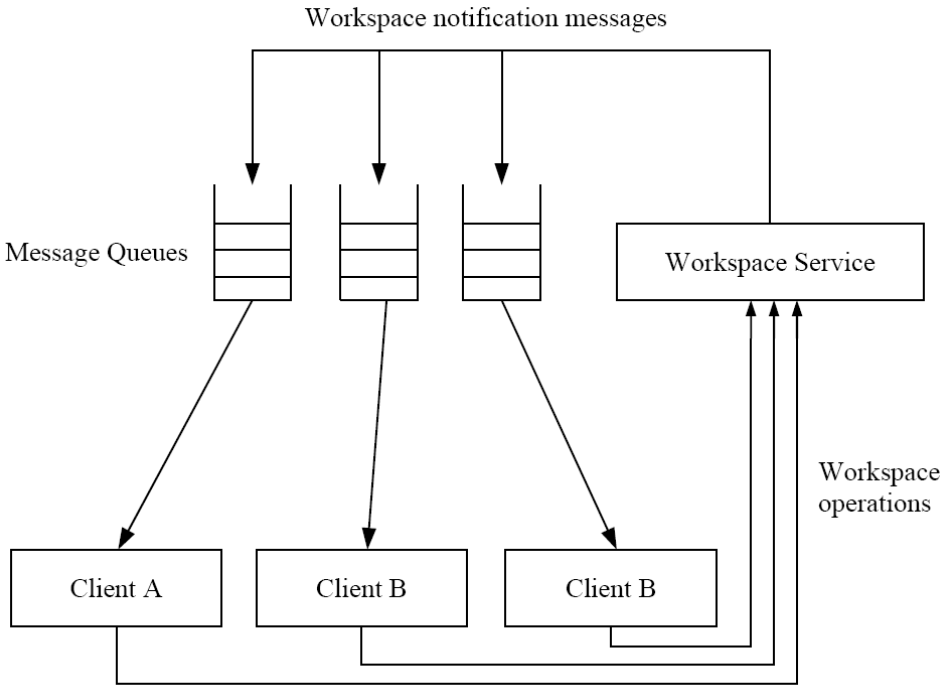
- `addView` and `removeView` are used to respectively add and remove a view from the published view list.
- `sendMessage` allows to send text messages to other users. message reception will be notified by the `ChatMessage` workspace event (see section 5.3)

5.2 Server structure

Most of the server side classes replicate the structure of client-side equivalents. This is due to the fact that the majority of server objects takes part in remoting interactions, either as a proxy or a serializable object.

5.3 Client-Server communication

The communication between client and server is handled by a *remoting gateway*. A remoting gateway is installed as a classic server-side application component. It intercepts all



26. Handling communication between client and the workspace through message queues.

HTTP requests related to remoting invocations, and performs the actual calls inside application server code. The gateway handles also the deserialization of call arguments into server-side types, and serializes back the result is a format readable by the client-side actionscript code (figure 25). QbViz currently uses WebOrb (39) as its remoting gateway, although other choices are possible (i.e. Fluorine, (53)).

There are two distinct patterns of communication needed by the framework:

- **Standard request-response:** the client invokes a method on a server-side object, and possibly waits for the asynchronous response.

- **Server-side events:** the server has to notify an event to the client (i.e. an action performed by another user in a collaborative session.

The standard request-response pattern, being initiated by the client, follows the normal remote invocation pattern, and poses no particular development difficulty. Server-side events instead, cannot be implemented directly, since remote method invocation is always unidirectional and goes from client to server (mapping exactly on HTTP request-response primitives). A straightforward way to solve this problem is to manage also server-side events with a request-response pattern. The implemented solution is shown in figure 26: the server keeps a set of *message queues*, one for each connected client. These queues contain event notifications (*messages*) directed to the respective client. Each client periodically sends a `Poll` request to the server which replies with the list of all the messages for that client, and then clears the queue.

All of workspace client notification are handled using message queues. Currently, QbViz implements four types of messages:

- `ChatMessage`: Contains a text message sent by a user to some or all of the other clients in the workspace. `ChatMessage` can also be used by the server to send textual notifications to specific clients.
- `ViewAddedMessage`: Notifies the user that a view has been added to the workspace. The message contains the unique name of the view, and optionally the URL of a *view thumbnail*, a small image that contains a snapshot of the view. This image can be used by the client application to offer *preview* functionality for visualizations.

- `ViewRemovedMessage`: Notifies the user that a view has been removed from the workspace.
- `ViewUpdateMessage`: Contains update information for a view. This message can transport three types of updates: *content updates*, *camera updates* or *state updates*.

  Content updates contain a complete view definition that is sent to all the users connected to a specific view, so that they can update the local view when a remote user makes some change in the view contents (i.e. he adds a visualizer).

  Camera updates contain just a viewpoint definition. When update filters are configured to allow so, camera updates are used to communicate viewpoint changes, without requiring a complete view refresh as a content update message would do.

  State updates carry information used to refresh the visualization preview only, i.e. a new thumbnail URL or an updated list of users connected to that view. State updates are the only update message which is sent to all the users connected to a workspace, and not only to the users that are actually working on the specific view.

5.4 Summary

This chapter presented the implementation of a prototype visualization framework based on the concepts introduced in chapter 4. The developed framework has been used to build *Hydroviz*, a test-case collaborative application targeted at hydrographic visualization. The description of Hydroviz will be the object of next chapter.

CHAPTER 6

HYDROVIZ, A TEST-CASE APPLICATION

This chapter presents *Hydroviz*, a test-case visualization application implemented using the QbViz framework prototype described in chapter 5. The application, targeted at hydrographic visualization, can be run from a normal web browser and makes use of both 3d visualization and collaborative features offered by QbViz.

6.1 Purpose of the application

The application has been inspired by the collaboration between the Electronic Visualization Laboratory at University of Illinois at Chicago and the University of Illinois at Urbana Champaign to visualize salinity and currents in Corpus Christi Bay, Texas. The application prototype is able to visualize a qualitative 3d model of the bay, and decorate it with salinity and current flow information under user control. Collaboration support and ease-of-access were also considered important requirements during the application design and implementation phases.

To achieve efficient data access, an additional requirement was to preprocess and convert the source dataset to a suitable format and store it on a database. Details about this operation are presented in next section.

Category	Data fields	Total size
Bathymetry	node ID; x, y coordinates (UTM); depth (meters)	145Kb
Salinity	timestamp (seconds); node ID; vertical layer ID; salinity	166Mb
Currents	timestamp (seconds); node ID; vertical layer ID; x, y velocity (m/s)	210Mb
Layers	layer ID; depth (meters)	1Kb

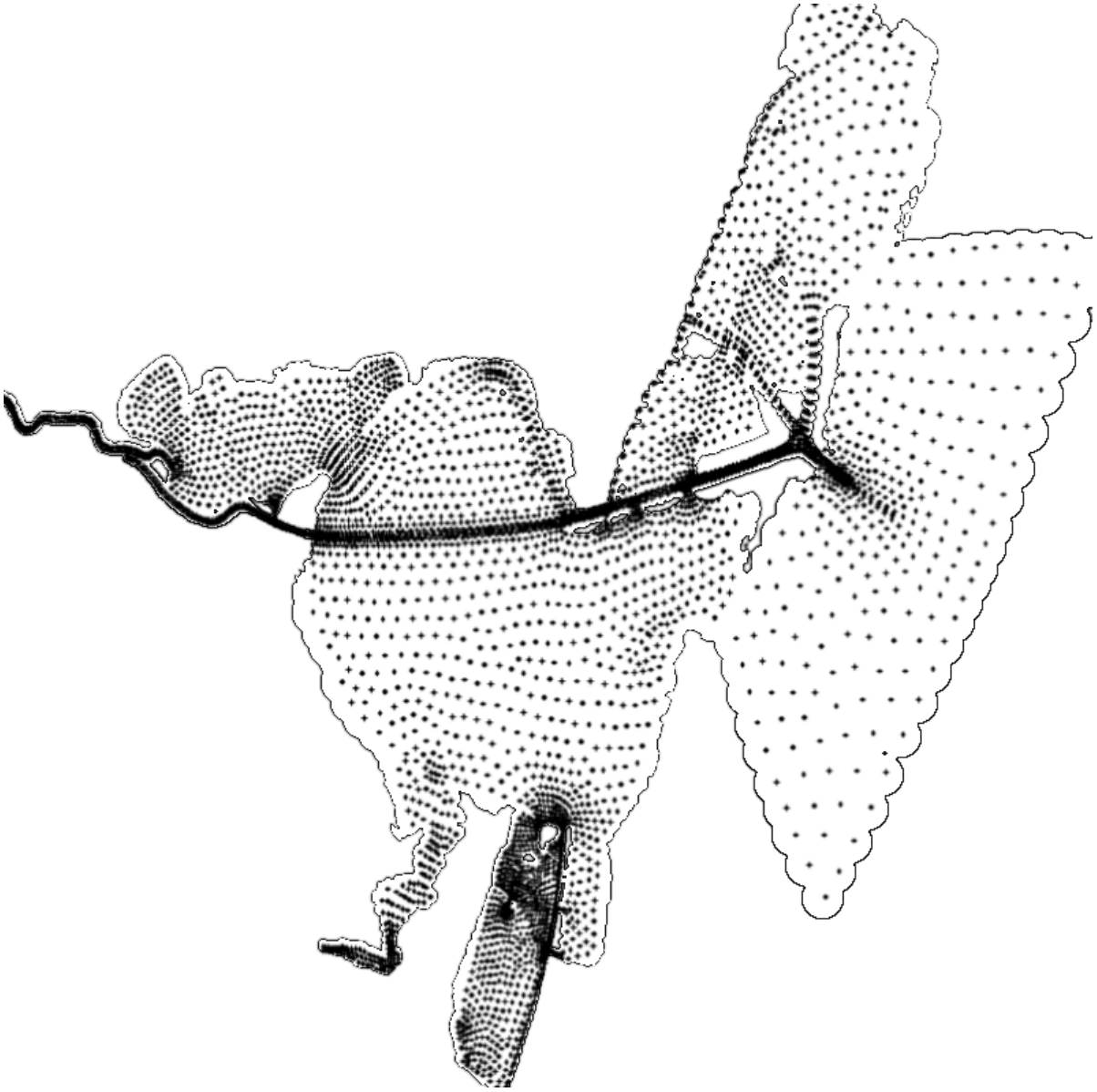
II

OVERVIEW OF SOURCE DATA STRUCTURE.

6.1.1 Available data

The original source data has been made available by Texas Water Development Board. Data collection points were distributed non-homogeneously over the bay (figure 27) and sampled salinity, current speed and direction at various depths. Data collection lasted about three days, with a sampling interval of two hours. There were 4216 distinct data nodes, collecting 21 depth samples on average for a total of 35 time snapshots. The total number of datapoints is slightly more than 3 millions. For each data collection point, the exact depth of the bay has also been made available. Source data has been provided as a set of plain text files. Each row in the these files contains a set of tab-separated records with information about a specific datapoint. The overall dataset has been split in four main groups. The exact format and size of data for these groups is reported in table II.

Source data has been preprocessed, and put inside a database for easier and optimized access by the application data service. The format of data stored on the database is illustrated in table III. The database representation is basically a compact version of source data. There



27. Distribution of measurement points in Corpus Christy Bay

Table name	Data fields	Total size
Datapoints	node ID; x, y coordinates (UTM); layer ID; salinity; x, y velocity (m/s); timestamp (seconds); detail	200Mb
Layers	layer ID; depth (meters)	1Kb

III

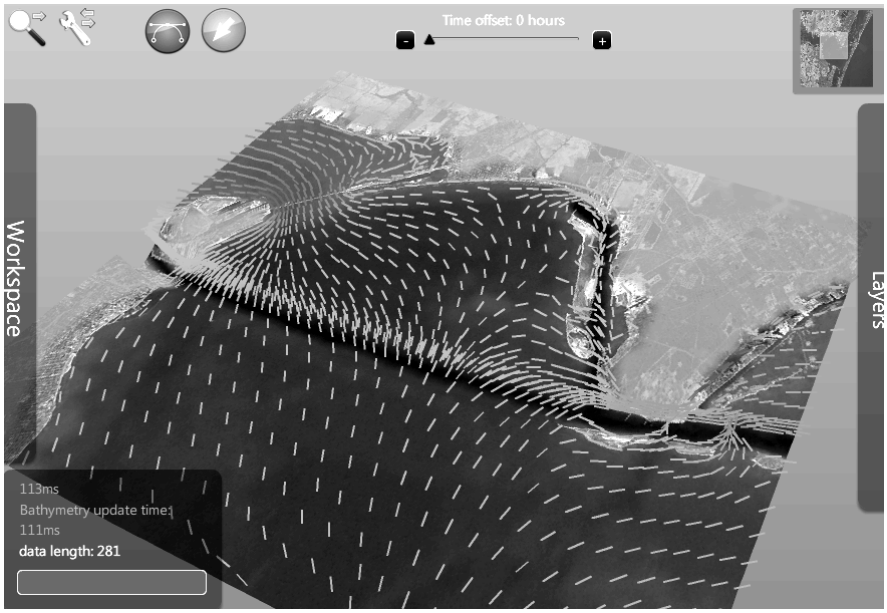
PREPROCESSED DATA STRUCTURE.

is a single field not present in the original data: *detail*. This field is calculated by the data preprocessing tool and is used to mark datapoints belonging to different detail levels, to allow for faster data retrieval by the *layer visualizer* (see section 6.3.2).

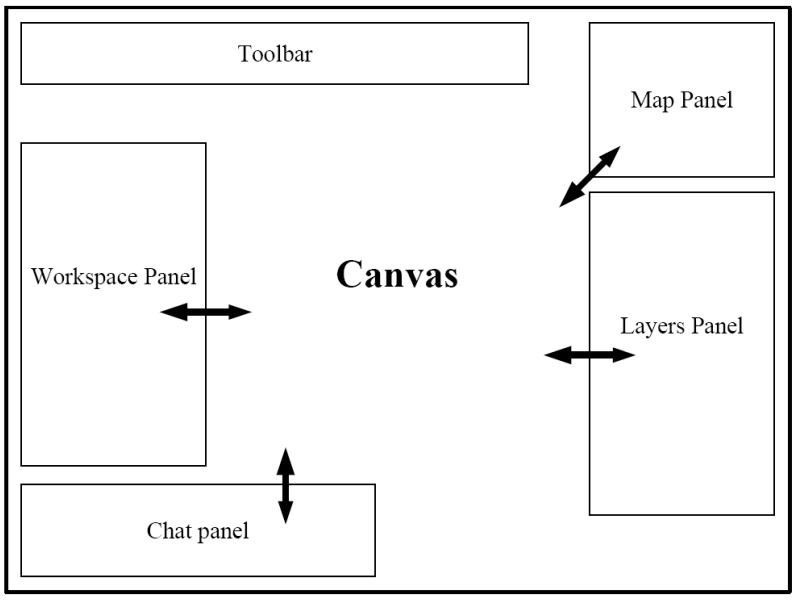
Bathymetry data is not stored on the database. Another preprocessing tool has been developed to convert it into a *heightmap*, which is used by the *bathymetry visualizer* to build a qualitative 3d model of the bay at runtime. Details about the bathymetry visualizer can be found in section 6.3.1

6.2 Application layout

Figure 28 shows a browser window running an instance of the Hydroviz application. The application has been designed to leave as much space as possible to the visualization. Much of the controls and manipulation tools offered by the application are grouped in panels that disappear or minimize themselves when the user is not interacting with them. The application controls and tools are collected into five main areas: the toolbar, the map panel, the layers panel, the workspace panel and the chat window (figure 29). The following sections will present each component of the hydroviz interface in detail.



28. A screenshot of the Hydvoriz client application running.



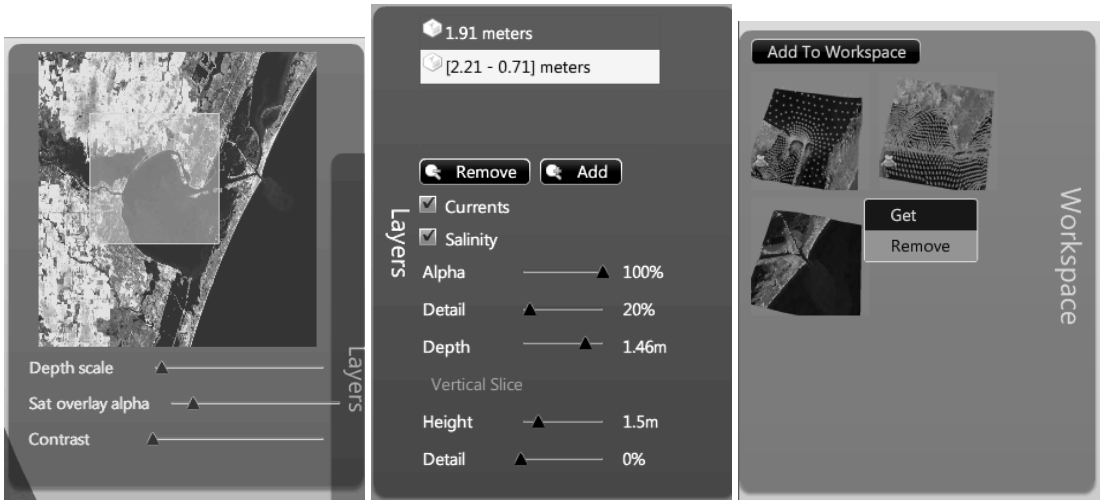
29. Hydroviz application layout. Arrows indicate how panels expand when selected

30. The Hydroviz toolbar.

6.2.1 The toolbar

The toolbar is placed on the top edge of the application. It contains buttons and controls that serve various purposes:

- **The collaboration filter buttons** are used to setup the update filters used in collaborative sessions. Clicking on a button will loop through the configuration options available for the relative filter. An example usage of collaboration filter will be described in section 6.4.6.
- **The streamlines panel button** opens a window used to configure visualiation options for interactive streamlines. The example presented in section 6.4.5 illustrates the usage of streamlines.
- **The datapoint selection toggle**, when enabled, allows the user to select single datapoints in the visualization and interact with them. When the selection toggle is disabled, user interactions with the visualization will affect the viewpoint position instead.
- **The timeline** is used to visualize data corresponding to the specified time offset from the start of data collection. The two (+) and (-) buttons at the sides of the slider are used to advance or bring back the time by a single sampling interval, which corresponds to



31. (from the left) The map, layers and workspace panel.

about 2 hours as explained in section 6.1.1. When the time offset is changed, all time-dependent visualizers instantiated in the view will be updated accordingly.

6.2.2 The map panel

The map panel (shown in figure 31 is used to define the current region of interest for the visualization. The user can define the visualized region by simply click-dragging to draw a rectangle over a satellite view of the bay. The map panel also contains controls used to fine-tune the generated 3d model of the bay: the user can define depth exaggeration factor, or adjust parameters used to generate the texture that will be attached to the 3d model.

6.2.3 The layers panel

The layers panel (figure 31) allows the user to add layers of data to the visualization, delete them or configure their parameters.

A layer is basically a visualization of a salinity and currents data for a specific region of the bay.

Each layer can show information for a specific depth under the water surface (a 'flat' layer), or show data for a depth interval (a 'slice' layer).

A layer can optionally visualize salinity or currents data information only. Data can also be visualized at different detail levels. See section 6.3.2 for a detailed explanation of layer visualization.

6.2.4 The workspace panel

The workspace panel (figure 31) is used in collaborative application sessions.

The workspace represents the shared repository where users can publish their views, or connect to views published by others to participate in cooperative visualization.

Each time a user adds a view to the workspace, a thumbnail of the view will be added to the workspace panel of all the users currently logged inside the application. Users can click on thumbnails to open a context menu that allows them to connect to the view, delete it or potentially perform other view-related operations.

Each view thumbnail can be decorated by two icons that indicate if the local user is actually connected to that view, or if other remote users are.

6.2.5 The chat panel

The chat panel offers basic chat functionality to logged on users. All the users connected to the application join a common chat room: users can chat with each other even when they are connected to different views.

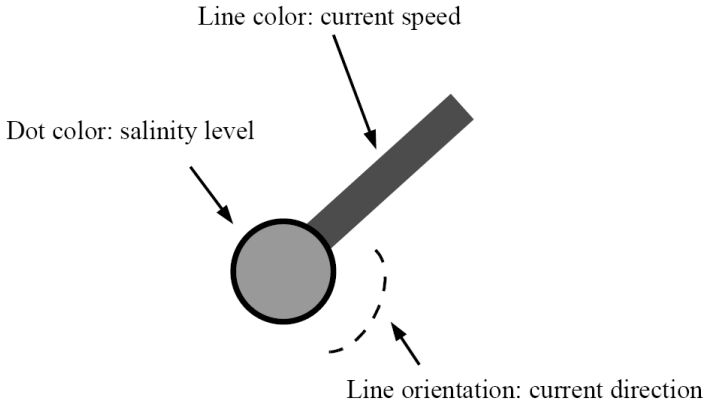
6.3 Visualizers

The Hydroviz application implements several ad-hoc visualizers: Bathymetry, Layer, Streamline and Annotation. All these visualizers can be part of a Hydroviz View. Three of them (Layer, Streamline and Annotation) can also be placed on the view multiple times and with different definitions, to build more complex views.

6.3.1 Bathymetry Visualizer

The bathymetry visualizer is used to display a qualitative 3D model of the bay. This representation is mainly based on a texture-mapped heightfield. Source data used to generate the model comes from bathymetry information available in the source dataset. The source bathymetry information is transformed into a heightmap (40). A heightmap is greyscale image, in which each pixel represents the height (or depth) of a specific point on a 3d plane (the heightfield). The plane is then deformed using the heightmap information, to generate the 3d displacement described in the heightmap. A detailed description of heightmap generation from the original, sparse bathymetry data can be found in appendix A.

It is worth noticing how in the case of bathymetry visualization the *source of data* for the visualizer is not a data cube, but an image (the heightmap), The bathymetry visualizer is therefore not associated to any datacube. This is a perfectly consistent situation for the framework. The only requirement for a visualizer is to expose a visualizer definition that identifies it inside the view. The way in which the visualizer actually loads its data is entirely up to the visualizer implementation. Data cubes represent just a common way to access data sources, their usage can be avoided completely, if a specific visualizer does not require them.



32. Representation of a datapoint glyph, indicating how data items are mapped to graphic features.

The bathymetry visualizer loads the heightmap using the standard flash image API. The image is retrieved from the web server through a common HTTP request. The visualizer does the same to retrieve the satellite imagery that will be attached on the 3d model to give it a more realistic look.

The bathymetry visualizer is also able to display specific areas of the bay. The region of interest is defined by the x and y filters in the visualizer definition. The filters define the horizontal and vertical intervals that identify the region of interest, in UTM coordinates.

6.3.2 Layer Visualizer

A layer visualizer displays salinity and currents data for a user-specified region and depth of the bay. Visualization of the datapoints is achieved using glyphs, (9)(10). Glyphs are symbols used in the visualization of multi-variate data: whose geometric (e.g., shape, size, orien-

tation) and non-geometric (e.g., color and texture) visual attributes are determined by the data they represent.

Glyphs used by the layer visualizer represent information about salinity, current direction and current speed using colors and glyph orientation. The configuration of a glyph is detailed in figure 32. The user can configure the format of glyphs visualized by a layer, deciding whether to show salinity information only, currents information, or both.

Additionally, the user can specify the detail level used by a layer visualizer. Higher detail levels lead to more dense visualizations (i.e., more glyphs will be displayed). The Hydroviz data service is able to quickly retrieve and send to the visualizer just the datapoints that will be actually shown for a specific detail level.

The layer visualizer also supports selection of glyphs. It is possible to click on a glyph and obtain more precise information about the corresponding datapoint (like the actual numerical values of salinity level and current speed). When clicking on a glyph, a contextual menu is shown, exposing additional operations that can be performed on the selected datapoint, like adding an annotation (section 6.3.4), or creating a streamline (6.3.3).

6.3.3 Streamline Visualizer

A streamline is a line for which a tangent in any point is parallel to the vector field. More formally, a streamline is calculated by integrating the following equation:

$$\frac{d\mathbf{x}}{d\tau} = \mathbf{v}(\mathbf{x}, t) \tag{6.1}$$

where **x** is the position in space, **v** is a velocity field, and t the time. The integration variable, τ, is a pseudo-time variable because the velocity field does not vary through time. A bucket of streamlines is commonly used in steady flow analysis because it expresses the structure of the field, and allows for the detection of vortices.

The hydroviz application is able to interactively display streamlines, starting them from user specified points. The user can access to the streamline generation tool from the datapoint context menu, as explained in section 6.3.2

Computation of the streamline is done on the server. This avoids having to compute the streamline multiple times on each client in collaborative scenarios, and reduces the amount of transferred data (a streamline datacube is usually much smaller that the relative layer datacube). Streamlines are sent to client as a list of points in UTM coordinates. Each point also carries information about sampled salinity and current speed. The streamline visualizer draws the set of points as a continuous 3d line, optionally decorating it with information about salinity and speed. The user can place multiple streamline visualizers on the view. Each streamline visualizer manages a single streamline.

6.3.4 Annotation Visualizer

The annotation visualizer allows the user to add georeferenced text notes to the view. It is possible to attach an annotation to a layer datapoint, selecting the relative option in the datapoint context menu.

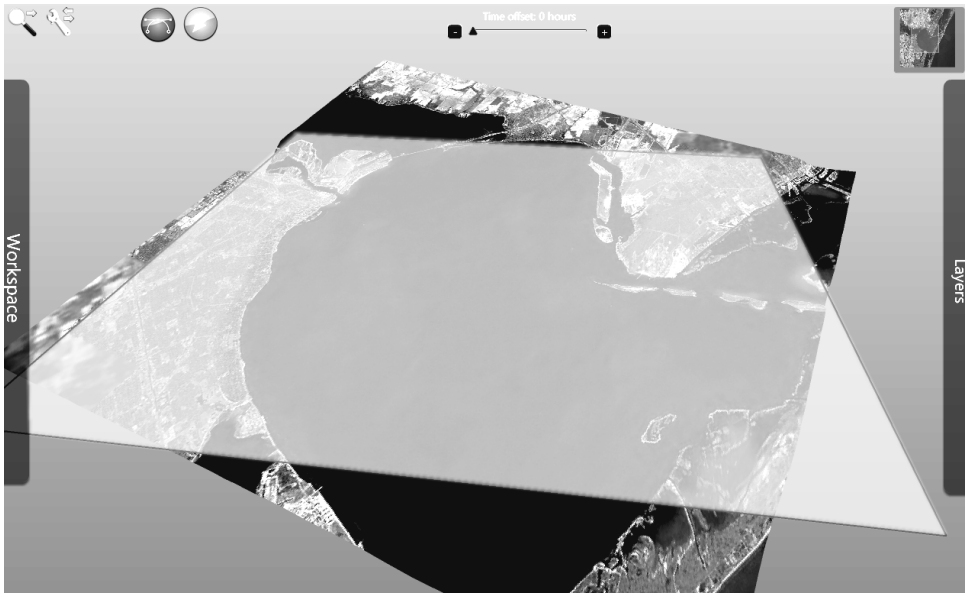
As for streamlines, each annotation is managed by its own annotation visualizer. The data needed by an annotation visualizer is really simple: a point in UTM coordinates and a string

33. Streamline visualization.

containing the annotation. For this reason, annotation visualizers do not use any data service. Information associated with an annotation is embedded directly in the visualizer definition as custom data. This is another example of data access flexibility offered by the QbViz framework.

6.4 Application usage scenarios

This section will present several examples of usage of the Hydroviz application. Each scenario will be introduced by a brief explanation of the task that the users is trying to accomplish. A detailed description of the interaction with the application will follow. The purpose of user scenarios is to identify several common interaction patterns with the application. The detailed



34. Model manipulation using impostors.

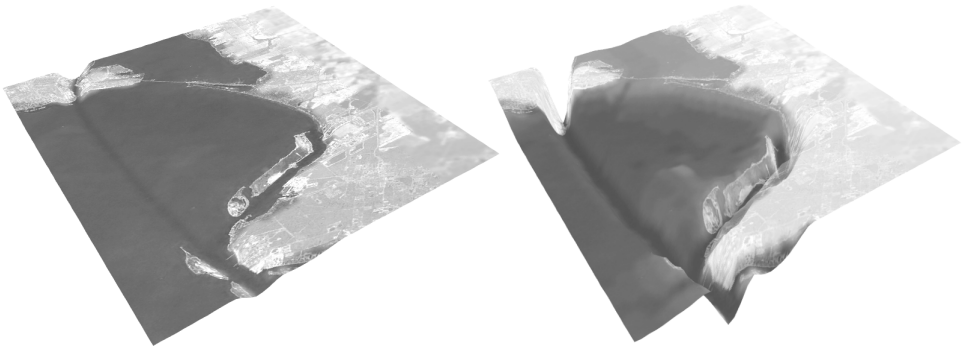
description of actions that users need to perform in order to achieve the task can be used to assess the functionality and ease of use of the application.

6.4.1 Basic 3d model exploration

The user wants to access the application, and simply explore the 3d model of the bay. He wants to observe the model from different viewpoints, and view some of its regions in detail.

1. *Login*: the user specifies its name in the login window. This name will be used to identify the user in the chat window and in the workspace panel.

2. *Manipulate the viewpoint*: after logging in the user is presented with an default view consisting only of the bay model. Clicking on the view and moving the mouse will rotate the



35. The same bay area visualized using different model parameters.

viewpoint around the model. Holding the `Shift` key while moving the mouse vertically will modify the zoom level. Holding the `Ctrl` key will change the viewpoint position.

When changing the viewpoint, a red indicator plane is shown (figure 34). The indicator plane will stay on screen and will give the user a hint on how the viewpoint is changing. When the mouse button is released, the indicator plane will be substituted by the actual updated view. This technique is used instead of real-time updates of the model to achieve good reaction times even on complex, data-rich visualizations, and on slower machines.

3. *Define the region of interest*: the user changes the currently visualized region by click-dragging on the map view in the map panel on the top-right corner of the application. The visualized model will be immediately updated to show just the area of the bay defined by the user.

4. *Tune the model visualization*: using again the map panel, the user can define some properties of the bay model, like the depth scale factor, the opacity and contrast of the texture

applied to the model. Figure 35 shows how different views of the model can be achieved manipulating these parameters.

6.4.2 Basic data visualization

The user has to visualize salinity information for a specific region and depth of the bay. Additionally, he wants to display current direction and speed data for a subset of that region.

1. *Define the region of interest*: the user interacts with the map panel to define the visualized region.
2. *Add a data layer*: through the layers panel, on the right side of the screen, the user adds a data layer to the visualization. By default, the data layer will show both salinity and currents information at a low detail level. The layer region of interest will correspond to the currently visualized region of the bay.
3. *Set layer properties*: the user selects the newly created layer on the layers panel, and sets the desired layer depth. Additionally, the users disables current visualization through the salinity checkbox.
4. *Add another data layer*: using again the layers panel, and defines its parameters.
5. *Change layer region*: to modify the second layer region of interest, the user selects the layer and goes to the map panel. The selected layer region of interest will be shown as a yellow box. To change it, the user click-drags on the map panel, while holding the `shift` button pressed. The final result is shown in figure 44, figures appendix.

6.4.3 Visualizing time dependent data

The User needs a qualitative view of salinity level variations over time for the entire bay area.

1. *Setup the view*: the user adds a data layer covering the entire bay area, disables currents visualization and sets a medium detail level.
2. *Manipulate the timeline*: the user manually advances the time offset, moving the timeline slider near the half of the timeline. The view is reloaded, showing salinity data for the specified time step.
3. *Compare data snapshots*: holding the `Ctrl` and `Shift` keys, the user clicks on the visualization and drags the mouse horizontally, obtaining a variable crossfade between the current visualization and the previous one. This allows the user to clearly identify the regions of the bay where salinity is changing. This feature can be used whenever a visualization is updated, and allows the user to quickly compare the new visualization with the previous one.

6.4.4 Using depth slices

After accomplishing task 6.4.3, the user wants to get more detailed information about the region were salinity has the greatest variation over time. In particular, he wants to visualize the differences in salinity levels at different depths.

1. *Setup the view*: the user defines region of the bay he wants to analyze on the map panel. The user deletes the previous data layer, and creates a new one whose region will au-

tomatically correspond to the currently visualized area. The user could also manually modify the region of interest of the previous layer, instead of creating a new one.

2. *Define a slice layer*: user manipulates the vertical slice height and detail parameters on the layer panel to create a set of "salinity columns" that show salinity information for each datapoint at various depths (figure 48, figures appendix). The layer detail can be regulated to limit the number of datapoints, and obtain clearer view.

3. *Get quantitative datapoint information*: the user enables datapoint selection, using on the fourth toolbar button from the left. Clicking on the view now lets the user select datapoints: the user can concentrate on a salinity column and click on it at various depths, to see the precise variations of salinity level.

6.4.5 Streamline visualization

The user needs to analyze the currents that flow through the bay. He wants to understand if water stream passing from several adjacent points tend to go in the same direction, or if they diverge. Additionally, he wants to see if those same flows change over time.

1. *Setup the view*: the user selects the region of the bay he wants to analyze, and adds a new data layer, with medium detail.

2. *Configure streamline visualization*: the user opens the streamline configuration panel, using the third button from the left in the upper toolbar. From the configuration panel, the user can decide the length of generated streamlines, and their detail level. The applica-

tion will warn the user if the specified parameters may lead to long streamline computations.

3. *Create streamlines*: the user enables datapoint selection (see section 6.4.4), and clicks on one point he wants to be the source of a streamline. On the context menu, the user chooses the `Streamline from here` option. This operation is repeated for each of the points the user wants as streamline sources. The result is shown in figure 46 and 47, figures appendix.

4. *Compare data snapshots*: the user can now manipulate the timeline, to see how currents starting from the selected points vary at different time offsets. It is possible to compare different time snapshots using the same technique applied in section 6.4.1.

6.4.6 Collaborative visualization

Two users want to setup a collaborative visualization session. In particular, a user wants to share his local view with another user and get feedback from him. Additionally, the first user wants to have his viewpoint controlled by the other client, so that both users see data from the same exact position.

1. *Login*: each user opens an application session inside his browser, and logs on using a recognisable user name. After logging in, the users can communicate through the chat panel to coordinate the collaborative session.

2. *Setup the view*: the first user creates the desired visualization, manipulating data layers and streamlines as explained in previous scenarios.

3. *Share the view*: the first user opens the workspace panel on the left side of the screen, and clicks on the `Add to workspace` button. A thumbnail of his current view appears in the workspace panel of all users. The second user can click on the view thumbnail and select the `Get view` context menu option to load the view in his local canvas.

4. *Manipulate the shared view*: update filters are configured by default to broadcast view updates (as indicated by the second button from the left on the upper toolbar). Each modification done on the visualization by any of the two users is sent to the other. The users can now work together on the visualization content, while communicating through the chat panel.

5. *Add an annotation*: the second user wants to attach an annotation to a specific datapoint. He enables datapoint selection, selects a layer and clicks on a datapoint. On the context menu he chooses the `Add annotation` option. An annotation appears inside the visualization of all users connected to the view, linked to the specified datapoint. The user can now add some line of text to the annotation, and click on the `Save` button to broadcast the annotation update. Any user can modify the annotation content, not only the one who created it.

6. *Synchronize the viewpoint*: the first user wants the second one to get control of his viewpoint. He clicks on the viewpoint settings button (the most-left button on the toolbar) until the **Input only** option is selected (represented by a green arrow pointing to the magnifier icon). Each time the second user changes his viewpoint, the update will be applied to first user local viewpoint too.

6.5 Summary

This chapter described the implementation and functionalities of Hydroviz, a web application aimed at collaborative visualization of hydrographic data. The application has been developed to test the potential of the supporting framework and of the flash technology on a real visualization scenario. The application design followed the criteria introduced in section 3.1 (namely: ease of use, 3d visualization and collaboration support, enjoyement). To evaluate the application, a set of interaction scenarios has been defined: these scenarios cover almost every aspect of the interaction between a user and the application. The concretization of these scenarios (i.e. the actual set of operations users has to perform to achieve the corresponding task) can be used as a first qualitative hint of the effectiveness of the application in respecting the identified requirements. The scenarios show how the majority of the tasks can be completed through simple operations, that require a limited training of end users.

The following chapter will present an evaluation of the overall work developed in this thesis.

CHAPTER 7

CONCLUSIONS

On previous chapters, the basic requirements for a web-based visualization system were presented and discussed. The choice of flash among other technologies was motivated mainly by its ease of use and power to create interactive and rich web applications. Diffusion of the technology also played an important role in the choice. A simple collaborative data visualization framework has been designed. The visualization pipeline has been structured to respect the standard Filter-Map-View visualization model. Architectural, as well as technological choices have been made to ease the implementation of the data access and collaborative components of the framework. The framework prototype, named QbViz, has been implemented as a set of ActionScript classes. The framework and has been used to develop Hydroviz, a collaborative web application to display salinity and currents data for the Corpus Christi Bay area. This chapter will present some final considerations about the overall work presented in this thesis, evaluating both the technological choice (using Flash to develop a web-based visualization application) and the actual developed system.

7.1 Evaluation

As explained in section 1.4, it is significant to evaluate the effectiveness of Flash, and of the developed visualization framework, using the set of requirements that have been identified for a web-based visualization system targeted at common users. These requirements were

presented in section 3.1. Each of the requirements will be recalled here to comment on the respective, specific aspect of the developed system.

7.1.1 Simple Access

Requirement: *the application should avoid the need for the user to install custom or third party browser extensions. Widespread and trusted technologies should be used whenever possible*.

Flash, being the most diffuse browser extension, clearly satisfies this requirement. Applications and content developed in Flash are seamlessy downloaded and executed in a trusted environment on user machines. Flash players releases are always backwards compatible with previous versions, and when content needs a player version newer than the one installed on the user machine, the user is conveniently guided through the update process, which usually takes just some minute. Also, flash executable (.swf) files use efficient compression techniques of both data and code. In the case of Hydroviz, the complete application is made up of a single .swf file of about 600Kb. This guarantees short loading times for the application.

7.1.2 Ease of use

Requirement: *the application should offer an easy to use interface, one that an untrained user can learn by himself with little or no assistance*.

The evaluation of this requirement was based mainly on user scenarios presented in section 6.4. The scenarios show how several functionalities of the applications, like model exploration, data analysis and collaborative work, can be managed through simple end user interaction. The user can learn how the interface works after a very short training. Also, the interface itself

offers serveral hints (i.e. tooltips associated to the most important user interface elements) that further guide the user in the interaction.

It is worth noticing how the positive evaluation of this requirement is not directly related to the technological choice of Flash as a foundation platform. This was rather a result of design of the application itself. Still it is important to underline how the technology *efficiently supported* the design choices, allowing for fast prototyping and tweaking of the interface through the FLEX framework.

7.1.3 Collaboration

Requirement: *the application should offer some form of collaborative visualization support, either synchronous or asynchronous*

Collaboration support has been integrated directly into the design of the conceptual visualization framework described in chapter 4. The application itself did not have to directly take care of any of the collaboration mechanics, since they were already part of the framework. The visualizers developed for the application were automatically collaboration-aware. The only part of the application code devoted to collaboration support was associated to user interface features, i.e. the implementation of the workspace and chat panels. As for the "ease of use" requirement, even in this case the platform choice is not directly relevant in the requirement evaluation, but the choice of a technology that offers an advanced but easy to use communication layer gives a strong advantage in developing collaborative features. Having collaboration as part of the framework kernel, together with the simplicity of the platform, led to a significant reduction of the developement effort, as it will be underlined in section 7.2

7.1.4 3D visualization

Requirement: *Rendering and manipulation of mildly complex 3D visualizations should be supported*.

It has been already underlined how the main limit of the presented implementation is its lack of hardware accelerated 3d support. The use of impostors during interactive manipulation of the view was a mandatory choice to guarantee application scalability: nonetheless, impostors proved to be intuitive enough to guarantee an easy interaction with the visualization. Another problem related to the lack of accelerated 3d arisen with the need to visualize time dependent data: rendering animated 3d models at acceptable frame rates would have been impossible. The only viable solution would be to perform on-demand prerendering of several views, and give the user the possibility to play back the generated images. This has been partially done adding support for snapshot comparison (as seen in user scenarios 6.4.1 and 6.4.5).

7.1.5 Enjoyment

Requirement: *the application should be functional, but also offer a rich interaction experience to users*.

Objectively evaluating this requirement is difficult, since the interaction experience offered by an application is difficult to assess by a simple inspection of the application itself. A detailed user testing and validation phase is foundamental to better define the success of the application in respecting this requirement: thorough user testing will be a high priority in future work on Hydroviz.

Area	Development time	Classes	Lines of code
QbViz client	40 days	40	3100
QbViz server	20 days	20	1000
Hydroviz client	30 days	20	3600
Hydroviz server	10 days	3	400
Hydroviz tools	10 days	4	700

IV

QBVIZ AND HYDROVIZ DEVELOPMENT STATISTICS.

Yet, It is important to note that the graphic experience of the current version of Hydroviz (i.e. disappearing panels, animated context menus, crossfade effects and so on) was developed with no significative difficulty: once defined the interface design, its implementation was straightforward. Also, many aspects of the interface can be customized through a simple style sheet, allowing designers to fine tune the application graphic style, without any coding effort.

The previous consideration underlines that, even if user testing would suggest some change in the application interactive experience, these changes would likely be easy to implement thanks to the chosen platform.

7.2 Development effort

Table IV shows the development effort and the amount of implemented code, for both the framework prototype and the test case application.

The final framework prototype is quite compact, consisting of 60 classes for about 4000 lines of code. Once the framework prototype was complete implementation of the actual Hydroviz application revealed to be an almost trivial task. Much of the application development

time was spent on visualizer implementation. This was necessary since the framework did not offer any visualizer template, and all the used visualizers had to be developed from scratch. The developed visualizers can be easily adapted for generic usage. The bathymetry visualizer for instance can be turned into a generic heightmap visualizer.

After developing a visualizer library on top of the framework, application development will tend to get more and more easier, requiring also less time. Development of the application visual interface was straightforward, thanks to the Flex framework. Visual interface features can also be modified and fine-tuned very easily using a standard style sheet.

7.3 Platform Choice

The presented evaluation showed how the choice of Flash to develop a web-based collaborative visualization application it not only possible, but also pretty effective in several aspects: these aspects involve both the end user and the developer of the application. Some of the advantages offered by the flash platform could be obtained even through other technologies, but paying a significative tradeoff in the developement effort.

It has also been underlined how flash presents important limitations when used for 3D visualization. Due to these limitations it is clear how Flash *does not completely replace* other technologies for scientific visualization over the web, but it represents a possible alternative that is worth considering before starting developement. The final choice depends on the exact visualization scenario that is being addressed.

This thesis suggested a possible guideline for this choice: if the visualization system does not deal with complex 3D visualization (in this regard, animation is a major source of com-

plexity), and is targeted at common users that need quick access to the application and may be pleased by additional graphic features, then Flash may be most effective choice.

7.4 Future Work

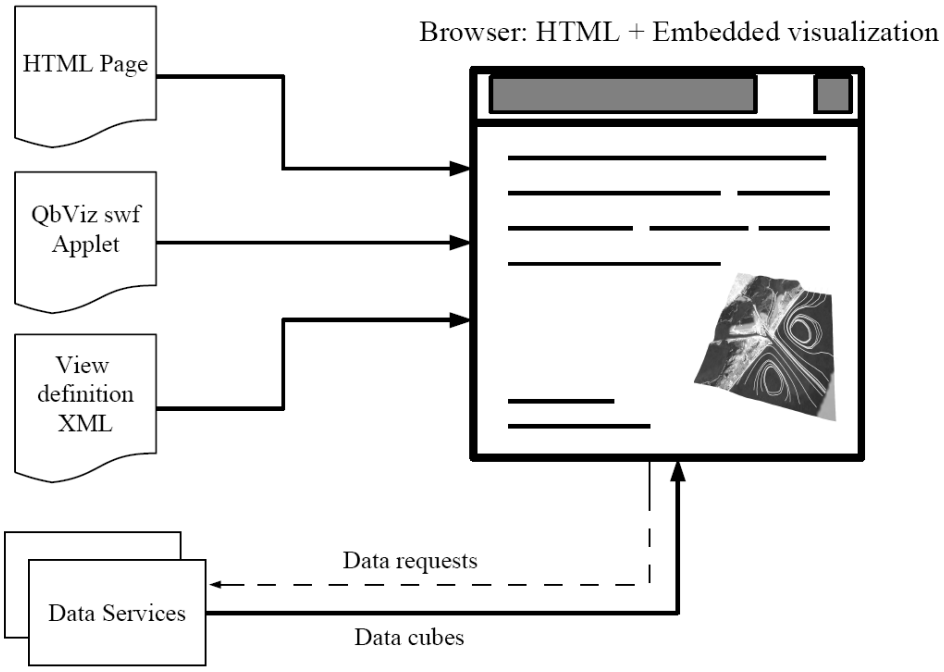
As already underlined, creating a basic visualizer library for QbViz would represent an important addition to the framework: the availability of visualizer templates would increase the ease of development of end user applications.

Another helpful addition would be the development of a generic *view builder* web application, which allows the users to connect to custom data services and use the available visualizers to create simple visualizations without the need of developing a custom application. Some effort in this direction has already been spent during the first iterations of framework development.

View definitions could easily support serialization and deserialization to a custom XML-based format. This would allow to have visualizations persistently saved on a web server, and then accessed by a QbViz applet through a HTTP request to embed them in a standard html page (figure 36).

7.4.1 Hydroviz

An important improvement of the Hydroviz application is represented by the support of data playback: the user should be able to add one or more data layers to the visualization, and animate them to see how they vary over time. Since real-time animation is not possible due to data transfer and software rendering limits. A solution could be on-demand animation generation. The user would specify the time interval and detail of the playback sequence: a



36. Embedding QbViz visualizations in a HTML document.

preprocessing step would obtain the required data and generate the set of images that can then be played back by the user. Viewpoint changes would be of course not possible during playback. Changing the viewpoint could automatically trigger a re-rendering step of all the animation frames: without the need of reloading the data this operation can be performed significantly faster.

Another improvement of Hydroviz would be the addition of salinity isosurface visualization: isosurfaces would allow for the easy identifications of 3d regions where salinity levels are constant and equal to a user-specified value.

As suggested in section 7.1.5, a detailed user testing and validation phase of the Hydroviz application could be an additional source of improvements and interface refactoring hints.

APPENDICES

Appendix A - Heightmap generation from sparse sample points

This appendix will illustrate the algorithm that generates the heightmap image used by the Hydroviz bathymetry visualizer described in section 6.3.1. The heightmap generation algorithm is the main component of the `HeightmapGen` tool of the Hydroviz software package.

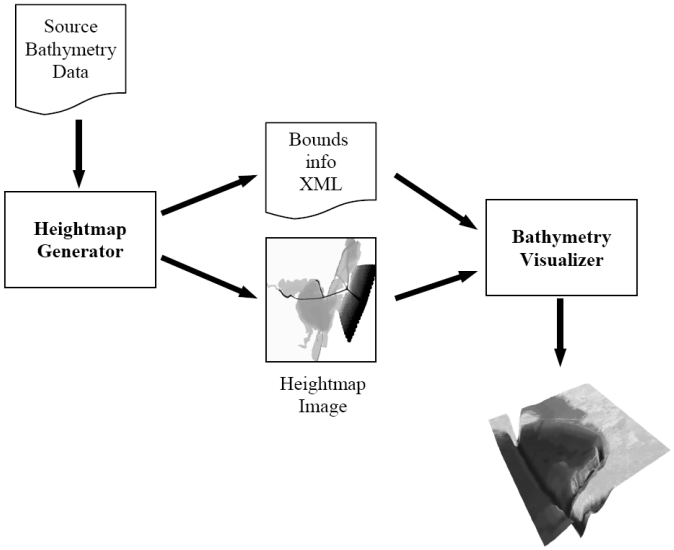
.1 Overview

Figure 37 shows the data transformation pipeline used for bathymetry information inside hydroviz. The `HeightmapGen` tool converts source bathymetry information to a heightmap image and a *heightmap descriptor*. The heightmap descriptor contains information (UTM latitude and longitude boundaries, bay depth maximum and minimum value) that is used in conjunction with the heightmap by the Hydroviz application to generate a qualitative 3D model of the bay.

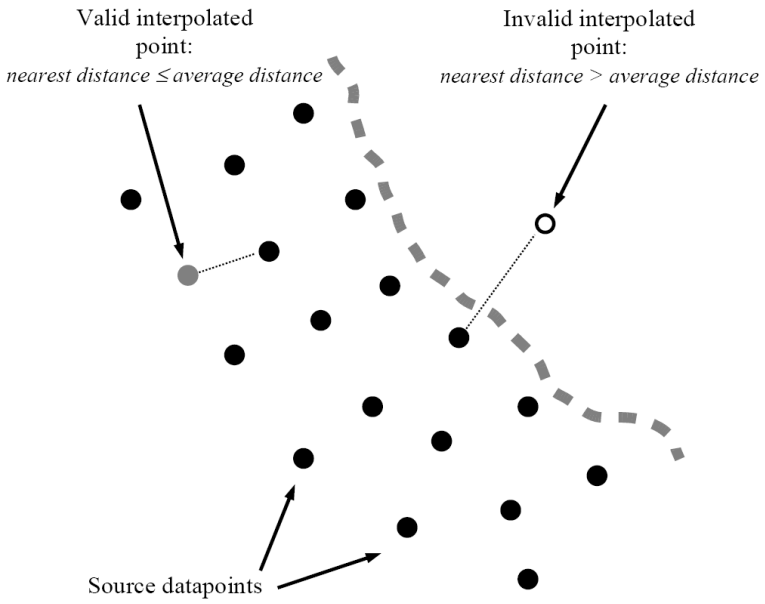
The source information is the set of bathymetry sampling points that are part of the Corpus Christi bay dataset. Source information is *nonomogeneous*: the distribution of sampling point on the bay does not correspond to a regular grid. Generating a heightmap basically means to map depth information to a dense, regular grid where each grid point corresponds to a pixel in the destination heightmap image, and then apply a reversible color mapping to convert the numerical values to actual pixel color values.

In the case of nonomogeneous data a heightmap generation algorithm should be able to:

- Interpolate depth information for points that are not covered by the source data.
- Identify areas that are not part of the sampling space (in the specific case in exam, these are represented by islands, and anything else outside of the actual water-covered area



37. The heightmap transformation pipeline.



38. Identification of the valid sampling area.

of the bay). Points inside these areas are not interpolated, and should be marked in the resulting heightmap (i.e. making them transparent). This is an important operation, since doing a simple interpolation for areas outside of the real sampling space would lead to non-significative heightmap values that would then result in a partially incorrect 3d model representation.

.2 Algorithm

Algorithm 1 HeightmapGen($Dataset, width, height, \alpha$)

1: $heightMap =$ EmptyHeightmap($width, height$)
2: **for all** $< lat, lon, depth >$ in $Dataset$ **do**
3: $< x, y, color >=$ Map($lat, lon, depth$)
4: **if** $heightMap(x, y) = \oslash$ **then**
5: $heightMap(x, y) = color$
6: **else**
7: **return** $\oslash$ {failure: two samples are mapped to the same heightmap point}
8: **end if**
9: **end for**
10: **for all** $< x, y >$ in $heightMap$ **do**
11: **if** $heightMap(x, y) = \oslash$ **then**
12: $heightMap(x, y) =$ InterpolateSample($x, y, heightMap, \alpha$)
13: **end if**
14: **end for**
15: **return** $heightMap$

The heightmap generation procedure, summarized in algorithm 1, is divided in two steps. Each bathymetry sample in the source dataset, containing information about latitude and longitude of a datapoint and the corresponding bay depth, is converted to *image space*. UTM

Algorithm 2 InterpolateSample$(x, y, heightMap, \alpha)$

```
1: samples = NearSamples(x, y, heightMap)
2: if samples ≠ ∅ then
3:     nearDistance = NearestSample(x, y, samples)
4:     meanDistance = MeanSampleDistance(samples)
5:     if nearDistance * α < meanDistance then
6:         return InterpolatedColor(x, y, samples)
7:     end if
8: end if
9: return ∅
```

coordinates are converted to (x, y) pixel positions in the heightmap, while the depth value is converted to a grayscale color. This mapping depends on the width and height of the destination heightmap, and on the number of grayscale color levels available for depth mapping. The mapping is performed by the Map function used in the first half of algorithm 1.

The second step of the algorithm performs interpolation for points inside the valid sampled area, and is entirely done in image space. The InterpolateSample function illustrated in algorithm 2 executes the interpolation for a specified (x, y) point of the heightmap. The actual interpolated color value is calculated by the InterpolatedColor function. This function can be implemented as a weighted average of near samples, or may just return the color of the nearest valid sample. InterpolateSample also performs the identification of points inside and outside of the valid sample area, using the intuitive yet functional criterion illustrated in figure 38. A point is considered inside the sample area (thus its value will be interpolated) if its distance from the nearest sample point is not significantly bigger that the average distance between local sample points. The α constant in algorithm 2 determines the acceptance threshold for

valid points, and has to be manually tuned depending on heightmap size and source dataset distribution. For the Corpus Christi bay dataset, α was set to about 0.6.

Appendix B - Source code

This appendix presents an excerpt from the Hydroviz application source code. The reported code implements the *layer visualizer* described in section 6.3.2. The layer visualizer source is made up of two distinct classes: LayerVisualizer, which inherits the Visualizer base QbViz class, and LayerDataPoint, which contains the draw logic for a single layer datapoint.

.1 LayerVisualizer source

```
public class LayerVisualizer extends Visualizer
{
  // Visualizer Class ID
    public static const ID: String = "Layer";

    private var myLayer: Sprite3DContainer;

    private var myX: Filter;
    private var myY: Filter;
    private var myScX: Number;
    private var myScY: Number;

/*
* Class constructor.
*/
    public function LayerVisualizer(info: VisualizerInfo): void
    {
        super(info);
    }

/*
* Perform a complete data layer visualizer update.
*/
    public override function update(): Boolean
    {
        var prof: Number = getTimer();

        clear();

        var roiCube: DataCube = canvas.cubes["ROI"];
        var fx: Filter = roiCube.info.getFilter("x");
        var fy: Filter = roiCube.info.getFilter("y");
        var scx: Number = (fx.max - fx.min);
        var scy: Number = (fy.max - fy.min);
```

```
        myX = fx;
        myY = fy;
        myScX = scx;
        myScY = scy;

myLayer = new Sprite3DContainer();
        myLayer.name = info.name;

        minL = 10000000;
        maxL = 0;
        minS = 10000000;
        maxS = 0;

        // Retrieve element offsets.
        var offsX: int = cube.dataBlock.getElementOffsetByIndex(0);
        var offsY: int = cube.dataBlock.getElementOffsetByIndex(1);
        var offsZ: int = cube.dataBlock.getElementOffsetByIndex(2);
        var offsSalt: int = cube.dataBlock.getElementOffsetByIndex(3);
        var offsVX: int = cube.dataBlock.getElementOffsetByIndex(4);
        var offsVY: int = cube.dataBlock.getElementOffsetByIndex(5);

        for(var i: int = 0; i < cube.dataBlock.length; i++)
        {
            var item: Object =
            {
                x: cube.dataBlock.getFloat(i, offsX),
                y: cube.dataBlock.getFloat(i, offsY),
                z: cube.dataBlock.getFloat(i, offsZ),
                salt: cube.dataBlock.getFloat(i, offsSalt),
                vx: cube.dataBlock.getFloat(i, offsVX),
                vy: cube.dataBlock.getFloat(i, offsVY)
            };

            var dataPoint: LayerDataPoint = new LayerDataPoint(
             this, item.x, item.y, -item.z);

            var x: Number = ((item.x - fx.min) / scx) - 0.5;
            var y: Number = ((item.y - fy.max) / scy) + 0.5;

            if(x >= -0.5 && x <= 0.5 && y >= -0.5 && y <= 0.5)
            {
                var l: Number  = Math.sqrt(
                 item.vx * item.vx + item.vy * item.vy);
                var dx: Number = item.vx / l * 0.02;
                var dy: Number = item.vy / l * 0.02;
```

```
            if(l > maxL)
            {
                maxL = l;
            }
            if(l < minL)
            {
                minL = l;
            }

            if(item.salt > maxS)
            {
                maxS = item.salt;
            }
            if(item.salt < minS)
            {
                minS = item.salt;
            }

            dataPoint.speed = l;
            dataPoint.salt = item.salt;
            dataPoint.vertices.push(
             new Vertex3D(x, y, -item.z));

            dataPoint.vertices.push(
             new Vertex3D(x + dx, y + dy, -item.z));

            myLayer.addSprite(dataPoint);
        }
    }

    var ds: Number = maxL - minL;
    var dsalt: Number = maxS - minS;
    for each(var dp: LayerDataPoint in myLayer.billboards)
    {
        var w: Number = (dp.speed - minL) / ds;
        var r: int = w * 255;
        var g: int = 255 - r;
        var b: int = 0;
        dp.color = r << 16 | g << 8 | b;

        w = (dp.salt - minS) / dsalt;
        r = 0;
        g = /*w **/ 255;
        b = 255 - (w * 255);
        dp.saltColor = r << 16 | g << 8 | b;
    }
```

```
            resetMetrics();
            resetAlpha();
            resetDrawFlags();

            canvas.rootNode.addChild(myLayer);

            //super.update();
            return true;
        }

    /*
    * Clear the visualization managed by this visualizer.
    */
        public override function clear(): void
        {
            if(myLayer != null)
            {
                canvas.rootNode.removeChild(myLayer);
            }
        }

    /*
    * Show the visualization managed by this visualizer.
    */
        public override function show(): void
        {
            if(myLayer != null)
            {
                myLayer.visible = true;
            }
        }

    /*
    * Hide the visualization managed by this visualizer.
    */
        public override function hide(): void
        {
            if(myLayer != null)
            {
                myLayer.visible = false;
            }
        }

    /*
    * Called when impostor visualization has been enabled by the canavas.
    */
        public override function enableImpostors(): void
```

```
    {
     // When impostors are enabled, do not draw anything.
        hide();
    }

/*
* Called when impostor visualization has been disabled by the canavas.
*/
    public override function disableImpostors(): void
    {
        show();
    }

/*
* Called when metrics settings have been modified.
*/
    public override function resetMetrics(): Boolean
    {
        if(myLayer != null)
        {
            var roiCube: DataCube = canvas.cubes["ROI"];
            var x: Filter = roiCube.info.getFilter("x");
            var y: Filter = roiCube.info.getFilter("y");

            var szx: Number = (x.max - x.min);
            var szy: Number = (y.max - y.min);

            myLayer.scaleZ =
             canvas.view.computeMetric("z meters").multiplier;
            myLayer.scaleX =
             szx * canvas.view.computeMetric("x utm").multiplier;
            myLayer.scaleY =
             szy * canvas.view.computeMetric("y utm").multiplier;
        }
        return true;
    }

/*
* Called when visualizer options have been modified.
*/
    public function resetDrawFlags(): void
    {
        alpha = Number(info.getProperty("alpha", 1));
        drawSalt = Boolean(info.getProperty("drawSalt", true));
        drawCurrents = Boolean(info.getProperty("drawCurrents", true));
    }
```

```
    public var alpha: Number;
    public var drawSalt: Boolean;
    public var drawCurrents: Boolean;
    public var minL: Number;
    public var maxL: Number;
    public var maxS: Number;
    public var minS: Number;
}
```

.2 LayerDataPoint source

```
public class LayerDataPoint extends Sprite3D
{
  private var myLayer: LayerVisualizer;

/*
* Class constructor.
*/
public function LayerDataPoint(
layer: LayerVisualizer, x: Number, y: Number, z: Number): void
{
          myLayer = layer;
          this.x = x;
          this.y = y;
          this.z = z;
  }

/*
* Draw the datapoint managed by this LayerDataPoint instance.
*/
public override function draw(container:Sprite): void
{
          var gfx: Graphics = container.graphics;

          var v1: Vertex2D = vertices[0].vertex2DInstance;
          var v2: Vertex2D = vertices[1].vertex2DInstance;

          if(myLayer.drawCurrents)
          {
              gfx.lineStyle(2, color, myLayer.alpha);
              gfx.moveTo(v1.x, v1.y);
              gfx.lineTo(v2.x, v2.y);
              gfx.moveTo(0, 0);
          }
          if(myLayer.drawSalt)
          {
              gfx.lineStyle(1, saltColor, myLayer.alpha / 2);
              gfx.beginFill(saltColor, myLayer.alpha);
              gfx.drawCircle(v1.x, v1.y, 3);
              gfx.endFill();
          }
          else
          {
              gfx.lineStyle(1, 0xffffff, myLayer.alpha / 2);
              gfx.beginFill(0xffffff, myLayer.alpha);
```

```
                gfx.drawCircle(v1.x, v1.y, 2);
                gfx.endFill();
            }
            gfx.lineStyle(0, 0, 0);
    }

    public var color: uint;
    public var saltColor: uint;
    public var speed: Number;
    public var salt: Number;
    public var x: Number;
    public var y: Number;
    public var z: Number;
}
```

Appendix C - Hydroviz server installation guide

This appendix provides a step-by-step guide on how to compile Hydroviz client and server source code, and how to setup a host machine for Hydroviz server components.

.1 Requirements

In order to setup a complete hydroviz host on a Windows machine the following components are needed:

1. A web server supporting ASP.Net application deployment. Microsoft IIS 6 server is a suggested choice and this guide specifically targets it, but other web servers may work as well.
2. An installed database management system, like MySQL. The database and web servers can be hosted on different machines, as long as the database host is accessible from the web server.
3. The complete Hydroviz / QbViz software package. The package contains both source code and precompiled versions of Hydroviz and QbViz components, data conversion tools and remoting gateway service components.
4. The Corpus Christi bay source dataset.
5. The Flash software development kit (SDK). This is not needed when using the precompiled version of the Hydroviz client.
6. The .Net Framework SDK version 2.0 or higher. The installation of a complete C# development environment is highly suggested. This is not needed when using the precompiled versions of QbViz and Hydroviz server components.

In following sections, all file and directory paths will be represented as relative to the local Hydroviz software package installation directory. For instance, if the software package is locally placed under `C:/Hydroviz` the path `/Client/File.as` will refer to `C:/Hydroviz/Client/File.as`

.2 Software package structure

The Hydroviz / QbViz software package is divided in several folders. What follows is a brief description of their content:

- `/HeightmapGen` contains the source code and binaries of the heightmap generation tool.
- `/DBUpdater` contains the source code and binaries of the database updater tool.
- `/DBEtc` contains utility files used to setup the DBMS.
- `/Client` contains the sources of the QbViz framework client component, and of its dependencies:
  - `/Client/Papervision` contains the sources of the Papervision library.
  - `/Client/QbViz` contains the actual ActionScript framework source.
- `/Server` contains the sources of the QbViz framework server component.
- `/Hydroviz` contains the sources and precompiled versions of Hydroviz:
  - `/Hydroviz/Client` contains the Hydroviz client code and the compiled .swf file.
  - `/Hydroviz/Server` contains the Hydroviz server code and binaries.
  - `/www` is the web application hosting root, and contains the remoting gateway service files.

.3 Compilation

.3.1 Server code

Compilation of server code is straightforward when a C# IDE, like Visual Studio 2005 or Visual C# Express is available. It is enough to open the `/BuildAll.sln` solution file inside the environment, and choose the `Rebuild Solution` command. Tool binaries will be put under the `bin` folder under their

respective source directory. QbViz and Hydroviz binaries (namely, `QbViz.dll` and `Hydroviz.dll`) will be automatically deployed in the web application binaries folder (`/www/bin`).

.3.2 Client Code

To compile the Hydroviz client code it is usually enough to slightly modify and run the batch file `/Hydroviz/Client/Build.bat`. The modification of the batch file consists in setting the `MXMLC` variable to point to the local flash compiler `mxmlc.exe`. For instance, if the Flash SDK is installed under `C:/Flash` the variable should be set to `C:/Flash/bin/mxmlc.exe`. A succesfull compilation will generate the executable flash file `Main.swf` and place it under `/www/Hydroviz`.

.4 Preprocessing data

After setting up a database system, the proper dataset and tables must be created and filled with processed source data. The `DBEtc` folder contains a sql script containing the necessary commands to setup the empty tables. How to run this script on a specific DBMS installation depends on the DBMS software itself. Database systems usually offer a SQL console or management user interface that will guide he user in this task.

Once the tables have been set up, they can be filled with data, using the DBUpdater tool. The tool syntax is:

```
DBUpdater <dataRoot> <DBMSaddr> <DBMSuser> <DBMSpswd>
```

where:

- **dataRoot** is the path to the directory where source data text files are stored.
- **DBMSaddr** is the network address of the DBMS service, in the form `<IPAddress>:<port>`.
- **DBMSuser** is the name of a user with at least write privileges on the hydroviz dataset tables.
- **DBMSpwd** is the password for the user specified on previous point.

The HeightmapGen tool can be used to generate a heightmap image from source bathymetry data. Running this tool is not strictly necessary, since a pre-generated heightmap image is already placed in `/www/hydroviz`. The tool has the following syntax:

```
HeightmapGen <dataRoot> <imgDest>
```

where:

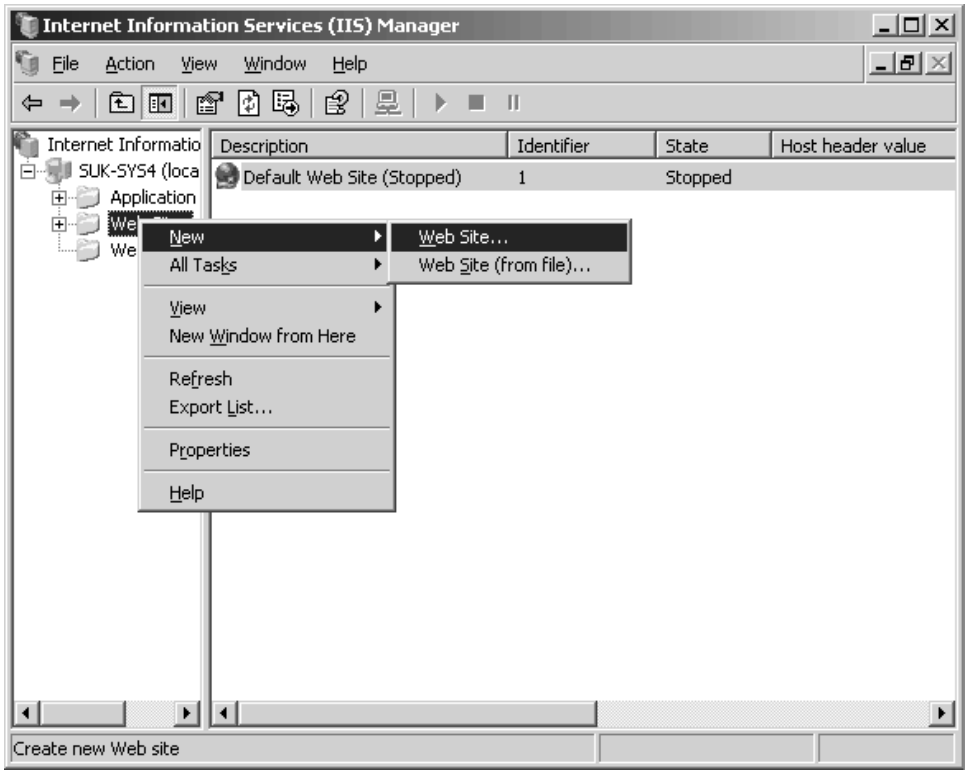
- **dataRoot** is the path to the directory where source data text files are stored.
- **imgDest** is destination path of generated image.

.5 Web server setup

Setting the web server to host the hydroviz application consists in the following two operations (a IIS 6 server installation is assumed):

1. Creating a new web site: this can be done selecting the "'Web Sites"' item in the left pane and choosing "'New Website"' from the context menu options as shown in figure 39. Server name and root directory can be freely defined by the user.
2. Adding a virtual application to the web site: done by selecting the newly created site and choosing the "'New application"' from the context menu. The application root must be set to the `/www` directory of the Hydroviz software package.

The last step consists in setting up the application configuration file, `/www/hydroviz/config.xml`. This file contains the address, user name and password that should be user by the hydroviz service to access data on the DBMS. The same values used for running the DBUpdater tool can be specified.



39. Hosting a web application on IIS.

After this step, the application is completely setup, and ready to be used. As an example, if the website name specified in web server setup is `MySite` and the virtual application name is `MyApp`, the hydroviz web application will be accessible at:

`http://machineURL/MySite/MyApp/hydroviz/iindex.html`.

BIBLIOGRAPHY

1. Francis T. Marchese, Jude Mercado, and Yi Pan: Adapting Single-User Visualization Software for Collaborative Use. Proceedings of the Seventh International Conference on Information Visualization (IV'03), 2003.

2. Pilar Herrero, and Angelica de Antonio: A Formal Awareness Model for 3D Web-based Collaborative Environments, SIGGROUP Bulletin, Vol 21, No.3, December 2000.

3. Rainer C. Splechtna, Anton L. Fuhrmann, and Rainer Wegenkittl: ARAS - Augmented Reality Aided Surgery System Description, http://www.vrvis.at/br1/aras/.

4. Jason Wood: COVISA, Collaborative Visualization and Scientific Analysis, Proceedings of Visualization, doi.ieeecomputersociety.org, 1996

5. Jason Wood, Helen Wright, and Ken Brodlie: CSCV - Computer Supported Collaborative Visualization, 1995

6. Alex Pang, Craig M. Wittenbrink, and Tom Goodman: CSpray: A Collaborative Scientific Visualization Application

7. G. Feichtinger, G. Fischel, E. Gröller, and A. Prskawetz: Despotism and Anarchy in Ancient China: Visualizing the Dynastic Cycle. Jahrbuch,Wirtschaftswissenschaften 47/1, Vandenhoeck & Ruprecht, 1-13, 1996.

8. Simon Su, R. Bowen Loftin, David T. Chen, Yung-Chin Fang, and Ching-Yao Lin: Distributed Collaborative Virtual Environment: PaulingWorld

9. Frank J. Post, Theo van Walsum, Frits H. Post, and Deborah Silver: Iconic techniques for feature visualization, Proceedings Visualization '95, IEEE Computer Society, 1995.

10. Thomas Nocke, Stefan Schlechtweg, and Heidrun Schumann: Icon-based visualization using mosaic metaphors, IEEE Information Visualization (IV'05), London, 2005

11. Millward Brown: Flash Player Penetration: Market Research, http://www.adobe.com/products/player_census/flashplayer/

12. Steve Casera, Hans-Heinrich Nägeli, and Peter Kropf: Improving Usability of Collaborative Visualization Systems, http://iiun.unine.ch/paral/zoomin, 2001

13. Russel M., and Taylor H.: Practical Scientific Visualization Examples, Computer Graphics, Vol. 34, No. 1., February 2000

14. Ken Brodlie, Jason Wood, and Helen Wright: Scientific Visualization: Some Novel Approaches to Learning, http://www.comp.leeds.ac.uk/vis/covisa/covisa.html

15. Alex Pang, and Kyle Smith: Spray rendering: Visualization using smart particles, Proceedings Visualization '93, pages 283-290, 1993.

16. Concurrent System Laboratory, University of California, Santa Cruz, USA: Spray and the REINAS Project, http://csl.cse.ucsc.edu/projects/reinas/

17. W. Schroeder, K. Martin, and B. Lorensen: The visualization toolkit: an object-oriented approach to 3D graphics, Prentice Hall, ISBN 0-13-199837-4, 1996

18. William J. Schroeder, Kenneth M. Martin, and William E. Lorensen: The design and implementation of an object-oriented toolkit for 3d graphics and visualization, Proceedings Visualization '96. IEEE Computer Society, 1996

19. Will Schroeder, Ken Martin, and Bill Lorensen: The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics, Prentice Hall, 1996.

20. Hervé Sanglard: Towards an Easy-to-learn and Extensible Platform for Scientific Visualization. PhD Thesis, University of Neuchâtel, Switzerland. 2001

21. Robert B. Haber, and David A. McNabb: Visualization idioms: A conceptual model for scientific visualization systems, Visualization in Scientific Computing. IEEE Computer Society Press, 1990.

22. T. J. Jankun Kelly: Visualizing Visualization: A Model and Framework for Visualization Exploration. PhD Thesis, University of California, Davis, 2003

23. Bahari Belaton, and Ken Brodlie: Model Centred Approach to Scientific Visualization, Journal of WSCG, 10(1), 2002

24. Bianchi Serique Meiguins, Rosevaldo Dias de Souza Júnior, Marcelo de Brito Garcia, and Aruanda Simões Gonçalves: Web-Based Collaborative 3D Information Visualization Tool, Proceedings of the Eighth International Conference on Information Visualisation (IV'04), 2004.

25. Ed Huai-hsin Chi, and John T. Riedl: An Operator Interaction Framework for Visualization Systems, 1998.

26. Andy Cedilnik, Berk Geveci, Kenneth Moreland, James Ahrens, and Jean Favre: Remote Large Data Visualization in the ParaView Framework, Eurographics Symposium on Parallel Graphics and Visualization, 2006.

27. Anton L. Fuhrmann: Scientific Visualization in Virtual Reality, ÖGAI Journal 21/1.

28. Zihiong Gao: Extending Scientific Visualization into the World Wide Web, Dalhouse University, Halifax, Nova Scotia, 1998.

29. Advanced Visual Systems Inc., Waltham, MA, USA: AVS, http://www.avs.com.

30. International Business Machine Inc., USA: Data Explorer, http://www.almaden.ibm.com/dx/.

31. The Numerical Algorithms Group Ltd., Oxford, UK: Iris Explorer, http://www.nag.co.uk.

32. Wolfram Research: Mathematica, http://www.wolfram.com/products/mathematica/index.html.

33. Adobe Systems Inc., San Jose, California, USA: Adobe Flash, http://en.wikipedia.org/wiki/Adobe_Flash.

34. Adobe Systems Inc., San Jose, California, USA: ActionScript 3.0 Language and Components Reference, http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/.

35. Adobe Systems Inc., San Jose, California, USA: Adobe Shockwave, http://en.wikipedia.org/wiki/Adobe_Shockwave

36. Adobe Systems Inc., San Jose, California, USA: Flex 3 Framework, http://www.adobe.com/products/flex/

37. Adobe Systems Inc., San Jose, California, USA: AMF3 Format Specification, http://download.macromedia.com/pub/labs/amf/amf3_spec_121207.pdf.

38. Microsoft Corporation, Redmond, Washington, USA: Silverlight, http://silverlight.net/.

39. Midnight Coders, Frisco, Texas, USA: WebOrb, Universal Connectivity Between AJAX, Flash, Flex Clients and .Net, Java Web Servers, http://www.themidnightcoders.com/weborb/.

40. Heightmap, http://en.wikipedia.org/wiki/Heightmap.

41. JOGL: Java bindings for OpenGL API with OpenGL 1.5 specification, https://jogl.dev.java.net/.

42. OpenGL, http://en.wikipedia.org/wiki/OpenGL.

43. JavaScript Language Specification, http://www.noc.garr.it/docum/jsspec.pdf.

44. Hypercosm, http://www.hypercosm.com/.

45. VirTools, http://www.virtools.com/.

46. Lingo, http://en.wikipedia.org/wiki/Lingo_(programming_language).

47. X3D Format Specification, http://www.web3d.org/x3d/specifications/x3d_specification.html.

48. The Java Virtual Machine Specification, http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html.

49. The MathWorks: Matlab and SIMULINK for technical computation, http://www.mathworks.com/.

50. Rendering(Computer Graphics), http://en.wikipedia.org/wiki/Rendering_(computer_graphics).

51. Virtual Reality, http://en.wikipedia.org/wiki/Virtual_reality.

52. Papervision 3D, http://code.google.com/p/papervision3d/.

53. Fluorine, an Open Source Flash Remoting Gateway, http://fluorine.thesilentgroup.com/fluorine/index.html.

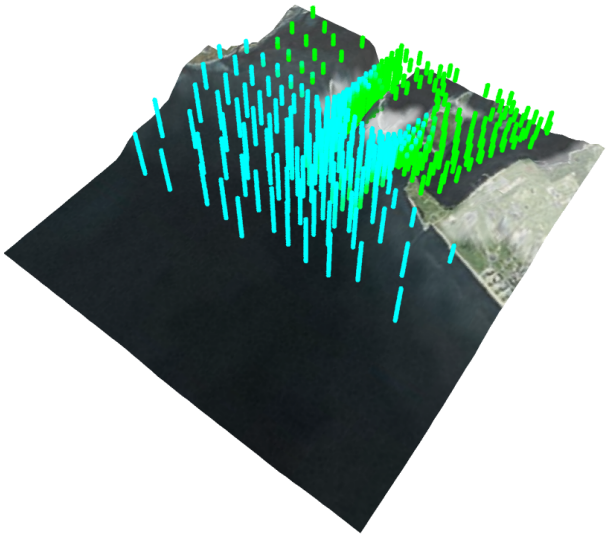
54. Swing, http://en.wikipedia.org/wiki/Swing_(Java).

55. Abstract Window Toolkit, http://en.wikipedia.org/wiki/Abstract_Window_Toolkit.

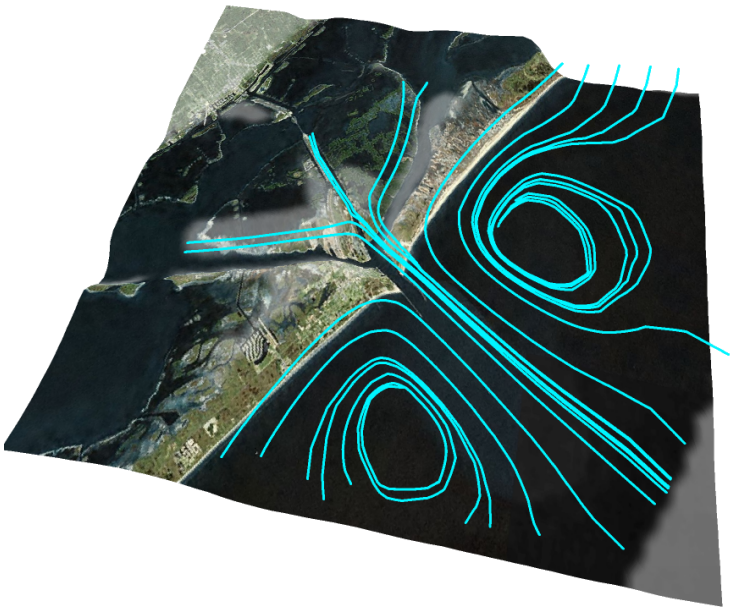
56. Painter's Algorithm, http://en.wikipedia.org/wiki/Painter's_algorithm.

57. Z-Buffering, http://en.wikipedia.org/wiki/Z-buffering.

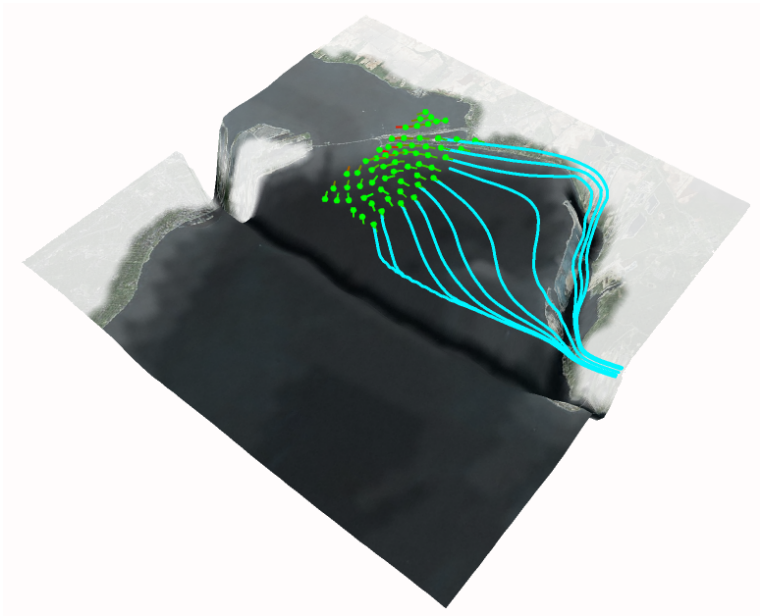
FIGURES



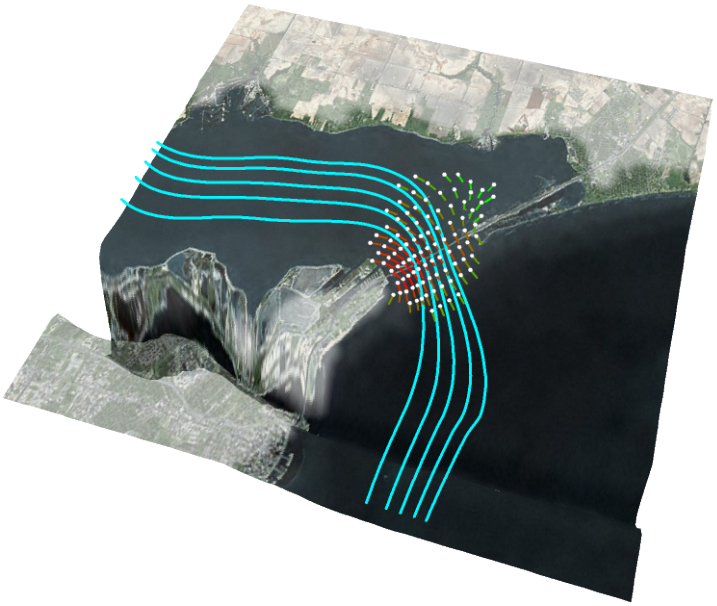
40. Depth slices



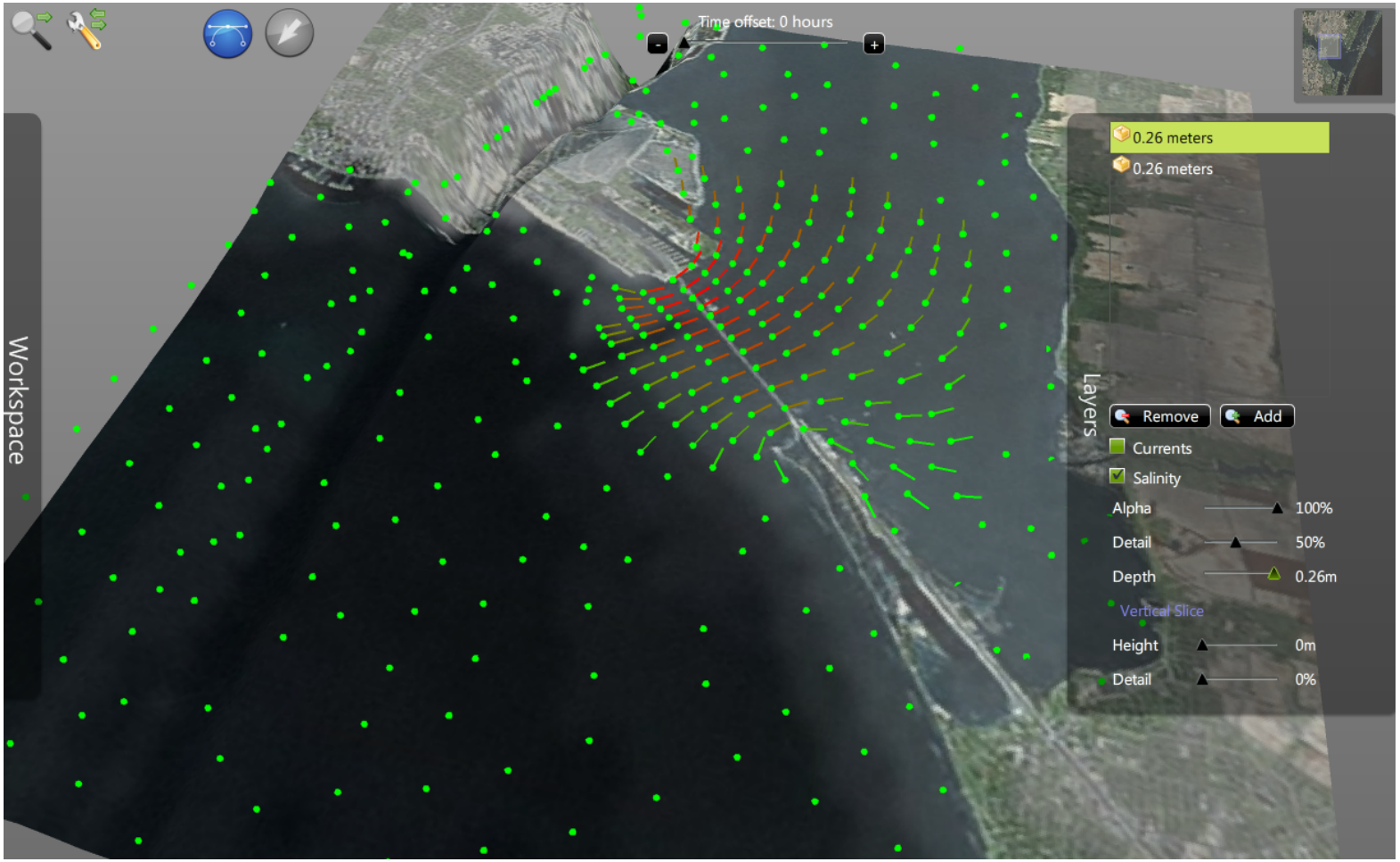
41. Streamlines



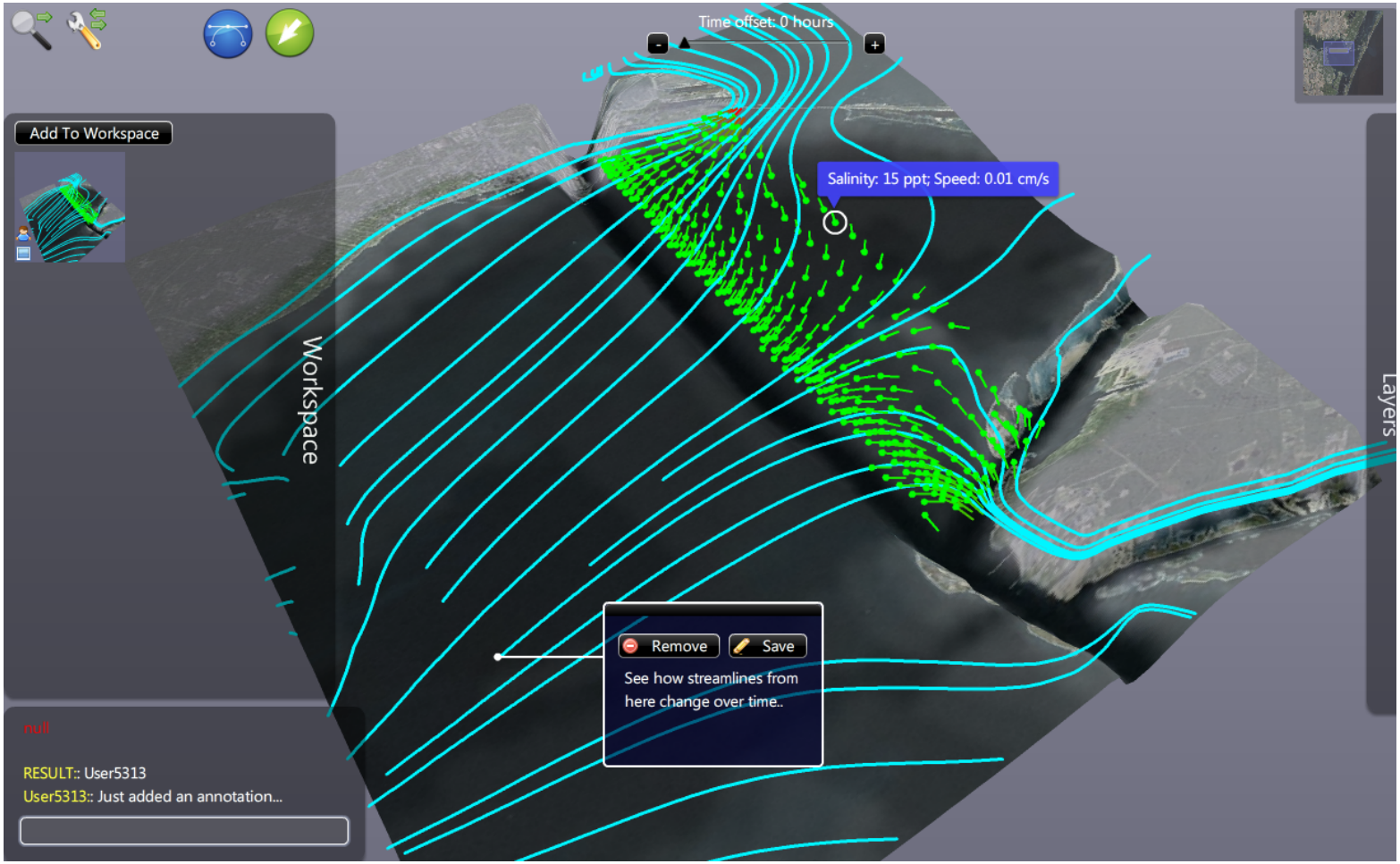
42. Mixed streamline and datapoint layer visualization



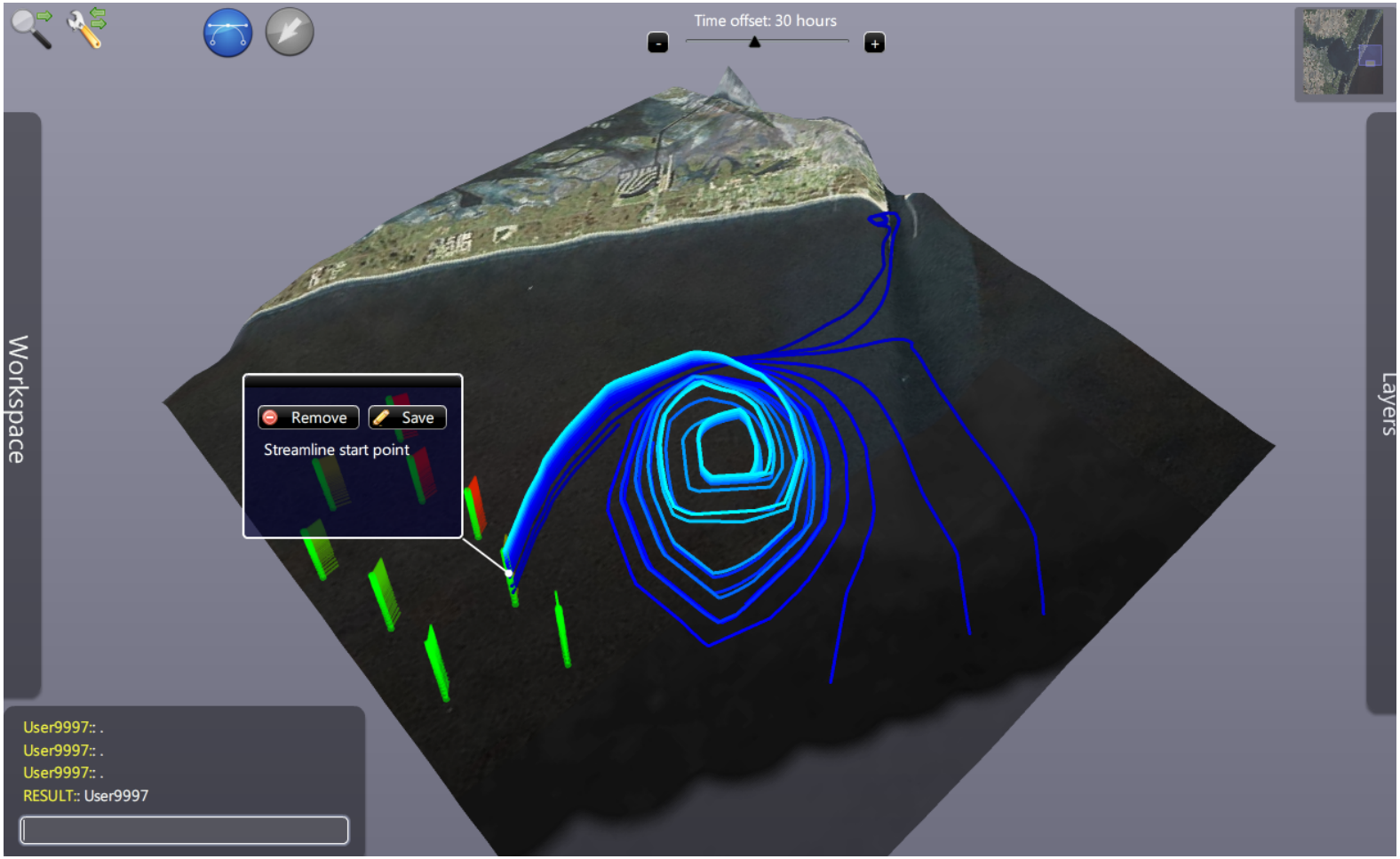
43. Mixed streamline and datapoint layer visualization



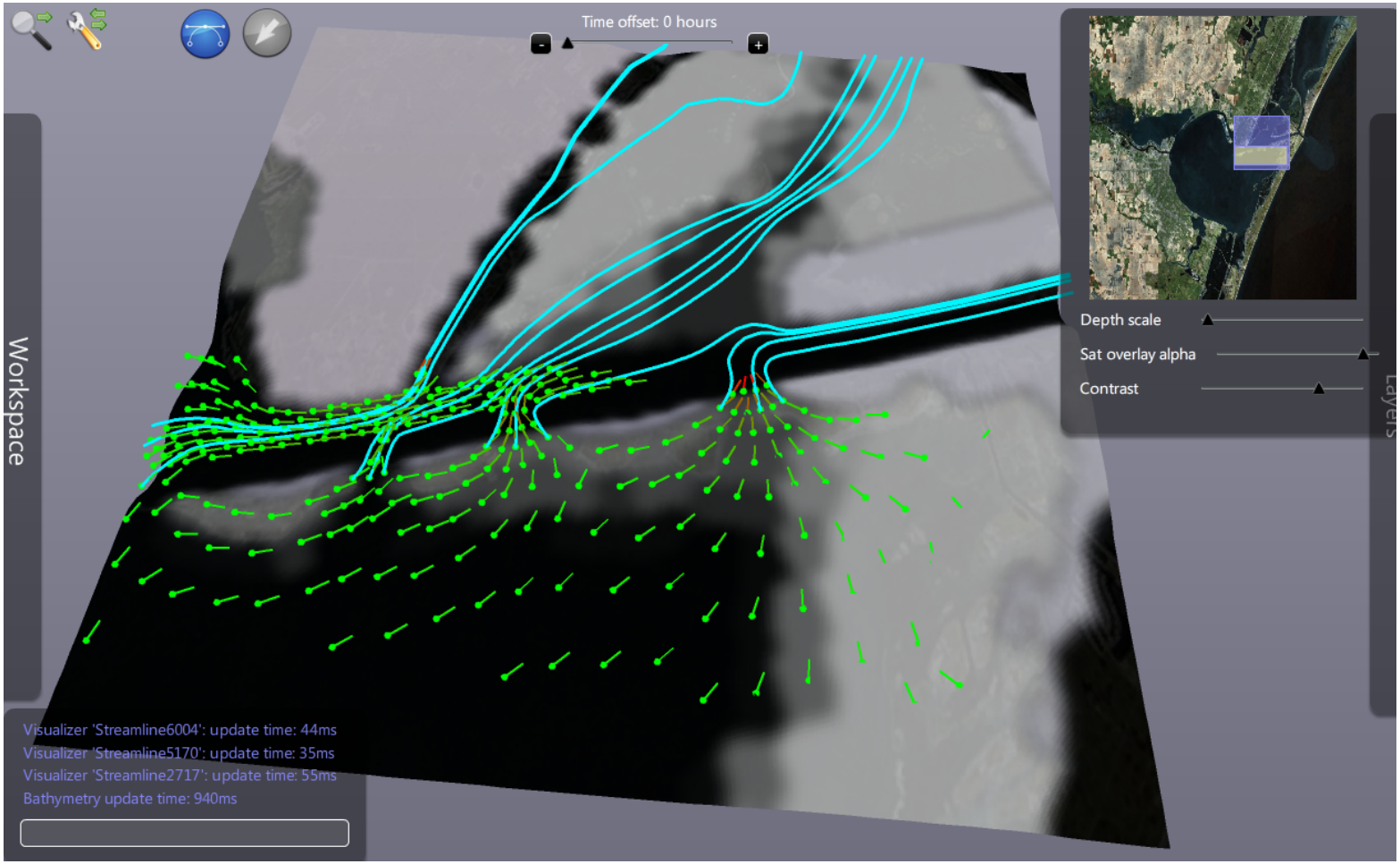
44. a screenshot of the Hydroviz application showing the bay model and two active data layers



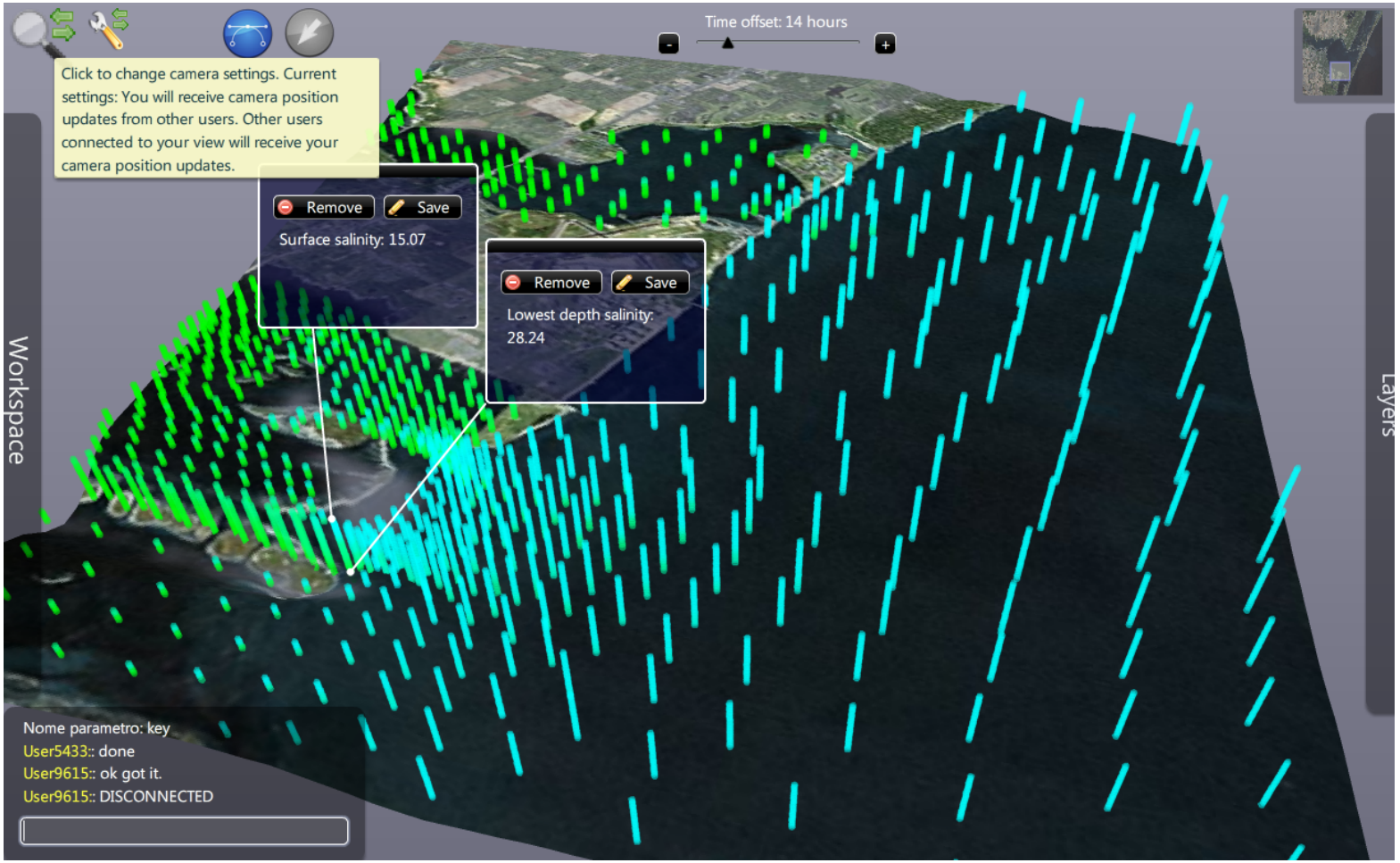
45. screenshot visualizing a data layer, streamlines and a geo-referenced annotation



46. Behavior of streamlines starting from the same point at different depths



47. Streamlines starting from significative points show the divergence of different current flows



48. Annotated depth slices