Towards an Easy-to-learn and Extensible Platform for Scientific Visualization

Hervé Sanglard
Computer Science Department
University of Neuchâtel
CH-2007 Neuchâtel

herve.sanglard@unine.ch

May 9, 2001

A mes deux rayons de soleil
Véronique et Justin

Remerciements

Cette thèse n'aurait pas pu voir le jour sans l'entourage, les encouragements et les idées de ma famille et de mes amis. C'est pourquoi j'aimerais avant toute chose remercier :

- mon épouse, Véronique, pour son amour et sa compréhension,
- mon fils, Justin, pour son beau sourire et sa joie de vivre,
- mes parents, qui ont su me passionner pour les études, m'encourager et m'entourer de leur affection,
- mon frère, Arnaud, ma famille et ma belle-famille,
- mes amis, et tout spécialement Luc Girardin, Marcel Maillard et Laurent Iff,
- mon directeur de thèse, Hans-Heinrich Nägeli, pour son soutien et son enthousiame,
- mes professeurs, et tout spécialement Pierre-Jean Erard,
- les membres du jury, John Maddocks, François Nyffeler et Daniel Thalmann,
- le Groupe Limnocéane et le Centre d'Hydrogéologie de l'Université de Neuchâtel et en particulier Redoine Tahiri, Michael Hauns, Eduard Zuur, Vincent Thunus,
- le groupe Parallélisme, les étudiants, doctorants et chercheurs qui ont collaboré à l'accomplissement de ce travail, notamment Steve Casera et Alain Conod,
- mes collègues de travail de l'Institut d'Informatique et compagnons d'étude, et tout spécialement Claude Fuhrer, Gianni Gasparotto, Robin Farine, Laurent Vuilleumier, Cédric Perret, Yves Salvadé, Emmanuel Wermeille et Jean-Daniel Scherrer.

Contents

Chapter 1

Introduction

Visualization can be defined as the mapping of data to a graphical representation by means of a computer in order to analyze them with the purpose of a better understanding. We all have seen the pretty pictures or the animations presented daily by the weather forecast team on TV. This is the archetype of a visualization process which, in this case, represents meteorological information with annotated pictograms expressing hours of sunshine, cloud covering or isobaric contour lines.

Using visualization, it is possible to infer at a glance a relation of cause and effect, for instance of isobaric contour lines and the wind at a given moment and at a certain place. This technique is powerful and helps researchers to study physical phenomena, for instance, to understand the effects of human activity on the atmosphere and the seas.

More precisely and formally, the purpose of visualization is to make a quantitative or qualitative analysis of multivariate data easier by concentrating a huge volume of information into a condensed visual representation. Visualization breaks the limits of bidimensional classical tables or graphs and provides an interactive means of presenting higher dimensional quantities and fields within a single view.

Visualization is a generic term used in several disciplines; its meaning can slightly vary according to its qualifier or the context in which it is used. Let us give an example: **scientific visualization** is sometimes distinguished from **information visualization** in that the latter refers mainly to the visualization of synthetic information, e.g. statistics, produced by human activities, whereas the major concern of scientific visualization is to represent data produced by numerical simulations or measures. But, generally speaking, even if this rough distinction is commonly accepted, it is not so easy to draw a strict frontier: does, for instance, the visualization of the solution of partial derivative equations belong to information visualization or scientific visualization?

Scientific visualization can be related to **computer graphics** because

techniques developed in this area are the foundations of scientific visualization. But these two disciplines have quite different goals: whereas computer graphics tries to produce realistic pictures and animations, the goal of scientific visualization is to help users to interpret the data and to draw conclusions. This makes a fundamental difference even if both disciplines are obviously close.

In this thesis, we will focus on scientific visualization and **restrict** this notion to the **visualization of physical, biological or chemical processes**, and of problems related to applied sciences and engineering. In these domains, the analysis of results produced by numerical simulation or of measured data becomes more and more important. In order to be successful, this task requires the collaboration between domain specialists and computer scientists with skills in the field of scientific visualization. Throughout this thesis, we will refer to the latter using the term of "visioneer" in order to distinguish him from other computer specialists, like numerical analysts or system managers.

1.1 Context

1.1.1 The challenge of a general solution

One of the challenges of scientific visualization is to address the numerous and specific needs of users coming from a wide range of research activities. Researchers as well as practitioners in disciplines ranging from physics to biology or from earth sciences to archaeology make demands on this powerful technology [9, 72, 122, 121]. Obviously, the needs in all these domains are extremely varied so that it is not possible to provide a general solution adapted to all visualization problems. These users have very different backgrounds and the tools they use are completely different from one activity to the other. However, it is possible to formulate the most important and general needs these communities have in common: they need practical tools to analyze data. In the course of their investigations, users face three typical situations:

- they guess the kind of effects which will happen, but do not know exactly **where** or **when** these effects will take place;
- they would like to know **which** kind of effects are happening at a certain location and at a certain instant;
- they do not know **which** kind of effects they will discover, neither **where** and **when** they will happen: it is the very purpose of the visualization to reveal it.

These situations impose very different requirements on the visualization environments. In the first case, a software able to provide an adequate, but

finite, set of tools required for the investigation can solve the problem. The detection of the features can be done interactively by a user or automatically by the software.

On the contrary, the last two situations are more complex, because in these cases, a finite set of tools may not be sufficient to visualize the phenomena. Often, specific tools have to be developed to succeed in this task. This is certainly the major problem of visualization and it is, in our opinion, an open research problem today.

1.1.2 Genesis of the thesis

At the beginning of this thesis, we initiated a collaboration with two groups of researchers who can be considered as representative users of our field of interest. It was interesting to note that their major and common requirement was expressed by the two sentences: "Could you help us to analyze the results with visualization? But, please, we do not want to program anything." Obviously they were aware of the difficulty of the problem and could not spend too much time on building the tool they needed.

Together with these users, we started to use existing visualization environments, but, as visioneers, we noticed that users had to be supported permanently. Indeed, visualization tools were either too complicated for them to master, or not well-suited for their real work. Thus we quickly considered that the solution offered by the existing systems was not economical, either for users nor for visioneers. It is the reason why we decided to search for a better solution.

1.1.3 Application domain

In this thesis, we will address the needs in scientific visualization of one, rather large, class of users. The main characteristics of these users can be summarized as follows:

- they are interested in physical phenomena related to natural science problems;
- they analyze three-dimensional, mostly time-dependent, data produced by numerical simulations or by measures.

As a typical example, we will consider the fluid dynamics problems that arise in oceanography or hydrogeology. In this kind of problems, the simulation of the physics produces data that are often large and multivariate, like vector fields or even tensor fields, and usually vary through time. Satisfying the complex visualization requirements of this category of users constitutes per se a solution for these cases where, for instance, time-dependency can be neglected or where data are simple scalar fields. Thus we shall concentrate on the visualization of results of **numerical simulations**, overlooking

the visualization of data acquired by measures, because all the interesting problems arise in the former case.

1.1.4 Technical possibilities

Numerical solvers have for a long time proven to be useful in understanding or predicting complex physical phenomena like those that play a role in weather forecasting or more simple problems, like the heating of a piece of aluminum. Such software passed from academia to industry and made possible the development and mastery of new technologies and helped to increase productivity.

At the very beginning, results produced by numerical simulations were analyzed and compared "by hand", with the help of ad hoc programs for instance, and the necessity to have more convenient methods to understand a huge volume of information quickly arose.

The development of peripherals, like monitors and printers, based on raster images gave the possibility to present more adequately the information elaborated by a computer. Numerous homemade solutions that tried to represent data graphically appeared. This technology helped to make the analysis of numerical simulation results much more effective.

Then, the increasing power of microprocessors contributed to simplify the user interfaces so that anybody can use a computer without being an expert. Many tasks which were then considered complex by the users could now be achieved simply and efficiently by moving a pointer and playing with icons. It can be claimed that the very takeoff of scientific visualization is due to these improvements: it started in the mid 80's due to an important development of the graphical workstations.

Today, the price and performance of these computers bring them onto the desk of many researchers. These researchers are stimulated by the conclusions they can draw thanks to the expressiveness of the generated pictures. Moreover, multimedia computers offer even more possibilities for analyzing the data. Researchers do not limit themselves to classical rendering techniques like pictures or movies, but try, for instance, to relate sound [56, 62] with data, and to investigate any technique that could be useful. The game industry boosts the development of devices like graphics or sound cards, and makes prices decrease. The ever growing power of processors and graphics cards makes possible an interactive exploration of the data using navigation tools like virtual cameras. Researchers are developing new hardware – called virtual reality peripherals – to make the user believe that he is completely immersed [16, 50] inside the data and can feel virtual objects expressing them as if they were real. New trends like collaborative and multidisciplinary problems require that data have to be exported as virtual objects and exchanged through the Internet.

All these requirements have a high impact on the way data can be vi-

sualized. It no longer suffices to produce static bidimensional cuts through data: but it must be possible to visualize data with fully interactive tools.

1.2 Goals and non-goals

The development of original visualization techniques is an exciting field of research. Numerous efficient techniques, both on the quantitative and qualitative plane, still have to be imagined. However, we think that it is also as important and complicated to search for good ways to bring these techniques onto the desk of users who are not visioneers through efficient visualization systems. For this particular reason, we will not try to develop any new or original visualization techniques in this thesis, but rather focus on the relationship between users of visualization systems and visioneers, who are responsible for tool development and user support.

As explained previously, the profusion of new hardware and libraries gives new possibilities to users of scientific visualization. The user community has become wider, and the problems to which this technique is applied have become more and more complex. Unfortunately, there is a price to pay for users of this technology because the difficulty of mastering three-dimensional graphical libraries increases with their functionalities. However, end users, even physicists or numerical analysts, cannot spend too much time learning graphical libraries and developing by themselves a set of tools adapted to their specific needs. Visualization is becoming a specialization of its own. For this reason, users either visualize data with one or several dedicated environments and accept restrictions to their functionality or they require help from a visioneer in order to build a specific application. In the latter case, users highly depend on visioneers because even little customizations can lead to a considerable development effort.

Our primary goals are, first, to achieve a clear separation between the different actors involved in the process in order to make users less dependent on visioneers, and, second, to propose new and convenient visualization paradigms in order to make the division of labor between them more effective. To meet these goals, it is of primary importance to define a new, simple and unique conceptual framework for three-dimensional visualization techniques. The standardization of bidimensional graphical techniques, e.g. bitmap, pixmap, color map, coloring systems, and standardization of operations applied to them, e.g. scaling, smoothing, rotating, led to the development of powerful and general purpose graphical software which are extensively used today. It is part of our thesis that unification of visualization techniques can be achieved in the context we defined above and that new kinds of visualization environments can benefit from this approach. Thus we will demonstrate that a platform based on these concepts can be **easy to learn**, so that users have the degree of autonomy they need, and

provide an adequate level of **extensibility** so as to be applicable in several disciplines.

1.3 Overview

We will start in **Chapter 2 (Limitations of existing systems)** with a short survey of the major visualization systems encountered in the domain of interest. The limitations of these systems as well as the major conceptual problems and technical difficulties the users and visioneers face in this discipline will be emphasized.

In **Chapter 3 (The visualization objects)** we, first, propose a new formalization of the visualization process that is closer to the needs of contemporary scientific visualization. Then, legacy visualization metaphors as well as abstractions commonly used in this discipline are discussed and a new metaphor, called a visualization object, that unifies visualization techniques is introduced. A few comparisons with older metaphors are undertaken and benefits of the visualization objects approach, for both users and visioneers, as well as the new possibilities offered by this concept are emphasized.

In **Chapter 4 (A new kind of visualization system)** we address the gap that exists between users and visioneers with regard to the complexity of the tools and propose a solution to make the division of labor more effective. We then examine more deeply the respective requirements of both users and visioneers and enumerate the architectural consequences this has on the development of an "ideal" visualization system. We explain how a visualization platform can benefit from the visualization objects concept and how the user interface can be improved. Finally, the advantages of this concept with regard to the extensibility of the platform as well as the implication it has on data management are considered.

To prove the soundness of those new ideas, a visualization platform, called ZoomIn, was built. In **Chapter 5 (ZoomIn)**, its general architecture is sketched, and we explain how the object-oriented paradigm made possible its construction and provided the versatility required for the implementation of most of its components.

Chapter 6 (Application to case studies) illustrates the usage of ZoomIn through case studies in oceanography and hydrogeology. A few results obtained by users thanks to the platform are given and the effort necessary for the customization of the platform to their real needs is estimated.

The last chapter **Chapter 7 (Conclusion)** summarizes the advantages and drawbacks of the approach presented in this thesis. The experience gained and the actual contribution are mentioned as well as unexpected problems we encountered. We will finally give our opinion about the perspectives in the field of scientific visualization and try to make a few pre-

dictions on tools that will, in the future, assist the users in the analysis of data.

Chapter 2

Limitations of existing systems

This chapter will begin with a short survey of existing visualization systems. Three different categories covering the large variety of visualization systems will be described. A few characteristic examples of visualization systems for each category will be given.

Then, a discussion about the major limitations that the users of these systems encounter – in the application domain we defined in the previous chapter – will follow. We will be interested in the specific limitations of each category in the relevant domains of interactivity, ease-of-use, extensibility, division of labor, reuse, and performance.

2.1 Classification of existing visualization systems

Numerous visualization systems have been developed on most computer architectures [65]. To provide a complete assessment of all these systems would be an impossible task. Indeed, because an assessment depends on the particular needs of the person who does it, it is very difficult to avoid subjectivity. To be exhaustive, it is necessary to take into account numerous criteria such as performance, ease-of-use, extensibility, scalability, price, support, documentation, hardware and operating systems supported as well as the community using the tool, the input and output data formats provided and so on.

As there are so many visualization tools having so many features, we will not try to give a complete survey. However it is useful for the discussion in the rest of the thesis to group them very roughly into three categories :

- special purpose applications,
- modular visualization environments, and

- libraries with toolkits.

2.1.1 Special purpose applications

Special purpose applications are dedicated to a particular type of visualization problem. They are sometimes called **"turnkey" applications**, because they seem "simple to use" and because complex functionalities required by the specific discipline are accessible thanks to simple manipulations. This kind of application has usually been built in order to satisfy the classical needs of a given user community. They work on special types of data related to the nodes of a grid. For instance, a visualization system for oceanography will work on scalar fields, like temperature or salinity, and on vector fields, like velocity, that are defined in a given domain; it allows the user to obtain different representations of these fields in selected regions of interest, like planar cuts or isosurfaces of a given scalar field.

Well-known and typical examples are for instance:

- **SciAn** [80, 102], developed at the Florida State University, is restricted to scalar and velocity fields and provides a limited set of tools (contour, mesh, isosurface, arrow, streamline, streamtube).
- **Fieldview** [48, 47], from Intelligent Light Inc., is a commercial visualization software that is dedicated to computational fluid dynamics and provides mainly five tools (computational surface, isosurface, streamline, probe, particles).
- **Fluent** [32], from Fluent Inc., is a commercial numerical solver for CFD that provides visualization tools. Usually, solvers integrate their own visualization environment.

To be more exhaustive, systems like EarthVision [29], Ensight [46], FAST [66], Geomview [104], HIGHEND [74, 97], Plot3D [67], Tecplot [3], UFAT [68], Vis5D [98], Visual3 [28], VolVis [117] can be classified in this category.

2.1.2 Modular visualization environments

These environments, also called application builders, are founded on the dataflow and visual programming paradigms [38]: they provide a number of predefined **modules** each categorized as **filter**, **map** or **render**. Roughly, **filters** deal with data management and conversion, **maps** transform data to geometries and **renders** display geometries onto the screen. The outputs and inputs channels of these modules have to be connected by the user. Hence the user sets up a **network** visually by drag and drop. The data traveling through this network are subject to various transformations and the results are displayed in a graphical window (see Fig. 2.1).

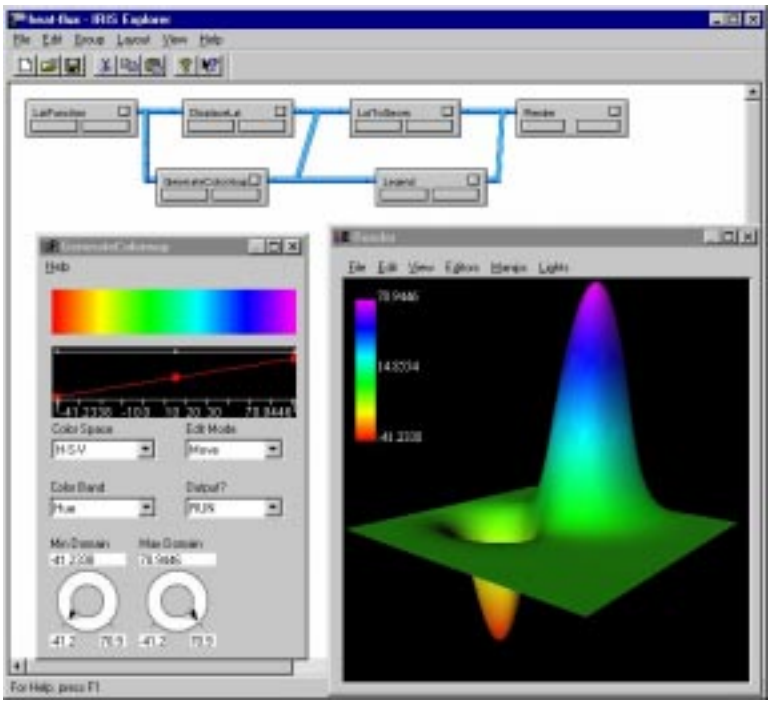


Figure 2.1: Modules network in a dataflow system

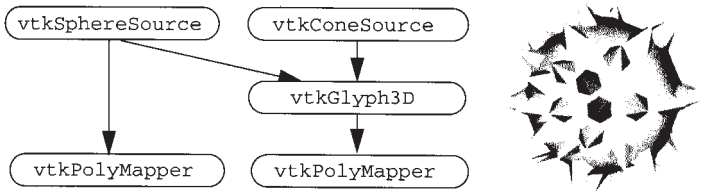
If there is a lack of functionality, the visioneer has the possibility to **program** a module and to **add** it to the environment. A particular usage of this kind of visualization systems consists in using them as a programming library to build a **special purpose application**. Well-known commercial systems like **AVS/Express** or **AVS5** from Advanced Visual Systems [113, 2], **Data Explorer** from IBM [49] and **Iris Explorer** from Silicon Graphics and NAG [96, 105] belong to this category.

2.1.3 Libraries with toolkits

In the last years, numerous libraries have been developed in order to provide multipurpose and standard tools for visualization and computer graphics in general. They are often provided together with toolkits that ease their use. Typical examples are Silicon Graphics' **OpenGL** and **OpenInventor** libraries [69, 119, 120].

Based on a higher level of abstraction, the **Visualization Toolkit** (VTK) [93] was developed recently. It is a portable object-oriented library for visioneers who want to quickly build applications or prototypes for their users. The model supported by this library is the **visualization pipeline** also called visualization network (see Fig. 2.2). The pipeline consists of ob-

jects to represent data (data objects) and objects to operate on data (process objects). This is very similar to the **dataflow paradigm** of modular visualization environments described previously, except that, in the case of the visualization pipeline, the execution is controlled implicitly (a process object executes only if its local input or parameters change). Whereas in the dataflow paradigm, execution is controlled explicitly by an executive component tracking the changes to the network and controlling the execution of the process objects.




```
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);
```

Figure 2.2: Example of a visualization network in VTK

2.1.4 Other systems

Of course, certain systems lie between two of those categories. For instance, different attempts have been made to make special purpose applications more flexible . The example of **Spray** [76, 21] seems to be exemplary in this respect. In this 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**.

2.2 Basic interactivity and difficulty-of-use

Interactivity as well as ease-of-use play an increasing role, because more and more users who are not experts in visualization want to take profit of this technology. Generally speaking, it can be stated that the majority of today's visualization systems offer modest interactive functionalities compared to other applications such as, for instance, the entertainment or desktop applications used daily by millions of people.

Among the major and particular difficulties encountered by visioneers in order to build efficient visualization systems are the complexity of transforming data into comprehensible visual information and the technical requirements imposed by the management of datasets.

2.2.1 Interactivity has severe implications

Due to the volume of data and the complexity of physical phenomena to visualize, the need of interactive systems offering an intuitive and natural "exploration of datasets" is of paramount importance [31]. By **intuitive**, we mean that the interface seems familar and easy-to-learn to the users, and, by **natural**, that it operates in such a way that a human ideally needs no instruction.

"The amount and ease in which insight is gained from the dataset is an indication of the success of the visualization process. Equally as important as the development of powerful visualization methods is the development of techniques and interface methods that make these powerful tools available to the users..." [75] Certainly, one of the best solutions is to provide the users with visualization systems that allow direct manipulations of tools the users are familar with. "The tools should be made accessible in an unencumbered, direct, and natural way – first to gain acceptance from the users, and second to promote unimpeded investigation of their data without having to think about the visualization methods... The interface should be so natural to use that it is almost invisible to the users. In this case, the measure of an interface's success is its apparent absence from the user's thought process while investigating their data set." [75]

This easiness is usually offered by special purpose applications but this is neither the case with modular visualization environments, where the user manipulates the modules network to act indirectly on the graphical objects, nor with programming libraries. In both systems the user has to develop an ad hoc graphical user interface – which requires skills in programming – in order to reach a satisfactory level of interactivity.

Interactivity has severe consequences on data management. Because the users want to explore datasets and to concentrate on regions of space where phenomena are expected, it should be possible to restrict the visualization to the neighborhood of a region of interest. Many of today's visualization

systems operate on a grid – or a mesh – and are not able to honour neither "local" access nor interpolated data between the nodes of the grid. This is certainly one of the main problems to solve in order to successfully build interactive visualization systems providing:

interactive selection: the ability for a user to select the regions of particular interest in the datasets, and, if needed to be able to re-run the simulation for a subset of the space-time domain;

computational steering: the ability to change parameters of the simulation while the simulation is progressing: for instance, to change the direction and intensity of the wind in an oceanographic simulation.

Such **interactive selection** or **computational steering** environments are becoming a mandatory requirement for large data sets, simply due to the volume of data to be analyzed [30], but the few prototypes having this capability are still under development [116].

2.2.2 Advances in virtual environments

Reaching a satisfactory level of interactivity imposes an effective navigation in 3D space as well as an intuitive manipulation of graphical objects. Indeed, many problems encountered by users are due to the fact that the projection of three-dimensional graphical objects onto bidimensional screens causes ambiguities and distortions.

Modern visualization systems have to accomodate the use of virtual reality peripherals – such as a spaceball, stereoglasses, helmets or datagloves – in order to help users to visualize three-dimensional datasets. A few visualization systems propose immersive environments [50, 95, 100] thanks to virtual reality peripherals.

At the moment, these peripherals are rare, very specific, cumbersome, expensive and have to be used in powerful near real-time interactive systems. The next task of engineers will be to improve their accuracy as well as to diminish the size of these devices. Furthermore, research is still needed in order to find appropriate methaphors of gesture [75] to interact directly with the virtual objects or to invoke commands inside the immersive environments because there is no more classical graphical user interface.

2.2.3 Collaborative work

The increasing performance of local area networks (LAN) and wide area networks (WAN) offer new perspectives for collaborative work. Except for a few systems [77], most visualization systems are engineered to be used in single-user mode and do not integrate the possibility to collaborate on visualization sessions from many computers, either locally – in the same site –, or remotely through the Internet.

Collaboration between researchers or practioners dispersed all around the world is becoming a requirement. Although this capability is desirable for many teams, integrating collaborative work into visualization systems is not straightforward. It has significant implications on software design because of the functionalities that have to be added to the user interface and because of the technical problems imposed by the networked architectures, such as:

- access control over graphical objects,
- private and shared views,
- synchronous and asynchronous collaboration modes,
- data access, and
- software portability and data format over heterogeneous networks.

2.2.4 Difficulty of use

Will Schroeder, Ken Martin and Bill Lorensen [93] wrote: "Visualization systems are by their very nature designed for human interaction. As a result they must be easy to use." Proclaiming that a certain visualization system is easy to use and that another is not, is risky. This quality depends very often on the user's level of expertise and on the specific functionalities that the system offers. But it is sure that, for non-specialists and for casual users, the necessity of reading and understanding hundreds pages of documentation or thousands of lines of code before obtaining any result is discouraging.

In the past, visualization systems were reserved for researchers of disciplines close to computer science. They had, and were able, to spend the time necessary to understand in detail and to master the visualization packages. But now, environments have to become easier to use in order to provide efficient solutions for users from other disciplines, e.g. for natural science researchers.

Today, certainly the best solution is offered by special purpose applications and the worst one by libraries with toolkits. Modular visualization environments (MVE) lie between these two extremes thanks to the visual programming paradigm. However, as pointed out by Earnshaw and Jern [30], large module suites are difficult to manage. Although their main advantage is their generality and their extensibility – which explains their popularity – their drawbacks appear quickly when users face the problem of using them. It is due to many reasons:

- They are not tailored to an application and extensive programming, albeit visual, is required to set up an application. "This is an obvious weakness because users expect to be able to turn it on and go" [13, 7].

- Because strong data type is usually required between modules, a profusion of modules with almost identical functionality have to be introduced, which discourages the novice.

- Because almost each module has its own parameters to tune, each one has its own interface widget; this leads quickly to a fragmented and incoherent graphical user interface. The complexity of maintaining all these elements grows rapidly[1] with the number of interconnected modules.

- The graphical user interface is not uniform between the applications and the functionalities are reduced because it is often necessary to redevelop them for each problem due to the lack of abstraction at the application level.

2.3 Limited extensibility and versatility

Just as for other kinds of software, extending a visualization system with functionalities for which it was not designed is very difficult, if not impossible. Unfortunately, the need to extend a visualization system is common due to the infinity of imaginable techniques for the visualization of data and due to the unpredictability of the way the interesting phenomena will be discovered. This imposes a high degree of versatility to general visualization systems.

Ideally, it should be possible to extend a visualization system with new visualization primitives, device and dataset drivers. But the reality is completely different because of the technical implications it has. In fact it is not simple to extend a visualization system or to customize it to a specific problem: it is usually reserved to visioneers or advanced users who have extensive experience in programming.

2.3.1 Special purpose applications are not extensible

At first glance, **special purpose applications** seem ideal for users because they allegedly are easy to use and well-suited for their specific needs. From the beginning, the user feels comfortable because, thanks to an easy to learn user interface, he does not need to program anything. This is the case when the visualization package is fully integrated with the numerical simulator, because the user does not even need to care about data management. However, these environments show their limitations rather soon as they usually are difficult to extend. The user has to be satisfied with the functionalities

[1] As depicted in the simple example of Figure 2.1, the user has to control 6 modules, each having its own control panel, and 7 links in order to produce only 1 graphical object (and its accompanying color scale) representing the function!

they offer and to adapt his way of thinking and working to the logic of the application. This is especially the case for visualization systems integrated with numerical simulators where only the manufacturer is able to release new or customized visualization tools.

In practice, however, the user is often forced to use several environments in order to perform his analysis. This leads to painful and time-consuming technical problems like data conversion from one format to another, not to speak of the difficulty of merging the results produced by different systems.

As is the case for Spray [76], a scripting language represents a powerful means of extending such a system. However, it is not always accessible to the casual user, who therefore remains dependent on a visioneer.

2.3.2 Extension of visualization systems requires expertise

Generally speaking, extending a visualization system with ad hoc visualization techniques always requires expertise. And because problems for which visualization is useful are varied, the need of extending systems appears rather quickly. It is not sufficient to master basic programming, e.g. to know a language like C/C++ : several software technical difficulties have to be tackled:

- access to the data;
- understanding of the 3D graphical system and primitives, and
- design and implementation of a user interface.

This kind of knowledge is far outside the reach of many users, especially those we consider, because they are not programmers and do not want to develop anything.

Certainly today's best solution for users is offered by modular visualization environments when used as application builders: the very last versions of modular visualization environments, such as **AVS Express** provide front-end and rapid application development wizards in order to minimize the work of programmers. These components provide an additional layer that helps to quickly build a prototype of the graphical user interface. This consists in implementing a specific graphical user interface hiding the network with a centralized command panel allowing direct manipulations of graphical objects. Thus, one can develop a special purpose application from scratch or by adapting existing modules.

Unfortunately these components shift the problem. Rapid development tools that generate code automatically certainly save time for tedious development such as, for instance, the design of dialog boxes and of pop-up menus. But time and expertise are needed to master this kind of tool, and

a great experience is required, as soon as one wants to undertake the development of prototypes or customize the samples produced by the wizards that are provided by these tools. "Visual programming is too limited for detailed control, so construction of complex low-level algorithms and user interfaces is not feasible." [93] and it is the main reason why "modular visualization environments require a considerable effort from users" [100] to be customized to their real needs.

2.3.3 Conclusion

To summarize briefly this section, we come to the conclusion that in our field of interest and for the users we consider:

- either visualization environments or libraries are extensible but require an important development or support effort from the visioneer in order to be usable by the user;
- or the application is easy to use but its extensibility is limited.

2.4 Inefficient division of labor

As the problems under study become more and more complex, a particular division of labor is taking place. All actors involved in the visualization and analysis process have to be specialists of their own domain in order to take up challenges and to achieve a good level of productivity. End users cannot spend too much time on tooling problems because they have to concentrate on their main activity. This situation imposes a new challenge to visualization systems.

The division of labor was not the primary concern of the people that developed general visualization systems because :

- the users were at the same time applied mathematicians, visioneers and end users, and
- the development of qualitative and ad hoc visualization techniques was the primary step.

Because the problem is emerging and because we did not find anything in the literature about it, we will try to give a rough evaluation of the situation based on our experiences with users.

The three graphs in Figure 2.3 depict the amount of effort invested through time in the customization of each category of existing visualization system for a typical problem.

In the case of special purpose applications, the visioneer intervenes only at the beginning of the investigation in order to convert the data or setup the software preferences (case 1). After that, the user is autonomous, and

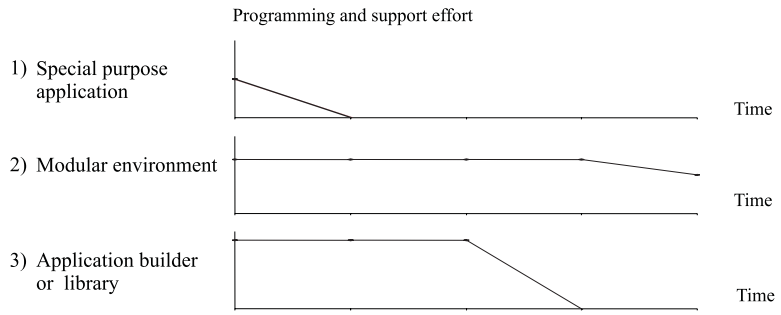


Figure 2.3: Programming and user support effort

usually does not need any further help from a visioneer because no customization of the application is possible.

Because of the complexity of modular visualization environments, the visioneer has to give the user intensive support during the whole process of data exploration, modifying the network or programming new visualization modules (case 2). Both the setup of a specific environment and its maintenance requires an important investment. The user is fully dependent on the visioneer from the beginning to the end of his investigations. This case is illustrated with the second graph in Figure 2.3.

As illustrated with the third graph in Figure 2.3 , our experience has shown that it requires more or less the same effort to develop a standalone application with a MVE – used as an application builder – as it takes with a good visualization library (case 3).

It is interesting to notice that there are two significant differences between case 2) and 3). The first difference resides in the fact that the visioneer's effort in case 3) is more important – because almost every component have to be developped from scratch –, but limited in time – because, once the application is running, users support is mimimum due to the fact that the application is customized to their particular needs. Moreover, in case 3), the application is generally not easily extensible, just as for case 1).

Thus it appears that none of these three situations provide a good solution in terms of division of labor. A visualization system providing a large autonomy for users and minimizing the customization and support effort of the visioneer has still to be found.

2.5 Insufficient reuse

The "recent" analysis and design methods [36, 12, 118] as well as object-oriented technology had an important impact on the development of software. The quality of a system not only depends on the functionalities it

offers once and for all, as is the case for special purpose applications, but also on its genericity, on the possibilities of reuse, on its customizability for a wide community and adaptability to new problems and new kinds of data. In order to increase productivity, reuse has become a major issue in every new development. Object-oriented analysis and design methods are efficient and can be applied successfully to scientific visualization systems [15]. Abstracting information and organizing it into class hierarchies as well as managing objects abstractly using polymorphism are powerful mechanisms that can drastically increase reuse in this discipline.

Several visualization systems such as Plot3D, UFAT and FAST [67, 53, 66] were not designed with reuse in mind, because, first, efficiency was the primary concern – they were developed around algorithms optimized for a specific task – and, second, they were engineered before the emergence of object-oriented technology.

The result is that much work is invested in implementing a set of applications that usually cannot be reused by other teams because they are not designed with a sufficient level of abstraction. There are numerous examples of powerful visualization techniques that were implemented in some package, but left unused because they could neither be conveniently integrated into existing applications nor linked together within a general environment accessible to the user. This is not very productive and rather frustrating both for the scientific visualization community and for users.

2.6 Low performance

2.6.1 Data management

Certainly one of the major criticism of today's visualization systems is their inability to handle large datasets. It can be stated that in the particular discipline of scientific visualization, the increase of computing performance is an order of magnitude below the needs of the community. On this subject Steve Bryson commented [100]:

> *"The problems addressed in real-time scientific visualization will, however, far outstrip the advances in technology. Increased computational capability will be used to perform simulations that produce far larger data sets, and contain far more phenomena. There are several ways in which this will occur: the spatial and temporal resolution will increase for enhanced accuracy, increasing the number of points and timesteps; and multi-disciplinary simulations will be performed, including many phenomena in a single data set. Thus the size of data sets will increase dramatically over the next few years."*

Because data transfer from mass storage to memory remains a bottleneck for many computers, datasets produced by number crunchers have to be organized and processed adequately. Even for middle sized datasets that currently reach many hundreds of megabytes, it is not possible to load them entirely into the memory of a classic workstation and to expect a good job from its virtual memory manager.

Many existing visualization systems [80, 48, 76, 98] load all timesteps of datasets into memory even if the user is interested in visualizing data in the neighborhood of a specific point and only for a given range of time. This situation is certainly due to the fact that today's systems are simple extensions of first generation visualization systems that were thought to work in batch mode, without any interactivity. At minimum, existing systems should be re-engineered in order to accomodate the volume of contemporary datasets.

Other visualization systems [113, 96, 93] use another data model called **dataflow**. As explained previously, the mechanism is the following : the data are traveling through modules that successively convert, transform or select portions of the original data. Finally, the data are mapped into a graphical primitive and rendered onto the screen. Because most filters do not accept every kind of data structure as input, some specific filters are inserted along the path in order to convert the data adequately. For instance, a filter can interpolate data from a finite element dataset to a rectilinear grid in order to be accepted by the next filter. Basic implementations of the dataflow model pass a copy of the data. This leads to double or triple the volume of information to manage even if avanced mechanisms avoiding copying of the data are added. Although such optimizations undoubtedly accelerate the response of systems, the performance is still insufficient for interactive visualizations:

> *"Any change to the data set necessitates reading it in once more and repeating the process. Thus interaction rates depend principally on the speed with which the dataflow pipeline can process the data, severely limiting the speed of interaction with large data sets". [84].*

> *"Because dataflow systems transform the entire data set, they are not well suited to the near-real-time visualization of time-varying data because of the very large amounts of data involved". [17]*

As mentioned by several authors [30, 84, 109], the increasing volume and variety of data, and the need of interactive visualization systems, require the development of a new models for data management providing among other things :

- scalability;
- the integration of visualization and simulation;

- the distribution of data over a LAN and/or a WAN;
- a good compromise between extensibility and performance,
- a common interface to access to disparate data objects, and
- the computation of derived data on the fly.

2.6.2 Rendering

Another bottleneck for interactive visualization systems appears when a large quantity of non-trivial graphical objects have to be rendered at a sufficient frame rate. The computations needed to render these objects are intensive. The stages of the graphical pipeline – such as face culling, levels of details, shading, z-buffering – are accomplished by software or hardware and require millions of floating point operations.

Although spectacular progress were made in the last few years, the limits are easily reached when numerous objects have to be rendered simultaneously at a sustained rate of 24 frames per second in order to create the illusion of dynamics. This causes delays and latencies that disturb the user during his manipulations. Moreover, because many visualization systems do not allow a fine control over the graphical objects and the region of user's interest, visual clutter is quickly reached as well as the latencies accompanying it.

Addressing this particular problem is out of the scope of this thesis. However, it is obvious that concepts allowing the development of ad hoc user interface able to improve the control of graphical objects are useful and can contribute indirectly to increase the global performance.

Chapter 3

Visualization objects

In this chapter a new model for the visualization process, called **interactive visualization process**, will be introduced. It takes account of the needs of interactivity imposed by the ever growing community of people who want to take profit of scientific visualization.

Then, we will point out one of the key problems end-users encounter in scientific visualization: the low abstraction level and the lack of convenient metaphors provided to users at the application level. We will try to explain why it is essential to remedy this problem. Nevertheless, a set of **metaphors** that were successfully used in existing visualization systems will be presented.

The next section will propose the central idea of this thesis : a **new conceptual framework unifying most visualization techniques**. It consists in a hierarchy of objects dedicated to scientific visualization, called the **visualization objects**, for which we will give an in-depth, semi-formal description. These objects will be compared to the metaphors described previously: the differences and benefits of this new approach will be emphasized.

Then, we will explain how this approach brings solutions to the main limitations of existing visualization systems that were described in the previous chapter. In particular, how a visualization platform satisfying both the requirements of **ease-of-learning and of extensibility** can be built thanks to this conceptual framework.

Finally, the benefits in terms of **rapid prototyping and reuse** will be emphasized and the new perspectives offered by the concept in the field of a **client-server achitecture, collaborative work and computational steering** in scientific visualization will be investigated.

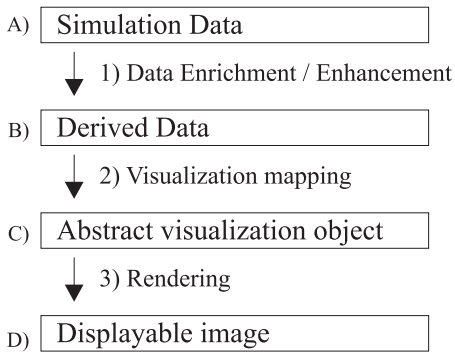


Figure 3.1: The visualization process

3.1 The visualization process

3.1.1 A classical visualization process formalization

Scientific visualization is usually viewed as a multi-stage process beginning from simulation results and ending at a displayable image. Since the early days of scientific visualization, several visioneers [39, 93, 8] proposed more or less detailed formalizations of this process. Haber and McNabb [38] gave their own formalization of the visualization process that was largely accepted and reused by the visioneer community.

As depicted in Figure 3.1, this process is composed of four stages (A–D) and three transformations (1–3):

1) Data enrichment and enhancement: This first transformation operates on the raw data (A) provided by the simulation and modifies it in one or more ways to derive data (B) for subsequent visualization operations.

2) Visualization mapping: The second transformation constructs an imaginary object called an "abstract visualization object" (AVO) (C) from the derived data produced by enhancement and enrichment. Typically, this involves mapping the simulation data into the attributes fields that describe the AVO.

3) Rendering: The last transformation operates on the AVO to produce a displayable image (D).

The sequence of these stages and transformations forms what Haber and McNabb [38] name a **visualization idiom** by analogy with a verbal idiom of a language. "Just as a listener has difficulty understanding a verbal idiom in a foreign language, a viewer is unable to interpret a visualization

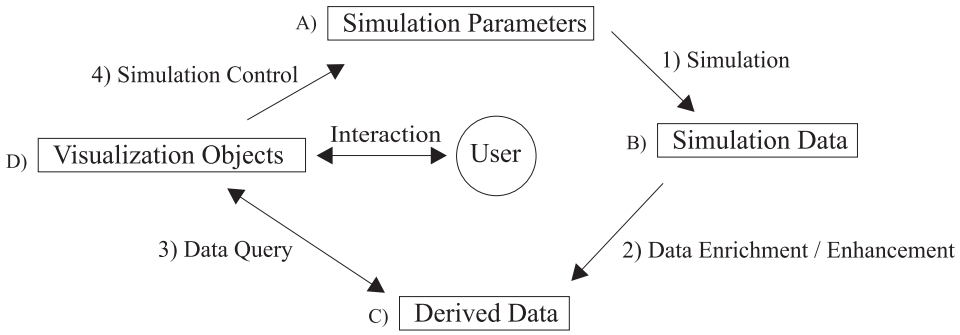


Figure 3.2: The interactive visualization process

display scientifically without an explicit understanding of the steps in the visualization idiom used to generate it."

This formalization and the idea of encapsulating each kind of transformation into a specific module type as well as the possibility for users to interconnect them interactively with visual programming techniques gave birth to the modular visualization environments paradigm.

However, when applied to our particular class of problems, this formalization is incomplete: first, it does not consider the user in the process and does not include the problem of interactivity. The visualization process is viewed as a batch oriented process in postprocessing mode and the user does not have the ability to use the AVO to interactively explore the dataset. Second, it reduces the results of the visualization process to an ultimate image that does not seem to offer any solution for interactive selection nor computational steering.

Because, our work aims at providing a solution for the requirements of interactivity allowing exploration of the datasets, we propose a new model by integrating the user into the process and by considering that user interaction – with the graphical objects produced – can feed again the "pipeline". Called the **interactive visualization process**, this new model will be described in the next section.

3.1.2 The interactive visualization process

Figure 3.2 illustrates what we define as the **interactive visualization process**. In this definition, the visualization process is no longer depicted as a "pipelined process", but it is represented by a **ring of processes** that are driven by the user.

The linking of processes starts with the **simulation** (1) according to the **simulation parameters** (A) that are given either statically or interactively by the user. The simulation generates **simulation data** (B) that are **enriched or enhanced** (2) in order to produce **derived data** (C).

Then, derived data are **queried** (3) in order to produce the **visualization objects** (D). The user interacts with the visualization objects in order to visualize the data by interactive exploration or to **control the simulation** (4) by updating the simulation parameters – either through the visualization objects representing them graphically, or directly, for instance, through another component of the graphical user interface.

By integrating the **simulation parameters** and the **simulation** into the whole process, we close the loop between the production of numerical data and the visualization system. This is the only means of tackling the problem of the production of huge data sets and of taking profit from the fact that only a small part of them is useful and really interesting during a particular interactive exploration. Thus, the visualization process is no more merely a postprocessor but is integrated with the simulation.

Notice that, in this process, derived data, which can be visualized at their turn, can result either from the visualization objects creation or from the user interaction – see the bidirectional arrows in 3). The **displayable image** stage as well as the **rendering** and **visualization mapping** transformations of the classic model are absent from this new model. They will be considered as a subprocess that produces the **visualization objects** (D). This point will be described more precisely in the 'Detailing visualization mapping' and 'The visualization objects formalization' sections of this chapter.

3.2 Higher abstraction levels

A sufficient abstraction level is decisive for a visualization system able to provide a large variety of visualization techniques through an easy-to-learn user interface. Indeed, as pointed out by Haber and McNabb [38], "most scientists forego visualization altogether unless provided with high-level tools that keep their energies focused on their science.".

For these users, abstractions at the application level is of paramount importance: this is perfectly illustrated by the popularity reached by modern software, for instance **wordprocessing**, **spreadsheets** and **databases**, due to the abstraction level they provide in the application layer. Without any doubt, thanks to expressive **metaphors** like **styles**, **cells** or **queries**, these software offer a high level of genericity and ease-of-use.

In the following section, we will examine representative examples among the most interesting metaphors that were imagined in the field of scientific visualization.

3.2.1 Visualization metaphors

Merriam-Webster's dictionary [61] defines **metaphor** as "a figure of speech in which a word or phrase literally denoting one kind of object or idea is used

in place of another to suggest a likeness or analogy between them". Applied to scientific visualization, we propose to define a **visualization metaphor** as "an idea abstracting the visualization mapping and allowing the users to investigate their data without having to think about the visualization methods".

Fundamental work has been achieved by other teams in order to provide well-suited metaphors for the visualization of datasets. This gave birth to several useful metaphors that will be described more precisely hereafter.

3.2.2 AVO

In their conceptual model [38], Haber and McNabb introduced the term **AVO**, an acronym for Abstract Visualization Object, as "an imaginary object with some extent in space and time", whose "attribute fields might include geometry, time, color, transparency, luminosity, reflectance, and surface texture". An **AVO** is the result of the visualization mapping stage of the visualization process seen above, i.e. the transformation mapping the derived data to the **AVO** fields according to a given **transfer function**. "The time and geometry fields in the AVO need not correspond to similar quantities in the simulation domain. For example, we can map the simulation z coordinate to time in the AVO model to produce an animation of sequential, horizontal slices through a volumetric domain [...] No fool-proof method exists to derive effective transfer functions, so interactive systems for real-time modifications of the transfer functions are effective tools for exploring computational data sets."

3.2.3 Spray rendering and sparts

"Spray rendering uses the metaphor of **spray cans** to paint and visualize datasets. Conceptually, visualization users grab and aim spray cans into their datasets. Depending on the type of paint in the can, different data features are highlighted and rendered. The paint particles are referred to as **sparts**, which stands for smart particles. They combine the power of both particles systems and behavioral animation to seek out and highlight features of interest in the dataset" [76, 78, 77]. These sparts have a specific behavior – which can be described interactively thanks to a simple scripting language – and produce a collection of local graphical shape, whose attributes depend on the value of the data they encounter. What is particularly interesting in this attempt is the fact that the **techniques of surface, volume and flow visualization are generalized** and that spray rendering allows **selective progressive refinement**, i.e. the ability for a user to concentrate on the phenomena occuring in a precise region of interest.

3.2.4 Glyphs and Glyphmaker

"**Glyphs** are graphical objects whose attributes – position, size, shape, color, orientation, etc. – are bound to data. These objects can be effective in depicting discrete data such as macromolecular structure and interactions, but they have also proved their usefulness in representing variables such as wind speed and direction in atmospheric dynamics simulations and observations. Glyphs owe their effectiveness to human eye-brain system's ability to discern finely resolved spatial relationships and differences in shape. They allow the user to display and correlate several variables at once. A researcher can, for example, distinguish two variables by binding them to the visually orthogonal representations for shape distortion and pseudocoloring" [83].

Glyphmaker allows non-expert users to customize their own graphical representations using a simple **glyph editor** and a point-and-click binding editor named **glyph binder**. In particular, users can create and then alter bindings to visual representations, bring in new data or glyphs with associated bindings, change ranges for bound data, and do these operations interactively.

3.2.5 Parametrized Geometric Objects

Inspired by the glyph metaphor described above, Mulder and van Wijk [115, 64] imagined a similar object called **Parametrized Geometric Object** (PGO). They built also an interactive tool named **3D PGO editor**, which is part of their **Computational Steering Environment** (CSE). In this environment, users bind graphical objects – like sphere, cubes or cylinder – to variables by parametrizing the geometry and attributes of their shape with data by means of the PGO editor. Because the goal of authors is to provide a general environment for computational steering, they have implemented a two-way communication between graphical objects and data so that users may also drive the simulation by interacting with graphical objects.

3.2.6 VTK actors

In **VTK** [93], Schroeder, Martin and Lorensen defined **actors** in order to represent abstractly any entity in a rendering scene. Used at the ultimate stage of the pipeline, an actor maintains a reference to the rendering properties and the defining geometry, that is given as input by a **mapper** object converting data to graphical primitives. At the end, all actors are collected and put into an **actors collection** which is given in input to a general **renderer**.

3.3 Detailing the visualization mapping

By carefully examining the definition of the metaphors described above, one notices two things. First, the authors of these metaphors systematically referred to the notion of **object** and, second, they introduced their own solution for the **binding** of data and graphical objects with high level mechanism, i.e. without having recourse to heavy programming techniques. Undoubtedly, this kind of approach supercedes the classic notion of image and hardcoded image processing techniques that were the foundations of most older visualization systems. Certainly the main reason for that is the fact that recent visualization systems have to integrate advanced interactive means in order to be accessible for casual users. Thus, graphical primitives are grouped into entities so that the users can manage well-identified objects.

In order to provide well-suited methaphors usable for a wide range of applications, it is essential to examine in greater detail the notion of visualization mapping. Haber and McNabb considered the visualization mapping as an atomic operation that maps the derived data to graphical primitives by means of a transfer function. However, it can be more interesting to refine this definition and to decompose the visualization mapping in three distinct steps:

1. the definition of the transfer function,

2. the result of the application of the function to given parameters, and

3. the results produced by the mapping.

As we will see in the following sections, separating the function from its application on parameters and from the results produced allows to imagine useful metaphors and abstractions for each of the three distinct steps.

3.3.1 Defining the transfer function

As explained in Section 3.1.2, the transfer function is the mapping of derived data onto a graphical representation. For instance, a transfer fonction can be the mapping of the magnitude of the velocity at a given instant and position onto the height of a cone representing also the velocity direction by the rotation angle around its gravity center.

For many visualization techniques, the transfer function is defined once and for all, but it is essential that the function can be adapted to fullfill the particular needs of users. Ideally, it should be possible to define this function interactively – as it is the case in Spray, Glyphmaker and CSE. But practically, because most mappings are non-trivial, only the expression power of a programming language can provide a general solution.

3.3.2 Mapping the function on parameters

After the definition of the function and the set of parameters, the second stage consists in applying the function to the instantiated parameters. This step corresponds to the visualization mapping transformation of the classical visualization process (see Section 3.1.1). It is generally achieved by the user or automatically by the system. Four types of parameters can be identified:

1. spatial parameters,
2. temporal parameters,
3. a source of data, for instance, the dataset and field of interest, and
4. rendering parameters such as, for instance, a color map for pseudocoloring.

The parameters can be introduced using several input devices as for instance keyboard, microphone, mouse or VR peripherals.

3.3.3 The results produced by the mapping

Then, the application of the function to the parameters produces a result. It mainly consists in a virtual object that is displayed onto the screen – by taking into account the rendering parameters – as a graphical object. Contrary to the classic visualization process (see Section 3.1.1), the result of the mapping is a virtual object existing in the mental representation of the user. This object is no longer reduced to a displayable image but the user is able to interact with it: this object is an indivisible entity that the user can control directly.

The mapping can also generate derived data: for example, the creation of an isosurface usually generates a closed surface whose volume can be an interesting quantity to visualize. Convenient metaphors will be proposed in the next section so that the users can have access to such conceptual objects in a direct and natural way.

3.4 The visualization objects formalization

The concept of **visualization objects** that will be proposed in this section, will define **three simple metaphors** – whose role can be easily understood by the user and which generalize most visualization techniques – corresponding to the three steps described in the previous section.

We propose to refine the concept of Haber and McNabb's **AVO** [38], with a slightly different meaning, using a hierarchy composed of three abstraction levels (Fig. 3.3):

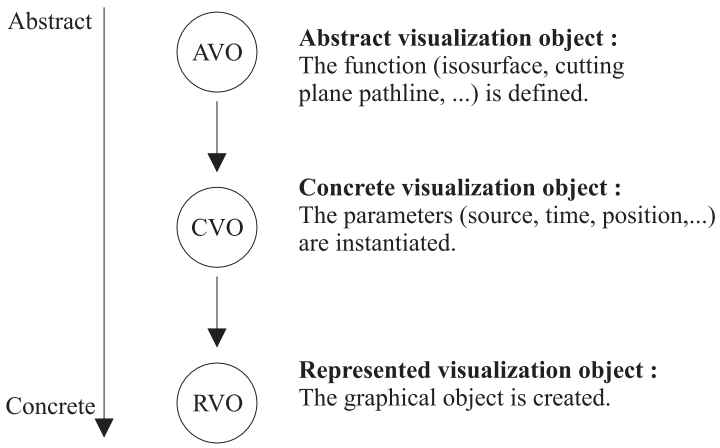


Figure 3.3: Visualization object hierarchy

- An **AVO** (Abstract Visualization Object) is a **function** that maps a **set of parameters** onto a geometric object with certain properties. Three of them can be identified because they play a particular role in the creation of the geometric object:
  - the **source**: the data on which the AVO acts;
  - the **time**: the time at which the AVO acts;
  - the **region of interest**: the place where the AVO acts.

  The cutting plane is an example of such an AVO, the source being a scalar field, the region of interest being a plane, and the color map being an additional parameter.

- A **CVO** (Concrete Visualization Object) is a 3D object which exists only in the conceptual world of the user. The "semi-transparent cutting plane of the temperature field in the plane $x = 7$ and at time $t = 22$" (with the corresponding color, reflectance and texture) is an example of a CVO. A CVO thus encompasses attributes including color, transparency, and so on. More formally:

  $$\mathrm{CVO} = \mathrm{AVO}(\mathit{parameters})$$

  where $\mathit{parameters} = \{\mathit{source}, \mathit{time}, \mathit{region\ of\ interest}, \ldots\}$

  In the example we mentioned, the source is the temperature field, the time is $t = 22$, the region of interest is the plane $x = 7$, the first additional parameter is the color map, and so on.

- For some purposes, it is necessary to isolate the **pure geometry** of a CVO; it will be called a **GVO** (Geometric Visualization Object). As an example, a user might consider the GVO "semi-transparent cutting plane of the temperature field in the plane $x = 7$ and at time $t = 22$", and, later on, view it as a non-transparent object whose coloring expresses the salinity; together with this attribute, the GVO becomes again a CVO.

- A **RVO** (Represented Visualization Object) is the approximation of the (ideal) CVO which is manipulated by the graphical system. Whereas in our example the CVO "semi-transparent cutting plane of the temperature field in the plane $x = 7$ and at time $t = 22$" has a **smooth contour**, the corresponding RVO is a **polygon** composed of a set of triangles which is projected onto the screen.

Several advantages will issue from this new approach as will be demonstrated in the next sections of this chapter. But first, let us compare this concept with the aforementioned metaphors.

3.5 Comparison with other abstractions

3.5.1 AVO, modules and visual programming

Haber and McNabb's definition of the AVO as well as their idea of providing the users with an environment in which "the sequence of transformations could be configurable at run-time thanks to visual programming" gave birth to modular visualization environments.

Although the visualization objects concept is inspired by the visualization process formalization given by Haber and McNabb, it differs fundamentally in the following points:

- The notion of AVO does not correspond to the same abstraction: in our case, the AVO provides an abstraction for the **visualization mapping** whereas in Haber and McNabb's definition it provides an abstraction for the **result** of the visualization mapping, which corresponds to the RVO in our concept.

- The degree of abstraction for users at the application level is higher in our concept: the dataflow paradigm and the modules – according to their authors [38] – were primarily an implementation choice in order to allow for a concurrent execution. This explains the functional redundancy of the modules in the modular visualization environments, and the strong data type of their inputs and outputs. On the contrary, the visualization objects concept is based on a top-down approach targeted to end-users. A visualization object can be seen as a **pattern of**

modules avoiding redundancy at the application level and responsible at the same time for data query, data transformation and visualization mapping.

- In our concept, the users interact with an environment that is object centric rather than data processing oriented.

- The purpose of dataflow and visual programming was to allow a definition of the visualization mapping interactively without having to write new code – what seems ideal – whereas in our case the visualization mapping is encapsulated and has to be implemented once and for all in the AVO. This is a deliberate choice, more realistic, because the task of defining the visual mapping becomes rapidly very complicated and recourse to complex programming is required. And we consider that it is not the role of end-users. Thanks to our approach, the division of labor can be more effective and productivity, as well as reuse, can be increased.

- Space and time are not treated as particular dimensions in modular visualization environments – this can be an advantage for multi-dimensional analysis, but this implies additional developments for our particular class of problems – especially in order to implement interactive selection mechanisms. In the visualization objects concept, space and time dimensions are treated as particular parameters.

3.5.2 Sparts

The concept of sparts is powerful – it has proved to be efficient, intuitive and versatile in several case studies – and it is very difficult to challenge it. The concept provides interactive exploratory visualization as well as the ability to define at run-time the behavior of a spart – the transfer function – using a simple scripting language. However, we see two reasons, one conceptual and one technical, why the visualization object concept is more powerful.

The conceptual limitation of sparts is the fact that certain visualization techniques cannot – or only with difficulty – be implemented using this concept because any visualization result has to be decomposable into simple graphical primitives released by the sparts. The graphical abstraction is not sufficient enough to accommodate visualization techniques such as, for instance, sophisticated probing (see Fig. C.4) or texture techniques (see Fig. C.3). The scripting language does not help to remedy this problem, because it is not general enough to cover the most important needs of the users.

The technical drawback – as granted by the authors themselves [20] – is due to the fact that the system only supports data that are sampled on a

regular grid. Because it does not provide any abstract interface to the data, every dataset has to be resampled.

3.5.3 PGO and Glyphmaker's glyphs

In the case of the Parametrized Geometric Objects in CSE and the Glyphmaker's glyphs, the binding of the graphical primitives to data, although interactive, is limited to simple shapes. Indeed, it is very difficult, for instance, to produce realistic texture representing a flow field with the help of a few line segments and boxes. In fact, the primary goal of PGOs and glyphs was to be used as local probes and not to accommodate more global visualization techniques like cutting planes or isosurfaces. As is the case for sparts, the graphical abstraction level is insufficient to describe a large set of visualization techniques.

3.5.4 Actors

The work achieved by the **VTK** team demonstrated that an object-oriented approach providing a unification of visualization techniques and an abstraction of the data source is feasible. At the moment, the performance is not completely satisfactory because the abstract interface to the datasets requires many more computations in order to query the data and because the render engine is not yet fully optimized. For many people this is not acceptable nor powerful enough, but we are convinced that the object-oriented approach with abstract interfaces is the only way to achieve a satisfying level of reusability.

Our own work goes in the same direction, but it relies on a higher level of abstraction and is closer to users' requirements. Indeed, the concept of actors generalizes graphical objects **only** in terms of their graphical properties, for instance position, color or shading and does not abstract the functional aspect of the visualization techniques, for which external mappers are responsible.

Furthermore, actors do not integrate any notion of time, although it is a crucial issue for the visualization of time-dependent datasets. The only workaround consists in managing this parameter outside of the actor – by means of dedicated homemade components – as is the case in modular visualization environments. There is no high-level event management associated to the actors, so that it is very difficult to implement standard mechanisms such as object selection or manipulation.

It can be stated that VTK is a excellent object-oriented implementation of a dataflow system – it is described as such by its authors themselves – whose goal is to provide the visioneers with a reusable toolkit so that they can build dedicated applications more efficiently. It does not offer any metaphor at the application level to the users we consider because of the

necessary recourse to programming. Compared to the visualization objects concept, actors offer an abstraction level equivalent to the RVO metaphor, but nothing more.

3.6 Easy to learn thanks to programming by example

We start with an analogy. Let us think about the difference between a general programming language and a spreadsheet processor. On the one hand, a general programming language is certainly more flexible and much more powerful than a spreadsheet processor. On the other hand, spreadsheet processors are used throughout the world by a high percentage of clerks in all kinds of businesses and administrations, not to mention their managers and numerous scientists; the number of their users far exceeds the number of programmers who use a general programming language. Most of these clerks do not have a clue of what computer scientists call programming; but, setting up a spreadsheet is definitely a form of programming, i.e. establishing more or less complex relations between quantities, whose value can be set afterwards and modified at will. The point is that

- a spreadsheet is, first of all, a rectangular table, which is a structure that is both ubiquitous and well understood by these clerks;
- notwithstanding this simple structure, the functionalities offered by a spreadsheet processor cover an unexpectedly high proportion of the computational needs of many of their users;
- the user establishes the relations between the entries in a **concrete** and **intuitive** way by setting up one instance and by transporting a dependency from one place to others; it is **programming "by example"**.

It is our thesis that, in order to be accessible to non-programmers, the visualization systems have to pass **from the abstraction level of a general programming language to the abstraction level of a spreadsheet**. Only if this condition is satisfied, will these environments, to a large extent, enable the end-users themselves to satisfy their own needs.

This is exactly what the visualization objects paradigm offers. It allows for **familiar and easy to learn manipulations**: for instance, copying of a cutting plane from one region of interest (ROI) to another, or from one time step to another, produces a new cutting plane that inherits the parameters of the original and refers to the new ROI or to the new time step. More precisely, what happens is the following:

i) the original cutting plane, a CVO, is the result of the application of an AVO (i.e. a function) with a set of parameters (e.g. $t = 22$, "semi-transparent object") to the ROI (plane $x = 7$); and

ii) the copy operation applies this AVO with the same set of parameters to the new ROI or to the new time step.

Such a move in space can, for instance, be applied to a probe so as to compare the values of a field in different points; a copy in time of a cutting plane shows the evolution of the field (see Fig. C.25). For the user, programming is thus replaced by defining a graphical function by its effect in a particular situation and by updating its parameters with copy and move operations.

3.7 Extensibility, rapid prototyping and reuse

Rapid prototyping is an important issue in scientific visualization. Dedicated visualization tools often have to be developed in order to satisfy the specific needs of users. Usually, a rough visualization tool is implemented and users test whether it is actually usable or not. In this situation, a platform offering the possibility to quickly implement a prototype, and to refine it afterwards, is useful.

The simple and unique conceptual framework offered by the visualization objects concept provides the unification of most visualization techniques. Thus, one can imagine a visualization platform in which new visualization techniques are implemented and inserted, for instance, in the form of plug-ins. Hence, a visioneer can concentrate on the visualization technique itself, and, as far as he conforms to the API provided by the platform, he is able to implement a new prototype rapidly. The platform does the rest.

An object-oriented implementation of the visualization object concept would certainly improve reuse of visualization techniques, on the one hand, because code is encapsulated – hence only local modification of the code are necessary as well as learning of the classes provided by the platform's API – and, on the other hand, thanks to the mechanism of inheritance.

3.8 Benefits for advanced architectures

The visualization objects concept offer new perspectives for collaborative work and computational steering in scientific visualization. Indeed, the three distinct abstraction levels (AVO, CVO, RVO) provided by the visualization concept make possible versatile configurations for a client-server architecture. Of particular interest are the lightweight client/high performance server architecture well-suited for remote visualization service and the

n-tier client-server architecture for computational steering and collaborative work. Both configurations will be described in the following sections.

3.8.1 Remote service for scientific visualization

Visualization of complex datasets requires high performance computing and, in most cases, help from a visioneer. Because many researchers do not have these resources at hand it could be useful that dedicated computing centers offer such a service.

Using the visualization objects concept, it is possible to imagine a client-server architecture allowing remote visualization for the community of end-users. This kind of service would use the performance of dedicated servers required to offer a large variety of classic visualization techniques. It would also be possible to develop specific tools for users on demand and to integrate these tools into customized libraries.

In this configuration, it is assumed that the data to be visualized are initially uploaded to the server. Based on a lightweight client software – such as a Web browser, for instance – all computations in order to build the RVOs are achieved by the server. From the client side, the technological independence provided by the RVO abstraction makes possible an implementation on most platforms. Conceptually, the visualization clients can be seen as a simple terminal able to display RVO and offering a sufficient level of interactivity to the users. Undoubtedly, it is technically feasible to implement such visualization terminals by means of the large variety of industry standards such as OpenGL, OpenInventor, VRML, Java3D or Direct3D.

In the client-server architecture depicted in Figure 3.4, the main functionalities that a RVO terminal has to provide are :

- the display of a collection of RVOs;
- a navigation interface and interactive tools for the selection of the visualization objects;
- a way to update the parameters of the visualization objects; and
- a menu offering allowable actions on visualization objects such as creation, update, deletion, and so on.

The modus operandi is the following : 1) the client requests actions on the visualization objects by transmitting their corresponding CVO – containing the set of the instantiated parameters – to the server; 2) the server builds the RVO according to the CVO and returns the RVO and its corresponding CVO; 3) a collection of CVOs is stored both on the client and the server so that the user is able to update the visualization objects parameters.

This configuration offers the following benefits:

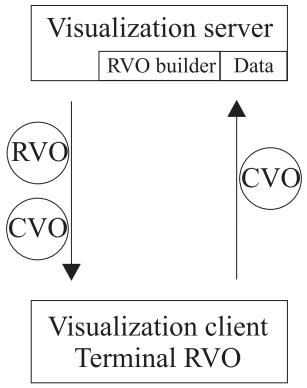


Figure 3.4: Client-server visualization with RVO terminal

- the client-server architecture can run accross a WAN (Internet) with a reasonable network bandwidth (a few Mbits/s);
- neither particular application installation nor tedious configuration on the client is required (ideally an Internet browser and a Java virtual machine would suffice), and
- the computational power and data management can be distributed transparently at the server side: for instance, the visualization can use several dedicated data servers and/or the production of RVOs can be achieved thanks to several visualization servers.

3.8.2 N-tier architecture

A recurring problem encountered in scientific visualization is to know where the data produced by the simulation have to be stored and where computations have to be performed. In certain situations, it can be more efficient to have the data located on the server, but in other ones it is necessary to transfer the complete dataset to the client – essentially for quick data access – that can be long for large datasets due to the limited network bandwidth.

Thanks to the separation of the functional aspect and the graphical result provided by the visualization objects framework, several n-tier configurations are allowed, depending on which tier is responsible for which kind of visualization object. The nature of the information exchanged between the tiers being of two kinds: either visualization objects or simulated or derived data. In Figure 3.5, for instance, a configuration with heavyweight clients is depicted. The clients are responsible to build the RVO according to a CVO and data, and the central server is responsible for data management as well as the coordination of clients if required. This is typically the situation we have with a cluster of high performance graphical workstations and

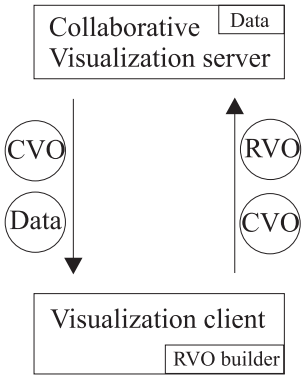


Figure 3.5: Client-server visualization with graphical workstations within a LAN

a dedicated data server connected within a local area network.

Moreover, because the source of data and the region of interest on which the visualization objects act are considered as CVO parameters, a set of versatile and efficient solutions for data management are offered. Indeed, first, the abstraction of the source of data allows for the setup of the datasets virtually everywhere on the network. Second, the time and region of interest parameters allow precise selection of a subset of the datasets. Thus the volume of data that has to be transfered between the server and clients can be minimized.

3.8.3 Towards collaborative work and computational steering

Collaborative work and computational steering are within scientific visualization arguably the most exciting challenges that will be addressed in the next decade.

Where collaborative work is concerned, the development of the Internet has multiplied by hundred the bandwidth of most connections between research centers all around the world. Internet backbones capable of a speed of more than 20 Mbits/s are no longer rare. This allows visualization systems for interdisciplinary teams – composed by specialists of several countries – in which it will be possible to investigate global and complex phenomena as, for instance, the influence of the oceans on the climate.

On the other hand, computational steering, defined [37] as "the capacity to control the execution of long-running, resource-intensive programs", can be considered as the ultimate goal of interactive computing. Implementation of a computational steering framework requires a successful integration of many aspects of scientific visualization and numerical simulation. These

aspects have to be effectively coordinated within an efficient computing environment.

We will not pretend here to solve either problem with a few paragraphs because, as stated by Johnson and Parker [51], "the creation of such a [computational steering] program is an extraordinary difficult task". And this is even more true for collaborative environments.

However, for the following reasons, we are convinced that the visualization object paradigm can offer conceptual solutions when addressing the problem of designing such complex visualization systems:

- Eight technologies have been identified by Bajaj [5] as most critical for a successful collaborative environment: 1) shared data management, 2) concurrency control, 3) distribution, 4) session control, 5) interaction control, 6) coordination control, 7) multimedia and graphics and 8) user interfaces. Obviously, concurrency control, session control, interaction control and coordination control require a set of atomic operations on some kind of "entity". This "entity" role can certainly be played by an extended version of the CVO encapsulating the additional information and parameters needed for these specific controls.

- The separation of the function achieved by the visualization object concept – the definition of what and how it has to be computed – from the result obtained – a graphical object and a set of properties – can be useful for the distribution of either computations or data.

- The interaction model offered by the general framework we propose – especially through direct manipulations of visualization objects – can very likely be augmented without any conceptual contradiction with the dedicated functionalities needed by a collaborative graphical user interface such as shared and private views, access rights management and sessions control. On the other hand, this framework does not offer any solution for the necessary multimedia – mainly phone and videoconference – integration, but it is conceptually compatible with this technology.

Let us conclude this chapter by trying to, first, sketch an n-tier architecture based visualization system able to combine both computational steering and collaborative work capabilities with the help of visualization objects and, second, to imagine a typical simulation-visualization session. As depicted in Figure 3.6, the proposed architecture would be composed of four tiers :

1. numerical simulator

2. data server

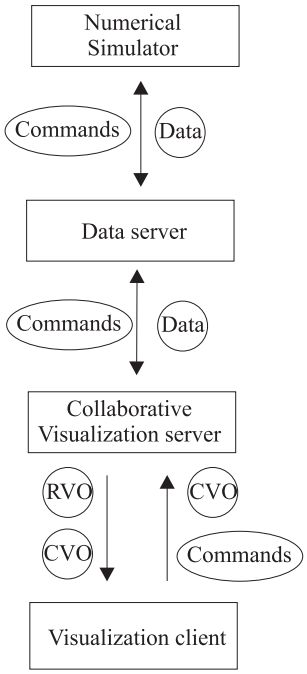


Figure 3.6: N-tier architecture for collaborative visualization

3. collaborative visualization server

4. visualization client

The visualization system is completely driven by the actions of several users connected into the collaborative environment thanks to their own visualization client. All actions are performed interactively on visualization objects that either represent the data or the simulation parameters as, for instance, the wind direction and strength. Commands like *start*, *stop*, *pause* or *resume the simulation* can be performed thanks to a specific command panel. Other commands dedicated to collaborative work, such as *open the session*, *share a view*, *give access rights to a visualization object* or *close the session* are also accessible to users through another command panel.

The users select regions of interest as well as time steps and, as is the case in the traditional non-collaborative framework, visualize the data by creating, updating and deleting visualization objects. In this process, the collaborative visualization server builds the visualization objects and is responsible for 1) control of concurrent access, 2) session control and 3) user coordination.

Behind the scene, the data server acts as a proxy and is responsible for the computation of derived data from raw data provided by the numerical simulator. The abstraction level offered by the parametrization of the source

of data, time and region of interest allows the data server to mix and match data that are either computed on the fly by the numerical simulator or pre-computed and stored on disk. Moreover the data server is able to interpret user commands and to control the execution of the numerical simulator by forwarding the commands if the data are not accessible in its cache memory.

Chapter 4

A new kind of visualization system

In Chapter 2 we saw why, in our field of interest, the adequate level for a visualization system is not currently achieved. **Special purpose applications** are easy to use but not extensible, while **modular visualization environments** and programming **libraries** require an important development effort even for small customizations.

We got the conviction that it would be possible to define a conceptual framework for all visualization techniques that helps to build a visualization environment satisfying both the requirements of ease-of-use **and** of extensibility.

This chapter starts by examining the needs of users and visioneers in more detail. It is of primary importance to understand the requirements from the point of view of each participant, because each of them has very specific needs.

One amongst the main advantages of the concept of visualization objects is the ability to build around it a visualization system having a classic graphical user interface derived from desktop applications. This point will be discussed in one of the following sections. We then consider more precisely how the concept of visualization object naturally leads to a visualization platform architecture which can be both extended and customized to the actual needs of users.

Finally, we will investigate data management because this aspect has a high impact on the performance of all visualization systems. We will propose an approach based on an interface to datasets rather than on datasets formats, as is often the case in existing applications. Finally, we will discuss the advantages and drawbacks of this approach.

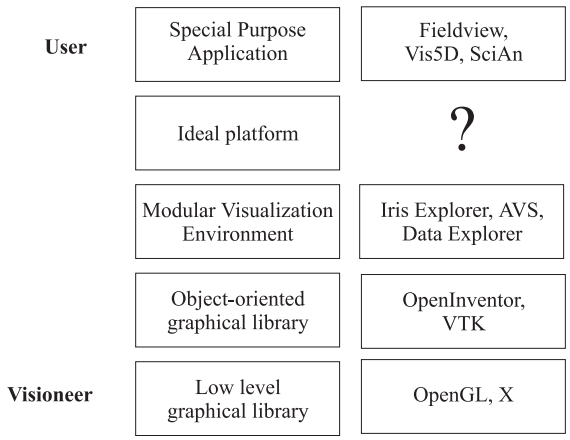


Figure 4.1: The missing layer

4.1 Bridging the gap

Figure 4.1 depicts a top-down view, from user's to visioneer's level, of the software is currently available. In the lowest layer, we find libraries which provide basic graphical functionalities, for instance line and triangle rendering. Rendering algorithms are usually implemented directly in the hardware graphical chips. On top of them, more advanced libraries were designed to enhance the productivity of developers thanks to the object-oriented technology. They offer off-the-shelf 3D graphical objects, like cube, cones and sphere, as well as components to organize and manipulate them. Generally speaking, it can be stated that these two layers require the skill of a visioneer, because programming is definitively not an easy task. Visioneers needs these libraries to develop special purpose applications.

From the usage point of view, modular visualization environments are layered on top of advanced graphical libraries, because, as we saw in the previous chapter, they address the needs of advanced users and visioneers. Finally, the special purpose application layer is, of course, dedicated both to casual and advanced users, but this particular kind of software is not extensible.

As suggested previously, our experience with different users showed that a layer is missing between **special purpose applications** and **modular environments**. This layer could be provided with some kind of customizable application which, from the user's point of view, looks like a special purpose application but provides, from the visioneer's point of view, an application programming interface (API) that is general enough to extend the platform in a reusable way with new functionalities as well as dataset drivers. In the remainder of this thesis we will refer to this kind of application as a

platform and we will lay the foundations required to build it.

4.2 A better division of labor

Coming back to the three graphs of the Chapter 2 concerning the customization effort, what we aim at is illustrated by the appended fourth graph (see Fig. 4.2): in an initial phase, an ideal platform should allow the visioneer to invest a **reasonable effort** to tailor a set of tools satisfying the needs of a particular user. Later on, the visioneer should be needed only for specific interventions, for instance to integrate an entirely new tool. In a certain sense, such a platform is extensible, but minimizes the total development effort through a maximum of reuse.

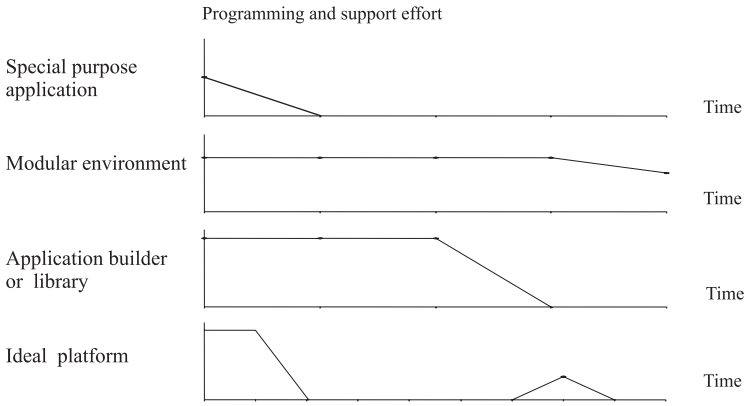


Figure 4.2: Division of labor

In such a situation, the roles of each participant in such an ideal visualization platform can be clearly identified and illustrated in Figure 4.3. The task of the visioneer is to customize a set of visualization tools and datasets drivers so that the user is able to analyze, all by himself, the data produced by a numerical solver.

4.3 Requirements

Satisfying all visualization needs of users, even in a restricted field of interest, with just one general environment is certainly an impossible task. Because there are so many ways to map data into graphical information, no environment is able to anticipate the needs of users. However, we think that it is possible to build an environment covering a high percentage of the needs by taking into account the major requirements of users and of visioneers.

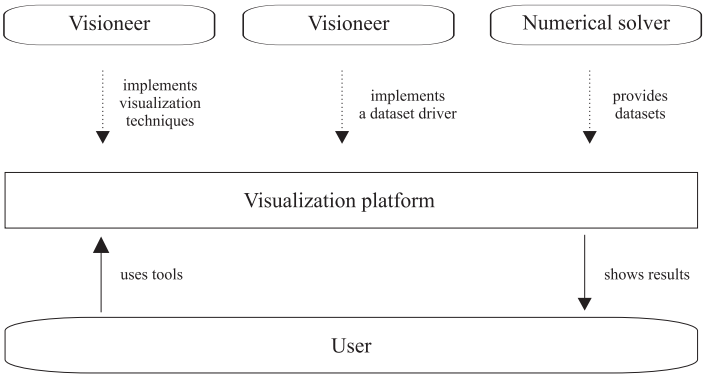


Figure 4.3: Roles of the actors

In this section we will rate the importance of the different requirements and identify the minimal set which an ideal environment has to satisfy.

4.3.1 User's needs

To be efficient, users need applications which are customized and easy to use. More precisely, the requirements such a platform has to meet are the following:

To offer an easy-to-learn graphical user interface: The platform must provide an **easy-to-learn graphical user interface** having the look and feel of desktop applications. It has to run in an environment the users are familiar with, and it should be possible to exchange data with common applications and to collaborate with other users.

To be extensible: It must be possible to **extend** the platform with **the most diverse visualization techniques**.

To facilitate the exploration of data: Analysis and exploration are not easy tasks and can in general be long and tedious because of the volume of data. It is the reason why, ideally, the user has to feel comfortable in the environment and use tools which make easy the detection of particular or unexpected phenomena. Interactivity plays an important role in this task and latency time caused by the data fetch or by the computation time needed to produce the visualization itself have to be avoided.

To accommodate adequate peripherals and collaborative work: It must be based on standard peripherals, i.e. a mouse and a keyboard, as the majority of users usually have no access to more sophisticated hardware today, but it must be able to accommodate virtual reality

peripherals and to be adapted to collaborative work in the future. Efficient environments have to integrate appropriate peripherals (many of them still have to be discovered) dedicated to the manipulation of objects living in three-dimensional space. For this reason, it is not sufficient to simply extend current two-dimensional graphical packages to the third dimension.

To provide direct and indirect manipulations means: Users must be able to **manipulate** directly (i.e. by point and click) and indirectly (i.e. setting or updating properties in dialog boxes) the obtained representations.

To manage time-dependent data: This requirement has to be addressed conveniently by modern visualization environments. Routinely, numerical solvers address time-dependent problems like unsteady flow simulations. In order to solve physical equations numerically, time is generally discretized and numerical solvers produce the results timestep by timestep. The continuity of time can then be recreated using either:

- flipbook technique: images are displayed one after the other, as in a cartoon;
- keyframe animation: a set of interpolated images is displayed between two reference frames.

Not all visualization environments provide adequate tools to manage the evolution in time of three-dimensional structures, like the movement of an eddy, so that physical phenomena can be more easily understood. Moreover, convenient means of presenting these time-dependent three-dimensional results to a large audience, on the Internet for instance, have still to be improved.

To handle a broad range of datasets formats types: The platform must be able to handle a broad range of datasets types.

4.3.2 Visioneer's needs

From the point of view of a visioneer, it is of primary importance that small customizations do not require a large development effort. This is very important at the beginning of the investigations as a large effort has to be invested to tailor a set of tools. But this should stay true when users ask for limited refinements or a new tool. To be productive, visioneers mainly need:

a sufficient level of abstraction: a sufficient level of abstraction is decisive to enhance reuse and customization of the visualization techniques.

a programming environment: by a programming environment, we mean that an ideal platform has to provide a set of basic components, e.g. a color map chooser, and facilities, e.g. selection mechanisms, which are needed in every visualization problem.

4.4 An easy-to-learn interface

What makes the concept of visualization object interesting is the fact that it naturally leads to a visualization platform which looks like a desktop application. In such a platform, visualization objects are produced by tools implementing the visualization techniques and which act directly on the data. By **easy-to-learn interface**, we mean that it seems natural for a user to undertake an action in order to get a result. The question of a user becomes "What kind of effect will I get if I undertake this action?" instead of "How can I customize the software in order to get such a result?".

Moreover, direct manipulation of visualization objects becomes possible because the platform manages objects as entities and the display no longer consists in producing a grid of pixels but in rendering a collection of individual objects. It can be stated that this concept provides similar advantages to those provided by line art software, in opposition to software based on "pure" bitmaps. It is well-known that with the second kind, tools work directly at the pixel level and that, once a modification is committed, it is no more possible to modify directly the properties of a painted "object".

Thus, it follows from the abstraction that most metaphors of desktop applications can be applied to visualization objects. Copy and move of visualization objects can be provided to users through the classic copy, cut and paste commands. Toolbars and properties sheets can offer creation, update of parameters, hiding and deletion mechanisms. Because the state of visualization objects can be encapsulated, it is possible to implement undo and redo capabilities as well as store and retrieve functionalities.

Of course, most of these features already exist in **special purpose applications** equipped with a reduced set of tools. But the point is that the concept of visualization object is general enough to support also the extension of the platform with customized visualization techniques benefiting automatically from all these capabilities as we will see in the next section.

4.5 Customization and extension

The object-centric orientation of the concept of visualization objects makes possible a quick and easy customization of a platform implementing it. The main advantage resides in the fact that a set of general functionalities can be provided at the platform level. They can be given through a general application programming interface (API) and the visioneer is able to concentrate

on the customization of the tools; hence they satisfy the actual needs of users, whereas the platform does the rest. In this situation, reuse can be greatly enhanced because tools can be designed to be as generic as possible.

Moreover the integration of a new tool is completely transparent for users and the platform offers a uniform graphical user interface for all tools and experiments.

4.6 On data management

The problem of data management is important for all types of visualization system. Unfortunately, it is difficult to solve this problem in a flexible, general and efficient way.

Today, almost all visualization systems work in postprocessing mode (see Fig. 4.4), i.e. the numerical solver produces and stores a huge volume of data that are analyzed **afterwards** with an ad hoc visualization environment. This is mainly due to the fact that in general the computer architecture needed for each task is dedicated. As the simulation process is usually computationally intensive, it must be run on a number cruncher, for instance on a parallel system. Data are then visualized on computers equipped with hardware graphical accelerators. In this approach, data management is mostly thought of in terms of dataset formats, because data have to be stored, and converters for each pair of solver/visualization software must be developed.

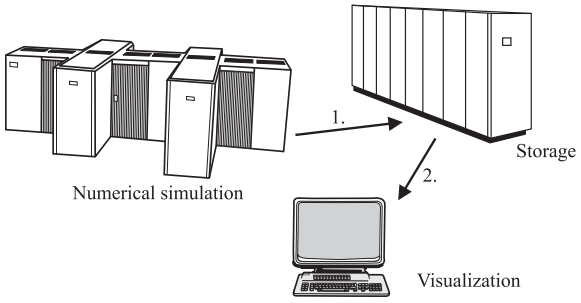


Figure 4.4: Pure postprocessing

Throughout this thesis, we needed to care about data management, even if it was not our primary concern, in order to be able to provide the users with visualization systems. We spent much time writing drivers for standard or proprietary datasets formats using reverse engineering and converting these datasets from one format to the other. It is the reason why we early on devised another approach consisting in defining an interface as a **set of methods of a data server** rather than by data formats; this would make the platform independent of the data source, which can then be implemented as a set of files or an area in shared memory, either local or remote. The

solution of a standard interface rather than a data format is used successfully since many years in the field of database management, where SQL was adopted as a standard. Assuming numerical solver vendors provide drivers for their datasets that conform to the standard, visioneers could avoid time consuming conversions and handle datasets more appropriately. Furthermore, this appears to be the best way to achieve a **complete separation** between data access and visualization, with the possibility of installing the numerical solver, the data server and the platform on different machines (see Fig. 4.5). This solution is often named the n-tier client/server architecture and is widely used in large management software involving high volumes of data.

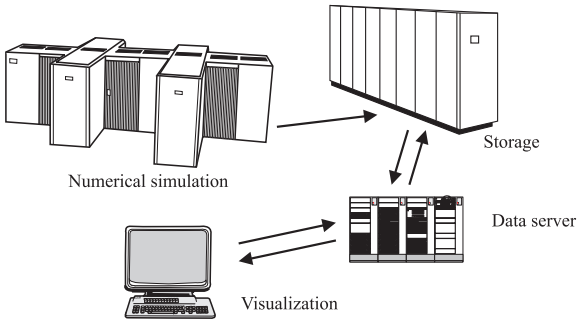


Figure 4.5: N-tier client/server architecture

Unfortunately the situation is more complicated in scientific visualization. The volume of data that has to be managed can reach many gigabytes and the response time has a major impact on interactivity. Another problem stems from the fact that the data server has to supply the visualization environment with interpolated or derived data: **derived data** are data produced by the evaluation of expressions based on **raw data** that are directly produced by the numerical simulation program. A classic example is a vector field given as raw data from which the scalar field of its norm has to be derived. As the complexity of the involved computation can be high, it may be impossible to provide these derived data as quickly as is required by the visualization. We think that these problems will have to be tackled in the framework of the emerging distributed object-oriented databases [52, 19] with powerful query languages.

Nevertheless the separation between data access and visualization has several advantages:

- it becomes possible to simultaneously visualize data contained in several datasets – in the case of interdisciplanary visualizations – or coming from simulations using different coordinate systems (e.g. meteorological and oceanographical simulations);

- datasets can contain data about objects other than scalar or vector fields, e.g. the bathymetry in an oceanographic simulation;
- the visualization of virtually any kind of dataset format without the need to convert every dataset to a given data format: the "only" thing that is needed is a dataset driver compatible with the interface;
- the user can choose among different drivers for the same dataset format which for instance implement different methods of data interpolation;
- it is possible to access huge datasets that do not entirely fit into memory;
- thus it even becomes possible to visualize data produced online by a numerical solver, i.e. without storing the complete dataset, or to accommodate future computational steering environments (see Fig. 4.6).

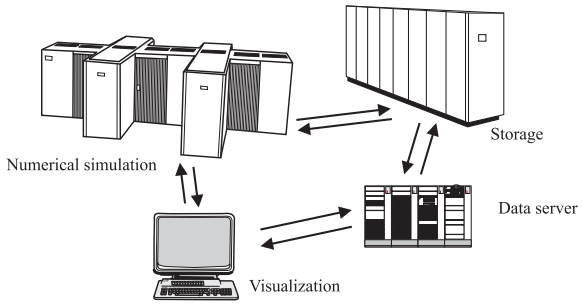


Figure 4.6: Computational steering

Certainly the major drawback of this general interface, is that the data server has to answer queries expressed in **physical** coordinates rather than in computational coordinates. In the case of simulations based on complex grids, the coordinate transformation and data interpolation are computationally intensive and require powerful processors. This is a large penalty when huge, unstructured and time-dependent datasets are involved; it explains partly why even a well-designed library like **VTK** [93] that offers an abstract interface to datasets is still not widely used outside the academic community.

However, it is assumed that the expected increase of the overall performance of the hardware and the possibility to distribute computational power over a network will soon enable an acceptable latency for the visualization of datasets of a few gigabytes through an abstract interface.

For all these reasons, we are convinced that an extensible visualization platform has to be based on an interface to datasets rather than on datasets formats.

Chapter 5

ZoomIn

ZoomIn was built in order to demonstrate that it is technically feasible to implement a visualization platform based on the concepts developed in this thesis. In this chapter, an overview of the architecture of the platform will be given and it will be shown how the problem of extensibility was solved thanks to object-oriented technology.

First, we will sketch the graphical user interface of the platform and describe its main components. Then, the extension capabilities of ZoomIn regarding visualization objects and dataset drivers will be examined in detail. This will be illustrated by means of a simple example in which a customized glyph for the analysis of the shear stress in a fluid is implemented. A short discussion about the choice of the hardware architecture, the language and the software libraries we used for the current implementation will follow, and finally, a short summary of the possible extensions of ZoomIn will conclude the chapter.

5.1 Graphical user interface

For the reasons mentioned in Chapter 3, the graphical user interface (GUI) of ZoomIn was designed so as to resemble those of classic desktop applications. It is composed of four main windows (see Fig. 5.1):

1. The **viewer** is a window which allows for both interactive navigation in 3D space and selection of graphical objects with a mouse. This window also contains the menu bar giving access to the main functionalities of the platform.

2. The **time control box** allows for an interactive navigation through time and provides flipbook animation facilities.

3. The **properties sheet** displays the properties of any selected visualization object; users can change these values. The buttons in its

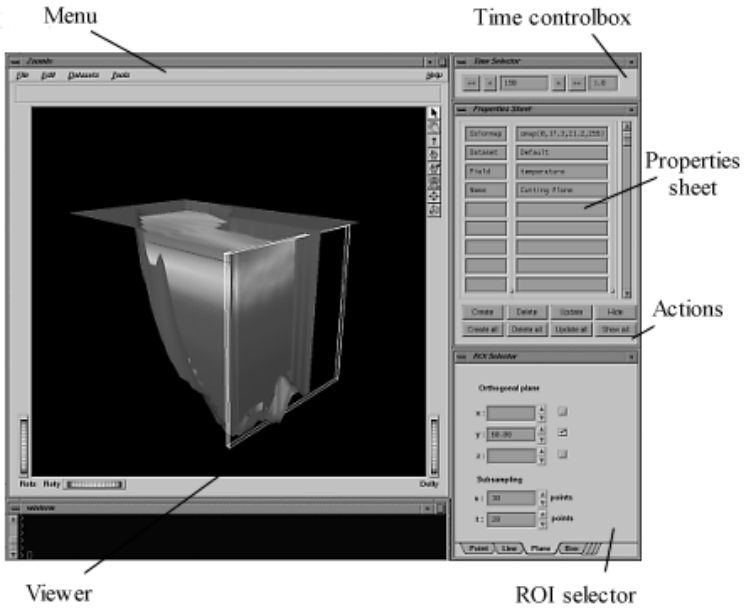


Figure 5.1: Overview of the graphical user interface

bottom part are used to create, update, delete and hide visualization objects.

4. The **ROI selector** is used to select the parameters of a region of interest. These parameters can be introduced by hand, with sliders, or interactively, by picking and dragging manipulators (see Fig. C.7) inside the viewer.

Typically, the progress of a visualization session can be summarized as follows:

1. The user creates a new visualization session with the classic **File/New** menu and is asked for a few parameters (e.g. the minimum and maximum real world coordinates and the temporal parameters).

2. He attaches external datasets using the **Datasets** menu in order to have access to the data.

3. He proceeds to the exploration of the data using the visualization objects provided by the **Tools** menu. After selecting a tool, a region of interest, and setting the properties, he invokes the creation of the desired visualization object with the **Create** button located in the properties sheet.

4. By picking a visualization object, the user can apply different actions to it either with the corresponding button of the properties sheet, or via the **Edit** menu:

   - to update it by simply modifying the entries in the properties sheet;
   - to hide it;
   - to delete it;
   - to copy, cut and paste it into time and space, whereby all replications of a visualization object remain in relation with each other, so that one can update (or delete) them all together;
   - to undo and redo the last **n** operations.

5. After their investigations, the user can save the state of the session in order to resume later on as well as print or output pictures, movies and VRML scenes by selecting the corresponding command in the **File** menu.

5.2 General architecture

Because of the technical requirements, implementing ZoomIn was not a simple task. Even for a prototype, the versatility we wanted for its components forced us to use advanced programming techniques. Fortunately, object-oriented technology helped us to achieve this goal. The analysis and design capabilities of this technology made the definition of abstract interfaces, mainly for visualization objects and datasets, relatively easy [88]. The general problem could be decomposed in simpler ones while keeping the code size small (about 25 thousands lines of code), the class number modest (about 50 classes), and the development time short (2 man-years).

5.2.1 Subsystems

The kernel is composed of a set of five subsystems (see Fig. 5.2), each of them being responsible for a specific task. They all collaborate tightly in most processes, like, for instance, the creation and update of the visualization objects. Their role will be described briefly in the following sections.

5.2.2 Core application

Playing the role of controller of the other four subsystems, the core application is also responsible for the graphical user interface (GUI). It relays the users commands to subsystems by means of a command processor [36, 118]

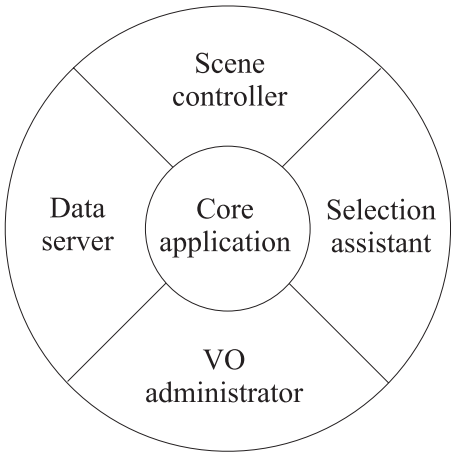


Figure 5.2: ZoomIn kernel

which maintains a stack of users actions. This technique has two advantages: first, calls to object methods can be decoupled from the GUI, which enhances the portability across different GUI systems and lets one envisage a three-tier client/server architecture. Second, this makes possible an implementation of **undo** and **redo** functionalities because a commands history is maintained by the command processor.

The core application manages the state of a visualization session as if it was a document. Thus, open and save mechanisms are provided by this subsystem. Moreover, it maintains a clipboard allowing one to copy and paste visualization objects used for the exploration of data during the session.

5.2.3 Scene controller

In order to allow for the visualization of time dependent simulations, the platform maintains a **scene** for each time step a user wants to explore. Concretely, a scene is a collection of CVOs. The scene controller can provide a simple flipbook animation while displaying the scenes one after the other, thus recreating, artificially the dynamics. Picking and manipulating visualization objects is achieved by this subsystem.

5.2.4 Selection assistant

The selection assistant keeps the information about selections and transmits them on demand to the other subsystems. For instance, when a user selects a visualization object, the scene controller automatically notifies the selection assistant that an object has been selected. Upon the creation and update of visualization objects, this subsystem delivers the information about the

region of interest, the time step and the visualization object which are currently selected.

5.2.5 VO administrator

The VO administrator is the principal subsystem of the platform because it acts as a central repository for the visualization objects (AVOs, CVOs, and RVOs) and manages their creation, update and deletion.

In this subsystem, the **Exemplar** instantiation mechanism [22] is implemented so as to instantiate visualization objects whose class was unknown at compilation time. This allows one to integrate completely new visualization objects into the platform, which can therefore control them without any need for recompilation. According to the exemplar technique, there is a unique AVO instance for each visualization object class. This unique exemplar is instantiated when the user creates a visualization object for the first time. A list of all exemplars is maintained by the VO administrator.

Each time a user invokes the creation of a visualization object, the following events happen:

1. an instance of a CVO parameterized by the time, ROI and properties is created;

2. the CVO instance is passed to the AVO exemplar method which builds and returns the RVO, and

3. the CVO and RVO are inserted into the scene corresponding to the current time step.

Afterwards, if a user invokes the copy and paste commands on a selected visualization object, it is cloned and a dependency link is maintained with the original visualization object (see Fig. 5.3). It is therefore possible to update one property of all dependent visualization objects with only one command, and, according to the same principle, to delete them all simultaneously.

Moreover, because the construction of a RVO can be time consuming, the VO administrator provides a late instantiation mechanism for the RVOs. These are built only when a user asks for the display of the scene in which they reside. Thus, a user does not have to wait for their creation when he asks for a copy or update of them for several time steps.

5.2.6 Data server

As stated previously, solving the problem of data management is hard. For the reasons invoked in Chapter 3, we decided to address it by defining abstract objects and interfaces (see Fig. 5.4) responsible for the delivery of the data:

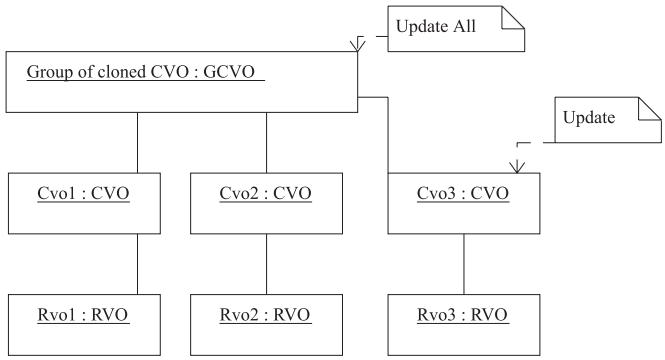


Figure 5.3: Dependencies between cloned VOs

- the **dataset**, whose role is to give access to scalar and vector fields and to the user-defined objects it contains;
- the **field**, which provides the raw data according to its type, i.e. a scalar or vector quantity;
- **user-defined objects** providing information which cannot be considered as a scalar or a vector fields as, for instance, geometrical information like the body of a car in a crash test simulation.

These objects are controlled by the data server subsystem which is subdivided into a dataset manager and an expression evaluator. The dataset manager is responsible for loading the datasets and acts as a central repository giving access to them. The expression evaluator combines raw data with mathematical and logical operators into expressions. Expressions can then be given as a source of data for a visualization object.

Datasets give access to the fields and the user-defined objects they contain. Due to the abstract interface, fields and user-defined objects have to answer queries expressed in physical coordinates. Thus, they are responsible for coordinate transformation and data interpolation if these are necessary. If no driver for a certain source of data is available, a visioneer can develop a particular dataset driver and integrate it into the platform by implementing some classes:

- the class of the new dataset by deriving it from the abstract dataset class;
- classes for the fields by deriving them from the abstract field class;
- classes for user-defined objects by deriving them from the abstract user-defined object class.

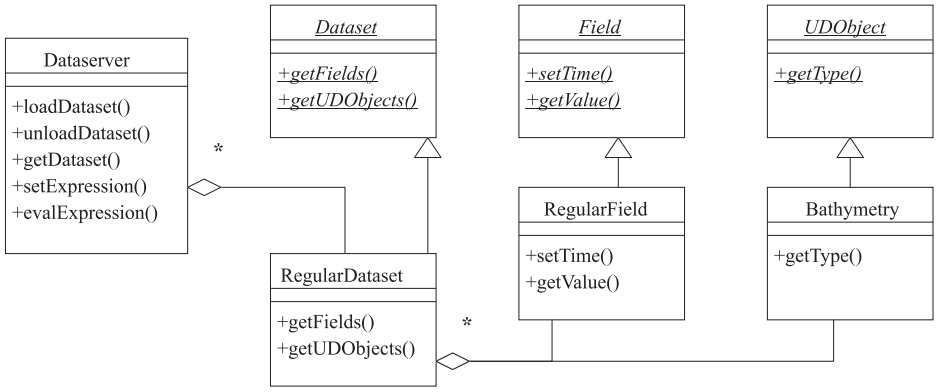


Figure 5.4: Class hierarchy for data management

As was the case for visualization objects, a mechanism allows for the integration of new datasets without having to recompile the platform. This mechanism was implemented within the data server.

5.3 Implementing the visualization objects

From the implementation point of view, the platform does in fact **not know** what the objects it manipulates really are. Thanks to the conceptual framework, an abstract interface was defined so that all visualization objects derived from it can be manipulated blindly by the platform.

Thus ZoomIn can accommodate the large majority of the techniques which are available in different packages, be it isosurfaces or cutting planes [35], particle traces [114, 54], textures [74], vortex extraction [6], glyphs or icons [81] (see Fig. C.1–C.8).

5.3.1 Implementing a new visualization object

Any visualization object is implemented with the help of three classes:

1. the AVO abstract class from which all visualization objects classes derive (in the sense of object-oriented programming);

2. the CVO class containing the time and ROI parameters as well as those introduced via the properties sheet, for instance the data source; all further commands, like copy and paste or delete, are applied to the instances of this class;

3. the RVO class, which is responsible for the visual aspect of the visualization object; it builds and maintains the graphical primitives composing it.

In order to add a new visualization object to the platform, the visioneer has in particular to take the following actions:

- to subclass a new AVO class from the abstract class AVO;
- to declare and to initialize the default properties of this new AVO class;
- to implement a method which declares the set of the supported ROIs;
- to implement the method that builds the RVO according to the parameters contained in the CVO; and
- to compile the source code of the new AVO and to link it within the library.

Inheriting from the AVO abstract class automatically provides – i.e. without any additional development by the visioneer – the functionalities offered by ZoomIn: the selection of region of interest, the modification of parameters using properties sheet, the creation-deletion and copy-cut-paste mechanism, the undo-redo capabilities as well as the the storage and retrieval of any visualization object. In order to simplify even more the implementation of a new visualization object class, a template file is provided to the visioneer.

A recent experiment proved that non-trivial visualization techniques (see Fig. C.26–C.29) could be implemented rapidly – within 125 hours – by a programmer who only knew about the API of ZoomIn [18]. Of particular interest is the fact that, once the visualization techniques were integrated into ZoomIn, users who did not write any line of code could use them very easily and interactively (see Fig. C.9–C.25).

5.3.2 Regions of interest

The region of interest (ROI) is one of the main parameters involved in the creation or update of almost any visualization object. It represents the portion of the spatial domain the user wants to concentrate on. The platform provides **direct** and **indirect** selection mechanisms. The **direct** selection mechanism is available through manipulators controlled by classic devices like a mouse or a space ball. In order to give precise values to distances or positions, **indirect** means like dialog boxes or sliders are required. This is particularly important as soon as the visual cluttering reaches a certain level: direct means then lead to long and troublesome manipulations. There are two types of ROI:

- the **basic ROI**: in the current implementation, it can be a point, a line, a plane, a box, or a sphere;

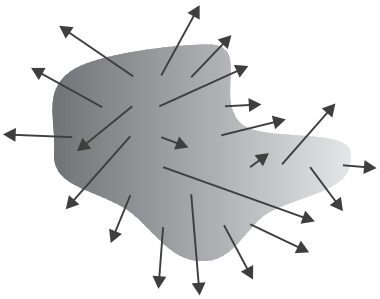


Figure 5.5: Hedgehog

- the **GVO ROI**: the geometrical shape of an existing CVO.

This definition unifies representations as diverse as, for instance:

- a color map (see Fig. C.9) representing the temperature on a plane (a basic ROI),
- a hedgehog (see Fig. 5.5) expressing the velocities on a temperature isosurface (a GVO ROI),
- a glyph expressing a concentration evolution (see Fig. C.20) on a set of points (a set of basic ROIs).

5.3.3 An example

It is not easy to find a striking visualization technique for the deformation of an infinitesimal volume of fluid at a given place and to monitor this deformation by the drag of the velocity field. The classical arrow plots or a bunch of streamlines do not give, at any rate, a precise idea of the local phenomena. For illustration purposes, we will develop a convenient AVO which applies the data to a graphical object composed of two parts: a reference cube of fluid, which is discretized in several smaller cubes, before and after the deformation. This AVO has the following parameters:

- the velocity vector field to be used;
- the duration of the effect of this velocity field;
- the color and the transparency factor for each of the two parts (to allow for comparisons between them);
- the size and the subsampling.

The pseudocode in Figure 5.6 gives an idea of the modest programming effort which is needed for the implementation. The visualization object depicted in Figure C.8 shows an example of the obtained result.

```
ziRVO*
ziStrainGlyph::makeRVO(ziCVO* cvo) {
  ziPropertyList cvopl = cvo->getProperties();
  float deltat =  cvopl["Delta time"].getValue();
  float refTransp = cvopl["Refcube transparency"].getValue();
  ...
  ziROI* roi = cvo->getROI();
  if(roi->getType() == ZI_ROI_POINT){
    ziRVO *rvo = new ziRVO(cvo);
    roi->getParameters(position);
    SoSeparator *strain = new SoSeparator;
    rvo->getRoot()->addChild(strain);
    strain->addChild(refCube(size, position, refTransp));
    strain->addChild(defCube(size, position, defTransp));
    return rvo;
  }
  else
    return NULL;
}

ziStrainGlyph::ziStrainGlyph(ziExemplar ex):ziAVO(ex){
  _properties["Name"] =
         ziProperty("Name", ZI_STRING, "StrainGlyph");
  _properties["Delta time"] =
         ziProperty("Delta time", ZI_FLOAT, "0.01");
  _properties["Refcube transparency"] =
         ziProperty("Refcube transparency", ZI_FLOAT, "0.0");
  ...
}
```

Figure 5.6: Implementing a new visualization object

5.4 Current state of implementation

ZoomIn is composed of a kernel and two shared libraries, one containing the visualization objects and the other containing the datasets drivers (see Fig. 5.7). Fulfilling the requirements imposed in this thesis, the platform can be extended at will and customized by compiling a set of visualization objects and linking them together into the shared library. It is possible to build several dedicated libraries depending on the tools that are necessary in a given application domain, for instance, one library for hydrogeology, and another one for oceanography. The shared libraries are loaded at runtime so that a fully customized application is provided to the user community.

Zoomin was developed in C++ under Unix. Generally speaking, the choice of the language and libraries as well as the computer architecture is always difficult. We decided to choose a Unix based Silicon Graphics work-

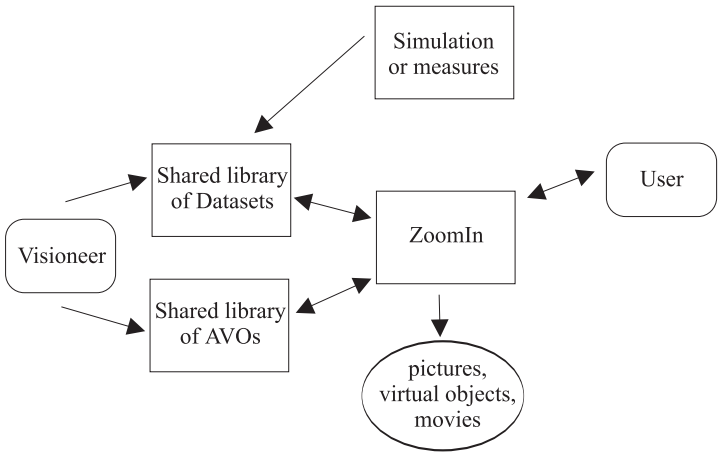


Figure 5.7: Data flow

station because, at the time we began the project, this vendor offered the best graphical performance of the market thanks to dedicated 3D hardware accelerators and powerful graphical libraries. The ability to run the numerical simulation on the same machine, thanks to the processor and internal architecture efficiency, was also an advantage.

For the development of ZoomIn we needed an object-oriented language because of the advanced capabilities we intended to implement in the platform; this holds in particular for the abstraction level required for the visualization objects. We wanted to use a widely supported object-oriented language so that we had the choice between C++ and Java. At that time however, Java was not mature and efficient enough, and had poor native 3D graphical libraries. Moreover we were afraid of interfacing Java with existing C++ libraries or with VRML. With regard to the graphical library, we hesitated for a long time between Open Inventor [119, 120], VTK [93], and OpenGL [69]. Open Inventor was chosen for two reasons: first, it is object-oriented – whereas OpenGL is not – saving us several thousands lines of code, and, second, it is more efficient than VTK, which is an important point to reach the level of interactivity users need.

Today the situation is different because graphical hardware of personal computers is competing with those of workstations due to the demand of the industry and the development of games. Object-oriented technology greatly helped us to keep the analysis and the design independent of the language and specific libraries so that it was possible to port ZoomIn with an effort of three man-months onto an Windows NT computer architecture using OpenInventor [**?**].

5.5 Further developments

Regarding the data management and the dataflow of ZoomIn, remaining problem has to be solved. A visualization technique can generate data – often related to the geometry of the graphical object produced – which have to be visualized at its turn. This can happen, for instance, if a user wants to reinject the numerical value of the volume delimited by an isosurface into the dataflow in order to be able to visualize it with another tool. A possible solution to this problem is to add a functionality to the data server: the ability to directly create virtual datasets, fields and user-defined objects from the visualization objects. Moreover, the abstract interface as well as the data server have to be augmented with functionalities dedicated to computational steering. Although this naturally fits into the concept of ZoomIn and seems technically feasible due to the platform's architecture, it has not been done in the current implementation.

Although the GVO ROI is a powerful abstraction allowing to combine different tools and to drastically enhance reuse, this mechanism is not yet available in the current implementation because of the technical difficulties it implies. Indeed, either the GVO ROI has to be computed for each demand, which leads to a heavy computation, or it has to be computed at the time of the creation of the RVO – as well as each time the RVO is modified – and stored within each RVO, which has a high memory cost.

Finally, it could be interesting to adapt the user interface to virtual reality equipments and to integrate functionalities allowing several users to collaborate within the same visualization session. Further developments in the field of computational steering can seriously be envisaged. Indeed, we think that the concepts developed in this thesis are general enough to support such extensions, and, thanks to the object-oriented concept, simplify the combination of these complementary technologies.

Chapter 6

Two case studies

At the beginning of this work, we initiated a collaboration with the geological Institute of the University of Neuchâtel. We were looking for representative users in order to validate our concepts and to test the new visualization platform ZoomIn. These researchers had advanced visualization needs but they had neither enough human resources, nor the skill required to write a customized application.

This collaboration was fruitful for both parts, beyond all expectations. It greatly helped us to refine the concepts and to implement them into something usable. "Our" users could successfully carry out their investigations, draw conclusions and illustrate publications with expressive pictures they generated by themselves.

In this chapter, an overview of the work we achieved together with them will be presented. We shall first expose the general context and these users' initial requirements. Then, we shall show how their needs could be satisfied by applying the concepts developed in this study. The main visualization results will be given in this chapter as well as a few problems encountered. At the end the experience we gained and the feedback we had from our users will be summarized.

6.1 Introduction

We started two collaborations because one group of geologists was interested in oceanography and another one in hydrogeology. In order to develop and validate their theories, both groups needed to visualize three-dimensional time-dependent field measures as well as results produced by numerical simulations.

The **Limnoceane Group** [73] was interested in the dispersion and the sedimentation of suspended particles in seas. They developed their own hydrodynamic model called **Prosper General Circulation Model** (PGCM) [123] based on a simplified version of the Navier-Stokes equations

using an hydrostatic approximation of the pressure and they implemented a corresponding numerical solver.

When we first met, they did not have any three-dimensional interactive visualization software to analyze their results. They only had a simple tool allowing one to print bidimensional cuts and a home-made animation tool for particles they developed by themselves using a two-dimensional graphical package. They realized that it was cumbersome to analyze the results using only static plots. For this reason, they were looking for an interactive three-dimensional visualization system that would make it possible to discover features which were not visible at all, or only with difficulty, with two-dimensional projections.

Our first approach was to use existing visualization software, such as **SciAn** [80, 102] and **Iris Explorer** [96, 105]. Unfortunately, **SciAn** is not extensible so that they were not able to analyze particle traces; on the other hand, **Iris Explorer** was to complex for them to use so that they never could master it and become autonomous.

The other group we collaborated with, the team of the Center of Hydrogeology (CHYN), was interested in the transport of colloids as potential carriers of contaminants in underground aquifers [40]. They bought a commercial solver for computational fluid dynamics and the visualization tool **Fieldview** because the one provided with the solver was not powerful enough.

Although there were some similarities between the problems the two groups tried to solve, they fundamentally differed on the following characteristics:

- Whereas the scale for oceanographers is the kilometer, it is the meter for hydrogeologists. As the relevant physical laws are scale dependent, the observed phenomena can be very different.

- In the case of the hydrogeologic problem, the main concern is for the steady state behavior whereas in the oceanographic problem, the phenomena are essentially unsteady.

- The background and the methods of both communities are quite different.

Thus, the tools required for the visualization could not be the same. The two following sections will explain in more detail how ZoomIn could successfully be applied in both cases.

6.2 Oceanography

6.2.1 Context

The Gulf of Lions is a wide bay of Mediterranean off the coast of Southern France, between Marseilles and Perpignan (see Fig. 6.1). For many years, the Limnoceane Group has been involved in the European project Metro-Med [34]. Metro-Med focusses on this area, because it is the place where the Rhone river flows into the sea with its load of pollutants. Moreover the continental shelf ends in this area and the shelf break is known to cause special effects on the circulation of water.

Figure 6.1: Gulf of Lions

It is the reason why, after the validation of the PGCM on the Lake of Neuchâtel [106], the Limnoceane Group parametrized it to study the Gulf of Lions [107].

6.2.2 Simulation

In this case study, a reference simulation of the hydrodynamics under wind forcing conditions is performed for a time period of eight days. This simulation, based on a $63 \times 39 \times 40$ nodes regular grid, consists of 7,680 time steps. It produces about 2 MB data per time step including the velocity, temperature and salinity value at each node.

The simulation starts with an initial situation corresponding to the mean monthly conditions. A wind forcing is applied for eight days. During the first four days, a uniform wind is blowing from N-W (300^o) with a constant speed of 15 m/s. At the end of the fourth day, the wind stops until the end of the simulation period.

6.2.3 Visualization

In many oceanographic problems, circulation of water is of primary importance because it usually has a direct impact on other quantities of interest, such as the salinity and the temperature, as well as on the dispersion, sedimentation, and resuspension of organic and inorganic matter in the form of particles. These particles move under the combined action of the advection by the fluid and of gravity. As these two forces may be of the same order of magnitude, the oceanographers' interest focussed on the role of downwelling and upwelling which accelerate, slow down, or even reverse the particles' vertical movement.

Because these phenomena are tightly coupled, we developed and integrated a variety of customized visualization techniques able to represent:

- the relations between temperature, salinity and velocity;
- the flow pattern, especially vortices and cells;
- the plume created by an input flow, in this case the Rhone.

As this study was the first real use of the platform, we had to implement the following AVOs from scratch:

- the cutting plane and the isosurface used to visualize water temperature and salinity;
- simple 3D conical glyphs expressing the direction and the magnitude of the velocity field; these glyphs are colored according to a scalar field, like salinity or temperature;
- a simple AVO representing a particle as a small colored cube, the color telling if the particle is sedimented or not;
- bathymetry with a transparency property; it is needed to provide a spatial reference frame to users;
- the contour line, which is useful to study the evolution of temperature and of salinity, as well as the behavior of the Rhone plume;
- the pathline;
- the particle tracer, to study the transport of particles.

Furthermore, three dataset drivers were implemented and added to the dataset drivers library: the first one gives access to the scalar and vector fields and provides trilinear interpolation because the data are referred to a rectilinear grid; the second one builds a bathymetry object from a two-dimensional depth map, and the third one provides access to a set of particles whose individual position is precomputed by the numerical solver.

6.2.4 Water circulation

The first application of ZoomIn in this case was to examine the evolution of the temperature and velocity through time.

When the wind blows, it causes a stress at the surface of the gulf. The water is pushed off the coast and, as the temperature of water is stratified, this movement brings cold water from the depth to the surface. This well-known phenomenon is called **upwelling** and is accompanied by a vertical shift of the isotherms which sink within downwelling zones and rise within upwellings. ZoomIn made possible a visual investigation of the up- and downwellings through interactive cuts and thanks to its animation capabilities (see Fig. C.9–C.12).

When the wind stops, inertial currents and internal waves appear due to the redistribution of the water masses and Earth rotation. In this situation, the users could observe one eddy in the south-western part of the Gulf (see Fig. C.13–C.14). They could also investigate the influence of the topography on the flow that exhibits important changes in the direction along the shelf break at a depth of about 30 meters.

6.2.5 Particles

Up to this point, ZoomIn was used as a pure postprocessing visualization tool. Velocity, temperature or salinity fields were first computed by a numerical solver and stored inside datasets. As seen previously, interactive tools integrated into ZoomIn like, e.g. arrow glyphs and cutting planes, were used to explore the vector and scalar fields.

After these first investigations, the users wanted to trace different classes of particles within the flow. For a visualization system, this requirement is harder to satisfy because it implies the visualization of data that have to be **derived** from the velocity field without losing interactivity, as users want to vary the particles' source and characteristics.

The classic method consists in computing, for a given source and settling velocity, the trajectories of these particles in the numerical solver and then visualizing the obtained results. However, this approach is not interactive, because it is necessary to run the numerical solver each time the source or one of the characteristics, such as the settling velocity, of the particles is changed.

An alternative consists in storing the velocity field computed by the solver and postprocessing it with another program, as was done with the Limnoceane Particle Transport Model (LPTM) [103], in order to get the advection of particles according to the velocity field and the varying characteristics. But even this approach is neither interactive nor integrated because it is still necessary to store the positions of particles for all time steps and to combine three different tools: the solver, the advection postprocessing and

the visualization program.

The versatility of ZoomIn made it possible to integrate LPTM as a particular tool inside the visualization platform [92]. The source of particles can be interactively given using the region of interest selector and the trajectories are obtained by solving the following differential equation using a fourth order Runge-Kutta [54] scheme applied to the ODE:

$$\frac{d\mathbf{p}}{dt} = \mathbf{v}(\mathbf{p}, t), \tag{6.1}$$

where $\mathbf{p}$ is the particle's location and $\mathbf{v}$ its velocity (including the settling velocity) at time t. The accuracy of the trajectory depends of course on the time resolution which is used; it is therefore necessary to store the velocity field produced by the solver at a convenient time rate.

In order to allow for a continuous release of particles, or to draw a pathline rather then the successive positions of particles a few parameters were added. The particles or segments of pathline can be colored according to a given scalar field, for instance the temperature, and a user-defined color map (see Fig. C.15–C.16).

Figure C.15 shows the trajectory of two particles released in the Rhone outlet that were dragged by the flow during eight days. The user could show that after the fourth day, when the wind is turned off, the particles sink according to the Eckmann spiral. This effect is due to the resultant of the Coriolis and inertial forces. Figure C.16 shows a similar situation, but in this case the particles have been interactively released from an imaginary straight line along the coast.

6.3 Hydrogeology

6.3.1 Hydrogeologic problem to solve

In this study, hydrogeologists are interested in the behavior of karst aquifers. These conduits may be imaged as a sequence of "storage" and connection elements, with irregular shapes in which the flow, containing dissolved karst particles, often changes from stagnant to fast (Fig. 6.2a).

A detailed study of the flow in different pool configurations has been undertaken [43] by using dye tracers that are injected into the flow. A laboratory scale model of a pool was built and some field studies were undertaken. To validate these experiments, the flow, as well as the tracer transport, within the domain of interest is simulated with a computational fluid dynamics model [42].

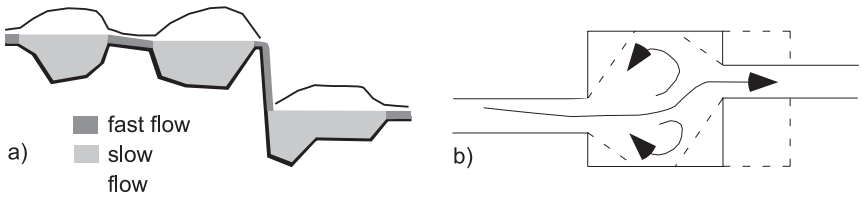


Figure 6.2: Conduits

6.3.2 Simulation

Typically, the flow in a conduit enlargement or pool is simulated with a 3D flow solver based on a finite volume method and a block structured grid with a resolution of $80 \times 75 \times 4$ nodes. The physical dimensions of the pool are approximately 1 m $\times$ 1 m, with an entry and exit channel, each 1 m long (see Fig. 6.2b). First, a steady state flow solution is computed. Then dye tracer is injected during 2 seconds, and the transport is calculated with an additional scalar transport equation on the velocities of the first solution.

The simulation covers 1000 seconds and is carried out with 10,000 time steps of 0.1 second. The data of every 50th time step is stored onto the solution file, which comes up to a size of 60 MB.

6.3.3 Visualization

The users' requirements on the visualization of the results obtained with the experiments and the simulation were the following:

- identification of the **general flow pattern**;
- representation of **time-dependent** and **spatial evolution of the tracer**;
- visualization of **shear stress** near the wall and floor of the pool;
- visualization of **particles' sedimentation**.

Selections of precise regions of interest and of several points of view were strong requirements of the users because they wished to explore the data interactively.

Since these users had a previous experience with a commercial solver, they required the provision of a visualization system that could directly read the dataset produced by the solver. Thus, they could avoid the repeated and tedious data conversions they had to run each time they would like to visualize new outputs. Thanks to the dataset abstract interface offered by ZoomIn, we could first write a specific driver. Because the grid used for the numerical simulation was a block structured grid, we had to write a specific

interpolation module with help of the **Visualization Toolkit** library [93]. The dataset interface of ZoomIn allowing integration of driver without any particular difficulty and with a 1 man-week effort.

With regard to visualization, thanks to the design of the platform, most of the AVOs developed for the previous case study could be reused and extended:

- cutting plane and isosurface to express turbulence and to visualize the propagation of the tracer;
- simple 3D conical glyphs expressing the direction and the magnitude of the velocity field;
- contour lines;
- streamlines to make the detection of vortices easy.

For these tools, users had a specific requirement. They needed an integration routine for the contour lines tool which could compute the surface comprised between each pair of neighbor isolines. A simple customization of this AVO provided the functionality within a couple of hours.

Furthermore, in order to fulfill their needs, we had to develop new AVOs for this study:

- a glyph representing the evolution in time of the tracer by a graph at the position of interest;
- a graph bar glyph to represent the quantities of several class of sedimented particles;
- an AVO with a transparency property to visualize the geometry of simple pools.

The development of these new tools required 1 man-week effort. This is a short time if one takes into consideration that the provided visualization system is intuitive, customized and fully interactive.

6.3.4 Major flow characteristics

The users' first objective was to recognize the general pattern (i.e. central flow path and recirculation zones) of the flow solution obtained by simulation. To get a general overview, they created a cutting plane of the speed distribution. They set the color scale between blue (0.0 m/s) and red (0.08 m/s). Coarsely gridded velocity arrows were superposed to indicate the global flow direction (Fig. C.17).

To recognize the shape and location of the principal eddy, they superposed a set of arrows, with the same length scaling but with a finer sampling,

in a region of interest covering the eddy (Fig. C.17). By means of the precise region of interest selection, the users discovered a smaller eddy, turning in the opposite sense, in the upper left corner of the pool. It was visualized with a still finer spacing of arrows. The users choose the arrow length scale four times larger and the base diameter smaller because lower velocity was prevailing here. The arrows in this region of interest were colored in red to distinguish them from the other arrows with a different scaling. Figure C.18 shows the small eddy in detail.

Vortices smaller than the computational grid size cannot be resolved, but are however important for the turbulent dispersion of tracer. The turbulent kinetic energy K is a measure for their intensity. Figure C.19 shows a cutting plane at $z = 0.02$ colored with the turbulent kinetic energy. An isosurface of K is superposed to emphasize the 3D structure of the zones of high turbulence. One of them is situated between the inflow jet and the main eddy; this zone is where tracer will be dispersed to the stagnant zone. In this picture some velocity arrows could be precisely added; they show the main eddy to those observers who are less familiar with the phenomenon.

6.3.5 Tracer transport

In general an isosurface gives a good 3D impression of the tracer density. However in the present problem, the turbulent eddy structure is mainly in the horizontal plane. Therefore a cutting plane was chosen; not only does it contains the essential information about the plume shape, but it also gives more information about the distribution inside the plume than an isosurface. Figure C.25 shows the tracer plume, thanks to a cutting plane and thanks to an automatically adjusted color map, at times 5, 25, 60, 115, 235 and 300 seconds.

In addition to this, the users were interested in monitoring the concentration as a function of time, $C(\mathbf{x}, t)$, for different points $\mathbf{x}$. An interactive **glyph probe** that draws a graph of $C(\mathbf{x_0}, t)$ for the desired location $\mathbf{x_0}$ and for a selected time interval, was developed (see Fig. C.20). The users can put these glyphs on several locations on the background of the tracer concentration cut. The scale of the glyphs is automatically adapted to the minimal and maximal concentration in the considered time interval at the reference position of the glyph.

6.4 Experience and users feedback

In both case studies, users highly appreciated that the tools could be customized to their real needs and within a reasonable period of time. This would be impossible with a special purpose application. Furthermore, through direct manipulation of visualization objects, the intuitive and simple user interface made them autonomous in their investigations. Precise selection

of a region of interest helped them to explore the data efficiently and to control the visual clutter. They liked the presentation facilities offered by the platform, a feature often underestimated by visualization system designers. Indeed, images, movies, videos and VRML scenes outputs are easily accessible through a popup menu and produce the adequate material they required for the presentation of their results.

Although users were enthusiastic, we remarked that, at the beginning, they hesitated to analyze their data with ZoomIn. The experience they gained with some visualization systems led to a certain conservatism. But, once all tools they needed were available, they understood the advantage of the platform.

One difficulty was to stay consistent with the general concept we had defined, without taking shortcuts just to provide as quickly as possible the required functionalities and customizations. Sometimes, we had to persuade users to keep tools sufficiently general so that they can reuse them in further studies. We were surprised to see how easy and fast it was to integrate new visualization techniques and to reuse existing tools. If we had to reach the same goal with libraries of modular visualization environments, it would have taken us several months of development for each study and we would obtain poorly reusable tools. We are convinced that, thanks to the visualization objects abstraction and the design of the platform, an experienced visioneer can customize a set of simple but not trivial tools in a few weeks.

Chapter 7

Conclusion

7.1 Summary

Existing visualization systems can roughly be classified into three categories: i) special purpose applications dedicated to particular domains, ii) modular visualization environments and iii) more general packages that require programming. The major advantage of applications in the first category is their specificity whereas the strength of those in other two categories is their generality. Experience has shown that, ideally, visualization systems have to satisfy both requirements. Because the way phenomena are discovered thanks to visualization is often unpredictable, systems have to be **extended** with new or customized visualization techniques. And because users usually want to concentrate on their main activity, systems have to be **efficient** and **easy to learn**. Each requirement, taken individually, is commonly offered in many visualization systems, but those that satisfy **both** requirements simultaneously are rare.

7.2 Contribution

Our work had two goals: i) finding concepts that abstract the visualization techniques and users' actions, and ii) building around these concepts a visualization platform that could satisfy simultaneously both requirements of ease-of-use and extensibility. The concept of **visualization objects**, i.e. functions mapping the data onto graphical objects, which can be directly manipulated by users, was defined. A usable prototype of a visualization platform based on these objects, called ZoomIn, was implemented and some experience was gained through case studies. Unfortunately, a quantitative comparison of the solution we propose with existing ones is difficult because most criteria are not easily measurable, and because, at the moment, we do not have enough experience with users. Nevertheless, our approach fulfills most of the requirements we wanted to meet at the beginning of this work.

For users, the major benefit of the visualization object concept is to provide uniform and intuitive tools, whose direct manipulation makes the **exploration** of three-dimensional time-dependent data interactive. Indeed, using ZoomIn, users immediately feel comfortable, can start working after a few minutes of explanations and rapidly acquire a level of autonomy they could not reach either with modular visualization environments or with libraries. Exploration is interactive and intuitive for two reasons. First, users **directly** manipulate geometric objects representing the data. Second, the ability to **apply the function** that is hidden behind a visualization techniques, here and there with simple commands appears very natural to them. They take actions and see instantaneously their effects on the visualization objects.

Furthermore, they highly appreciate that visualization objects can be **customized to their real needs** and can be fully integrated into the platform without any modification of the graphical user interface. This is a crucial advantage over tools developed with libraries or modular visualization environments that cannot be integrated easily into existing visualization systems, or require a specific graphical user interface. Moreover, because users know that customization is possible and that it does not require too much programming effort, they are encouraged to ask for non-trivial refinements of visualization techniques, or even better, to propose original tools.

For the programmer, the benefits of the visualization objects concept mainly resides in the **generalization of visualization techniques** it offers. Thus, a visualization platform integrating this concept is quickly **extensible** with new visualization tools and offers a good level of **reusability**. The implementation of ZoomIn demonstrated that the development of such a platform is technically feasible and actually shortens the development time required to integrate a new visualization tool.

7.3 Perspectives

In order to conclude this thesis, we will try to give a few perspectives for the future of scientific visualization. Trying to predict the future of a technology is risky. Usually people who risk a prediction either overestimate short term changes, because of the inertia induced by users' habits, or underestimate long term improvements gained by innovation which are, by essence, unpredictable. Everybody has in mind the unexpected explosion of the information technology caused by the arrival of the World Wide Web in 1994. Such innovation had – and will have – a considerable impact on our life and economy. Of course, improvements in the field of numerical simulation and scientific visualization cannot have the same impact because this activity is less fundamental than information exchange. In our opinion, scientific visualization advances will be achieved, first, thanks to technical improve-

ments and, second, indirectly, due to the needs of numerical simulation in the industry.

As discussed in Chapter 2, technical improvements in hardware and software as well as theoretical research to find better algorithms and to increase software quality are still expected. Moreover, sophisticated virtual reality peripherals will make interaction in space and time much more easier. From the hardware side, the near horizon offers gigaherz CPU, terabyte disks and memories as well as 10 gigabits per second networks. Within the last ten years, the ratio price/performance has decreased at a minimum by a factor 10. The performance of a CHF 200'000.- SGI engine in 1990 is now achieved by a CHF 20'000.- graphical workstation. Non-trivial three-dimensional and time-dependent visualizations, which required high performance workstations 5 years ago, are achieved today by means of a highend laptop. And there is no reason to think that this will not continue.

Until now a large effort has been invested in numerical simulation in order to, first, test and validate this technology, second, to optimize computations of numerical simulator, and third, in order to compare the results obtained with the reality. It can be stated that this research was successful in several areas and that now it is interesting to address more complex problems and to generalize the use to this technology. Obviously numerical simulations will become more and more common in the industry for two reasons:

- time can be saved during the research and development cycles – a rough simulation makes it possible to skip one cycle by early detection of major problems;

- quality assurance is now becoming a strong requirement for industry – fine simulations are able to detect hidden flaws and long simulations can anticipate material overwork.

But it is also clear that analyzing the time-dependent three-dimensional volume data produced by numerical simulators is not trivial and very time consuming, especially with rudimentary visualization tools. Today, it is common that the users spend more days in analyzing data than the machine needs for the computations. Thus, the use of numerical simulations by a wide community will not be possible without an important investment in the research and development of efficient visualization systems.

This also implies that software editors will begin to develop solutions because of their interest in this new market. They will try to offer solutions for the great majority of customers. Thus scientific visualization will be more and more spread off the circuit of academic researchers. Efficiency, versatility and ease-of-use will become major requirements for visualization systems. Engineers will need platforms that minimize their development effort – what could be achieved thanks to object-oriented dedicated components usable from several applications – and the users will need efficient,

easy-to-learn visualization systems that seem familiar for them and whose usage is consistent with other applications.

Whatever visualization system will be used, a fundamental problem will remain: the difficulty of extracting non-trivial information using a 3D graphical object – especially if the graphical object is not common and visualizes high-dimensional information. Working with a complex visualization object, for instance, with a sophisticated probe, and interpreting the data correctly requires skill from users. In the future, it is doubtless that virtual reality peripherals will improve this perception thanks to the feeling of immersion. Nevertheless, adequate visualization techniques for many application domains have still to be discovered. In this perspective, we think that ZoomIn and the concept of visualization objects offer an excellent environment for rapid prototyping.

The role of scientific visualization is becoming more and more important in many research and industrial activities. Successfully bringing visualization systems onto the desk of a large community of users is a great challenge. We experienced that providing users with a profusion of dedicated applications and with more powerful hardware is not sufficient if they want to solve their problems efficiently. Scientific visualization is still a young discipline and paradigms have to be found to minimize both users' and programmers' workload. The concepts developed in this thesis aim at making a step towards an easy-to-learn and extensible platform that gives users the autonomy they need without having to sacrifice the versatility required in essence by scientific visualization.

Bibliography

[1] *Proceedings Visualization '93*. IEEE Computer Society, 1993.

[2] Advanced Visual Systems Inc., Waltham, MA, USA. AVS. `http://www.avs.com`.

[3] Amtec Engineering Inc., Bellevue, WA, USA. Tecplot. `http://www.amtec.com`.

[4] Ricardo Avila, Taosong He, Lichan Hong, Arie Kaufman, Hanspeter Pfister, Claudio Silva, Lisa Sobierajski, and Sidney Wang. Volvis: A diversified volume visualization system. In Bergeron and Kaufman [9], pages 31–38.

[5] C. Bajaj. Scientific and multimedia toolkits. In *Pacific Graphics '95*, 1995.

[6] David C. Banks and Bart A. Singer. A predictor-corrector technique for visualizing unsteady flow. *Transactions on Visualization and Computer Graphics, Vol 1, No. 2*, pages 151–163, 1995.

[7] Sander Beliën and Rik Leenders. Comparison of visualization techniques and packages. Technical report, SARA Academic Computing Services Amsterdam, 1995. `http://www.sara.nl/Rik/REPORT.update`.

[8] Michael Bender, Ralf Klein, Andreas Disch, and Achim Ebert. A functional framework for web-based information visualization systems. *Transactions on Visualization and Computer Graphics, Vol 6, No. 1*, pages 8–23, 2000.

[9] R. Daniel Bergeron and Arie E. Kaufman, editors. *Proceedings Visualization '94*. IEEE Computer Society, 1994.

[10] Lawrence D. Bergman, Bernice E. Rogowitz, and Lloyd A. Treinish. A rule-based tool for assisting colormap selection. In Nielson and Silver [72], pages 118–125.

[11] Etienne Berner. Développement d'une base de données distribuée pour le logiciel ZoomIn. Technical report, Computer Science Department, University of Neuchâtel, Switzerland, 1998.

[12] Grady Booch. *Object-Oriented Analysis and Design with Applications, 2nd ed.* Benjamin-Cummings, 1994.

[13] K. W. Brodlie, J. R. Gallop, A. J. Grant, J. Haswell, W. T. Hewitt, S. Larkin, C. C. Lilley, H. Morphet, A. Townend, J. Wood, and H. Wright. Review of visualization systems 2nd edition. Technical report, Advisory Group on Computer Graphics UK, 1991. `http://www.agocg.ac.uk/`.

[14] M. Brown, C. Harris, R. Kriz, and M. Vigil. How visualization applications drive tool selection: One product can't do it all. In Nielson and Rosenblum [71], pages 345–347.

[15] Steve Bryson. Interaction of objects in a virual environment: a two-point paradigm. Technical report, NASA Ames Research Center, 1991.

[16] Steve Bryson and Sandy Johan. Time management, simultaneity and time-critical computation in interactive unsteady visualization environments. In Yagel and Nielson [122], pages 255–261.

[17] Steve Bryson, Sandy Johan, and Leslie Schlecht. An extensible interactive visualization framework for the virtual windtunnel. In *Proceedings Virtual Reality Annual International Symposium '97.* IEEE Computer Society, 1997.

[18] S. Casera. Intégration d'un nouvel outil dans ZoomIn: Textures représentant un champ vectoriel. Technical report, Computer Science Department, University of Neuchâtel, 2000.

[19] R. G. G. Cattell, editor. *The Object Database Standard : ODMG-93, Release 1.1.* Morgan Kaufmann, 1994.

[20] Michael Clifton and Alex Pang. Cutting planes and beyond. Technical report. `http://www.cse.ucsc.edu/research/slvg/cut.html`.

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

[22] James O. Coplien. *Advanced C++ for Users, Programming Styles and Idioms.* Addison-Wesley, 1992.

[23] Roger Crawfis, Nelson Max, and Barry Becker. Vector field visualization. *Computer Graphics and Applications, Vol. 14, Num. 5*, pages 50–56, 1994.

[24] Dave Darmofal and Robert Haimes. Visualization of 3-d vector fields: Variations on a stream. *AIAA Paper 92-0074*, 1992.

[25] Willem C. de Leeuw and Jarke J. van Wijk. A probe for local field visualization. In *Proceedings Visualization '93*, pages 39–45, 1993.

[26] Wim de Leeuw and Robert van Liere. Comparing lic and spot noise. In *Proceedings Visualization '98*, pages 359–365, 1998.

[27] Thierry Delmarcelle and Lambertus Hesselink. A unified framework for flow visualization. In *Computer Visualization, Graphics Techniques for Scientific and Engineering Analysis*, pages 129–170. CRC Press, 1995.

[28] Department of Aeronautics and Astronautics, Massachusetts Institute of Technology (MIT), USA. Visual3. `http://raphael.mit.edu/visual3/visual3.html`.

[29] Digital Graphics Inc., Alameda, CA, USA. EarthVision. `http://www.dgi.com`.

[30] Rae A. Earnshaw and Mikael Jern. Fundamental approaches to intractive real-time visualization systems. In *Scientific Visualization*, pages 223–237. Academic Press Ltd, 1994.

[31] Jose Encarnacao, Jim Foley, Steve Bryson, Steen K. Feiner, and Nahum Gershon. Research issues in perception and user interfact. *Computer Graphics and Applications, March 1994*, pages 67–73, 1994.

[32] Fluent Inc., Lebanon, NH, USA. Fluent. `http://www.fluent.com`.

[33] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer graphics, Principles and practice, 2nd edition.* Addison-Wesley, 1990.

[34] Christos Fragakis. Metro-Med, 1997. `http://erato.fl.ariadne-t.gr/metromed2/home.html`.

[35] Richard S. Gallagher. Scalar visualization techniques. In *Computer Visualization, Graphics Techniques for Scientific and Engineering Analysis*, pages 89–127. CRC Press, 1995.

[36] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[37] W. Gu, J. Vetter, and K. Schwan. An annoted bibliography of interactive program steering. Technical report, College of Computing, Georgia Institute of Technology, Atlanta, 1994. `http://www.cc.gatech.edu/systems/projects/Steering/`.

[38] R. B. Haber and D. A. McNabb. Visualization idioms : A conceptual model for scientific visualization systems. In *Visualization in Scientific Computing*, pages 74–93. IEEE Computer Society Press, 1990.

[39] Hans Hagen. Visualization of large datasets. In *Scientific Visualization*, pages 187–198. Academic Press Ltd, 1994.

[40] M. Hauns, O. Atteia, and F. Hermann. Application of a computational fluid dynamics model to cave river hydrodynamics. In *Proceedings of the 12th international Congress of Speleology, La Chaux-de-Fonds, Switzerland*, pages 141–145, 1997.

[41] M. Hauns, F. Hermann, and O. Atteia. Application of a CFD model to cave river hydrodynamics. *12th International Congress of Speleology, La Chaux-de-Fonds, Switzerland*, 1997.

[42] M. Hauns, P.-Y. Jeannin, and F. Herrmann. Tracer transport in undergound rivers in karst: tailing effect and channel geometry. In *Bulletin d'Hydrogéologie Num. 16*. CHYN, Université de Neuchâtel, Switzerland, 1997.

[43] Michael Hauns. *Modeling racer and particle transport under turbulent flow conditions in karst coduit structures*. PhD thesis, University of Neuchâtel, Switzerland, 1999.

[44] Michael Hauns and Olivier Atteia. Colloid particle dynamics in karst aquifers, 1997. `http://www-chyn.Unine.ch/colloid/`.

[45] William C. Hill and James D. Hollan. Deixis and the future of visualization excellence. In Nielson and Rosenblum [71], pages 314–320.

[46] Computational Engineering International Inc. Ensight. `http://www.ceintl.com`.

[47] Intelligent Light Inc., Lyndhurst, NJ, USA. Fieldview. `http://www.ilight.com`.

[48] Intelligent Light Inc., Lyndhurst, NJ, USA. *Fieldview User's Manual*, 1996.

[49] International Business Machine Inc., USA. Data Explorer. `http://www.almaden.ibm.com/dx/`.

[50] Vijendra Jaswal. Cavevis:distributed real-time visualization of time-varying scalar and vector fields using the cave virtual reality theater. In Yagel and Hagen [121], pages 301–308.

[51] C. R. Johnson and S. G. Parker. Applications in computational medicine using scirun: A computational steering programming environment. In H.W. Meuer, editor, *Supercomputer '95*, pages 2–19, 1995.

[52] Won Kim. *Modern Database Systems: The Object Model, Interoperability, and Beyond.* Addison-Wesley, 1995.

[53] David A. Lane. Ufat - a particle tracer for time-dependent flow fields. In *Proceedings Visualization '94*, pages 257–264, 1994.

[54] David A. Lane. Scientific visualization of large-scale unsteady fluid flows. In *Scientific Visualization, Overviews-Methodologies-Techniques*, pages 125–145, 1997.

[55] Wilfrid Lefer and Michel Grave, editors. *Eight Eurographics Workshop on Visualization in Scientific Computing.* The EuroGraphics Association, Springer, 1997.

[56] Suresh K. Lodha, Catherine M. Wilson, and Robert E. Sheehan. Listen: Sounding uncertainty visualization. In Yagel and Nielson [122], pages 189–193.

[57] H. Löffelmann, L. Mroz, and E. Gröller. Hierarchical Streamarrows for the Visualization of Dynamical Systems. In Lefer and Grave [55], pages 155–153.

[58] W. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of SIGGRAPH '87, Computer Graphics 21, Vol. 4, June*, 1987.

[59] B. Lucas, G. Abram, N. Collins, D. Epstein, D.Gresh, and K. McAuliffe. An architecture for a scientifc visualization system. In *Proceedings Visualization '92*, pages 107–114, 1992.

[60] J.-C. Maréchal. *Etude et modélistaion de l'hydraulique et du transport dans les drains karstiques.* PhD thesis, University of Neuchâtel, Switzerland, 1994.

[61] Merriam-Webster. Merriam-Webster's Collegiate Dictionary. `http://www.m-w.com`.

[62] R. Minghim and A.R. Forrest. An illustrated analysis of sonification for scientific visualization. In Nielson and Silver [72], pages 110–117.

[63] C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In Bergeron and Kaufman [9], pages 281–287.

[64] Jurriaan D. Mulder and Jarke J. van Wijk. 3d computational steering with parametrized geometric objects. In Nielson and Silver [72], pages 304–311.

[65] NASA Ames Research Center. NAS Annotated Bibliography of Scientific Visualization Sites. `http://www.nas.nasa.gov/Groups/VisTech/visWeblets.html`.

[66] NASA Ames Research Center, USA. Flow Analysis Software Toolkit. `http://science.nas.nasa.gov/Software/FAST/`.

[67] NASA Ames Research Center, USA. Plot3D. `http://science.nas.nasa.gov/`.

[68] NASA Ames Research Center, USA. Unsteady Flow Analysis Toolkit. `http://science.nas.nasa.gov/Software/UFAT/`.

[69] Jakie Neider, Tom Davis, and Mason Woo. *Open GL Programming Guide, The Official Guide to Learning OpenGL, Release 1.* Addison-Wesley, 1993.

[70] Gregory M. Nielson, Hans Hagen, and Heinrich Müller. *Scientific Visualization, Overviews-Methodologies-Techniques.* IEEE Computer Society, 1997.

[71] Gregory M. Nielson and Larry Rosenblum, editors. *Proceedings Visualization '91.* IEEE Computer Society, 1991.

[72] Gregory M. Nielson and Deborah Silver, editors. *Proceedings Visualization '95.* IEEE Computer Society, 1995.

[73] François Nyffeler. The LIMNOCEANE group, 1997. `http://www-geol.unine.ch`.

[74] Hans-Georg Pagendarm and Thomas Gerhold. Flow visualization in a hypersonic fin/ramp flow. In Nielson and Silver [72], pages 379–382.

[75] Alex Pang and Michael Clifton. Metaphors for visualization. `http://www.cse.ucsc.edu/research/slvg/ginzu.html`.

[76] Alex Pang and Kyle Smith. Spray rendering: Visualization using smart particles. In *Proceedings Visualization '93*, pages 283–290, 1993.

[77] Alex Pang and Craig Wittenbrink. Collaborative 3d visualization with cspray. *Computer Graphics and Applications, March-April 1997*, pages 32–41, 1997.

[78] Alex Pang and Craig M. Wittenbrink. Spray rendering as a modular visualization environment. *Computer Graphics, May 1995*, pages 33–36, 1995.

[79] Dave Pape, Carolina Cruz-Neira, and Marek Czernuszenko. *CAVE User's Guide.* Electronic Visualization Laboratory, University of Illinois, Chicago, 1996.

[80] Eric Pepke and Jim Lyons. *SciAn User's Guide.* Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida, 1993.

[81] Frank J. Post, Theo van Walsum, Frits H. Post, and Deborah Silver. Iconic techniques for feature visualization. In Nielson and Silver [72], pages 288–295.

[82] Jef Raskin. *The Humane Interface, New Directions for Designing Interactive Systems.* Addison-Wesley, 2000.

[83] William Ribarski, Eric Ayers, John Eble, and Sougata Mukherjea. Glyphmaker: Creating customized visualizations of complex data. *Computer, July 1994*, pages 57–64, 1994.

[84] Philip K. Robertson, Rae A. Earnshaw, Daniel Thalmann, Michel Grave, Julian Gallop, and Eric M. De Jong. Research issues in the foundations of visualization. *Computer Graphics and Applications, March 1994*, pages 73–76, 1994.

[85] David F. Rogers. *Procedural elements for computer graphics.* McGraw-Hill, New-York, 1985.

[86] Bernice E. Rogowitz and Lloyd A. Treinish. Data visualization: the end of the rainbow. *IEEE Spectrum, December 1998*, pages 52–59, 1998.

[87] Martin Roth and Ronald Peikert. Flow visualization for turbo machinery design. In Yagel and Nielson [122], pages 381–388.

[88] Hervé Sanglard. ZoomIn Implementation Guide. Technical report, Computer Science Department, University of Neuchâtel, Switzerland, 1999.

[89] Hervé Sanglard and Michael Hauns. Visualizing numerical flow simulations of karst aquifers. In *GraphiCon '98 Conference Proceedings, Moscow*, pages 272–276, 1998.

[90] Hervé Sanglard and Hans-Heinrich Nägeli. An end user oriented platform for scientific visualization. In *International Conference on Information Visualization '97, London, IEEE Computer Society*, pages 235–239, 1997.

[91] Hervé Sanglard and Hans-Heinrich Nägeli. Zoomin, an intuitive and extensible platform for scientific visualization, 1998. `http://iiun.unine.ch/ZoomIn/`.

[92] Hervé Sanglard, Redoine Tahiri, Hans-Heinrich Nägeli, and François Nyffeler. Zoomin - a new kind of visualization platform for oceanography. In *5th Metro-med Workshop Proceedings, Aix-en-Provence - France*, March 1999.

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

[94] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. In Yagel and Nielson [122], pages 93–100.

[95] Martin Schultz, Thomas Reuding, and Thomas Ertl. Analyzing engineering simulations in a virtual environment. *Computer Graphics and Applications, Vol. 18, Number 6*, pages 46–52, 1998.

[96] Silicon Graphics Inc., Mountain View. *IRIS Explorer Module Writer's Guide*, 1992.

[97] Software Technology for Fluid Mechanics, Deutsche Forschungsanstalt fuer Luft- und Raumfahrt. HIGHEND. `http://www.ts.go.dlr.de`.

[98] Space Science and Engineering Center, University of Wisconsin, USA. Vis5D. `ftp://iris.ssec.wisc.edu/pub/vis5d/`.

[99] Alexander Stepanov and Meng Lee. *The Standard Template Library.* Silicon Graphics Inc., Hewlett-Packard Lab., 1994.

[100] Steve Bryson. Virtual Environments in Scientific Visualization. `http://www.nas.nasa.gov/~bryson/papers.html`.

[101] Penny Storms, editor. *International Conference on Information Visualization '97.* IEEE Computer Society, 1997.

[102] Supercomputer Computations Research Institute, Florida State University, Tallahassee, Florida, USA. SciAn. `http://www.scri.fsu.edu/~lyons/scian/`.

[103] Redoine Tahiri, Hervé Sanglard, François Nyffeler, and Hans-Heinrich Nägeli. Comparison between the transport of particles and tracer dispersion in coastal areas. In *5th Metro-med Workshop Proceedings, Aix-en-Provence*, March 1999.

[104] The Geometry Center, University of Minnesota, USA. Geomview. `http://www.geom.umn.edu/software/download/geomview.html`.

[105] The Numerical Algorithms Group Ltd., Oxford, UK. Iris Explorer. `http://www.nag.co.uk`.

[106] V. Thunus, E. A. H. Zuur, P. Lambert, and C.-H. Godet. A numerical simulation of transport and sedimentation of suspended particles in lake Neuchâtel. *Ecologae geol. Helv. 87/2*, pages 385–402, 1994.

[107] Vincent Thunus. *Modélisation à mesoéchelle de l'hydrodynamique et du transport de particules en milieu marin semi-ouvert*. PhD thesis, Université de Neuchâtel, Suisse, 1996.

[108] Jens Trapp and Hans-Georg Pagendarm. A prototype for a www-based visualization service. In *Proceedings of the Eight Eurographics Workshop on Visualization in Scientific Computing, April 1997, Boulogne sur Mer, France*. The Eurographics Association, 1997.

[109] Lloyd Treinish, Ravi Kulkarni, Mike Folk, Greg Goucher, and Russ Rew. Data models, structures and access software for scientific visualization. In *Proceedings Visualization '93* [1], pages 355–360.

[110] Edward R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 1983.

[111] Edward R. Tufte. *Envisioning Information.* Graphics Press, 1990.

[112] Edward R. Tufte. *Visual Explanation: Images and Quantities, Evidence and Narrative.* Graphics Press, 1997.

[113] C. Upson. The application visualization system : A computational environment for scientific visualization. *Computer Graphics and Applications, 9(4)*, pages 30–42, 1989.

[114] Jarke J. van Wijk, Andrea J. S. Hin, Willem C. de Leeuw, and Frits H. Post. Three ways to show 3d fluid flow. *Computer Graphics and Applications, Vol. 14, Num. 5*, pages 33–39, 1994.

[115] Jarke J. van Wijk and R. van Liere. An environment for computational steering. Technical Report CS R9448, Center for Mathematics and Computer Science (CWI) Amsterdam, 1994.

[116] Jarke J. van Wijk and Robert van Liere. An environment for computational steering. In *Scientific Visualization, Overviews-Methodologies-Techniques*, pages 103–124, 1997.

[117] Visualization Laboratory, State University of New York, USA. VolVis. `http://www.cs.sunysb.edu/~vislab/`.

[118] John Vlissides, James Coplien, and Norman Kerth. *Pattern Languages of Program Design 2.* Addison-Wesley, 1996.

[119] Josie Wernecke. *The Inventor Mentor, Programming Object-Oriented 3D Graphics with Open Inventor, Release 2.* Addison-Wesley, 1994.

[120] Josie Wernecke. *The Inventor Toolmaker, Extending Open Inventor, Release 2.* Addison-Wesley, 1994.

[121] Roni Yagel and Hans Hagen, editors. *Proceedings Visualization '97.* IEEE Computer Society, 1997.

[122] Roni Yagel and Gregory M. Nielson, editors. *Proceedings Visualization '96.* IEEE Computer Society, 1996.

[123] E. A. H. Zuur. The prosper general circulation model, an adapted form of the sandia ocean modelling method : a numerical description. *Int. J. Num. Meth. Fluids, Vol. 13*, pages 251–263, 1991.

Appendix A

Glossary of visualization techniques

A.1 Visualization techniques

Scientific visualization has to satisfy the needs of a wide range of domains of interest. Unfortunately, there is no universal visualization technique that could be applied in every situation. Several specific visualization techniques have been developed for years to fulfill user needs. Most of them have been refined or optimized, new ones appear every year, and others disappeared because they were supplanted by more powerful ones. Among them, a few emerged and became standard, e.g. cutting planes, isosurfaces or particle traces. They seem familiar to both communities of users and visioneers.

Several approaches to visualize the data have been investigated. They address the problems differently, are more or less appropriate for certain kinds of application, and have their own specificities. Some techniques are based on surface rendering and aim at representing 3D structures by drawing their shape and rendering them with illumination models like Phong or Gouraud shading [85, 33]. Others are based on direct volume rendering [4], e.g. ray casting, and use 2D projection as well as transparency levels to render the data without conversion to surface representation.

An overview of the main visualization techniques encountered in our domain of interest follows. Each technique is illustrated with one or more examples.

A.1.1 Scalar techniques

In science and engineering, three-dimensional scalar fields are ubiquitous. From oceanography to aeronautics, it is common to work with this kind of fields. Whether time-dependent or not, three-dimensional scalar fields are not trivial to analyze, and require adequate tools. To reach this goal, the

following scalar visualization techniques were developed and optimized:

Cutting plane, volume slicing: This technique [35] consists in "restricting" a scalar field to a virtual plane parallel to one of the axes (orthogonal planes) or to any plane given e.g by one of its point and a normal. These cuts are then colored accordingly to the scalar value using a color map (see Fig. C.9, C.17, C.25).

Isoline: An isoline is the set of all points of a two-dimensional surface (usually a plane) having the same value. Often a set of isolines corresponding to different values are drawn so that it is possible to make a comparison between them. Isolines are usually colored according to the corresponding scalar value (see Fig. C.1, C.23, C.24).

Isosurface: The isosurface [58, 63] is the generalization of the isoline in three-dimensional space. It is defined as the set of all points of the 3D space having the same scalar value. This is one of the most common visualization techniques used to visualize three-dimensional scalar data sets (see Fig. C.6, C.19).

A.1.2 Vector techniques

Visualizing three-dimensional vector fields is much more difficult than visualizing scalar fields. One of the main problems is to represent appropriately the three (or more) components of the vector at a single point of the space. The perspective projection onto the screen causes ambiguities and distortions and it is difficult to interpret such images correctly.

In the problems we consider, the most common vector field is the velocity of a fluid. Users are interested in the general behavior of a fluid or in particular phenomena at locations where velocity vanishes, for instance near a wall or inside a vortex. They need very specific visualization techniques to be able to track these features.

Moreover vector fields usually vary through time. In the terminology of computational fluid dynamics, a velocity field is called steady (time-independent) or unsteady (time-dependent). Another point of interest is to observe the change of state between laminar flow (the fluid flows in layers and all elements in a small neighborhood have about the same velocity) and turbulent flow (velocities of neighboring elements can have large and random variations). Vector field visualization techniques have to accommodate all these cases and many efforts have been invested to provide adequate tools:

Streamline: 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:

$$\frac{d\mathbf{x}}{d\tau} = \mathbf{v}(\mathbf{x}, t) \tag{A.1}$$

where $\mathbf{x}$ is the position in space, $\mathbf{v}$ 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 (see Fig. C.22). Streamribbons, streamtubes or stream surfaces [24, 114] are variations of this technique; they are obtained by linking two or many streamlines side by side. Because the shape obtained can be shaded, the perception of depth is improved and this leads to an easier recognition of the three-dimensional structure of the flow.

Particle path: The trajectory of a particle released into a fluid is called a particle path. In this case, particles are injected into a velocity field and their behavior is simulated by taking into account gravity or another force. In several research projects, e.g. oceanography, particles of matter play an important role for the understanding of pollution or sedimentation problems. (see Fig. C.15, C.16, C.23, C.24).

Pathline: The pathline is the "particle path" of a virtual and infinitesimal (mass less and electrically neutral) particle. This technique [54] is commonly used to study the structure of a time-dependent flow (see Fig. C.5). The pathline can be computed by integrating the equation

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}(\mathbf{x}, t) \tag{A.2}$$

where $\mathbf{v}$ is again the velocity field.

Streakline: A streakline is a line joining the positions, at a given instant, of all particles that have been released previously from a specified location called the seed [53]. In hydrodynamics, streaklines are simulated by releasing hydrogen bubbles rapidly from the seed. It is to be noted that streamlines, pathlines and streaklines are one and the same in steady flows.

Textures: Some original visualization techniques based on texture mapping provide qualitative results (see Fig. C.3). De Leeuw and van Liere define them as follows [26]:

> "A line integral convolution (LIC) is generated by convolution of an input texture with a one-dimensional filter kernel. The shape of the kernel is determined by the shape of the streamline through the pixel. A pixel in the final texture is determined by the weighted sum of a number of pixels along a line in the input texture:

$$C_{out}(i,j) = \sum_{p \subset \tau} C_{in}(p) h(p)$$

where τ is the set of pixels in the input texture used for convolution, $C_{in}(p)$ is the value of the input texture pixel at grid cell p and $h(p)$ is the convolution filter.

A spot noise texture is generated by blending together a large number of small intensity fucntions at random positions on a plane. The shape of the intensity functions is deformed in relation to the vector field. Spot noise is described by the following equation:

$$f(\mathbf{x}) = \sum a_i \, h(\mathbf{x} - \mathbf{x_i})$$

in which $h(\mathbf{x})$ is called the spot function. It is an intensity function which has non zero value only in the neighborhood of the origin. a_i is a random scaling factor with a zero mean and $\mathbf{x_i}$ is a random position."

These techniques are useful to compare synthetic pictures and photography [74] obtained in a wind tunnel in order to validate numerical simulations and to detect vortices in aeronautics applications.

Vortex extraction: This technique is often based on isosurfaces of simple or derived scalar fields like the pressure or the magnitude of vorticity ($|\nabla \times \mathbf{v}|$). It is also possible to detect the vortex core by means of fields like the enstrophy (squared magnitude of vorticity), the curvature or the helicity (the dot product of the normalized velocity and vorticity) (see Fig. C.6). In some engineering discipline, like aeronautics or turbomachinery, it is of primary importance to detect vortices, so that several means to track and to visualize them automatically within a flow have been developed [87, 6].

A.1.3 Other techniques

It is often necessary to visualize simultaneously data of very different kinds. For instance, it is generally not sufficient to represent the shear stress, a tensor field, inside an incompressible fluid with scalar or vector visualization techniques. Adequate techniques are required to visualize multivariate variables and to observe the relations between their components. Furthermore, other techniques have to be developed in order to visualize objects, which are in essence not defined by a field, for instance, objects advected within a flow or the boundaries of a domain. The following examples illustrate techniques which were developed to satisfy specific needs:

Hyperstreamline: Hyperstreamlines (see Fig. C.2) are streamlines of the field of the eigenvectors of a symmetric tensor. They are built as follows: a geometric primitive, for instance a circle, sweeps along one of the eigenvector fields while stretching in the transverse plane under the combined action of the two other orthogonal eigenvector fields. The hyperstreamline is obtained by linking together the deformed primitives along the trajectory and is color coded according to another scalar variable, for instance the amplitude of the longitudinal eigenvalue [27].

Probe: Probing consists in the interactive displacement in space and time of a virtual instrument whose shape, color or other properties express the values it measures at its tip (see Fig. C.20). Interactive probes are very useful for the exploration of datasets, especially when the user does not know in advance, which is often the case, where or when the phenomena will take place, and generally when he does not know which kind of phenomena he will discover [25, 81].

Geometrical information: Visualizing geometrical information like a domain boundary or an obstacle put within a flow is of primary importance. This can be necessary in order to provide a reference frame in the three-dimensional space (see Fig. C.5, C.22) or merely to represent the value of a quantity, e.g. the pressure on the boundary surface of an aircraft's wing.

Appendix B

Sources

```
//
// Class name :             ziAVO
// Name :                   Abstract Visualization Object
// Role :                   An abstract class from which to subclass any
//                          visualization object
// Class hierarchy :        ziObject,ziSubject
// Description :            The AVO abstract class from which all specific
//                          visualization objects classes derive
//
// Implementation details : This abstract class allows for the
//                          instantiation of an object of any derived class
//                          unknown at compile time.
//
//                          It is built on the exemplar pattern described in
//                          'Advanced C++ : Programming styles and idioms'
//                          written by James. O. Coplien, Addison Wesley
//
//                          A static class member maintains a 'list' of
//                          exemplar instances (one for each derived class).
//                          When the make method of the unique abstract
//                          class exemplar is called (with the unique id of
//                          the required class as a parameter), the method
//                          'make' of each exemplar contained in the list
//                          is invoked and it returns the object if the
//                          id's comparison has succeeded.

#ifndef _ziAVO_h
#define _ziAVO_h

#include "ziProperty.h"
#include "ziSubject.h"
#include "ziObject.h"
#include "ziTime.h"
#include "ziROI.h"
```

```
#include "ziCVO.h"
#include "ziString.h"
#include "list.h"

class ziExemplar;
class ziCVO;
class ziRVO;
class ziGVO;
class ziROI;
class ziProperty;
class ziAVO;

typedef long ZI_AVO_CLASS_ID;
typedef long ZI_AVO_ID;
typedef list<ziAVO*> ziExemplarList;

const ZI_AVO_ID ZI_NO_AVO_ID = -1;

class ziAVO: public ziObject, public ziSubject{
protected:
  static ziAVO *list;
  ziAVO *next;
  ZI_AVO_ID _id;
  ziPropertyList _properties;

public:
  ziAVO();
  ziAVO(ziExemplar);
  virtual ziAVO* make(ZI_AVO_CLASS_ID cid);
  virtual void getSupportedROI(ziROITypeList &rtlist);
  virtual ziRVO* makeRVO(ziCVO* cvo);
  virtual ziGVO* getGVO(ziCVO* cvo);
  virtual ~ziAVO();
  virtual void update();

  static ziPropertyList getClassPropertyList(ZI_AVO_CLASS_ID cid);
  virtual ziPropertyList getClassPropertyList(ZI_AVO_CLASS_ID cid, bool &ok);
  virtual ziPropertyList getInstancePropertyList() {return _properties;}

  ZI_AVO_ID getId();
  void setId(ZI_AVO_ID id);

  virtual ZI_AVO_CLASS_ID getClassId();

  static void getExemplarList(ziExemplarList& exemplars);
  static void deleteExemplarList();
};
#endif // _ziAVO_h
```

```
//
// Class name :             ziCVO
// Name :                   Concrete Visualization Object
// Role :                   A class encapsulating the visualization
//                          objects parameters
// Class hierarchy :        ziObject,ziObserver

#ifndef _ziCVO_h
#define _ziCVO_h

#include <iostream.h>
#include "ziTime.h"
#include "ziProperty.h"
#include "ziObject.h"
#include "ziObserver.h"
#include "map.h"


class ziAVO;
class ziRVO;
class ziGVO;
class ziROI;
class ziCVO;

typedef long ZI_CVO_ID;
const ZI_CVO_ID ZI_NO_CVO_ID = -1;

struct _ltCVOid {
  bool operator()(ZI_CVO_ID id1,ZI_CVO_ID id2) const{
    return id1<id2;
  }
};
typedef map<ZI_CVO_ID, ziCVO*, _ltCVOid> ziCVOSet;

class ziCVO: public ziObject, public ziObserver{
  // Serialization
  friend istream& operator>>(istream&, ziCVO *&);
  friend ostream& operator<<(ostream&, ziCVO *&);

 protected:
  ZI_CVO_ID _id;
  bool _rvoIsBuilt;
  ziRVO* _rvo;
  ziAVO* _parent;
  ziROI* _roi;
  ziTimeStamp _time;
  ziPropertyList _properties;

 public:
```

```
  ziCVO(ziAVO* avo, ziROI* roi, const ziTimeStamp& time,
ziPropertyList pl, bool buildRVO=False);
  ziCVO(const ziCVO& cvo);
  ziCVO& operator=(const ziCVO& cvo);

  void update(ZI_SUBJECT_MESSAGE message);

  bool isRVOBuilt();
  void buildRVO();

  bool isValid();

  static ZI_CVO_ID _counter;
  ZI_CVO_ID getId();

  ziPropertyList getProperties();
  void setProperty(const ziProperty& property);

  ziAVO* getAVO();

  ziRVO* getRVO();
  void setRVO(ziRVO* rvo);

  ziGVO* getGVO();

  void setROI(ziROI* roi);
  ziROI* getROI();

  void setTime(const ziTimeStamp& time);
  ziTimeStamp getTime();

  ~ziCVO();
};

#endif // _ziCVO_h
```

```
//
// Class name :             ziRVO
// Name :                   Represented Visualization Object
// Role :                   A class encapsulating the graphical primitives of
//                          the visualization object
// Class hierarchy :        ziObject
// Implementation details : In the current implementation it uses an
//                          SoSeparator node from the SGI's OpenInventor
//                          library
//

#ifndef _ziRVO_h
#define _ziRVO_h

#include "ziObject.h"
#include "map.h"

class SoSeparator;
class SoNode;
class ziString;
class ziCVO;

typedef long ZI_RVO_ID;
const ZI_RVO_ID ZI_NO_RVO_ID = -1;

class ziRVO : public ziObject{
public:
  ziRVO(ziCVO* cvo);
  ~ziRVO();
  SoSeparator* getRoot();

  ziCVO* getCVO();

  static ZI_RVO_ID _counter;
  ZI_RVO_ID getId();

  static ziRVO* getRVOById(ZI_RVO_ID id);

  const char* getClassName();

private:
  SoSeparator* _node;
  ziCVO* _parent;
  ZI_RVO_ID _id;
};
#endif // _ziRVO_h
```

```
//
// Class name :             ziDataset
// Name :                   Dataset
// Role :                   Provide an abstract interface to the datasets
// Class hierarchy :        ziObject
// Implementation details : It is built from the exemplar design pattern
//                          described in
//                          'Advanced C++ : Programming styles and idioms'
//                          written by James. O. Coplien,  Addison Wesley


#ifndef _ziDataset_h
#define _ziDataset_h

#include <iostream.h>
#include "ziExemplar.h"
#include "ziObject.h"
#include "ziField.h"
#include "ziUDObject.h"
#include "ziTime.h"
#include "ziProperty.h"
#include "ziCVO.h"


typedef unsigned long ZI_DATASET_ID;

class ziExemplar;

class ziDataset: public ziObject{
  // Serialization
  friend istream& operator>>(istream&, ziDataset *&);
  friend ostream& operator<<(ostream&, ziDataset *&);

 protected:
  static ziDataset *list;
  ziDataset *next;
  virtual void Read(istream& is){}
  virtual void Write(ostream& os){}
  virtual ZI_DATASET_ID GetId() {return 0;}

public:

  ziDataset(){next=0;}
  ziDataset(ziExemplar){next=list; list=this;}
  virtual ziDataset* make(ZI_DATASET_ID id, ziPropertyList* pl);

  virtual void set_sfield(int index){};
  virtual void set_vfield(int index){};
  virtual ziError setTime(const ziTimeStamp& time){return ZI_NO_ERROR;};
```

```
  virtual ziError loadTime(ziTimeStamp *time_table,int size){return ZI_NO_ERROR;};

  virtual ziFieldList* getFields(){return NULL;};
  virtual ziUDObjectList* getUDObjects(const ziTimeStamp& time){return NULL;};
  virtual ziUDObject* getObject(ziCVO* cvo){return NULL;};
  virtual void showProperties(){};
  virtual void hideProperties(){};
  virtual ziPropertyList* getProperties(){return NULL;};
  virtual ~ziDataset(){}


  // To free the list of exemplar (one exemplar for one concrete object class)
  static void deleteExemplarList();
};

#endif // _ziDataset_h
```

```
//
// Class name :             ziField
// Name :                   Field
// Role :                   Provide an abstract interface to the fields of
//                          a dataset
// Class hierarchy :        ziObject

#ifndef _ziField_h
#define _ziField_h

#include "ziObject.h"
#include "ziType.h"
#include "ziError.h"
#include "ziCoordinate.h"
#include "ziProperty.h"
#include "ziString.h"
#include "ziTime.h"
#include "iterator.h"
#include "map.h"

class ziField : public ziObject {
  friend ostream& operator<<(ostream &os, const ziField& field);
public:

  enum {ZI_NO_FLAG=0, ZI_WAIT=1, ZI_NO_WAIT=2};
  enum {ZI_READY=0, ZI_NOT_READY=1};

  virtual ziString getName();
  virtual ziPropertyList* getProperties();
  virtual ziType getType();

  virtual ziStatus getScalarValues(ziCoordinate coordinates[],
   ziScalar values[], ziError errors[],
   int size,
   ziFlag flag=ZI_WAIT);

  virtual ziStatus getVectorValues(ziCoordinate coordinates[],
   ziVector values[], ziError errors[],
   int size,
   ziFlag flag=ZI_WAIT);

  virtual const char* getClassName(){return "ziField";}
 ~ziField();

protected:
  ziField(const ziString& name, ziType type);
  ziField();
  ziString _name;
  ziType _type;
```

```
  ziPropertyList _properties;
};

typedef map<ziString, ziField*, _ltstr> ziFieldList;
typedef ziFieldList::iterator ziFieldListIterator;


#endif // _ziField_h
```

Appendix C

Figures

Most pictures presented in this appendix, as well as movies and VRML scenes, are available on the Web site `http://iiun.unine.ch/ZoomIn`.

Figure C.1: Isolines on a cutting plane representing a snapshot in the evolution of a tracer injected near the outlet of the Rhone in the Gulf of Lions.

Figure C.2: Hyperstreamlines visualizing a stress tensor induced by two compressive forces (original work by T. Delmarcelle and L. Hesselink [27]).

Figure C.3: Line integral convolution texture of the velocity field on different slices of the curvilinear grid from a hemisphere-cylinder (original work by L. Hesselink and T. Loser).

Figure C.4: A probe showing the local flow field in detail, including the velocity field and its gradient (original work by J. J. van Wijk, A. Hin, W. de Leeuw and F. Post [114]).

Figure C.5: Pathlines around la Motte in the Lake of Neuchatel (Switzerland).

Figure C.6: Vortex extraction and tracking by means of isosurfaces of the vorticity magnitude (original work by D. Silver and S. Xin Wang [81]).

Figure C.7: Selection of a parallelipipedic region of interest with an ad hoc manipulator.

Figure C.8: Glyph representing the deformation of an infinitesimal volume of fluid by the drag of the velocity field.

Figure C.9–C.12: Horizontal and transverse cutting planes representing the temperature in the Gulf of Lions; the displayed phenomenon is characteristic of an upwelling situation.

Figure C.13: Arrows plot representing the velocity field at 30 m depth in the Gulf of Lions.

Figure C.14: Zoom into the main eddy.

Figure C.15: Two particles coming from the Rhone outlet and sinking according to an Eckmann spiral.

Figure C.16: Pathlines in the Gulf of Lions released interactively from an imaginery straight line along the coast.

Figure C.17: Cutting plane representing the velocity magnitude and arrows plot identifying the principal eddy during the simulation of the water circulation in a pool of an underground aquifer.

Figure C.18: Zoom into the upperleft small eddy of the previous figure showing a vortex swirling in the opposite direction.

Figure C.19: Isosurface of the turbulent kinetic energy (in gray) visualizing the 3D structure of the zones of high turbulence.

Figure C.20: Concentration glyphs showing the evolution in time of a tracer injected into the flow.

Figure C.22: Streamlines identifying two principal vortices with a different pool configuration and 3D Manhattan graphs representing the deposition of particulate matter on the floor.

Figure C.21: Zoom on the concentration glyphs.

Figure C.23: Top view of a comparison between a release of particles and diffusive tracer at the outlet of the Rhone river.

Figure C.24: Side view of the same comparison.

Figure C.25: Evolution of a tracer injected into the flow.

Figure C.26–C.29: Four variations of texture based visualizations of a magnetic dipole using line integral convolution (LIC).

Figure C.30: ZoomIn graphical user interface under SGI Irix. Application to the field of oceanography.

Figure C.31: ZoomIn graphical user interface under Microsoft Windows NT. Application to the field of medicine.

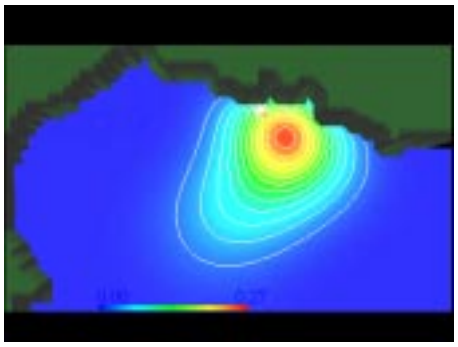


Fig. C.1: Isocontour

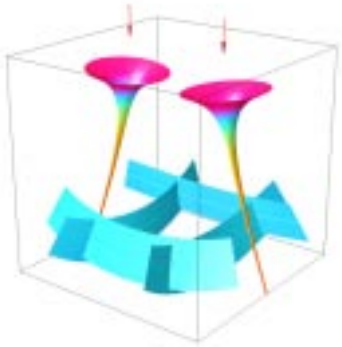


Fig. C.2: Hyperstreamlines

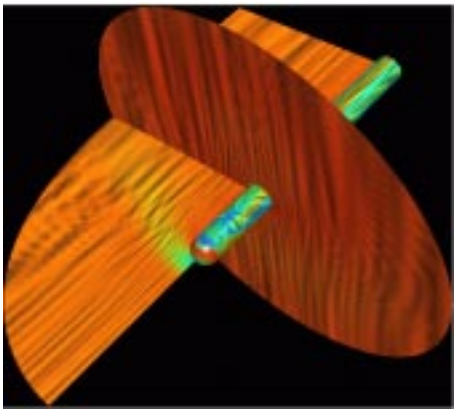


Fig. C.3: LIC textures

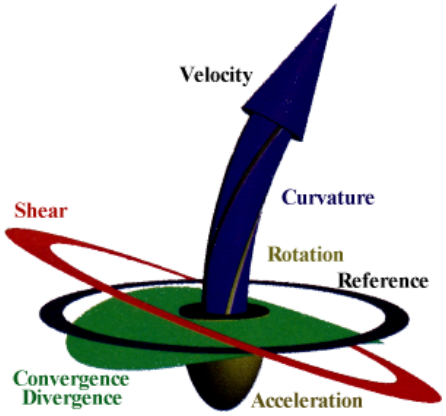


Fig. C.4: Iconic probe

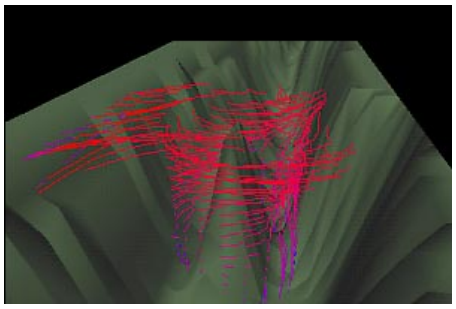


Fig. C.5: Pathlines

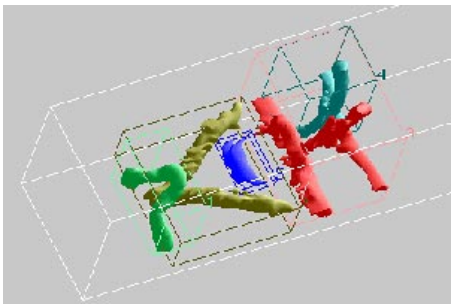


Fig. C.6: Vortex extraction

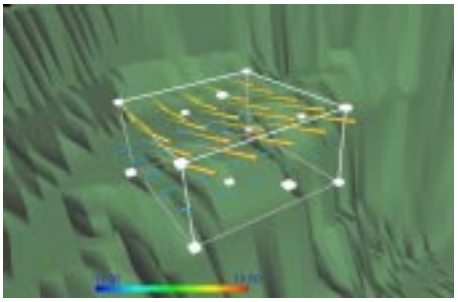


Fig. C.7: Manipulator

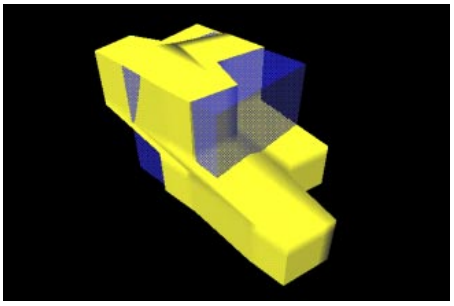


Fig. C.8: Strain glyph

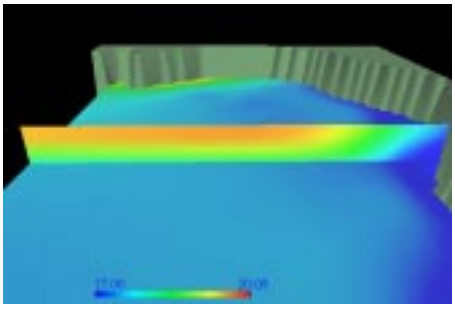


Fig. C.9: Cut after 25 hours

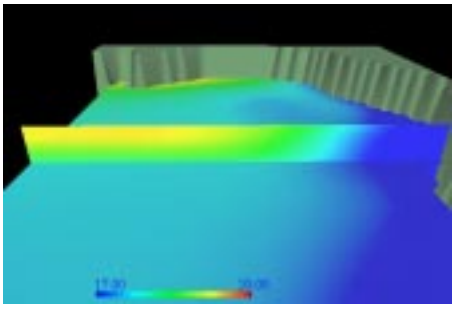


Fig. C.10: Cut after 49 hours

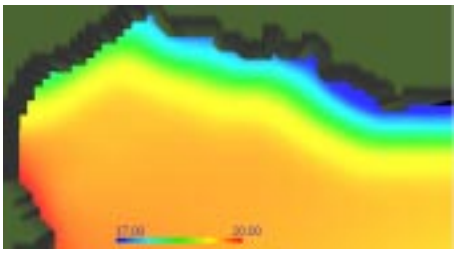


Fig. C.11: Upwelling after 25 hours

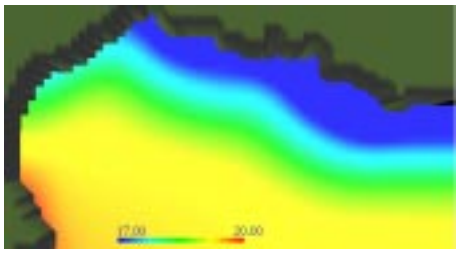


Fig. C.12: Upwelling after 49 hours

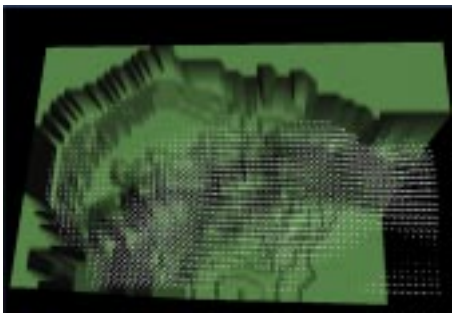


Fig. C.13: Circulation

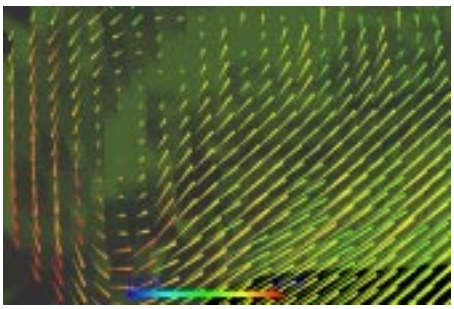


Fig. C.14: Zoom into the main eddy

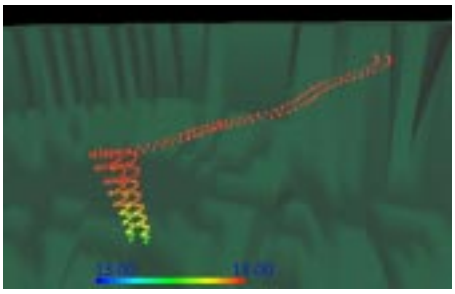


Fig. C.15: Particle paths

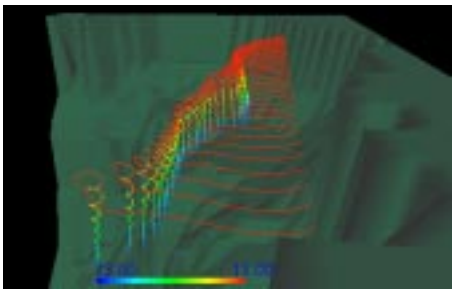


Fig. C.16: Particle paths

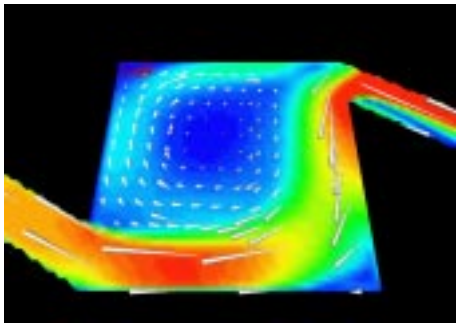


Fig. C.17: Speed and principal eddy

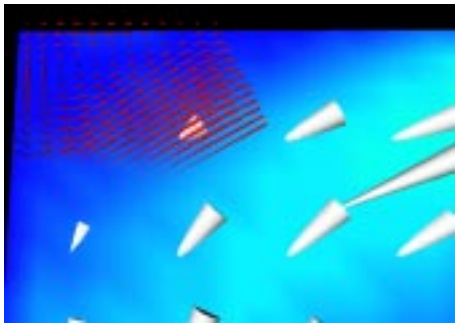


Fig. C.18: Zoom into the small eddy

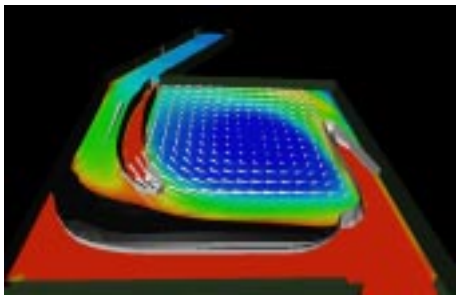


Fig. C.19: Turbulent kinetic energy

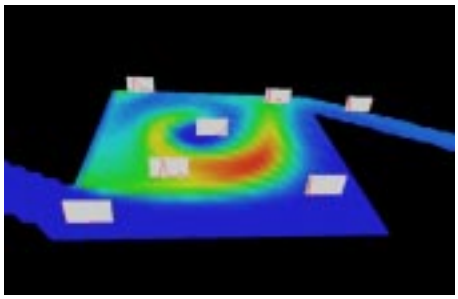


Fig. C.20: Concentration glyphs

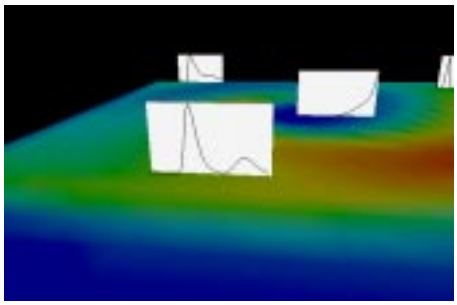


Fig. C.21: Concentration glyphs

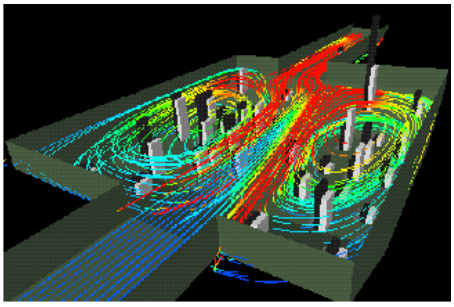


Fig. C.22: Streamlines and glyphs

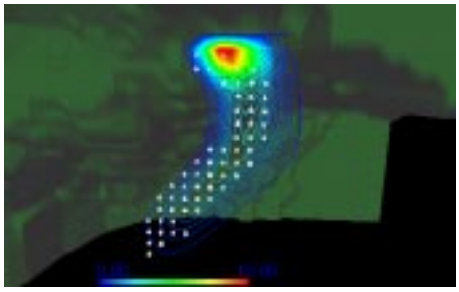


Fig. C.23: Particles and isocontour

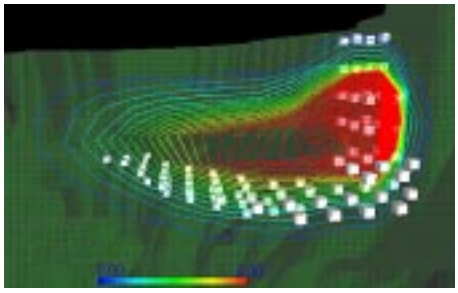
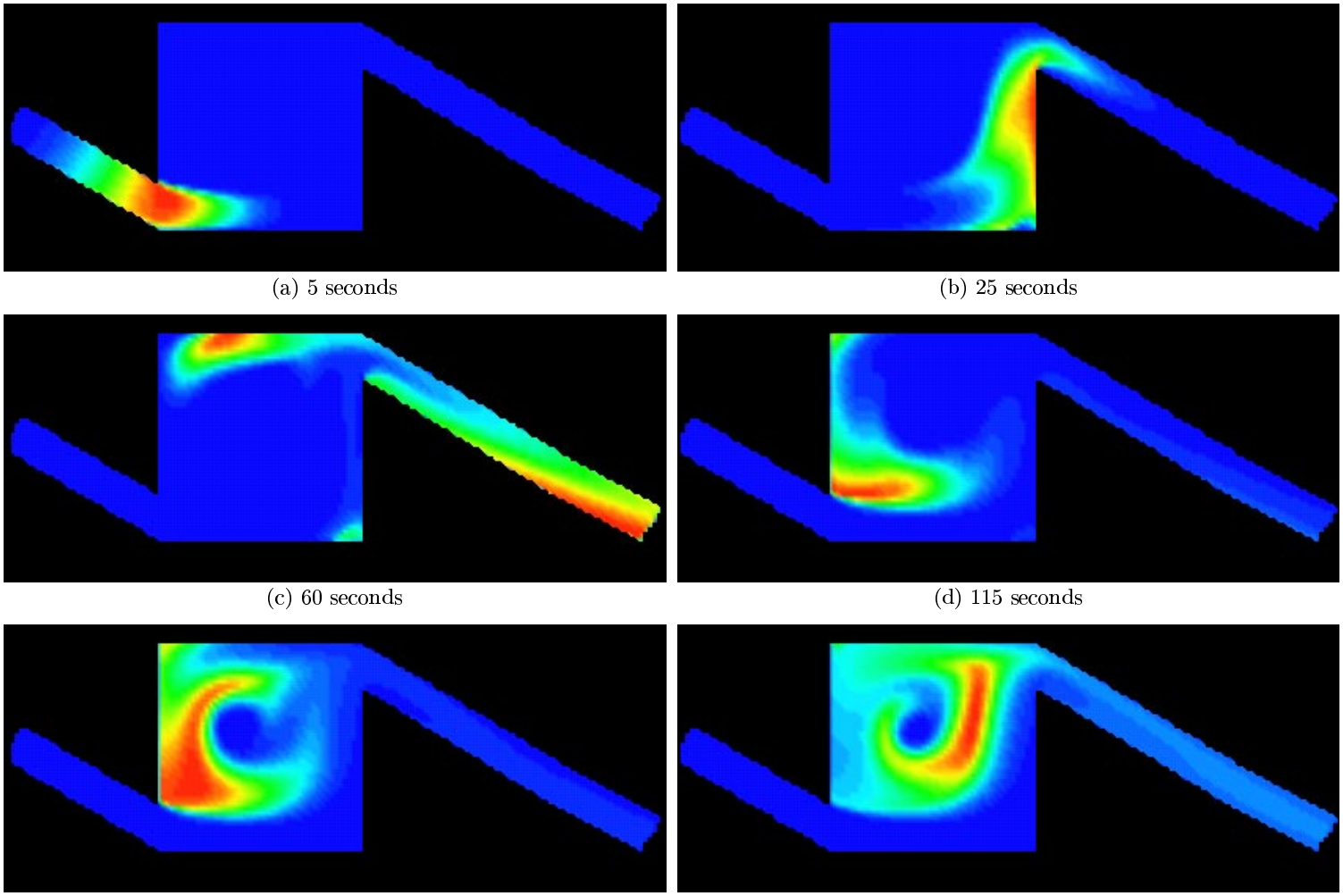


Fig. C.24: Particles and isocontour



(a) 5 seconds

(b) 25 seconds

(c) 60 seconds

(d) 115 seconds

(e) 235 seconds

(f) 300 seconds

Fig. C.25: Tracer plume

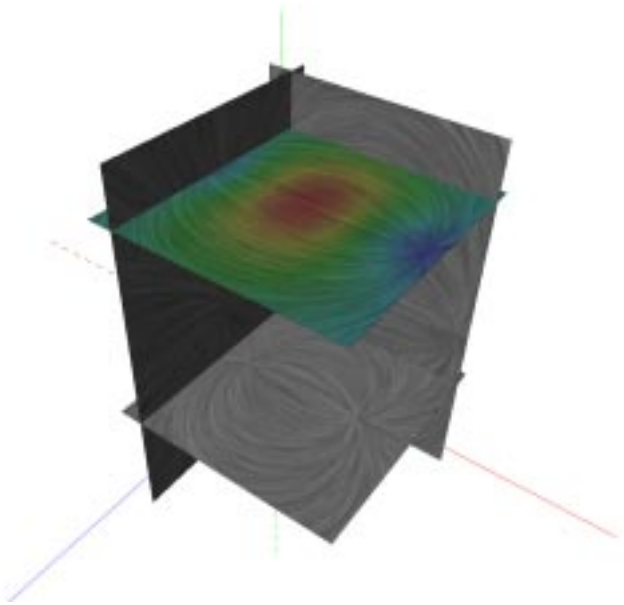


Fig. C.26: Line Integral Convolution applied to a dipole

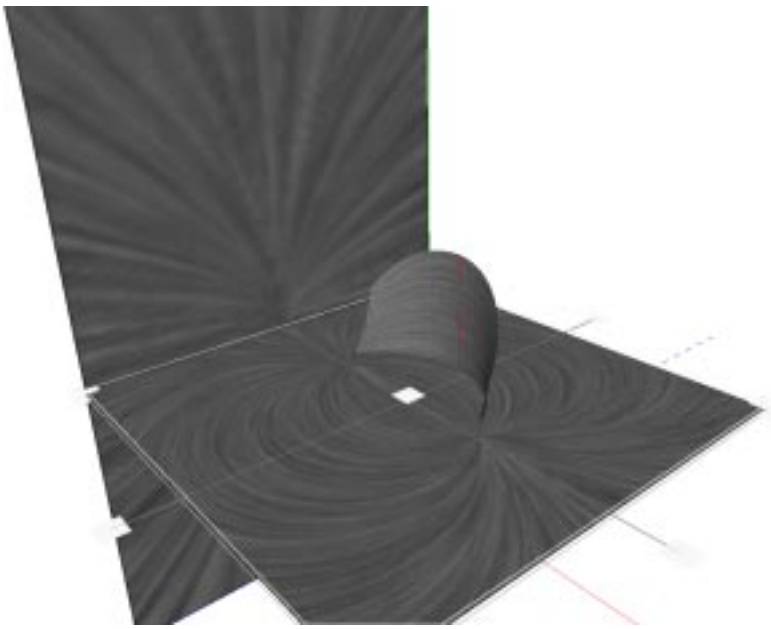


Fig. C.27: Line Integral Convolution applied to a curved patch

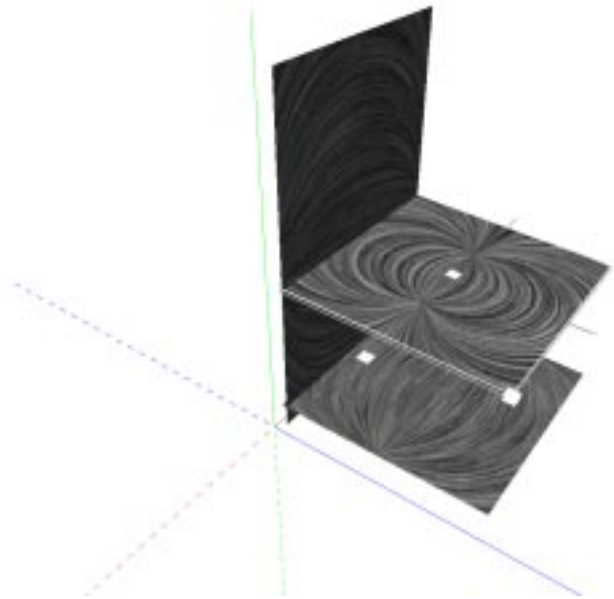


Fig. C.28: Enhanced Line Integral Convolution

Fig. C.29: Oriented Line Integral Convolution (OLIC)

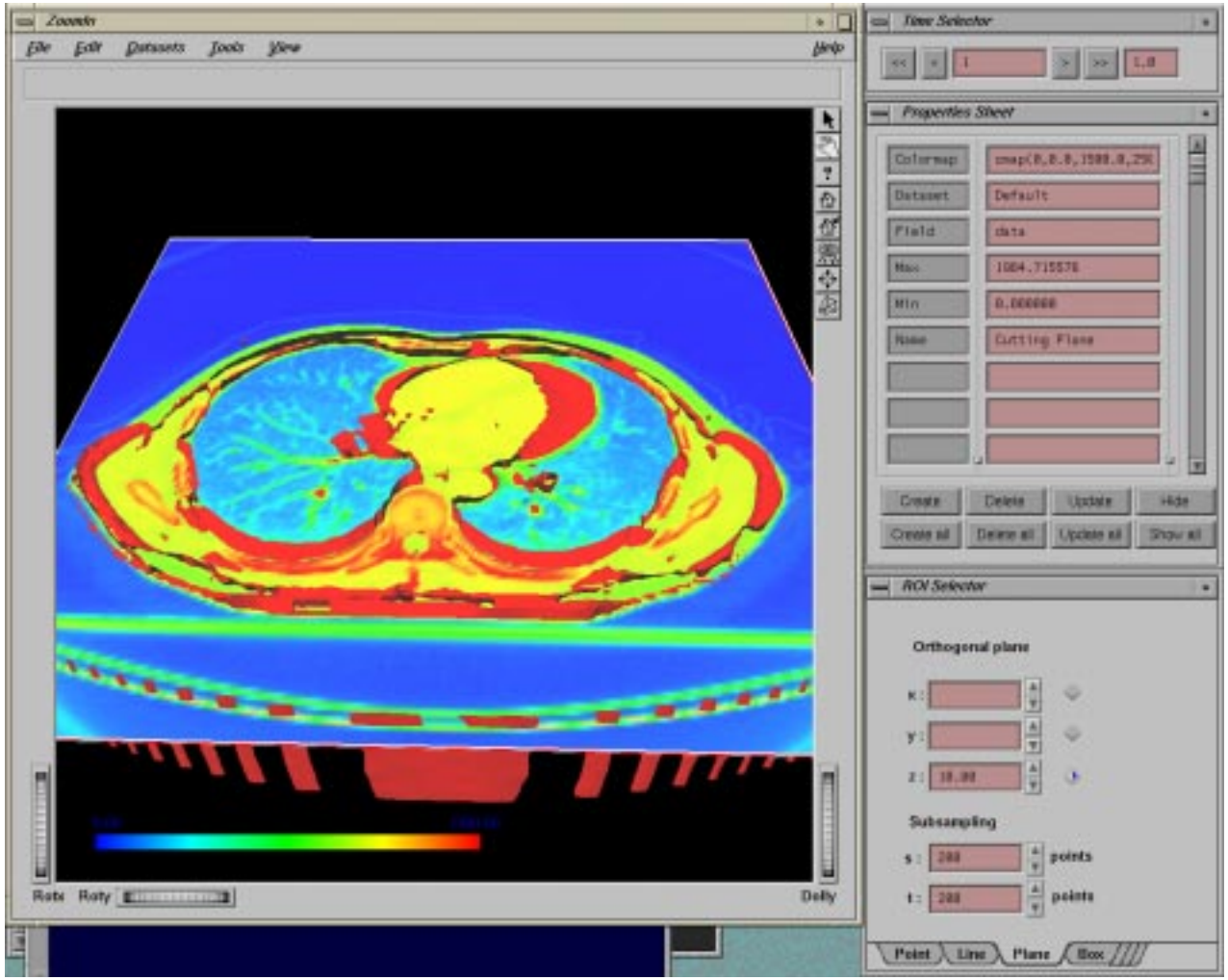


Fig. C.30: ZoomIn GUI under SGI Irix

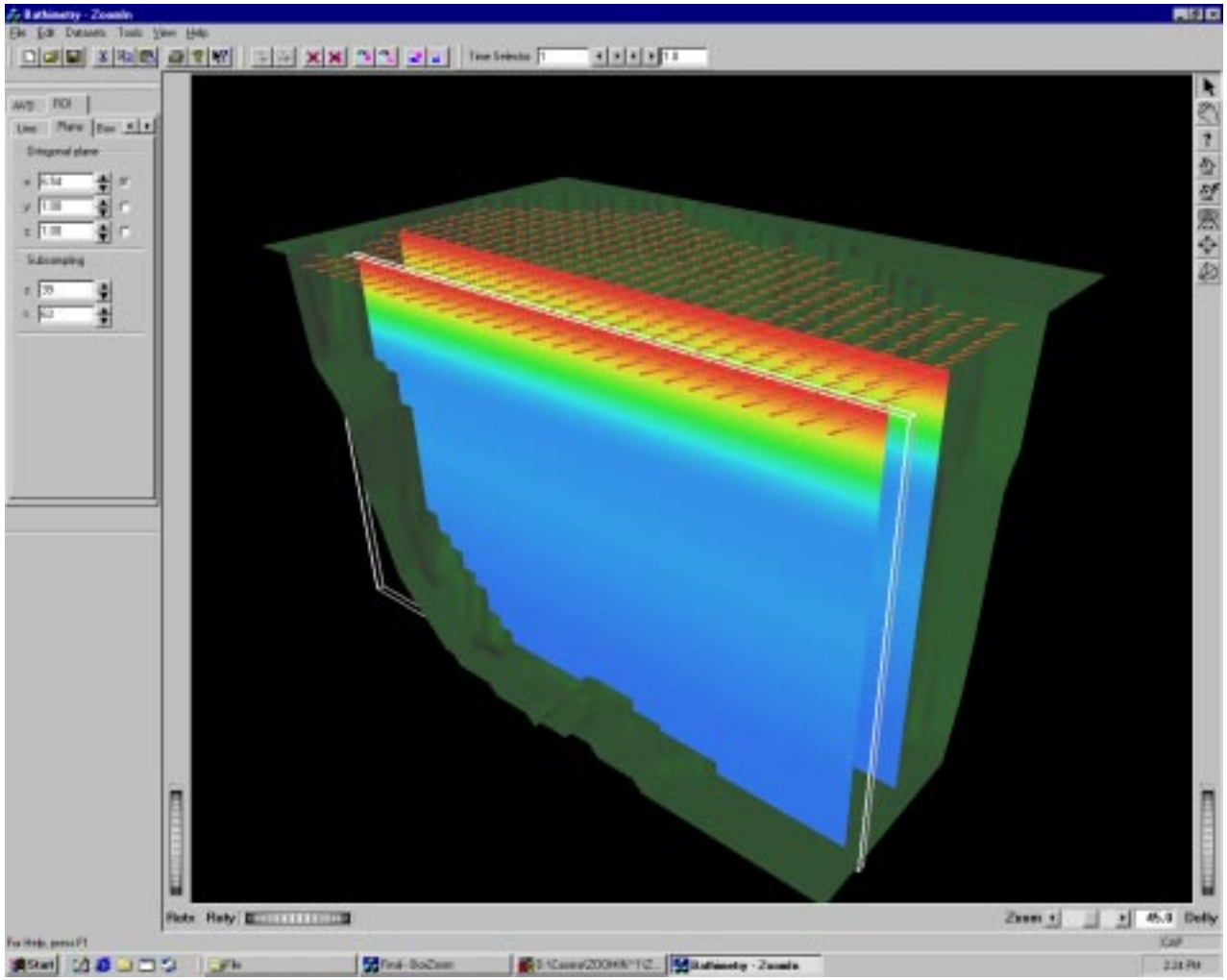


Fig. C.31: ZoomIn GUI under Windows NT