University of Alberta

Library Release Form

Name of Author: Ayman H. Ammoura

Title of Thesis: Data Mining In Immersive Virtual Reality

Degree: Master of Science

Year this Degree Granted: 2001

Ayman H. Ammoura
313 - 11030 82 street
Edmonton, Alberta
Canada, T5K 1L9

Date: _____

A picture is worth a thousand words,
and a virtual world has at least that many pictures.

University of Alberta

Data Mining In Immersive Virtual Reality

by

Ayman H. Ammoura

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2001

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Data Mining In Immersive Virtual Reality** submitted by Ayman H. Ammoura in partial fulfillment of the requirements for the degree of **Master of Science**.

Osmar R. Zaïane

Edward H. Cornell

Randy Goebel

Date: _____

To my heroine, guide,
and best friend ...
to my wife Elizabeth Zeppetzauer

Abstract

We propose a framework for visualizing remote distributed data sources using a multi-user immersive virtual reality environment. DIVE-ON is a system prototype that consolidates distributed data sources into a multidimensional data model (data cube), transports user-specified views to a CAVE, and presents various data attributes and mining results as virtual objects in 3D-stereo interactive virtual reality. In this environment, the user navigates through data by walking or flying, and interacts with its objects simply by "reaching out" for them. To maintain subsystem independence, DIVE-ON intra-module communication is provided via well established protocols, CORBA and SOAP. To support large sets of data while maintaining an interactive frame rate we propose the new concept of VOLAP. This data structure is well suited for indexing the levels of abstraction and the spatial decomposition of each associated virtual world.

Preface

This thesis contain a number of terms that are frequently used in their respective fields. We use the ***bold italic*** font to indicate that there is a glossary item associated.

Acknowledgements

I would like to thank Dr. Osmar Zaïane for his support and motivation all through this journey. The main idea behind visualizing OLAP operations in a CAVE environment is due to Osmar who believed that I can take this project on. Special thanks to Dr. Mark Green who provided the very first and most important guidelines on how to produce working examples within the VizRoom. Thanks also goes to Lloyd White with whom I have worked side by side for hours on end trying to configure and adjust various VizRoom glitches. Special thanks also to Peter Woytiuk and Pablo Figueroa and Ehud Sharlin who were always ready to answer all inquiries with a smile. I would also like to thank Paul Ferry for the constructive discussions and support regarding data visualization environments. Last but most certainly not least, my partner Marc Perron with whom I have coauthored my first conference paper.

The architecture of the **Virtual Data Warehouse** (VDW) was the contribution of another student of Dr. Zaïane, Yuan Ji. Yuan extended the original idea of a simple data cube constructor to a distributed heterogeneous virtual warehouse which became an integral part of our research. The term **CAVE** is the property of the Electronic Visualization Laboratory (EVL) at the university of Illinois. In our system we used the **MR-Toolkit** developed by Dr. Mark Green and Dr. Chris Shaw to communicate with the CAVE components. The term **CAVElet** is the property of Dr. Mark Green.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Immersive Virtual Reality (**IVR**) has the potential of becoming a powerful tool for the visualization of scientific data sets and models. Virtual reality technology has only recently started to address some of the requirements that make it possible to do "real work" [55]. On a different front, data mining methods and associated technologies have shown tremendous growth in the last decade. Commercial applications are constantly announcing new products that target specific tasks and types of data. In this thesis, the essence of a task-specific immersive virtual environment and the application of such technology for the purpose of data mining is presented. The challenges of moving in this new direction are discussed along with the solutions proposed.

The recent rapid development of data mining is a response to the fact that technology enables data collection, classification and storage at a rate far exceeding that with which we can analyze it. To better support the operations usually associated with data analysis and mining, researchers have developed the concept of a data warehouse to model voluminous data in a way that promotes the transformation of information into knowledge. Since vision is by far the human's most dominant sense, many researchers have targeted visualization as the means by which data is presented for analysis. Our proposed system **DIVE-ON** (Datamining in an Immersed Virtual Environment Over a Network) takes data warehouse visualization a step further by leveraging the human natural skills in virtual reality (VR). Using the sensorimotor skills gained in childhood, one maneuvers through the natural world and acquires spatial knowledge almost unconsciously. To support such nat-

ural skills, we have constructed an IVR environment that uses motion-trackers to acquire movement data, and then simulate the kinesthetic feedback through image transformation. This provides the user with a correlation between orientation and movement, to support a navigation interface that is transparent and highly capable in examining spatial data patterns [8, 83, 86]. DIVE-ON combines advances in virtual reality (VR), computer graphics, data mining, and distributed data warehousing into one flexible system that can be used for visual data mining with little or no training.

As presented in [5], DIVE-ON constructs a *virtual data warehouse* from a set of distributed Data Base Management Systems (DBMS). Information needed during a visualization session is communicated between the visualization module and the virtual data warehouse as extended markup language (XML) documents using Common Object Request Broker Architecture (CORBA) and Simple Object Access Protocol (SOAP) technologies (Chapter 7). The data warehouse uses a global schema that describes the location of the information needed for building an N-dimensional data cube or any of its subsequently derived subsets. Once the immersed user specifies the particular data dimensions to be viewed, the warehouse queries the individual sources, assembles the resultant cuboid as an XML document, and forwards it for visualization. In this thesis we also present a new structure,**VOLAP** (Visual OLAP) in Chapter 6, which is a hybrid spatial data structure designed to address the demands for ***real-time rendering*** and *interactive visual mining operations*[1] through the recursive spatial decomposition of the materialized "OLAP regions[2]."

1.1 Visual Exploration in Virtual Reality

The term *data visualization* refers to a process in which data is transformed so that certain attributes are pictorially depicted in a manner that promotes understanding. The concept is by no means new and actually dates back to the early 1950s[3], when one of the very first practical data visualization tools came about. However at the

[1]OLAP: OnLine Analytic Processing is a set of operations that allow interactive access to different data dimensions.

[2]OLAP Regions defined in Chapter 6.

[3]By visualization here the author refers to the use of electronics. Visualization on its own dates much further back than 1950.

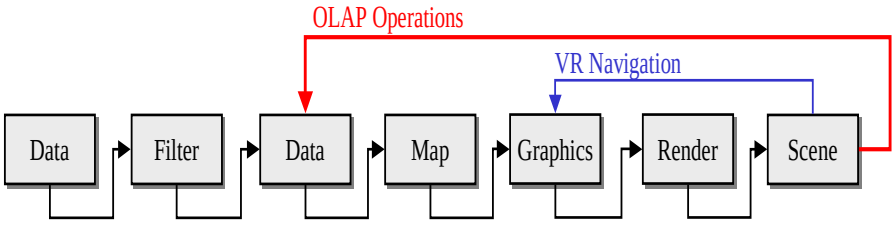


Figure 1.1: A typical data visualization process in IVR. Raw data is transformed to build a virtual world that is rendered one scene at a time depending on the user's location.

time it was not called a data visualization tool, it was called the *oscilloscope* and provided real-time data plotting solution. As the technology matured enough, more and more sophisticated visual tools established a niche and became an integral part in every imaginable field.

The main phases in a general data visualization process can be presented as in Figure 1.1; DIVE-ON is made up of modules that roughly correspond to each phase. The original data is first extracted and passed through a filter (preprocessing) that selects the most useful data in terms of the visualization goals. Filtering the data could be a simple numeric function that screens certain scalers or it could involve the construction of a new storage structure that is built around the main theme of the analysis. By "main theme" we mean the main concepts or topics of the analysis. For instance, if the purpose of the visualization is to examine the market trends, then that would constitute the main theme. Other themes may be budgeting, total revenue, employee productivity, atmospheric Ozone levels, etc. In the case of the DIVE-ON system, this step involves the building of an N-dimensional data warehouse (Chapter 3) that is then accessed through the interactive data mining operations (***OLAP loop***).

Next comes the synthesis of a virtual world based on the data obtained. In this step, a normalization process takes place so that only the "meaning" or significance of a scaler value is emphasized[4]. These normalized values are then encoded into virtual objects according to some metaphor. Once all aspects of the virtual world have been defined in terms of graphics primitives(OpenGL [91]), the system enters a ***visualization loop*** that is constantly updated to reflect the user's viewpoint of the

[4]In appendix B the normalization function Min-Max used is described.

world as they navigate through different scenes. As it will be discussed in detail in Chapter 4 and 5, while exploring the data warehouse the user can perform various operations that result in the construction of a new virtual world as new data sets are imported and mapped.

1.1.1 Goals of Visualization

The design of a data visualization system is usually proceeded by identifying a specific goal to be attained. In some instances, the purpose of the visualization is to obtain some sort of an insight that may provide hints on the formulation of an understanding of data patterns. Later visualization sessions may seek to accentuate or prove this understanding. As Daniel Keim pinpoints in [57], the first class of data visualization is the inquisitive case of *explorative analysis* where the user has no prior hypothesis about the data. The visualization process is then geared towards the development on an understanding of the trends or patterns that may lead to the formulation of a hypothesis about the data set or model. *Confirmative analysis* on the other hand is a visualization process in which a hypothesis is known a priori and the visualization session seeks to validate or to provide a "visual proof" of its integrity. Finally, certain facts about a given data set may be already established and the purpose of the visualization is then an appropriate *presentation* that highlights these facts. Usually the presentation of facts that have already been supported by a validated hypothesis emphasizes certain attributes about the data so that whatever is of an interest is clearly visible to an audience or a reader. In this last case, the developer is free to experiment and choose between different metaphors to arrive at the most effective presentation. For example, in a conference room people often use pie-charts to quickly illustrate certain facts using a data set as an evidence [58].

1.1.2 What is IVR?

Virtual Reality (VR) is a field in computer science that is built around the human visual and sensorimotor[5] systems. To better understand this concept, let us do an experiment. Walk into an area where there is no one else but you and look around at the objects that make up the space. Walls, chairs, pictures and anything else

[5]Our ability to use perceived environment events through the perceptual system (sensori) to guide our exploratory movement (motor).

that happens to be present. Fix your eyes at a particular point or object and start to reposition your head by standing, sitting or walking. You will notice that every single point around you will "come to life" and your view is under constant change as long as you are moving. Comparing this to a computer graphics scene, it seems that as you move, your entire surroundings get "redrawn" at an incredible refresh rate. This is the essence of Immersed Virtual Reality (IVR). The environment is called virtual because it is computer generated, and immersive because it provides its user with a wide egocentric field of view that creates the sense of "being there." Chapters 5 and 4 deal with this topic in detail.

With this understanding it should be clear that the ability to render realistic imagery is only a secondary measure of the quality of an IVR system. Of primary importance is the speed and accuracy of coordinating the appropriate image transformation with the user's motion; or more precisely, how accurately can the system provide the user with kinesthetic feedback.

1.1.3 What is a CAVE-Type IVR?

While the gathering and building of information can be done from any location, the actual visualization experience takes advantage of the most sophisticated virtual reality environment in Canada at the University of Alberta. This facility is called ***VizRoom*** that is formally known as CAVE Theater (CAVE and VizRoom are used interchangeably). **CAVE** is a recursive acronym (Cave Automatic Virtual Environment)[23] and refers to a visualization environment that places the user within three (9.5 X 9.5) feet walls. Each of these walls is back-projected with a high-resolution projector that delivers the rendered graphics at 120 frames per second. The graphics projected are in stereo (60 frames per second for each eye), which enables DIVE-ON to create stereoscopic views that can be seen by the user by wearing lightweight shutter glasses. To be able to power such high graphical demands, the CAVE in VizRoom is powered by two SGI Onyx2 InfiniteReality Rack systems. These are 4-processor specialized graphics engines running at 195MHz R10000 IP27.

1.2 Defining a Data Warehouse

The most widely known definition of a data warehouse is the definition given by Bill Inmon, "a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process" [53]. This definition is comprehensive and distinguishes data warehouses from all other data repositories. It is built not to facilitate the day-to-day operations of an organization (OLTP) but to provide predictions, patterns, anomalies or evidence upon which certain corporate decisions can be made. Constructing the data warehouse requires the transformation of traditional data models (usually the ER model) that exist on DBMS systems into a multidimensional subject-oriented data model [66]. Data warehouses are built with a *central theme* in mind, which is the goal of the analysis task prompting the warehouse creation. The notion of dimension can be thought of as the association of a DBMS entity or a perspective according to which an organization wants to view their data. For example, a company may want to construct a data warehouse for the purpose of budget analysis. In this case, the central theme could be *dollars budgeted* while some of the possible dimensions may be *location, product* and *time*. With such a warehouse, we are able to instantly obtain budgeting information regarding a given *product* in a *location* for a specific *time*. To be able to better support OLAP operations, a data warehouse is often implemented as a hierarchical N-dimensional data model that is called data cube. The data warehouses generated by DIVE-ON are indeed N-dimensional data cubes [35]. The need for a hierarchical data cube model becomes clearer when we discuss the data cube constructor module.

1.2.1 The Data Cube

Jim Gray et al. [37] introduced the CUBE-BY operator which provided a mean for restructuring databases to accommodate real-time analytic queries. The data warehouse often is an implementation of an N-dimensional data cube. There are two important concepts that are an integral part of the data cube, namely dimensions and measures. A *data dimension* is a collection of some database attributes that relate in some logical manner. The concept of a *measure* is related to the main theme of the data cube. For example, a data cube that is constructed to study product sale patterns would include the measure "units sold" that indicates the number of

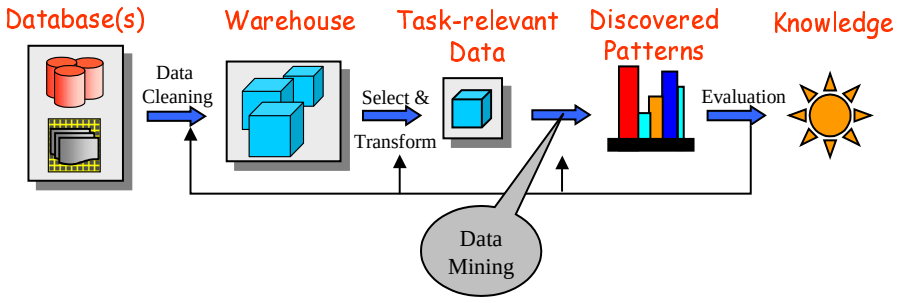


Figure 1.2: The different phases that make up the KDD process

items sold with respect to data dimensions $(X, Y, \ldots)$. Unlike what the name "data cube" suggests, it is often the case that a data cube is constructed from more than three dimensions. Data cubes are often referred to as N-dimensional data cubes, $N \geq 2$, a 2D data cube is simply a table.

1.2.2 The KDD Process

The two terms *data mining* and *knowledge discovery* are often used interchangeably. In actuality however, data mining is only a phase in the process of knowledge discovery. Knowledge discovery in databases (**KDD**) is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [29]. This process consists of several phases, each of which distills data to a form that is usable by the following phase. Figure 1.2 outlines this process. Row data is the beginning of the process and patterns that are defined and formulated in a specific manner is the end product. Whether the starting data is in the form of a flat file or a sophisticated DBMS system, it is referred to as "raw" data in the KDD process because it may be in a form that does not facilitate the understanding of hidden patterns and trends that are deemed interesting with respect to the goals of the analysis [41].

The entire process of nontrivial extraction of implicit, potentially useful and previously unknown information from a database consists of three main phases (each phase is further subdivided). First is the *preprocessing phase* where irrelevant and incomplete data is removed. Preprocessing, also known as data cleaning, transforms

the raw data found in a DBMS into a collection of complete relevant data items. The second phase is *data integration* and consolidation which combines several, possibly heterogeneous, preprocessed sources into a homogeneous source that is suitable for mining. The data warehouse (N-dimensional cube) is usually built in this phase (Figure 1.2). The purpose of constructing a data warehouse is to create a consolidated view of the important data that will aid the decision making process. For example, to analyze sale patterns for an international company, one would construct a data cube that focuses only on figures that relate to sales and disregarding irrelevant information that may exist on the flat files or DBMS of a given local branch or location. The next step in the KDD process is the *data mining* phase where mining algorithms are applied to the entire data or some subsets of it.

The final step in the process is the analysis of the patterns output by the data mining algorithms. This final step is where our immersive environment fits in. The need for visualization is stressed given the fact that the last two steps are usually applied in iterations. Using visualization techniques to explore the results, allows for fast pattern evaluation so that, if needed, the previously applied data mining algorithm can be "fine tuned" and reapplied as often as necessary.

1.2.3 Constructing a Data Warehouse

The industry seems to be split exactly in the middle when addressing implementation issues of the data warehouse. Since the introduction of the concept of a data warehouse, implementations seem to follow the ER (entity-relationship) model that have been very successful is building DBMS systems. This model is called Relational OLAP (**ROLAP**) which uses tables to build the data cube. The other is **MOLAP**, which is a multidimensional OLAP model. As the name suggests, MOLAP uses multidimensional arrays to implement the data cube. As will be illustrated in Chapter 3 both models have a task-specific advantage.

1.3 DIVE-ON: A System Overview

Abstractly, DIVE-ON consists of three task-specific subsystems. Figure 1.3 shows the various layers comprising the complete system, from the data sources to the visualization environment. The first subsystem is the Virtual Data Warehouse (**VDW**)

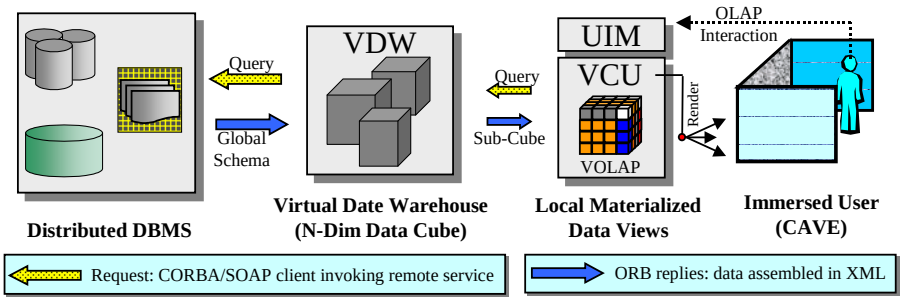


Figure 1.3: The three components of the DIVE-ON system

(see Figure VDW), which is responsible for creating and managing the data warehouse over the distributed data sources. The second subsystem is the Visualization Control Unit (**VCU**), which is responsible for the creation and the management of the immersed environment to insure that the "reality" in virtual reality is not compromised (Figure 1.3, details in Figure 4.5).

The User Interface Manager (**UIM**) (Figure 1.3), details in Figure 4.4) handles the direct application-control interaction as well as the automatic interaction that provides the kinesthetic feedback for navigation and aggregate manipulation. Inter and intra subsystem data exchange is provided by a set of specialized interfaces which implement specific protocols to guarantee extendibility and subsystem independence. This communication takes the form of client and server applications, using both Common Object Request Broker Architecture (CORBA) over TCP/IP and Simple Object Access Protocol (SOAP) over HTTP.

The rest of this paper is organized as follows. Our immersive display technology is presented, loosely corresponding to a CAVE. We include our motivation for using this environment, and virtual reality in general. We then present the software architecture of the Virtual Data Warehouse(**VDW**), and explain how the XML-based (XMDQL) queries are created and distributed amongst the various data sources.

1.4 Thesis Motivation

It has been estimated that the total worldwide OLAP market, including implementation services, was about just over $3 billion for the year 2000. It showed a 40 percent compound growth rate over the four years until 1997, 45 percent in 1998,

20 percent in 1999 and about 22.5 percent in 2000 [74]. Immersive virtual reality has been evolving in a different way. IVR applications now include way-finding and walkthroughs in possibly hostile environments, virtual prototyping (vehicles and spacecrafts), medicine (visualized surgeries, MRI, training and planning), "experiences" applied as clinical therapy (relieving Vietnam experiences to treat post-traumatic stress disorder), and entertainment of course [55]. With DIVE-ON, we sought to identify the requirements, characteristics and structure needed to build a system that bridges these two rapidly evolving fields of computing science. Using the methods of data warehousing, our system would then be able to allow a user to explore multidimensional data using an interface that facilitates spatial data exploration.

It is known that vision is by far the human's most dominant sense (more than 50 percent of the human neurons are devoted to vision) [31]; as a result, pictorially depicting a set of information effectively increases the communication bandwidth between human and medium. Humans do, however, have another natural skill that can be used in providing a transparent human-computer interface. This is our sensorimotor system. It has been shown that IVR can indeed exploit our abilities in 3D navigation to effectively relay complex spatial relationships [83, 55, 42]. With DIVE-ON, we use the human's visual abilities to convey certain facts about the data and use the human's natural spatial pattern recognition skills in presenting spatial correlations among such facts.

1.5 Thesis Contribution

We have built a system that, to date, is the first that provides the user with a data mining toolset in immersive virtual reality. In the solution presented in this work, every attempt has been made to provide the user with a task-specific working environment by addressing several problems inherent to data visualization in general and IVR in particular. Occlusion problems are addressed by introducing the concept of a clutched and hand-coupled floating menus (5.3.1) and by providing a two different visual encoding of data (4.2.1). Navigation, orientation and aggregate query functions are all provided in a manner that does not disengage the feeling of presence within the IVR (5.2).

One of the most significant issues in any immersed virtual environment is the ability to successfully immerse the user in VR. As will be discussed in Chapter 5, immersion is directly related to the system's ability to provide the user with a fast and accurate visual feed back corresponding to their motion within the CAVE. This is how we simulate the kinesthetic feed back that is an integral part of our sensorimotor system. As presented in [30], the frame latency rate of a system is directly related to what is known as ***VR sickness***. With this regard, we have designed the VOLAP structure to directly address scalability versus reality issues. VOLAP was first presented in [4] as a hybrid data structure that allows DIVE-ON to provide a "realistic" experience regardless of the size of data in the environment. This is accomplished through guaranteeing the frame rate in IVR regardless of the number of the data aggregates present.

VOLAP also addresses the concern of interactive OLAP operations through the materialization of various OLAP regions based on the concept hierarchy describing the data dimensions. Since the introduction of the CUBE-BY operator [37], there has been a great deal of research on database representations aimed at devising a spatial data structure that is well suited for indexing the data cube [13, 28, 38, 43, 1]. In the graphics community on the other hand, a very important issue is the ability to partition the VR world into chunks so that they can be rendered at a reasonable rate of speed [3, 27, 60, 79, 89]. The reason that VOLAP was needed is to be able to combine both advantages into one structure and hence be able to obtain an interactive OLAP operations at a guaranteed frame rate. The VOLAP structure that is presented in this thesis consists of a data part, the VOLAP-cube, and recursive partitioning spatial index called the VOLAP-tree (see Chapter 6).

1.6 Thesis Organization

This thesis is organized analogously to the way DIVE-ON was built; each module or main task is presented in a separate chapter. First, some related work is reviewed in Chapter 2. The VR research is extensive but our focus will remain on research that address questions that relate to the implementation of the DIVE-ON system, namely user interaction concepts. Then, we present some of the work done in applying the CAVE technology to the arena of data visualization. This is followed by presenting

the main concepts of a data warehouse. In Chapter 4 and 5 we present the virtual environment that is created by the system. Chapter 4 discusses how the environment is built while Chapter 5 presents the interaction techniques available. In Chapter 6 the VOLAP structure that we have designed to address problems of fast interactive visualization is presented.

The communication layer that connects system into one unit is presented in Chapter 7. In this chapter we illustrate how CORBA is implemented so that the data can be transported to the CAVE hare at the University of Alberta. Finally we present a summary of the work presented and our future agenda that highlights the future direction for expanding and improving on the system.

Chapter 2

Related Work

To the best of our knowledge, there is no published work that addresses the main points of the DIVE-ON system; specifically, providing the user with an immersive virtual environment for interactive OLAP operations on a remote data warehouse. However, the research in the fields of VR, data mining and computer-human interaction is extensive. It is the amalgamation of these into one body that is novel in our research. In this chapter some related work from these fields is presented.

During the past few years there has been a significant increase in the number of commercial applications that have been developing applications that relate to Jim Gray's concept of the data cube introduced in 1997 [37]. The potential of ability to gain an understanding that will aid corporations in their decision making process is too attractive to miss. As a result, we have been witnessing a good deal of new and old companies that have been targeting the market segment [7, 22, 51, 52, 80, 48]. The visual aspects of these products is limited to the standard desktop display.

2.1 Visual CAVE Applications

In this section we present some of the software applications that use the IVR environment of the CAVE. CAVE applications have been used in training people as reported by Kenyon et al. [59]. In virtual reality, the cost and risks of providing a hands-on training is minimum. What is very important in this study, [59], is the fact that what is learned in a virtual environment is retained in the real world; however, as suggested by the study, the converse is not true. CAVE applications also have immediate applications in other sciences including chemistry. Bethel has

illustrated how the CAVE environment can be used to simulate chemical flooding in a controlled environment [12]. Two of the best surveys in the field are those of VanDam and Brook in which a classification and an in-depth discussion of various IVR work is presented [55, 83].

The VR team at the Electronic Visualization Laboratory (**EVL**) have undertaken a research agenda that provide means of making the CAVE environment more suitable for scientific data visualizations through collaborating remote VR users [61, 62, 71]. During the past *ten* years, the EVL[1] team has been focused primarily on building a frame work that can be used for multi VR environments over a network. Some of their older and mature work includes the frame work CAVEern which was used to implement the the following application.

CAVEvis [54] is a set of tools for the interactive visualization and exploration of large sets of time varying scalars and vector fields such as those from the NCSA [33] simulations using the CAVE environment. The authors have addressed the problem of interactive frame rates by dividing the computations between several machines that run asynchronously. In this work, the authors also emphasized the fact that a great deal of effort was required to ensure that every part of the visualization process is running in synchronization without ***blocking***. It is also noted that another significant portion of the work was devoted towards the ability to render massive amounts of data interactively; how that was done remains unpublished. The interaction within the CAVEvis was not a main focus in the work, as it was presented through the use of 2D-based widgets that appear at the zero-parallax. The next section will take a closer look at the human-computer interaction factors.

2.2 How To Interact With Virtual Worlds

As compared to the exponential growth of data mining, VR applications that perform real work [42] has been in a state slow progression. This is because while the scientific community seems to agree on the fact that IVR applications have tremendous advantages [55, 83], they seem to be in a total disarray when it come to *how the VR should be presented and conceived by the user.* The difficulty is how the

[1]The EVL new home page is at (http://www.evl.uic.edu/) which contain links to 48 online publications.

user interaction should be performed in certain tasks and environments. In this section some recent research that explored some research issues in IVR interaction is presented.

In 1999 Robert Lindeman et al.[63] provided us with a good insight into the type of user interfaces that seem to be most effective in immersive VR. Differences and similarities between 2D and 3D computer-human interaction techniques are presented here. The authors feel that IVR is soon to be a means by which people will do "real work," not just entertainment. As a result, they examine several different interaction techniques and focus their study on the usability of some types of VR. This demands a clear understanding of how to classify different VR computer-human interaction techniques. For that purpose the authors illustrate how to perform a user interface decomposition called *widget-level* decomposition which decomposes the user interface components based on their manipulations. Another form of classification is outlined to include *discrete* and *continuous* actions. The interaction scenarios presented are different from those needed in the VizRoom, but the ideas presented can be easily applied. For example, one of the main IVR interaction techniques presented uses a *cyber-glove* interface, which is a device used to enter options and command in the form of hand gesture while wearing a "wired" glove. From this idea we have adopted the *3D pointer* in our application.

S. Feiner et al. in [30], present their view of IVR interaction as "**Windows on the world**." The authors describe different means of using already established 2D windowing techniques to implement 3D windows in immersive VR. Menus, or windows, are classified into three categories based on their association with the VR environment. The *worldfixed* menu is a set of menus that will always appear in the same exact position in the virtual world[2]. *Object-fixed* menus are those that are associate with only one single object in the world. The last category is the *view-fixed* menu which always appear in the same position relative to the current user view. This menu classification provides a new perspective to the problem of maintaining a usable interaction metaphors in VR. This view seem to be restrictive in a large VR world. It is not often possible to locate a world-fixed menu if the current user location is far enough so that it is no longer visible. However, the idea

[2]For further illustration of what these types of menus mean see Chapter 5

of an object-fixed window, or menu, is effective. In our work every data object is associated with a fixed panel that can be set visible whenever the user queries that particular aggregate. We refer to this in Chapter 5 as *aggregate-based* interaction.

Kurt Thearling et al. [82] provides a very good look at the most recent work in the arena of data visualization. Although the work is *not* intended for IVR applications, most of the ideas were instrumental in understanding some of the inherent difficulties in data visualization. For example, from this work it became evident to us that *orienteering* is one of the most fundamental issues in building a cognitive map of the IVR. As a result, early in the project we began to examine possible solutions to properly orient the user. Some of the ideas included maintaining a grid (see 5.1) that defines the three axes (x, y, z) and a 3D navigation pointer based on the 1986 Skitter of Erik Bier [14].

Thearling et al. also emphasize the importance of "trusting the model." This notion is very simple yet extremely important. Since our system is to produce a visual form of remote data aggregates, decisions are going to be made. Therefore, it is imperative to maintain the users trust in the projected data. This trust can be generated by being able to provide the user with the reasons that made a cube big, red, small or blue. In simple terms, if the user encounters an interesting object they should be provided with means to query the exact data lineage that contributed in creating this particular object. In our project this is equivalent to using the 3D pointer to select an object. This will set its associated panel visible and display the attributes values and measures of that aggregate.

2.3 VOLAP Related Work

As will be presented in Chapter 6, DIVE-ON deals with the duel problem of interactive OLAP operations while maintaining an interactive frame-rate by devising a suitable data structure. The data visualized at any given time is a subset of the N-dimensional data warehouse that exists somewhere across a network. As a result, the needed structure should be simple and able to handle both requirements simultaneously. The notion of using specialized data structure for the purpose of accelerated rendering of various visualization tasks is by no means a new concept [60, 89]. However devising a structure that is well suited for fast rendering and in-

teractive OLAP operations is indeed new. Upon initializing, the VCU requests the metadata that identifies the data sources from the VDW that is then used to construct the VOLAP structure. The meta data is also needed to provide a map for the VCU instructing it on how to aggregate the data required for building the various OLAP views. Next we present a family of trees that have shown good performance results.

The **R*-tree** (1990) which was optimized to the **X-tree** (1996) and later modified to include the concept hierarchies defining the data dimensions, the **DC-tree** (2000), are well optimized data structures designed for fast data warehouse updates and interactive range-query execution [11, 10, 28]. Such data structures create a representation of the data that optimizes range queries and points searches algorithms. As far as the computer graphics community is concerned, there have been a continuous array of new methods that make interactive rendering possible in many data-intensive applications. These methods are geared to render scenes that involve massive light and texture calculations that make up a VR world. Data is obtained partitioned and then rendered [21, 27, 64, 79].

Recently (2000), Ertl et al. [27] published some of their work to address one of the main problems that motivated us to develop the VOLAP structure. In their specific problem, they need to visualize three-dimensional data sets that result from real-time simulations or measurements. Such data streams can be enormous and, similar to DIVE-ON, interactive rendering becomes impossible. Their solution involves a *lossy compression* of the data in such a way that the number of cells which have to be processed by the visualization mapping is reduced. It is then up to the user to choose between quality or speed. Clearly this is not a feasible solution as far as our application is concerned.

Aliaga et al. also have published a very interesting solution to the problem of guaranteed frame rate [3]. Unlike T. Ertl, their method does indeed preserve the integrity of the data. Their creative solution involves the use of images[3] to *replace distant geometry.* The algorithm presented automatically chooses a subset of the model to display as an image so as to render no more than a preset number of

[3]These images are built from the VR world as JPEG and texture mapped at the appropriate distance.

geometric primitives. This work significantly improves the system frame rate for walkthrough applications of complex models. This is also significant for DIVE-ON from another important perspective. In our future agenda we plane to investigate this method to enhance the data mining views in conjunction with our VOLAP structure.

Before ending this chapter, it is important to mention the **3DVDM** (3D Visual Data Mining) group at Aalborg University in Denmark. We became aware of their direction in 2000 but to date, and to our knowledge, there has been no published work. It seems that they are working on immersed virtual data mining in a cube CAVE (a CAVE with six walls). This means that the user would have a ceiling and a floor that are also back projected. Seven faculty members are involved in this work, which indicates a serious commitment. The interested reader should check this page for interesting future announcements: (`http://www.cs.auc.dk/3DVDM/index.html`)

Chapter 3

Data Warehousing

The term data warehousing refers to the process of constructing a data storage that facilitates the viewing of data on different levels of abstraction. In the following chapter main concepts about a general warehouse are presented. This is followed by a presentation of the Virtual Data Warehouse (VDW) which provides a centralized summary of distributed Data sources. The VDW was constructed by Yuan Ji[1] in a collaborative effort to make DIVE-ON a data-source independent system [5].

3.1 Multidimensionality in Data Analysis

A fundamental principle in designing a data warehouse is the identification of data dimensions that are related to the main theme in mind. The information presented in a $(n \times m)$ table defines a two dimensional table. For example considering one of the tables in Figure 3.1, the attributes along the first column and first row define the *domain* of these data dimensions. The first column entries could represent the dimension PRODUCT while the first row could represent the dimension TIME (four quarters are seen in the figure).

Finding a set of tables that are related with respect to another concept, produces a third dimension. For instance, in our example above we have placed a set of TIME and PRODUCT tables of four locations together namely Lethbridge, Red Deer, Calgary, and Edmonton. The information presented in these tables can be used to build a three dimensional data cube with respect to (LOCATION, TIME, PRODUCT). The cells of a table correspond to the total revenue of a particular vehicle during

[1] Yuan was a course-based Student of Dr. Osmar Zaïane, 2001.

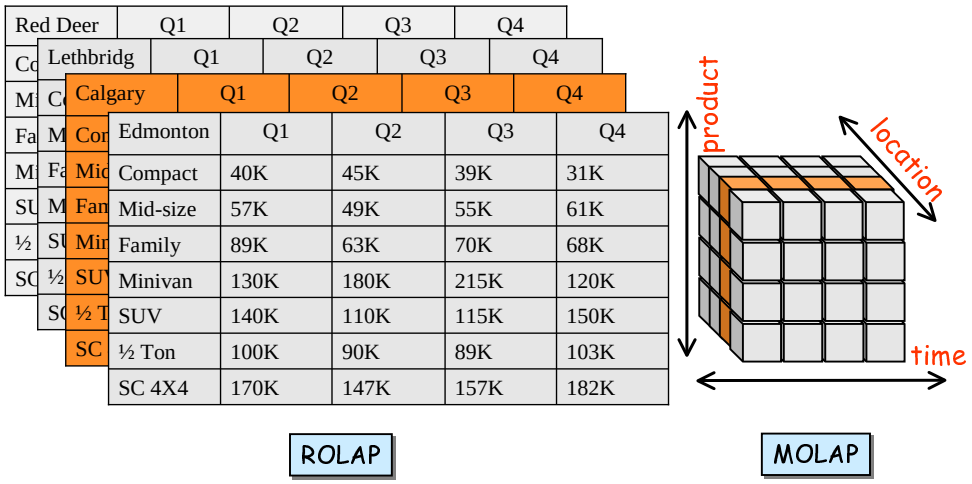


Figure 3.1: Multidimensional data models. ROLAP uses relational tables to store the different GROUP-BY aggregates. The MOLAP uses contiguous arrays.

a given quarter. These values make up the *measures* of the presented data cube. These measures are easiest thought of as a multivariable function, $f(t, p, l)$ where t, p, l are some time, product and location respectively. The figure here shows only one such function per (t, p, l), a data cube usually contains several such functions as we will illustrate in Chapter 4.

3.1.1 Data Abstraction and Concept Hierarchies

For the purpose of data mining, its is important to provide a way that enables the user to vary the level of detail on demand. Every dimension can be thought of as a perspective or a logical organization of entities according to which an organization wants to view its data. A dimension definition should also include a *concept hierarchy* that further describes the dimension in terms of a sequence of mappings between low-level concepts and higher-level concepts. That is, a concept hierarchy is an expert's interpretation of how a given dimension could be abstracted on varying levels. Each level in this concept hierarchy defines a level of abstraction. For example carrying on with the above described dimensions, a typical LOCATION dimension can be abstracted as seen in Figure 3.2. The general schema that describes each conceptual level is called the *hierarchy schema*. The root of most concept hierarchies is often reserved for the keyword ***ALL***. ALL can be defined as the super

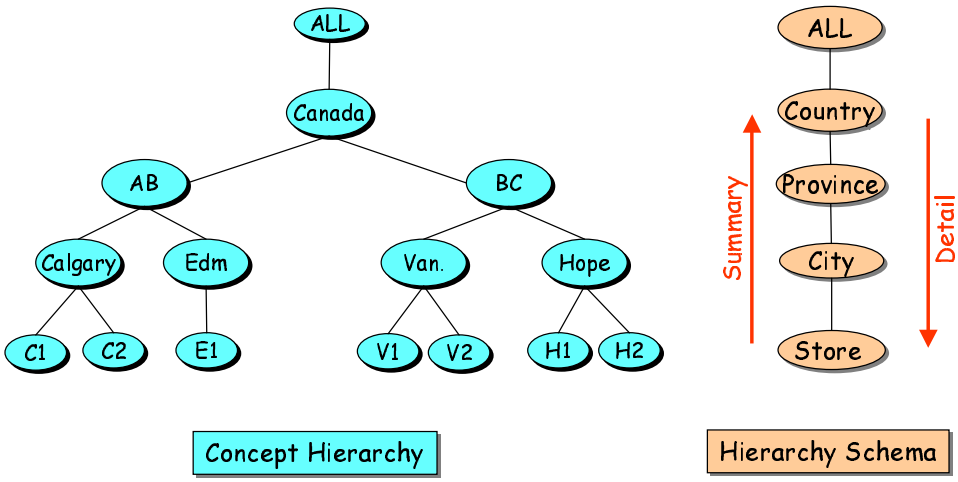


Figure 3.2: The concept hierarchy and hierarchy schema for the dimension LOCATION.

aggregate that encompasses the maximum degree of abstraction of a dimension. While the various levels of summary or abstraction are sets of attributes, ALL is a single attribute value. This means that ALL for a given dimension corresponds to only one cell in 1D, one column in 2D, a plane in 3D etc. In general, given an N-dimensional data cube, we define the n-tuple:

$$(ALL_1, ALL_2, \ldots, ALL_n) \in \Re$$

Each functional attribute in the schema defines a set of attribute values on that level in the concept hierarchy. One could then think of a concept hierarchy object as an instance of a hierarchy schema class. Hierarchical information presentation in the form of different levels of granularity helps support aggregation and summarization for the purpose of informed decision-making. Data viewed at the "store" level (*less summarized*) is said to be at a low abstraction level. As one moves up the hierarchy, from "store" to "city," the data can be viewed at a higher abstraction level providing less detail (*more summarized*).

It is important to point out that this process can not possibly be fully automated for the general case due to the fact that the concept of dimension is a user defined perspective that relies heavily on the central theme being considered. For example, the typical dimension shown above, LOCATION, can be further described

by another concept hierarchy such as *continent*, *country*, *region*, *district* without changing any of the actual data in the DBMS. The raw data may describe locations using only store ID numbers. In such cases, a human expert is needed to define which "districts" falls within which "region," the regions making up a "country" and so on. In the next section we will look at how concept hierarchies can be used through specific operations so that the data warehouse can be analyzed.

3.2 OLAP: Online Analytic Processing

Typical relational database systems are well optimized for query processing and Online Transaction Processing (OLTP), which minimizes the time needed for systematic daily operations of an organization. These operations consist of a well-structured and repetitive set of atomic transactions that occur in short bursts. What is needed in the case of data warehouses is a set of operations for the use of knowledge workers (upper management and analysts) so that historical data can be quickly presented in various views and degrees of abstraction. These operations are called Online Analytic Processing (**OLAP**).

Once a multidimensional data model has been constructed, the user is able to view and manipulate this consolidated information by applying the appropriate sequence of different OLAP operations. OLAP operations allow the interactive exploration of a data warehouse and the manipulation of the level of abstraction of the presented information. These operations can be functionally classified as data selection operations and data aggregation operations. Given a data cube, a *selection* operation is typically performed to obtain a subset of the data. OLAP data selection operations are closely relate to the SQL operator `SELECT`. On the other hand, the OLAP *aggregation* operations apply mathematical functions on a given set of data to yield a new set of data that , typically, differs in granularity from the input set. Before discussing these operations we first look at the different types of aggregation functions that can be applied.

3.2.1 Types of Aggregate Functions

There are three general classes of aggregate functions any of which could be used to obtain different views of the data warehouse [81]. *Distributive* functions are the

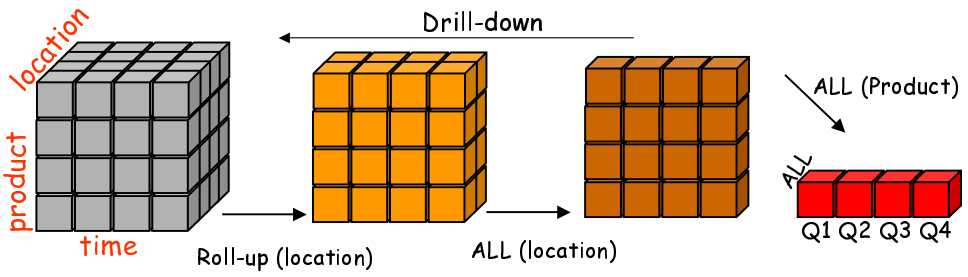


Figure 3.3: Illustrating the ROLL-UP and DRILL-DOWN operations using the data cube in Figure 3.1.

simplest and are based on partitioning the input into disjoint data sets that can be separately aggregated and combined at a later stage. Such functions include `COUNT()`, `MIN()`, `MAX()`, and `SUM()`. *Algebraic* functions on the other hand rely on the characteristics of the entire data set to compute their values, for example, `AVERAGE()`, and `STDDEV()` are such functions. Finally there is the *holistic* aggregate function class which includes `MEDIAN()`, `RANK()`, `MODE()`, etc. Such functions posses the property that there is no constant bound on the size of the storage needed to describe a sub aggregate. That is, holistic aggregate functions cannot be computed in several partitions and then combined for a comprehensive value.

3.2.2 OLAP: Data Summary Operation

Mining a hierarchical multidimensional wealth of specific data requires the implementation of a set of operations that allows the user to traverse this hierarchy. This means that there has to exist a set of OLAP operation that modifies a given data view to be more general or more detailed. Such operations rely heavily on the concept hierarchies described in the previous section. The **ROLL-UP** operation performs aggregation on the specified dimension of the data cube, effectively moving the view to a higher level of abstraction. The opposite operation is called **DRILL-DOWN**.

To illustrate, consider the example in Figure 3.3 which is based on the example introduced at the beginning of this chapter. If we are currently viewing the three dimensions (LOCATION, TIME, PRODUCT) on the level "city," "quarter," and "item" respectively, we can ROLL-UP to the higher level of abstrac-

tion on LOCATION[2] to produce a more compact set of data for an easier view. The figure shows, from left to right, two applications of ROLL-UP on LOCATION that produced a two dimensional slice in the third array. This indicates that the dimension LOCATION, at that point, is being viewed at the level ALL. Progressing on from the third array, applying a ROLL-UP to ALL products forms the last (rightmost) four-cell array. This final array consists of the four *numbers* $(ALL, ALL, Q_1), (ALL, ALL, Q_2), (ALL, ALL, Q_3), (ALL, ALL, Q_4)$. When ROLL-UP is continuously applied to all dimension, the final view will consist of a single numeric value (see 3.1.1 above). Next we present some of the OLAP operations that are provided for the purpose of selecting subsets of the data cube.

3.2.3 OLAP: Data Selection Operations

In a particular OLAP view it is often the case that the user wishes to focus the analysis on a particular subset of the data without changing its granularity. This functionality is provided through the OLAP data selection operations. Such operations include **SLICE**, **DICE**, and **PIVOT**. While the first two operations allow the user to provide constraints on the domains of the data dimensions viewed, PIVOT simply swaps the bindings[3] of the data dimensions.

Slicing the data cube involves the selection of a specific value along one of its dimension. Using the example in Figure 3.1, providing the constraint (LOCATION = "Calgary") produces a slice in the 3D data cube. This slice is the 2D table that appears behind the first table in a darker colour. To be able provide constraints on the domains of more than one dimension, the user can DICE the data cube. This dice is a mere subset of the data presented on the full domains. The PIVOT operations, also called rotate, is especially useful for visualization purposes. Given the data cube discussed thus far, (LOCATION, TIME, PRODUCT), a pivot operation allows the user to view the cube (TIME, LOCATION, PRODUCT) instead. With DIVE-ON, in the IVR the user can perform the PIVOT operation to visually inspect different "layouts" of the virtual objects.

DIVE-ON does support DRILL-DOWN, ROLL-UP, SLICE, DICE and PIVOT

[2]By executing the command ROLL-UP(location)

[3]In visualization three data dimension are bound to the three coordinates X, Y, and Z. PIVOT allows a dynamic rebinding.

operations within the created virtual world. The user is capable of inputting the parameters needed for each operation via a set of task-specific interaction techniques that are managed by the UIM. How these are managed and controlled by the user is the topic of Chapter 5.

3.3 The MOLAP and ROLAP Models

In the previous section we have seen how OLAP operations are defined on the data cube; in this section we will compare the main two methods for computing the data cube. To implement the concept of multidimensionality data dimensions can be assembled in the form of related tables of in the form of multidimensional arrays (Figure 3.1). In a warehouse, the next step after the definition of the data dimensions and the associated concept hierarchies is to compute the CUBE-BY operator introduced by Gray et al. in 1996. This operator computes group-by aggregations over all possible subsets of the data dimensions. That is, the power set of the set of dimensions has to computed. Whether this computation is deferred, partially materialized or fully materialized is large area of research and the reader is referred to the work of Harinarayan et al. [43]. For example, for an N-dimensional data cube, the total number of GROUP-BYs that must be calculated are:

$$T = \prod_{i=1}^{n} (L_i + 1)$$

where L_i is the number of levels associated with the concept hierarchy of dimension i - 1 (ALL is not included). For our work on the VCU side, the problem is a much simpler but parallel in essence since only a small number of data dimensions are visualized at any given time. The available algorithms belong to two classes, relational OLAP algorithms (ROLAP) and multidimensional OLAP algorithms (MOLAP). Some examples of the ROLAP implementations are provided by MetaCube from Informix [52] and MicroStrategy [51]. MOLAP commercial solutions include Essbase by Hyperion [80], Express from Oracle [22]. The example in Figure 3.1 illustrates the main idea of MOLAP and ROLAP side by side. While ROLAP is based on relational tables, MOLAP is based on the use of contiguous multidimensional arrays.

ROLAP algorithms that compute the data cube are called **value based**. By definition ROLAP algorithms use relational tables as their data structures. This means that a "cell" in a logical multidimensional space is represented in the system as a tuple, with some attributes that identify the location of the tuple in the multidimensional space. In addition, there should also be an attribute that contains the actual data value of that cell [94].

Using our example in Figure 3.1, a cell in a ROLAP model that represents 40k revenue for LOCATION = "Edmonton", PRODUCT = "Compact" in the TIME = "Q_1 would be presented by the tuple (Edmonton, Compact, Q_1, 40K). By contrast, MOLAP systems use the position in which a data item exists to infer the associated attributes. For that reason they are called **position-based** structure. Using MOLAP to store the above mentioned sale would require storing only a single value, namely 40K. As we will see in the Chapter 6, our VOLAP structure is also a position-based structure.

3.4 DIVE-ON: The Virtual Warehouse

The Virtual Data Warehouse (VDW) is a conceptualization of a centralized data warehouse that includes a set of distributed data sources and a shell (DCC-Shell) that is responsible for managing and querying these sources (Figure 3.4). The DCC-shell does not store any actual transaction (raw data). Instead, these transactions are left on their original sites while the DCC builds and updates a global multidimensional model of the available dimensions and measures, hence the name "virtual warehouse". This approach provides the VDW clients a constantly updated and global view of an N-dimensional data cube in a manner that makes the source distribution transparent. Although the DCC-Shell does not store raw data, it maintains a pool of meta-data (cube global schema) that is synchronized with all data sources. These meta-data are prepared when constructing the VDW, and when a part is modified, all sites must be updated to avoid inconsistencies. Besides meta-data, DCC-Shell also maintains a resource allocation table that includes information pertaining to the location, data organization, and the communication method of each data source. All the meta-data and resource allocation data are in XML format, so it is easy to understand and maintain, therefore making the whole system extensible

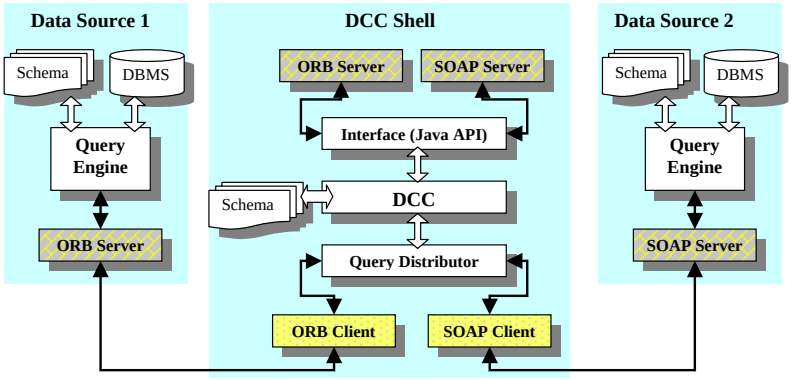


Figure 3.4: The Virtual Data Warehouse (VDW) architecture.

and flexible.

Similar to a traditional warehouse, the virtual warehouse provides querying capabilities to a requesting client. There are three classes of query functions that are available to a client. First is the *Warehouse query*, which is designed to provide the client with basic VDW structure including the number of data cubes, and the measures and main theme of each cube. Many physically or conceptually different data cubes could reside in the virtual data warehouse. The *Cube schema query* provides the meta-data of one specific data cube in the VDW. This meta-data includes a depiction of all available dimensions, measures, and the concept hierarchies that further describe each dimension. Finally, *Cube data query* is used to obtain an entire N-dimensional cube or any subset of it. This is particularly useful in applications such as the VCU, which handles only a subset of an N-dimensional data cube along with some of the measures available (only two measures currently visualized) [93]). All query requests and responds are in XML format, analogous to the way meta-data is stored within the DCC-shell. In the next section, the relay of XML messages within the system is presented which is then followed by schema formulation and the XMDQL language.

3.4.1 Schema Formulation

Since the VDW maintains a global and an up-to-date schema that combines all the distribute resources, it is important to choose a representation that facilitates reads, updates, and modifies. For this reason, the schema is represented using XML.

Constructing an XML schema for the warehouse general structure is rather simple, consequently we illustrate our method by discussing the data cube schema that is used to fulfill the *cube schema query*.

The *CubeSchema* is defined mainly by a set of measures and dimensions. Within an XML document of the VDW, the root element CubeSchema has sub-elements of Measures and Dimensions, which consist of Measure elements and Dimension elements. Each measure element has a *name* attribute to identify itself, and an *aggregationFunction* attribute with value "SUM" or "COUNT" to express how to aggregate the measure data. Measure also has several sub-elements such as *Title*, *DataType* and *Description*. Here is an example of Measure elements:

```
<Measures>
  <Measure name="Unit_Sales" aggregationFunction="SUM">
   <Title>Unit Sales</Title>
   <DataType>double</DataType>
  </Measure>
  <Measure name="Store_Cost" aggregationFunction="SUM">
   <Title>Store Cost</Title>
   <DataType>double</DataType>
  </Measure>
  <Measure name="Store_Sales" aggregationFunction="SUM">
   <Title>Store Sales</Title>
   <DataType>double</DataType>
  </Measure>
  <Measure name="Sales_Count" aggregationFunction="Count">
   <Title>Store Cost</Title>
   <DataType>double</DataType>
  </Measure>
</Measures>
```

The concept hierarchy defined on a dimension is expressed using levels of the ontology used. It is straightforward to write down dimension information in XML format given that the attributes forming the concept hierarchy are related by a

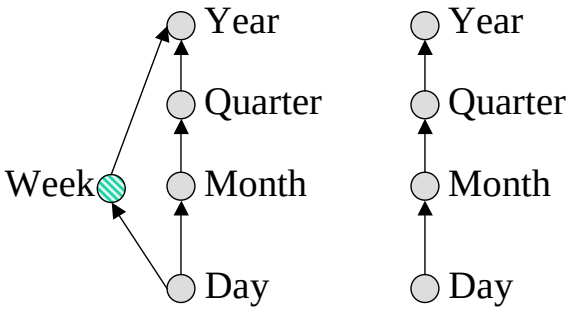


Figure 3.5: A lattice (partial order) and a tree (total order) concept hierarchy for the dimension *time*.

total order relation (Figure 3.5). In some instances however, the attributes of a dimension may be organized in a partial order forming a lattice [41]. For example, attributes of the dimension "time" could be organized as a partial relation such as: "$day < \{month < quarter; week\} < year$". In such a relation, a ROLL-UP operation from the level "day" is considered ambiguous since there are two ways to ascend the concept hierarchy, namely "week" or "month". It is therefore not possible to represent the concept hierarchy as a tree since partial order produces a cycle. Removing the attribute "week" produces the totally ordered concept hierarchy (Figure 3.5).

To solve the problem of partially ordered concept hierarchies, the attributes that cause partial ordering are considered as *properties* of the dimensions in the XML representation. For example, as in Figure 3.5 putting "week" into "day" as a property transforms the relation from partial to total order. So the Time dimension has five levels: {Year, Quarter, Month, Day}. Day level has a property "Day-of-Week". The XML fragment could be then written as follows:

```
<Dimension name="Time">
   <Description>Time dimension for Cube</Description>
   <Levels number="4">
    <Level name="Year" Title="Year"/>
    <Level name="Quarter" Title="Quarter"/>
    <Level name="Month" Title="Month"/>
    <Level name="Day" Title="Day" type="base">
     <Property name="Day_of_Week" datatype="String" />
```

```
  </level>
 </Levels>
 <Unit name="1997">
  <Unit name="Q1">
   <Unit name="January">
    <Unit name="1" baseID="1"> <Property>Wednesday</Property></Unit>
    <Unit name="2" baseID="2"> <Property>Thursday</Property></Unit>
    <Unit name="3" baseID="3"> <Property>Friday</Property></Unit>

    . . .

   </Unit>
  </Unit>
</Dimension>
```

3.4.2 XML Multidimensional Query Language (XMDQL)

To be able to interact with the VDW, a form of querying must be in place. We proposed in [5] an XML-based declarative query language, XMDQL, a language that allows the user to express multidimensional queries on the VDW. The concept of a special multidimensional query language was first proposed (still not finalized) by Pilot software[4] as an industry standard. They called their language MDSQL. In OLAP terminology this type of query is equivalent to slicing and dicing the data cube. DIVE-ON defines XMDQL as a query language that is formatted in XML to query a cube data from VDW; it also provides functionality similar to that of MDX (Multidimensional Expressions). The result of executing an XMDQL query could be a cell, a two-dimensional slice, or a multidimensional sub-cube. To specify a cube, XMDQL must contain information about the four basic subjects: (1) The cube being queried, (2) dimensions projected in the result cube, (3) slices in each dimension and (4) the members from a non projected dimension on which data will be filtered for members from projected dimensions. The basic form of the XMDQL

[4]Recently, Pilot software had been purchased by Accrue Software INC.

is as follows:

```
<XMDQL>
  <SELECT>
    Project dimensions and slices
  </SELECT>
  <FROM>
      Which cube to query
  </FROM>
  <WHERE>
    Filtering constraints
  </WHERE>
</XMDQL>
```

For example, given a data cube with four dimensions: Location, Time, Product and Customer, the following XMDQL query would retrieve the sales of office products in the USA for each quarter in 1997. The Product and Time dimensions are projected dimensions, each in one slice, while the Location dimension is expressed in the WHERE clause as a filter.

```
<XMDQL>
  <SELECT>
    <Measure name="Store_Sales">
    <Axis dimension="Product">
      <Slice type="mono" title="${name}">
        <Path>Office.*</Path>
      </Slice>
    </Axis>
    <Axis dimension="Time">
      <Slice type="mono" title="Quarter ${name}">
        <Path>1997.*</Path>
      </Slice>
    </Axis>
  </SELECT>
```

```
  <FROM>
    <Cube name="AllElectronics"/>
  </FROM>
  <WHERE>
    <Condition dimension="Location">
      <Path>N_America.USA</Path>
    </Condition>
  </WHERE>
</XMDQL>
```

The *Axis* element represents projected dimension, and the *Slice* element represents each piece of a slice cut from this dimension. The *Path* element indicates the conceptual level from the concept hierarchy to retrieve the data from, as well as the constituents of the data to retrieve. For instance, the path="Office.*" means showing all children of *Office* such as *computer, fax, copier, etc.*

3.4.3 Query Distribution and Execution

A VCU query is first received by the DCC-Shell interface (through ORB/SOAP Server) and forwarded to the Query Distributor through the DCC. The Query Distributor analyzes the query and, without translating the XMDQL query, broadcasts it to all the relevant data sources based on the resource allocation table that contains information pertaining to the sources content and their communication protocols. Each data source then translates the incoming query into an appropriate form. A specific wrapper transforms the XMDQL query from the global schema to a set of queries in the local schema after pruning attributes that are not applicable to the local data source. Each data source executes the query (or set of queries) and forms an XML document that contains the needed cube data. This information is relayed back to the VDW via the server software installed on the distributed sites. All the returned information is then merged by the VDW to form an N-dimensional data cube. For our visualization purposes, the DCC then extracts the VCU-requested data cube and sends it back to be rendered in the CAVE. The distribution information could be represented in XML as follows:

```
<Distribution dimension="Store">
```

```
 <Component path="N_America.USA"
            mart="DataMart1">USA sales data</Component>
 <Component path="N_America.Canada"
            mart="DataMart1">Canada sales data</Component>
</Distribution>
```

Chapter 4

Synthesizing and Managing Virtual Worlds

This chapter presents the methods adopted for transforming the multidimensional data into virtual objects in immersive VR; the IVR environment chosen is the CAVE. The reasons that make the CAVE environment especially useful for this type of application are also presented. Occlusion problems are to be expected and we have implemented some measures that reduces the number of occlude objects from a viewpoint. We first begin by describing some data visualization avenues that have been explored since the 1970s.

Iconographic data visualization is a technique that employs *icons* or *glyphs*, whose visual attributes are bound to the data being examined. Many researchers have experimented with the idea of using highly detailed icons or glyphs to represent a direct mapping between numerical and visual measure [73, 31]. Such applications place great emphasis on the quantity of a measure and how it can be represented as accurately as possible. Other types of visualization techniques aim to produce a collective effect such as gradients or islands of contrasting textures, which correspond to structures in the data [54, 86]. In contrast, DIVE-ON creates a visualization environment on a conceptual level. The objects of the virtual world are not designed to tell the user that the total sale of a branch was an X amount of dollars for example; it is designed to convey the significance of this amount. It should be noted here that the latter information is preserved and the user can inquire about the exact data lineage by pointing at the object (See Chapter 5).

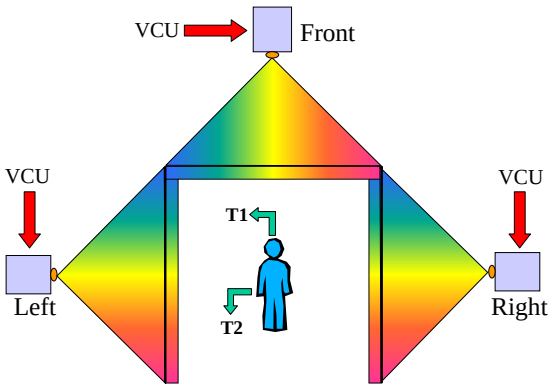


Figure 4.1: A CAVE user within the three back-projected walls. T1, T2: The head and hand-held tracker data stream respectively (Real-time).

4.1 Why Use The CAVE Environment?

While the gathering and building of information can be done from any location, the actual visualization experience takes advantage of the state-of-the-art virtual reality environment in Canada at the University of Alberta. VizRoom is a three-walled CAVE implementation that include two tracking devices (Figure 4.1). As in most CAVE installations, the graphics rendered are in stereo so that depth perception can be simulated. Inside this room, the user is outfitted with one or more tracking devices that provide a real-time stream of data representing their location within the room.

Using this type of environment for visualization over a desktop is justified by two psychophysical experiences; **immersion** and **presence** [8]. Regardless of how realistic the desktop graphics appear, the user is merely "looking at" a computer-determined point of view [83]. But within the walls of the CAVE the user is presented with an *egocentric* frame of reference which effectively immerses the user into VR. In Figure 4.6 (A) a user is presented with an exocentric view of the data space as they must have traveled far into the virtual space. In (B) a user is seen with an egocentric frame of reference where they are able to walk and interact with the data around them.

Standing at the centre of the CAVE, the available field of view is a user-centric 270 degree angle. Users can span this view just as they would in a natural setting, by simply turning their heads [23]. Presence is the sense of "being there" and

is enhanced by simulating the *kinesthetic* feedback gained while walking through a scene [42]. The user's head location and orientation within the three walls are monitored via a light-weight head tracker. The tracking information is used to update the views using the magnitude and direction of the user's most recent motion vector (quaternion). Presence is also enhanced by the use of a hand-held wand that is used to perform OLAP operations, probe the data objects, control the environment, and navigate the IVE.

Immersion and presence, when enhanced by peripheral vision and depth perception, are important factors that help improve situational awareness, context, spatial judgment, and navigation and locomotion [83]. As argued by Pauline Baker [8], these factors make navigation within a 3D model world *practically natural*, and dramatically easier than trying to maneuver around three dimensions using 2D-based graphical user interfaces.

4.1.1 Head Mounted Displays

Since the early years of VR research, great research and development effort were directed to Head Mounted Displays (HMD) to create an immersive virtual environment. HMD consists of a helmet that the user wears which includes a display. This helmet is connected to a computer that update the internal display. Some HMD helmets are equipped with a single display while others provide a display for each eye to enable stereoscopic graphics. The input to a typical HMD system is obtained from what is known in the literature as the digital glove. This glove is equipped with a tracker and several finger sensors that can be used to form gesture commands. The helmet in most older HMD systems were tethered or attached to a ceiling-mounted boom; however, most of the later models developed were geared towards free ranging helmets. CAVE technology was chosen for DIVE-ON over HMD systems for several reasons including:

- The user is free to move naturally without the constraints of the head mounted display (HMD).
- As mentioned above, the DIVE-ON is essentially a decision support tool and most likely will be used by a group or a team of analysts.

(a) (b)

Figure 4.2: A team of immersed users discussing the "dollars sold" data cube. (a) Using cubes. (b) A user pointing the direction of flight within 3D-lit spheres.

- Within the CAVE any view at anytime is instantly available to all for examination and discussion (Figure 4.2.a). Within the walls of the CAVE one is able to naturally communicate with others, like hand use to point out an interesting data item to others.

- It is easier to increase realism in a CAVE environment since both the left view and the right view are already rendered (on the left and right walls). This enables the user access to readily available views in a natural way, by simply turning their head. With the HMD, the head orientation is tracked and used to trigger image rotation in correspondence with the user's head rotation. This process presents an unnecessary set of computations that must be performed with every head rotation.

- From previous experiments, hygienic factors were a big issue for some users. After all, wearing a helmet with an inside display while walking around will definitely make most of us sweat.

Since explorative visualization should be thought of as a *task driven* and not a data driven process, the next section illustrates how our virtual objects are created in light of what we seek to accomplish.

4.2 Spatially Encoding Data as Visual Cues

DIVE-ON creates a visualization environment on a conceptual level and, unlike most iconographic data visualization systems, DIVE-ON is not primarily concerned with quantitative measures. For example, the VR environment is not designed to tell the user that the total sale of a branch was X dollars; rather it is designed to convey the significance of this amount with respect to its context. Once an "interesting" locality has been identified, the user is capable of extracting the original data lineage.

The primary abstraction of DIVE-ON is based on graphical rendering of data cubes. Selected data are extracted from the VDW (Chapter 3) after which relevant attributes are encoded in graphical objects and then rendered in the virtual world. The VCU interprets the three-dimensional cube it receives from the VDW as a three variable function. Each of the three data dimensions is associated with one of the three physical dimensions, namely X, Y, and Z. Since each entry in the data cube is a structure containing two measures M_1 and M_2, the VCU simply plots the two functions M_1(x, y, z) and M_2(x, y, z) in $\Re^3$.

We recognize that there are many alternatives for encoding the data cube measures as graphical objects. Our current prototype uses cube or sphere size and colour to provide the user with visual cues on measure contrasts. For example, if we are focused on the theme "dollars sold" (Figure 4.2), we assume the OLAP user is not primarily interested in the details that in year t the total sale of product p at store s was \$100,000.00. Instead, the VCU provides a context by associating these measures with visual cues that are bound to the virtual objects. In this case, the first cue we use is *size*[1], which is associated with the measure M_1 (dollars sold). After normalization, $M_1(x_t, y_p, z_s)$ is used to render a cube (or a sphere) of appropriate size, centered at position (x_t, y_p, z_s), for some t, p, and s within the data range, as shown in Figure 4.2. A VR user "walking" among these virtual objects becomes almost instantly aware of the relative significance of each value, without the need for specific numeric data.

Our prototype uses an object's *colour* as a second visual cue, by normalizing a measure M_2, and mapping to a discrete 8-colour palette. For example, we can encode

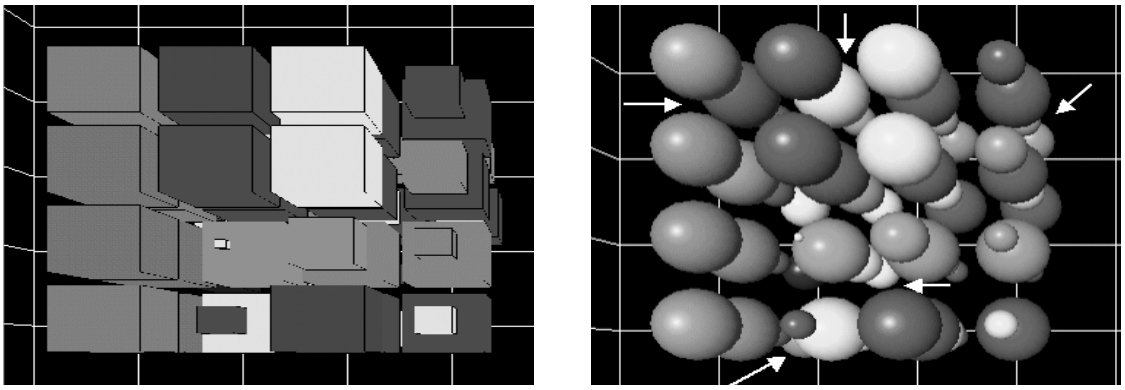
[1] Using the size of a virtual object as a cue depends on the volume not the height of the object.

any abstract data mining "interestingness" measure from "red" to "blue." For example, at the lowest level of aggregation (high granularity), colour can represent the deviation from the mean along one of the dimensions. This is particularly useful for market fluctuation analysis. Similarly, if the user is viewing highly summarized data, colour can be a very effective way to locate anomalies at a lower level. For example, the M_2 value for a month object can represent the maximum M_2 of any of the days it aggregates. In Figure 4.2, each virtual object represents the total revenue for a given year. The colour "red" indicates that one particular month deviates significantly from the rest of the year. We expect the OLAP analyst will *reach* in virtual reality and "select" that object, in order to understand the deviation, and inquire about the exact figures for that year. Similarly, the user may be interested in understanding the stability which dominates a product category (a "blue" object).

Figure 4.2 presents the IVR created by rendering objects that embody the above-described use of visual cues. The three dimensions X, Y, and Z are mapped to the data cube dimensions Product, Supplier and Location respectively. Colour is associated, in this example, with the profit margin while size is associated with gross revenue [93]. The user may request the data attributes to be encoded using cubes or spheres; while rendering cubes is much faster, using spheres reveals more information using the same virtual space. Next we will illustrate the difference between the two models as we discuss the occlusion problem.

4.2.1 Spheres, Cubes and Occlusion

Using spheres makes it easier to distinguish between objects that are within close proximity to one another. To illustrate, Figure 4.3 shows two snapshots of a small-scale version of DIVE-ON running on a PC. Both images represent the same data, from the same viewpoint, and using the same distance between object centres. The arrows in (Figure 4.3.b) point at few of the objects that were previously invisible in (Figure 4.3.a). It is apparent that using spheres "exposes" more data items within the same display area. Spheres are indeed a better presentation for the data for two main reasons. First, to be able to distinguish between a "filled circle" and a sphere the graphics engine must perform shading and light calculations of the surface rendered. The variation of the shadow intensity on the surface of the sphere



(a) (b)

Figure 4.3: Same data attributes from the same viewpoint using cubes and spheres: (a) Aggregates as cubes (b) The arrows point some of the data objects that are occluded in (a).

is a significant visual cue that aids the user in locating where one ends and another begins. Second, a sphere occludes fewer objects simply because they require less volume. Suppose that we are representing an aggregate of a size-attribute r using a cube of length r and a sphere of diameter r. Since the volume of a sphere is approximately $(0.476r^3)$ less than that of a cube, using a sphere instead of a cube clears exactly that amount of volume for the line of sight to penetrate through. We will address the occlusion problem further when dealing with user interaction in (5.3.1).

4.3 The User Interface Manager (UIM)

To support various VR devices, tools, and environments it was necessary to separate the creation and the application of the virtual world. The User Interface Manager (UIM) is the subsystem that is responsible for the reception, filtration, formulation, and channeling of all available input streams. All tracker signals feed into the UIM for preprocessing (Figure 4.4, (TI) Tracker Interface). Constant update of the user's location is necessary to determine the initial location of the floating menu, the active menu number, and the current menu choices. This information is used by the VCU to immerse the user with the graphical objects by providing him/her with the sense of presence. To create such an environment, the body motion data collected by the

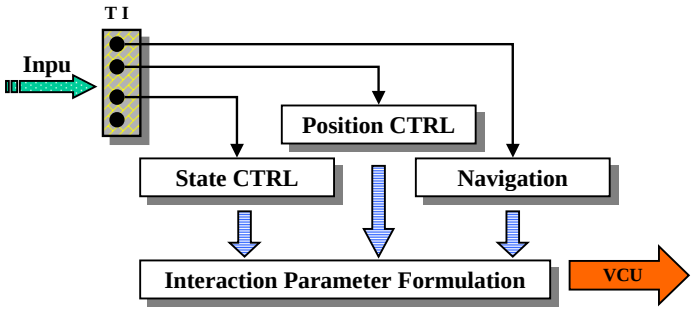


Figure 4.4: User Interface Manager. The Tracker Interface (TI) receives the real-time tracker input stream and channels it according to type. The set of interaction parameters is then fed to the VCU.

UIM (T1 in Figure 4.1) is used to translate the stereo graphics in a manner that simulates the real world. For example, if the user walks forward, the appropriate image translation would be backwards to create the illusion of "walking" through the data. The data stream (T2 in Figure 4.1), emitted from the user's hand, is used to track the position of the 3D menu in the virtual world. This is why they are called "floating menus," because their location is mapped, with six degrees of freedom (6-DOF), to the user's hand [65].

In almost every IVR application the essence of computer-human interaction can be categorized as object manipulation, viewpoint manipulation, or application control [42]. The reason for this taxonomy is the fact that simulating reality is all about simulating the changing views around us along with the ability to interact with the objects that make up these views. As a result, an interface that is highly transparent must use the human sensorimotor system to its advantage by providing the proper visual feedback to the user's motion. Understanding these issues helped us provide an interface that requires little or no instruction; however, it should be clear that the user is required to have an understanding of the data source and OLAP operations. All the interactive capabilities of DIVE-ON have been grouped in adherence with the above categorization. The users chosen in our experiments were people that are familiar with the architecture of the data warehouse being viewed and with the terminology and methodologies of data mining.

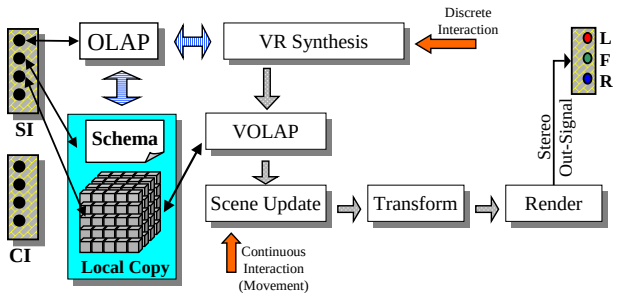
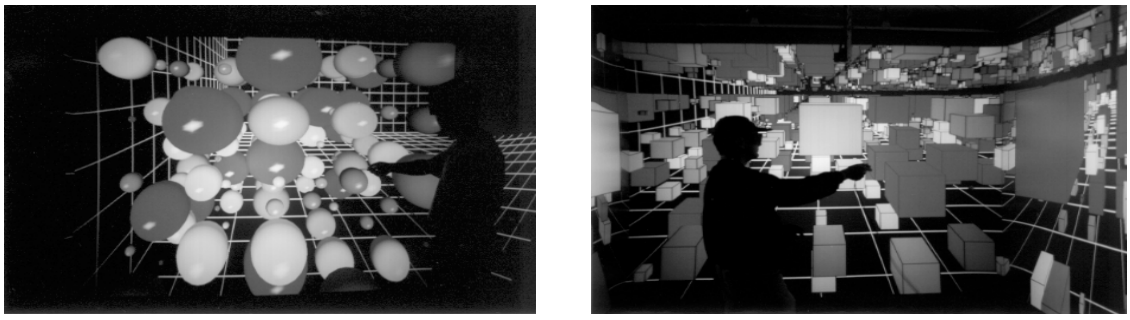


Figure 4.5: The VCU Architecture. SI and CI are the SOAP and CORBA client Interfaces respectively. Output is channeled to Left, Front, and Right projection stereo signals.

4.4 The Visualization Control Unit (VCU)

The VCU is the module responsible for synthesizing and managing the VR environment for data visualization and exploration as it is described in this chapter. As in Figure 1.3, the VCU duties are bound only to the the visualization environment. This means that the specifics of the VDW should be of no concern to the VCU developer and vice versa. To implement this abstract view, each of the VCU and VDW are constructed within a wrapper that provides the only means of relaying messages between the two subsystems . A simple communication protocol (Section 7.3) that defines a set of requests (VCU to DCC) and their corresponding replays (DCC to VCU) is implemented in DIVE-ON. This design effectively hides the implementation details and allows the VCU and the DCC to be independent of one another. After the DCC completes the creation of the N-dimensional data cube it signals the VCU (via the DCC-Shell). Since we are generating a 3D virtual world, only a subset of the available dimensions can be viewed at any given time [93]. The given dimensions and measures that are chosen by the user are extracted from the N-dimensional data cube and a sub-data cube is passed to the VCU for rendering. In light of the above discussion, what level of abstraction does the sub-cube represent? In other words, is the sub-cube highly summarized or highly abstracted? Since the user will be placed within an immersive environment , it is imperative that the delay to the user's actions is kept at a minimal. For this reason, DIVE-ON relies on the VCU to perform the data aggregation required (generating less detailed data). This effectively reduces network dependence to a minimum.



(a) (b)

Figure 4.6: (A) An exocentric frame of reference using spheres. The user has traveled "outside" the data space. (B) An egocentric frame of reference using cubes. The user is pointing in the direction of flight (Chapter 5)

As illustrated in Figure 4.5, the VCU contains a module called "VOLAP." This module is used to provide DIVE-ON with the scalability needed when handling massive amounts of data. As the amount of rendered objects increases, the frame rate of the image transformations decreases (Chapter 6). This effectively reduces the sense of presence since the latency between the user's motion and the image update has increased. To prevent such situations, only portions of the virtual world that are visible to the user are actually rendered. The VCU uses the input from the trackers via interaction parameters fed by through the UIM. The discrete interaction parameters describe the user's command as it pertains to the data being viewed. The continuous interaction parameters describe the user's whereabouts in the environment. More details on interaction is provided in the next chapter.

Chapter 5

IVR Interaction Technics

One of the most fundamental issues in IVR interaction is to build a model that the user can *trust* [85]. In the process of accepting the virtual world as a valid source of visual stimulants, the user would naturally expect it to exhibit similar characteristics to the real world. This trust can be "implemented" in an IVR system by providing a reliable and consistent frame of reference and visual cues. If the system does provide functionality control through some means, then it must adhere to the general principals that governed the design of the VR scene. For example, to gain from immersing the user in a CAVE, the system must be able to provide an egocentric[1] frame of reference. It has been shown that once the user "believes" in this metaphor, all subsequent reactions to the users actions must also be presented using the same frame of reference [8, 42]. As a result, by simply migrating the successful GUI widgets from the desktop environment to the walls of the CAVE would be distracting to the user as they are instantly transformed from *being* in the world to *looking at* the world.

5.1 Orienteering

Orienteering is one of the most important design considerations in any VR system. If you were suddenly transported into a foreign environment where everything around you were fundamentally very similar how would you begin exploring this new world? Before taking a single step, you must first find some structural clues as to how this

[1] The user is the centre of the display as opposed to *exocentric* frame of reference where the user is "looking at" the display.

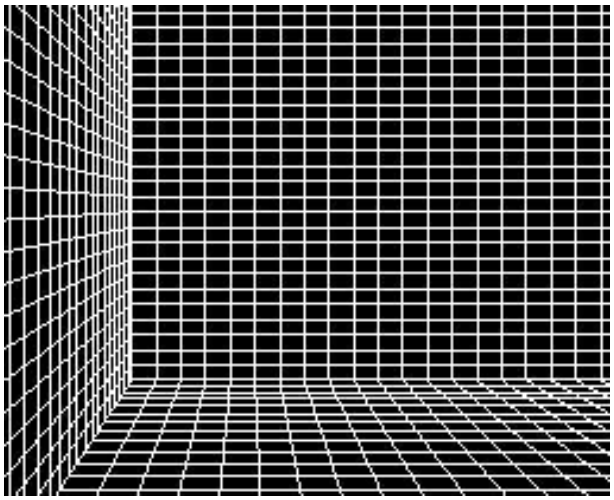


Figure 5.1: Orienteering the user in the CAVE (U of A holodeck). The reference grid is a constant reminder to the whereabouts of the three coordinates.

environment is assembled. This is done by examining the way objects are arranged within the environment and how they relate to one another [72]. The next natural process involves the identification of some reference points that can be used in maintaining a concept of origin.

It is fair to assume that the user of the system is familiar with the data sources being visualized. What is needed then is a means by which the layout of the 3D world is mapped to the conceptual correlation in the mind of the user. The system will clearly identify the three coordinates and their bindings to the data dimensions being viewed. As the users walk within the environment, they may need to know where they are located with respect to the domain of the three dimensions. For that purpose, a pointer device is used to clearly indicate the (x, y, z) coordinates, in terms of the three active dimensions in the VR scene (Section 5.6). In addition, the user needs a frame of reference for perspective and to determine the size of objects in the world [63]. Our solution in this regard involves drawing a *reference grid* as illustrated in Figure 5.1. This grid also serves as a reference for drawing the objects whose maximum volume is determined by the dimension of the grid spacing. Furthermore, during navigation the user is provided with a pointer that can be used to obtain the current coordinate and data dimension bindings (Figure 5.8).

5.2 Continuous Interaction

The part "continuous" of the term refers to the tracker's continuous input stream as it reports to the UIM its updated location. In this class of interaction techniques the user plays a *passive* role in the interaction process. In other words, continuous interaction is performed in a manner that is completely transparent to the user. It is this notion that makes the VCU capable of providing the kinesthetic feedback needed for immersive VR.

The user provides this transparent input by wearing an electromagnetic tracker that creates a data structure containing its exact location with respect to a preset origin within the 2.75 x 2.75 x 2.75 metre volume (9.5 foot). This structure is received by the UIM, which calculates the corresponding relative (x, y, z) coordinates of the device, forms the interaction parameter and forwards it to the VCU. By attaching this device to the user's head, the VCU induces the user's head location and orientation. This process is important since creating realistic virtual reality involves the calculation of all perspectives from the point of view of the user. As a result, the user's motion within the room is mapped to an appropriate image transformation on all three walls thus creating a realistic immersion in a virtual environment.

Another electromagnetic tracker is used to maintain information pertaining to the user's hand location. This hand-held tracker is a tool that enables the user to point, query and manipulate the data objects in the environment. The location of the user's hand is mapped to the location of a 3D pointer to create a ***hand-coupled*** virtual pointer. This pointer in conjunction with a time-cue[2] creates an easy to use and understand input technique. As will be seen shortly, the hand-held tracker also provides an input command centre through the incorporation of three external buttons that are a part of the tracker's hardware.

5.3 Discrete Interaction

Discrete interaction refers to the human-computer interaction that is deliberately initiated by the user to fulfill an exact need, request or action. This class includes

[2]Using time as an input cue is a well-practiced technique on desktop GUIs. If the mouse is left on an icon for longer than time t an action takes place.

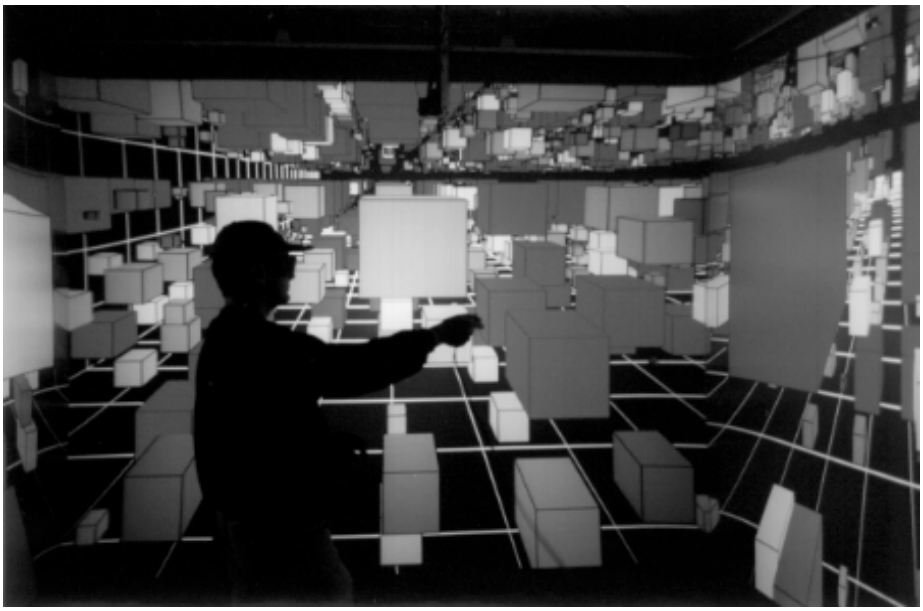


Figure 5.2: A user flying along the X coordinate. The flight speed is increased simply by stretching the arm away from the body in the desired direction.

all navigational functions, object query, and menu commands. Unlike continuous interaction that is transparent to the user and controls a basic IVR functionality, discrete interaction can be initiated and terminated anytime at the user's request.

The hand-held tracker is equipped with three buttons that the user can use to access the systems main functions. The UIM is a ***state machine***, this means that it's interpretation of a "button-pressed" action depends on the current state of discrete interaction. For instance, if the first button is pressed while the menu is active then the UIM would interpret that as a "scroll" through the options command. But if the menu has not been activated yet, then the first button would activate the menu system. Similarly, the second button could mean an "execute" command or currently selected "object query" command. The menu interaction is provided by a special type of 3D menus that are integrated with the virtual world. Details will be provided in the following sections.

5.3.1 Floating Hand-Coupled Menus

Displaying a menu, or a panel, in virtual reality will occlude parts of the virtual scene. To alleviate this problem, researchers have proposed various solutions that

Figure 5.3: The hand-coupled menus with clutching and 6-DOF allow the user to place the menu at any arbitrary point in the IVR.

mainly involve texture mapping and translucency. While the use of texture-mapped text produces a new set of problems (Section 5.5), displaying text on a translucent panel (menu) makes it hard too read against certain colours in the environment. Resulting from these observations, we have designed the concept of *hand-coupled floating menus* that will guarantee the legibility of text as well as the reduction[3] of the occlusion problem. In this section we will first introduce such menus then we will introduce a new enhancement that we called clutching.

The hand-held tracker is equipped with three buttons that can be used to pop a menu, scroll through it and execute an option. This process is very similar to the 2D CRT-based working environment. Early experiments with world-fixed and display-fixed windows [30] have proven to be ineffective since the former tends to obstruct the view of data items behind it and the latter is hard to locate due to the size of virtual world. As a result, the location of the menu is chosen to be that of the hand-held tracker and moves in correspondence to its motion creating the effect of a floating menu in IVR. Such a menu can be popped at any time and "tossed" aside while the user continues to examine the data available before making a choice. The menu system enables the user to execute all OLAP, environment, remote and local functions by scrolling and locating the appropriate action (Figure 5.3). We are also experimenting with a new menu system on a rotating torus. The torus minimizes

[3]A Hand-coupled menu does not eliminate the occlusion problem. However, considering that the area of focus of our vision is significantly smaller than the field of view available in the CAVE, occlusion is practically eliminated.

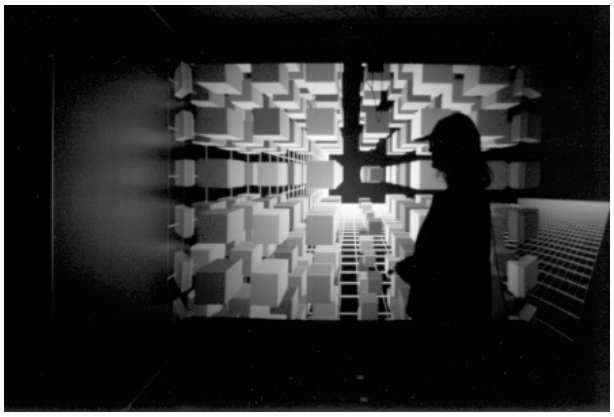


Figure 5.4: Global view of the cube space.

the occlusion by the menu while maximizing the options and the usability of the menu. It certainly reduces the number of selections in order to trigger a desired operation.

As mentioned above, the position of the hand held tracker is mapped to the virtual world by rendering a *virtual pointer* that point at different virtual objects as the user's hand moves in the real world. The user can perform the two OLAP operations *slice* and *dice* by using this virtual pointer to identify the parameters needed for the operations. To obtain a slice of the data, the user would restrict the current range along one or more axis within a specific interval. For example, if the user is only interested in data that pertains to 1997 and 1998 they would point at any data item of 1997, initiate the slice operation, navigate to any 1999 data item and finally point at it and terminate the slice operation. The dice operation is analogous and requires the users to specify restrictions along the X, Y, and the Z axis.

5.3.2 Clutching applied in 3D

The idea of clutching is very simple, and correlates to what almost everyone does without thinking the minute they set in front of a desktop. To illustrate, think of the situation in Figure 5.6. The objective is to click on the link given the current mouse and cursor positions. Automatically, you would lift the mouse, move your hand, and put the mouse back down to continue sliding it upwards. This is clutching in 2D. To apply this in the IVR so that the user can clutch-on or clutch-off the menu

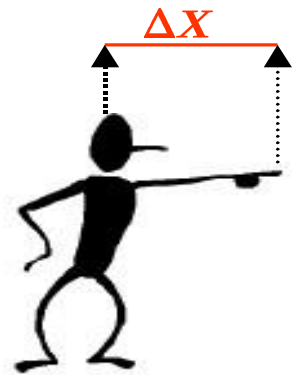


Figure 5.5: The horizontal distance between the user's head and hand is used for clutching and navigation control.

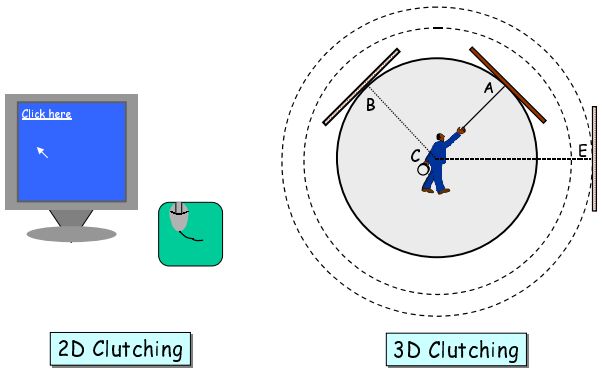


Figure 5.6: Migrating clutching from 2D to 3D. Hand-coupled menus (right) are always positioned tangent to the sphere centered at the current user location C. To position the menu tangent to sphere with larger radius, clutching can be used.

is very effective and was implemented due to what some users had commented. The problem noticed is that some users felt that having a panel floating around the CAVE with every hand move was simply "annoying." Consider the example in Figure 5.6. If the panel is too close, say at A, and the user wishes to "push it away" to E, then they can clutch-off, move the arm back, clutch-on and move forward again. In the next three sections, we will provide the three categories of discrete interaction that use the clutched and hand-coupled menu system.

5.4 System-Based Interaction

The system-based interaction refers to the application control needed to instruct the VDW and the VCU regarding what and how data is viewed. During initialization, the system sends a message to the VDW requesting the set of all available N dimensions. The corresponding XML document is received by the VCU and the user is presented with a list from which they specify the three data dimensions to be visualized (one for each of X, Y and Z). At that point the VDW constructs the corresponding 3D data cube at the lowest level of abstraction and sends it to the VCU along with another XML document that contains the corresponding three concept hierarchies. Our primary concern in the above design is that OLAP operations be performed locally, within the VCU, to minimize the use of a possibly congested network.

All system-based interaction is provided through the use of a 3D floating menu hierarchy (Figure 5.3), which is initiated by pressing the first button on the hand-held tracker. Unlike its 2D ancestor, 3D windowing techniques are anything but standard. Deering's [26] hybrid 2D/3D menu widgets use the current position of the hand-held tracker (wand in their case) to pop a menu in a fixed relative position. Wloka et al. [90] employed a similar idea but instead of pointing at the menu items they use the buttons of the 3D mouse (hand-held tracker in our case) to scroll through the available options. Lindeman et al. [63] report that both methods lack physical support for manipulation and hence provide limited user precision; Instead, they introduced the idea of "hand-held window" which associates a physical object held by the user to a window that appears as an object in VR. Although all the above are plausible solutions they seem to ignore either the importance of kinesthetic feedback or the annoyance of having to carry extra gadgets to experience VR. In the following, we present how these observations and experimentation helped in designing the user interface manager (UIM).

Initially the UIM implemented a menu system similar to that of Deering's with six degrees of freedom (6-DOF) to allow menus to flow freely in VR in total synchronization with the user's hand. The orientation quaternion (Appendix A) stream transmitted from the hand held tracker was used to rotate the menu surface so that

its normal is constantly orthogonal to user's line of sight. As evaluation of the system began, it was noted that most users had problems with menu control. It seemed that the 6-DOF was overwhelming and the extreme sensitivity of the tracker had made the menu jittery and diverted the focus of the study from data mining to "menu controlling."

Another problem appears when the user is descending through the menu tree to execute an OLAP filtering operation that requires the specification of a data attribute. Since the pointer (discussed next) is no longer available while the menu is active, it is not possible to interact with the aggregates to obtain the required attributes; it is either that they have to remember the parameters needed beforehand or restart from the root menu again. To circumvent this we introduce the clutched 3D menu system, which can be described as "grab-and-toss" menu system. After the first button on the hand-held tracker activates the 6-DOF menus, the third acts as a clutch to temporarily remove the association with the user's hand. When the third button is pressed again the association picks up from the previous setting. This provides the user with greater control over the menu treating it as any other object in VR. Clutching also creates the added feature where the user can progressively "push" or "pull" the menu to a desired position. Most important of all however, clutching makes it possible to simultaneously perform system and aggregate-based interaction since the hand-held tracker can disengage menu repositioning at any time without having to traverse from the root menu again. This concept of clutching that is implemented here is the 3D equivalent of the 2D version that is an integral part of mouse use. In 2D, when the mouse reaches the end of the pad the user clutches the mouse by lifting it away from the surface of the pad.

5.5 Aggregate-Based Interaction

Taking advantage of the study performed by Wehrend et al. [88], one learns that at the very top of the general user goals during the visual exploration of scientific data is to *identify* what they are looking at and to *locate* what they are after. Based on this result the aggregate-based interaction has focused primarily on instantly providing the lineage associated with each visual cue for any given object. As discussed earlier, the size and colour of every object is the result of normalized

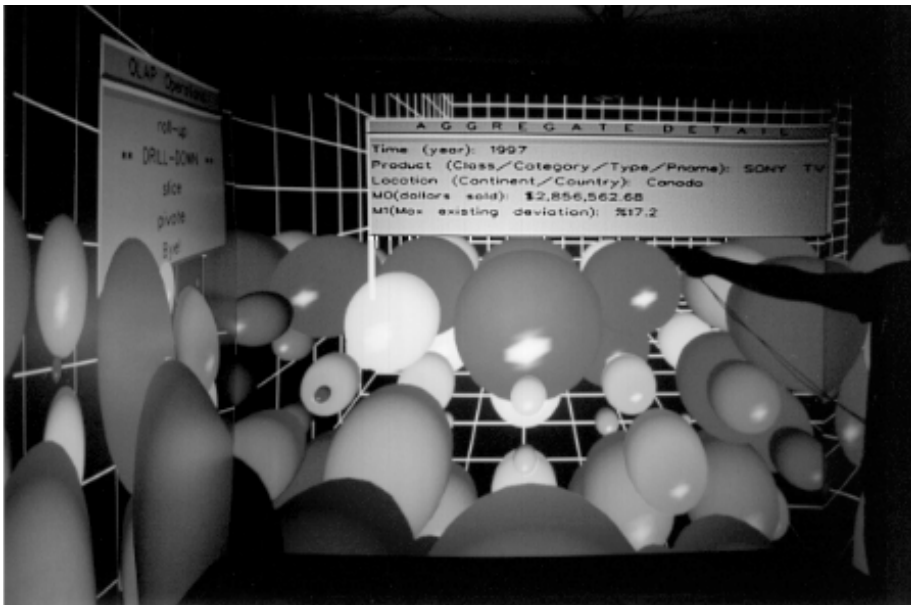


Figure 5.7: Aggregate-Based interaction. Providing text on demand in IVR is a better solution than texture mapping text for all object in the VR environment

measures in the data warehouse. Since normalization is irreversible, it is important that the system maintains the original data used before rendering. The location of the hand-held tracker is obtained and used to draw a pointer in the environment. When the pointer is placed "close" to an object, the second button will activate a small text panel that displays all information that is pertinent to that particular aggregate (Figure fig:aggregate). This information is sufficient to identify all aspects of that particular aggregate, which include the current level of abstraction along each of the three dimensions and the actual value of the first and second measure. The text panel pops as a 3D box that is not occluded by any other object and made to point to the particular object being selected.

As shown from the studies of Colin Ware et al. [87], it is best if the use of text in IVR is kept at a minimum. It is difficult to read text on a plane with norm that is at an angle of more than 45 degrees from the line of sight. As a result, when the user moves around the virtual world the VCU has to continuously monitor the displayed text and to maintain legibility. Performing this for every aggregate in the world is computationally prohibitive and scene cluttering. As a result, in our implementation the VCU provides text only on demand by providing aggregate-based interaction.

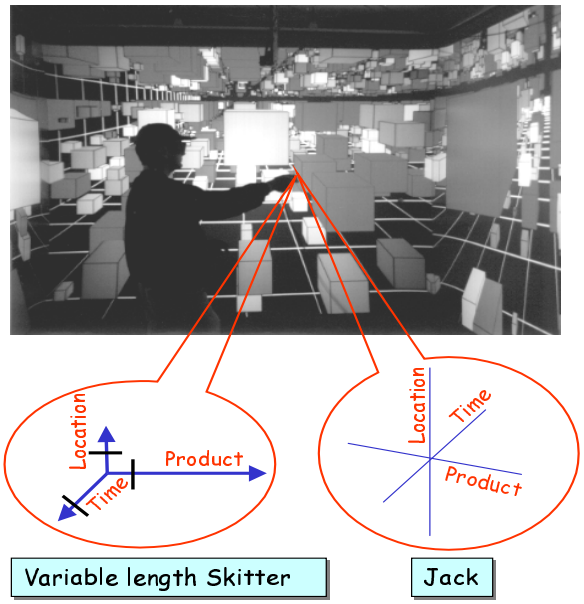


Figure 5.8: The 3D pointer is converted in flight mode to a cross-hair pointer (Skitter or Jack). This will always inform the user to the current bindings between the data dimensions and the coordinates

The user is able to inquire about any aggregate by holding the pointer in the vicinity of that object for a short period of time to activate the text panel associated with it.

5.6 Environment-Based Interaction

This type of interaction is responsible for the *viewpoint manipulation* and aids in building and using a cognitive map of the VR [42, 83]. As seen above, the ability to locate a given aggregate within the environment is the second most sought after operation in VR data exploration. Providing effective navigational means is essential since, depending on the aggregation level, only a fraction of the data may fall within the clipping planes (Figure 5.9). The third button of the pointer is used for navigation and location control (Figure 5.8). Pressing this button provides the user with two movement options, *specified-coordinate* movement and *specified-trajectory* movement. Both motion options simulate motion by manipulating the viewpoint of the user within the IVR.

The *specified-trajectory* movement (Figure 5.8) type is what most frequently re-

ferred to as flying. Using the third button, the user can turn this mode "on" to start flying to the desired destination. The idea is simple, the user points in the direction they wish to fly to and the VCU will translate the virtual world accordingly. The quaternion that specifies the orientation of the hand held tracker (flight direction) is used to obtain the vector needed to prepare the needed transformation matrix (Appendix A). To control the flight speed the hand-head absolute distance is calculated from the current location of both trackers and used as a measure for the speed of image translations (Figure 5.5).

$$\Delta_x = |Head_x - Hand_x|$$

This Δ_x is then used to produce the translation matrix T:

$$T = \begin{pmatrix} 1 & 0 & 0 & \Delta_x \\ 0 & 1 & 0 & Hand_y \\ 0 & 0 & 1 & Hand_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To keep the user oriented in both the ***virtual space*** and the ***data space***, we have modified the original *skitter* introduced by Eric Bier [14]. The length of each of the three orthogonal vectors is proportional to the number of data items along the data dimension it represents (Figure 5.8). As a reminder of the current OLAP view, the name of the concept hierarchy level of each of the three data dimensions is displayed on the skitter according to the (X, Y, Z) bindings. In addition, we have added a marker on each vector indicating the user's current position. As the user flies through the environment, the marker "slides" along the vector accordingly. In other words, the length of the three vectors provides a mental image of the absolute data space while the markers provide the relative position in the virtual space.

Specified-coordinate movement, or transporting, provides the user with navigational means that directly changes their current location within the virtual world. This is accomplished through the menu system where the user is provides with a planner cellular map of any of the three (X, Y), (X, Z) or (Y, Z) planes indicating the cell that includes the user's current location. The pointer can be then used to point at a new destination cell. Once the new cell has been selected, the user is transported to that new location by modifying the head tracker's position in virtual space.

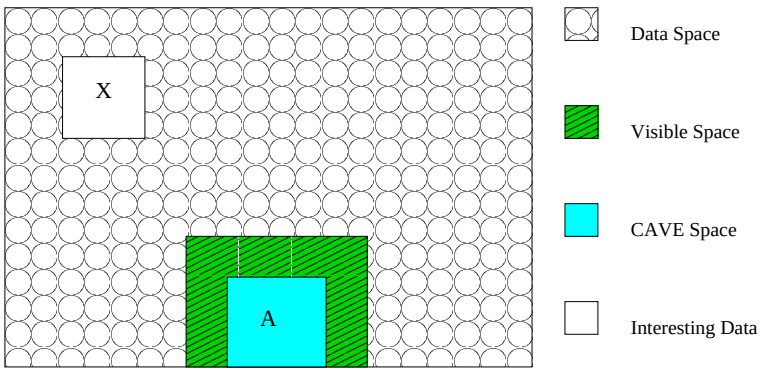


Figure 5.9: A cellular map of two dimensions. From the current user location in region A (defining the current CAVE view), navigation means must be provided to get at "interesting" data in the vicinity X.

Transporting the user within the IVR is needed since a given OLAP view may contain an extremely large number of aggregates, a situation that calls for fast and immediate navigational functionality. For example, it may be the case that the data being visualized reflects the daily transaction for a ten-year period for a corporation with two hundred locations. In a drill-down mode the model may include millions of data items, a situation that makes flying within the IVR slow and completely ineffective, if not useless. In such situations the user may activate a menu that presents a location indicator on a large scale enabling the user to quickly identify their current location and point to the desired destination. This operation will instantly move the user into the new location. In other words, the user is virtually taken out of the cube for a global view, then back in at a different location. Figure 5.4 shows an example of a global view of the data cube. This view can be rotated on any of the three axes to provide the required perspective.

Chapter 6

Introducing VOLAP: A Visually-Oriented OLAP Structures

The motivation and components of the DIVE-ON Visual OLAP (**VOLAP**) structure is presented in this chapter. VOLAP is a new concept that we have first introduced in [93] as a recursive spatial partitioning of virtual space to accommodate the needs for interactive rendering of large[1] virtual environments. The decomposition was done using a hierarchical 3D data structure, octree, which is a well studied structure since the mid 1980s. Rendering only the octants that define a neighbourhood of the user's current location keeps the system's frame rate independent form the number of aggregates available. Using the octree does indeed meet the demands for interactive rendering but does not address the execution of OLAP operations. In the original design, to perform a ROLL-UP or a DRILL-DOWN the system would compute, or read[2], the new data, synthesize the new virtual world and rebuild the corresponding octree.

6.1 Why VOLAP?

To begin, we first illustrate the motivation behind developing this new type of data structure when there is a variety of specialized and general data structures to

[1] Large virtual environment refers to the amount of geometry, or graphical primitives, that make up the world.

[2] To ROLL-UP on a view, an aggregate function has to be computed. To DRILL-DOWN on a view, the original (unaggregated) data has to be acquired.

choose from. There are two issues to consider. First is the need to index a fully materialized three-dimensional data cube on the VCU side (University of Alberta CAVE). Second, to provide a system with non-degrading frame rate this data must also be indexed in terms of spatially correlated chunks. Using a data structure that meets both requirement is essential in building a tool such as DIVE-ON.

Going back to Figure 1.1, the data that comprises an OLAP view is mapped to produce a virtual world which is then rendered as a set of scenes. The recursive partitioning of the octree works well for rendering only segments of the VR related to the current scene. However, when the user performs an OLAP operation a new virtual world must be synthesized, partitioned, and a new scene must be rendered. As a result, we have proposed the concept of VOLAP, which consists of a data structure called **VOLAP-cube** and its associated index, the **VOLAP-tree** [4]. To provide an interactive OLAP operations, a fully materialized 3D data cube is computed and stored in contiguous areas of an array called the VOLAP-cube. The VOLAP-tree is a hybrid structure consisting of two layers. The upper layer indexes the regions of the *data space* in the VOLAP-cube and the lower layer indexes the *virtual space* associated with each region. These concepts are detailed in the following sections.

6.2 Definitions

A presentation of the terms used is given first to describe the data received and the structures built by the VCU. A data cube can be thought of as a set of independent attributes forming what is known as dimensions and a set of dependent attributes that make the measures or the main theme of the data cube [37, 28]. In a K-dimensional data cube, a data dimension D_i where $0 \leq i \leq k$, is a set of attributes that describe all levels of the hierarchy schema (Section 3.1.1) of the dimension. These attributes do not have to be actual data attributes that exist in the database, they could exist only on a conceptual level to designate a particular aggregate. For example, if the data source is a simple flat file that contain only source transactions for a company based on store ID only, then an analyst can define different groups of these ID's to define a city, province and a country (see Figure 6.4).

A **data dimension** D_i is a set of s attributes:

$$D_i = \{d_1, \ldots, d_s\}$$

The concept hierarchy information is received from the virtual warehouse as a set of strings to describe the way data should be aggregated. An actual hierarchy is constructed by the VCU and used for building the OLAP-cube and performing OLAP operations (Figure 6.4 and 6.5).

A **concept hierarchy** Φ_i is a tree with the following properties: -

- The nodes of the tree are precisely those defined by the set D_i defined above.
- The root of the tree, level 0, must be the special attribute "ALL."
- The edges of the tree define the relationship "abstraction-of" when descending the tree and the relationship "generalized as" when ascending the tree.

Level of abstraction is a level of the schema hierarchy. We use Φ_{ij} to refer to the name of a level of abstraction j in the concept hierarchy of dimension D_i. The number of levels in a concept hierarchy corresponds to the depth of the tree and is denoted by $|\Phi|$. We also use $\| \Phi_{ij} \|$ to denote the set of attributes of dimension i at level j.

Using the above notation, at this point it should be clear that for all dimensions D_i where $1 \leq i \leq s$, the following holds:

$$\| \Phi_{i0} \| = \{ALL\}$$

To summarize the above, D_i is a set of attributes; organizing these attributes logically into a tree we obtain Φ_i. Hence, if $|\Phi_i| = d$ then

$$\| \Phi_i \| = \bigcup_{j=0}^{d} \| \Phi_{ij} \| = D_i$$

An **OLAP region** is a set of contiguous data cells that can be defined by specifying only two point, (x_0, y_0, z_0) and the point (x_k, y_k, z_k). These points can be thought of as the lower front left and the upper back right of the minimum bounding bounding box. Using the notation above, an OLAP region that contains the data aggregates of $\Phi_{1i}, \Phi_{2j}, \Phi_{3k}$ is denoted by R_{ijk}.

A **VOLAP-cube** V with two measures can be defined as:

$$V \subseteq D_0 \times D_1 \times D_2 \times \Re^3 \times \Re^3$$

More precisely, an OLAP-cube is the union of all OLAP regions over the three currently viewable data dimensions:

$$\bigcup R_{ijk} = \bigcup_{i=0}^{|\Phi_1|} \bigcup_{j=0}^{|\Phi_2|} \bigcup_{k=0}^{|\Phi_3|} (R_{ijk})$$

This means that a data item in the VOLAP-cube can be defined as 5-tuple (considering that we are dealing only with two measures)

$$(a_1, a_2, a_3, x_1, x_2)$$

Where $a_i \in D_i$ and $x_j \in \Re^3$

A **VOLAP-tree** is a hybrid tree that is used to spatially index the OLAP regions of a VOLAP-cube. Each OLAP region is associated with a VOLAP-tree leaf that points to the second layer of the tree, which contains an octree-based partitioning of that region.

6.3 Hierarchical Representation Of Decomposed Regions

Due to the numerous applications of spatial data structures, there is an abundance of methods for indexing volume data, and solid models [21, 79]. Mainly, the concept of spatial partitioning involves a divide and conquer algorithm. Starting with a region (volume, area, solid, etc.) partition its dimensions according to some scheme and recursively apply the same algorithm to each sub-region. Two of the well known and general spatial decomposition data structures are the KD-tree and the octree. In this section we give a brief introduction of both.

6.3.1 The Octree

Octrees are hierarchical data structure that are well suited for the representation of volume data regardless of the topography of the represented volume (regular hexahedra or irregular surfaces). Since octrees decompose the volume in terms of 8 octants (Figure 6.1), controlling volume traversal can be achieved via simple and

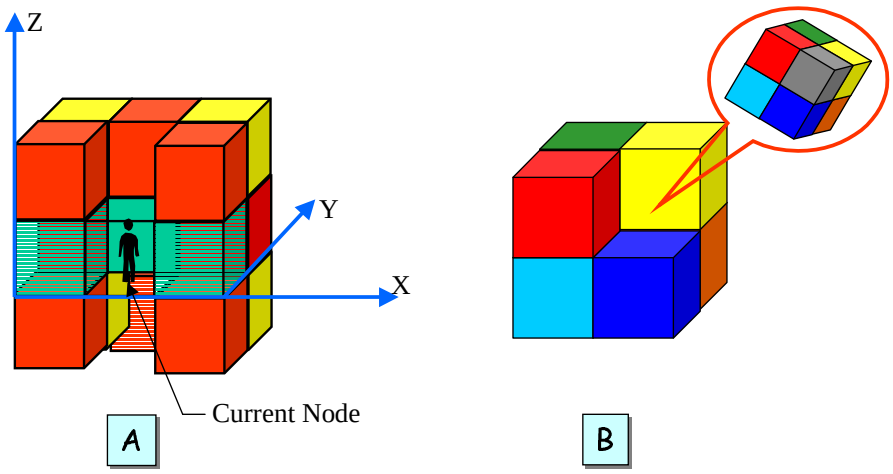


Figure 6.1: Octrees are used to recursively partition volumes. (A) The virtual space is partitioned by an octree, the octants (nodes) shown define the neighbourhood of the current octant (Section 6.4.1). (B) Illustrating the recursive partitioning of the octree where each octant is further subdivided into 8 octants.

inexpensive algorithms [77]. Typically, each coordinate direction is divided into two subvolumes, each of which is then recursively subdivided. For example, using point octrees a volume with resolution $2^k \times 2^k \times 2^k$ can be represented by an octree of depth k [89, 77]. Figure 6.1 (B) illustrate the main principle of recursively subdividing the volume into eight octants.

6.3.2 The KD-Tree

The KD-tree is essentially a binary search tree that can be adapted to index K dimensions (hence KD-tree) instead of just one. It inherits from the binary search tree the speed and the simplicity of implementation making it a good candidate for the spatial indexing of 3D region data. In terms of search efficiency, the ***point*** KD-tree search algorithm time complexity is $O(\log_2 N)$ where N is the number of data points in the tree. Each dimension of the data is represented by a level of the tree. Starting from the root, level 0, level 1 is a binary partition of the x coordinate, level 2 partitions the y coordinate and level 3 the z coordinate. This process recursively continues to work on each region where level 4 partitions x again and so on. Figure 6.7 illustrates the VOLAP-tree which contains an upper portion that indexes the regions in the data regions shown. This example is a 2D illustration, for this reason

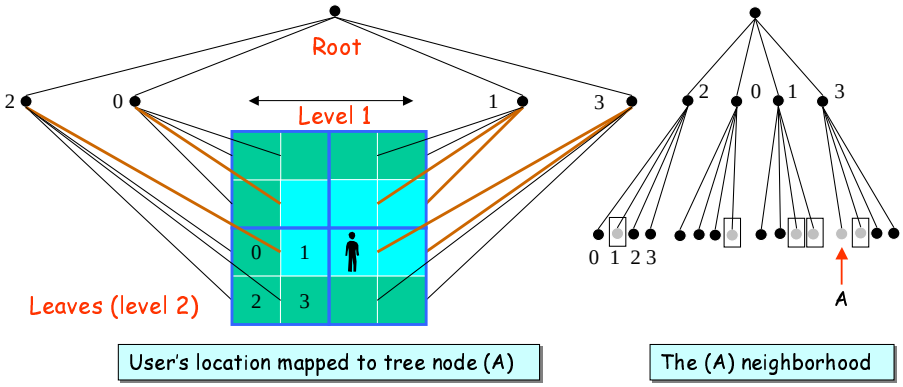


Figure 6.2: The 2D relative of the octree (quadtree) (A) As the user moves in the CAVE, their location is dynamically mapped to tree nodes. (B) Only leaves are used to specify a location.

the partitioning is on X,Y then X again.

6.4 Applying Spatial Decomposition in a CAVE

In this section we will illustrate how a scene is managed and updated using spatial data structures as the user explores the virtual space in our 3-wall implementation of the CAVE (VizRoom). The fundamental idea is to create a mapping between the discretized (X,Y,Z) coordinates of the user's head tracker in the virtual space and the 3D array index containing the data space. Once the location parameters are obtained, we use the attributes of each node to locate the data "nearest" to the user. Nearness here is defined in terms of a neighbourhood of octants.

To illustrate we will use Figure 6.2 which is a 2D relative of the octree called the (***quadtree***). In this figure, one can think of this 2D slice as the "floor" of the CAVE; as the user walks through the data, different neighbouring tiles (region) are selected. Each tile corresponds to a node in the quadtree. The light coloured nodes are the nodes that comprise the neighbourhood from the position where the user is located (leaf node 0 of the third quadrant). The leaf node **A** in the quadtree represents the current user location. We call node **A** the *current node*. Each node is bound by a globally defined capacity that determines the maximum dimensions of each partition. The node capacity is also used to terminate the recursion when the tree is constructed. It should be noted here that the neighbourhood in Figure

Table 6.1: The complete δ neighbourhood of an octree node

Neighbour	Relation
L,R,U, D,B,F	Max. exposure with these 6 face-neighbouring nodes
LD, LU, LB, LF, RD, RU, RB, RF, DB, DF, UB, UF	These are the 12 edge-neighbouring nodes
LDB, LDF, LUB, LUF, RDB, RDF, RUB, RUF	These are the 8 vertex-neighbouring nodes

6.2 is defined as such for the purpose of visualization using three back-projected walls, left, right, and front. Next we will discuss the neighbourhood characteristics pertaining to octrees in general and to our 3-wall CAVE in specific.

6.4.1 Neighbourhood in Octrees

Given a node in an octree, there are 26 nodes that define its neighbourhood. A node, or an octant, in the octree can be thought of as a bounding box that contains a portion of the virtual space (Figure 6.1 (B)). Each node is adjacent to other nodes through its 6 faces, 12 edges and 8 vertices. Table 6.1 contains a listing of these 26 neighbouring nodes using **U** for up, **D** for down, **L** for left, **R** for right, **F** for front, and **B** for back.

In our particular IVR environment only a subset of this neighbourhood is needed. As seen in Figure 6.1 (A), as the user stands within the IVR in a 3-wall CAVE, rendering octants with a **B** adjacency component can be considered a waste of resources. This is the case since views that lie behind the user could not be seen (no back wall). The neighbourhood subset of all visible octants is shown in Figure 6.1 (A) and is specified in Table 6.2.

The VCU instantiates a neighbourhood object at initialization to maintain the currently rendered neighbourhood. A set of pointers are constantly updated to refer to the octree nodes corresponding to the octants of Table 6.2. The class Neighbourhood contains the following data members:

```
class Neighbourhood {
private:
```

Table 6.2: The neighbourhood subset of visible octants in a 3-wall CAVE.

Neighbour	Relation
L,R,F,	Sharing a face
LD, LU, LF, RD, RU, RF, DF, UF	Sharing An Edge
LDF, LUF, RDF, RUF	Sharing A Vertex

```
   OctreeNode *L, *R, *F; // Only 3 face neighbors
   OctreeNode *LU, *LD, *LF, *RU, *RD, *RF, *UF, *DF;
   OctreeNode *LUF, *LDF, *RUF, *RDF; // 4-vertex neighbors
...}
```

Neighbour finding can be achieved through a variety of algorithms. Hanan Samet in 1989 [77] proposed an elegant algorithm for traversing the octree to locate the neighbourhood of a given node. Begin at the current node, ascend the tree until a common ancestor is found. The ancestor in question is a node that allows us to step in the direction of the current neighbour sought. Then retrace by descending back down in search of the neighbouring node. With this method, there is no need for neighbour pointers, just 8 child pointers per node are sufficient.

Since VCU partitions the virtual space with respect to volume, all internal octree nodes are of the same dimension. This dimension is the globally set node tolerance or threshold. As a result, a very simple algorithm can be applied to obtain the neighbourhood of a node P:

Current Node = P
Get absolute dimensions of P
Pick a point in each neighbouring octant
Use OctreeNode* locateNode (int x, int y, int z) to construct
the neighbourhood list.

The octree we use here is called a *region octree* since the leaves do not contain single data point (point octree) but rather a volume of data points. If the node tolerance is K, then the resulting octree will have depth $d = S/2K$, where S is the

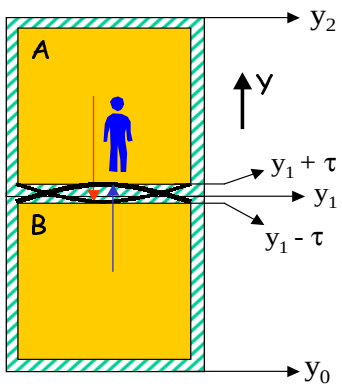


Figure 6.3: Illustrating the characteristics of octree nodes with elastic boundaries. Such boundaries are needed to reduce the jitter when node boundaries are crossed in the IVR.

maximum cardinality of all three dimensions. As a result, starting from the root, the "locateNode" method has complexity $O(d)$, which is indeed acceptable in an interactive environment.

6.4.2 The Boundary Problem

As the discretized VR space is mapped to data cells there will exist a set of invisible boundary lines in the environment that would result in a new neighbourhood search. This boundary problem is a result of the user's continues motion in a direction that exceeds the tolerance of the current node (the data that must be displayed are not in that node). Spatially partitioning VR using the standard octree node structure for real-time interaction will produce a jitter when the boundary lines are crossed as the tree is traversed to obtain the new data.

The VCU provides a solution to this problem by proposing the idea of elastic octree node boundary. As in Figure 6.3 As the user moves from node **A** to node **B** the current node variable "should" be updated from **A** to **B** at point p where $p \leq y_1$. Instead, just as we have used a node tolerance for partitioning the data space, we use node tolerance in the virtual space τ. Violating the boundary of node **A** is tolerated by τ along any of the three coordinates. It is only when the user is more than τ units in **B** that the current node variable is updated. This effectively provides the nodes with an elastic boundary so that there is *no* rigid virtual space "line" along which a transition from one neighbourhood to another must occur. Each node is hence

surrounded by a region in which the value of the current node is undefined[3] without the knowledge of the prior current node. The algorithm that provides this elastic boundary to the current node is rather simple and can be presented as follows:

Current node $= P$
updated user location $L = (X, Y, Z)$
tolerance $= \tau$
  If $|L - boundary(P)| > \tau$
    Current node $= M$ (neighbour of P in the direction of the move)
    Locate the M-neighbourhood.
    Render the M-neighbourhood.
Repeat.

6.5 Building The VOLAP-cube

This section presents the steps taken in collecting a set of OLAP views into one coherent and easy to index array called the VOLAP-cube. Having all OLAP views materialized ahead of time sacrifices storage for interactivity. The VOLAP-cube is a three dimensional array that stores these materialized views according to the level of abstraction described by their associated concept hierarchies. Following the way MOLAP algorithms compute the data cube [94], the 3D array provides position information about its regions. For example, in Figure 6.1 (B) if the octants represent data stored in a contiguous array then only two points are needed to reference any given octant. This is the main observation that lead us to designing the VOLAP-cube. Given three concept hierarchies Φ_1, Φ_2, and Φ_3 then the number of OLAP views on these three data dimensions is given by:

$$Number of OLAP regions = \prod_{i=1}^{3} |\Phi_i|$$

The VCU builds the VOLAP-cube using three pieces of information; the concept hierarchy, an aggregation function, and the data measures corresponding the lowest level of abstraction on all dimensions. In other words, the data measures expected are those that correspond to the GROUP-BY $(\Phi_{1l}, \Phi_{2m}, \Phi_{3n})$ where l, m, n are the

[3] At initialization a current node value is predetermined.

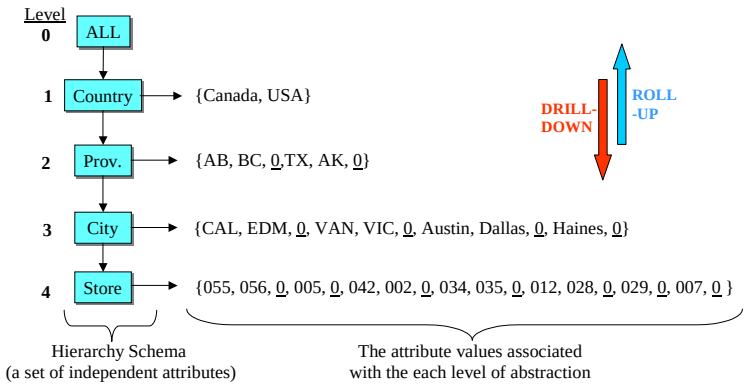


Figure 6.4: The VDW provides the VCU with a schema that is needed for building the actual concept hierarchy for each data dimension. This schema is for a simple LOCATION dimension.

depth of the first, second and third concept hierarchy respectively. To illustrate how the VOLAP-cube is build we will first show how the concept hierarchy is built using the VDW information and then proceed to show how this structure is used to assemble the various OLAP regions into a VOLAP-cube.

Consider the simple example in Figures 6.4, 6.5, and 6.6. The first piece of information needed from the VDW is the hierarchy schema and the attribute values associated with each. In our LOCATION example, the hierarchy schema information received by the VCU is in the form seen in Figure 6.4 which contains five levels of abstraction. Each level of abstraction is accompanied by a pointer to a zero-delimited name set containing the children of each node on the previous level. For example, the set {AB, BC, 0, TX, AK, 0} indicates that level "1" of the concept hierarchy has two countries each of which has two children nodes on the province level, level "2." The character "0" is used as a delimiter between the children of consecutive nodes on the previous level.

Using this information, the VCU proceeds to build the concept hierarchy as defined in Section 6.2. The concept hierarchy tree that corresponds to the schema described above is shown in Figure 6.5. Constructing this structure is a simple pointer-based general purpose tree. Each node contains information pertaining to its name, the schema it belongs to, and a reference number (shown as an integer at the bottom left of each node). For all the available data dimensions, the VCU will build a concept hierarchy such as the one seen here for the dimension LOCATION.

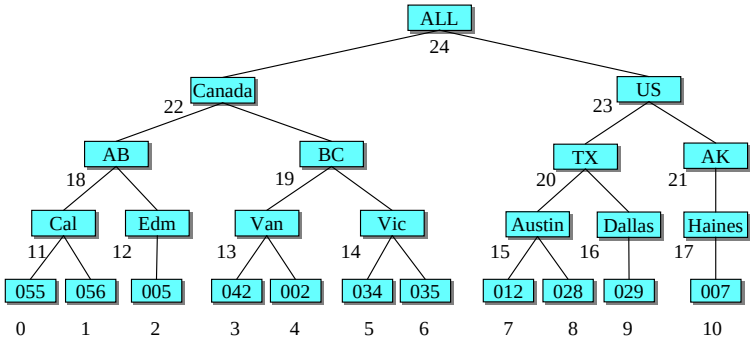


Figure 6.5: Using the schema from the VDW, an actual concept hierarchy for each dimension is created. This concept hierarchy corresponds to the schema of Figure 6.4.

Next we will discuss the use of the reference associated with each node.

6.5.1 Cell Access in The VOLAP-cube

Using the aggregation function as specified by the VDW, the VCU will then begin to materialize all OLAP regions that correspond to the ROLL-UP and DRILL-DOWN operations (Section 3.1.1). The data measures that are transmitted from the VDW are assumed to be at the lowest level of abstraction on all involved dimensions. This makes it possible to generate all aggregates by cyclically rolling up all dimensions until the OLAP region $R_{ALL,ALL,ALL}$, which happens to be a single cell, is reached. To illustrate how these regions are assembled, we consider the simple case where we have only two data dimensions, the LOCATION as described in the concept hierarchy of Figure 6.5 and the dimension PRODUCT. The resultant two dimensional VOLAP-cube is shown in Figure 6.6.

The 2D array is filled beginning at location (0,0) to include the measure associated with $M_1(055, item0)$ where item0 is leftmost leaf in the concept hierarchy of PRODUCT. Since the store IDs {005, 056, ..., 007} are at the lowest level, the associated nodes on the LOCATION concept hierarchy (leaves) are provided with the reference sequence {0,1, ..., 10} (see Figure 6.5). This is to reference their first index in the VOLAP-cube. This process continues up the LOCATION tree until the OLAP region $R_{ALL,item}$ which is a the column vector $(24, 0)$ to $(24, k)$ where $0 \leq k \leq |\Phi_{location,item}|$. The next iteration would then proceed with the next level of the product hierarchy, the OLAP region $R_{Store,Brand}$.

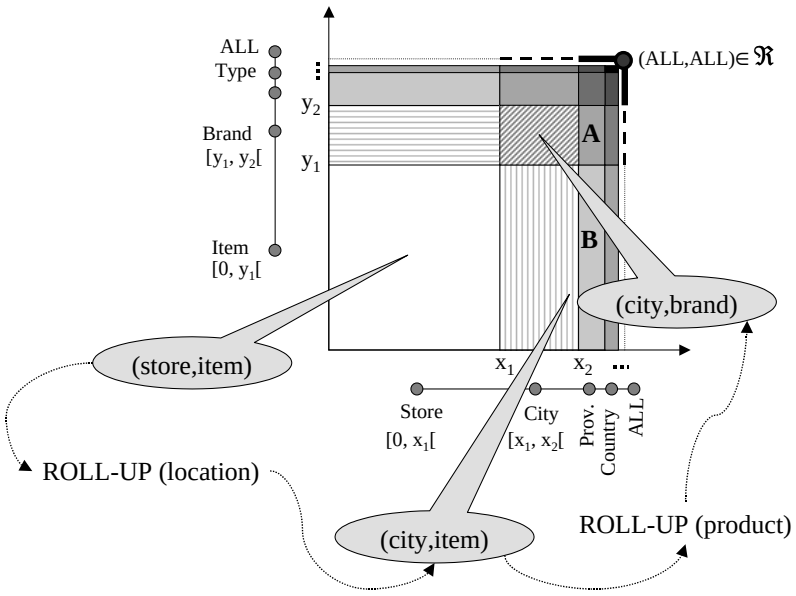


Figure 6.6: Performing OLAP aggregation operations using the VOLAP-cube is as simple as specifying the two points that contain the desired OLAP region.

Having built the VOLAP-cube as described above, lets illustrate how the data that comprises the view (City, Brand) could be located. Consider the dimension LOCATION of Figure 6.5, descend the tree from the root always following the leftmost child. Once at the level desired, City, the two references 11 of "Cal" and 18 of its parent specify the interval [11,18[. This is precisely the VOLAP-cube first index range of the region we are looking for. Similarly, the second index range of the PRODUCT dimension is obtained and we have effectively located the two points needed for specifying the OLAP-region sought.

6.6 The VOLAP-tree

The VOLAP-tree is a structure that is used to provide a quick consolidated index for all the VOLAP information presented thus far. Figure 6.7 shows a partial VOLAP-tree that is used to index the VOLAP-cube constructed in the previous sections. Since the example presented uses only two dimensions, the VOLAP-tree seen is a two dimensional tree. However, in practice the VOLAP-tree is essentially a three dimensional KD-tree (3D-tree) with the exception that leaf nodes are used as references to the pre-partitioned virtual spaces corresponding to the OLAP regions.

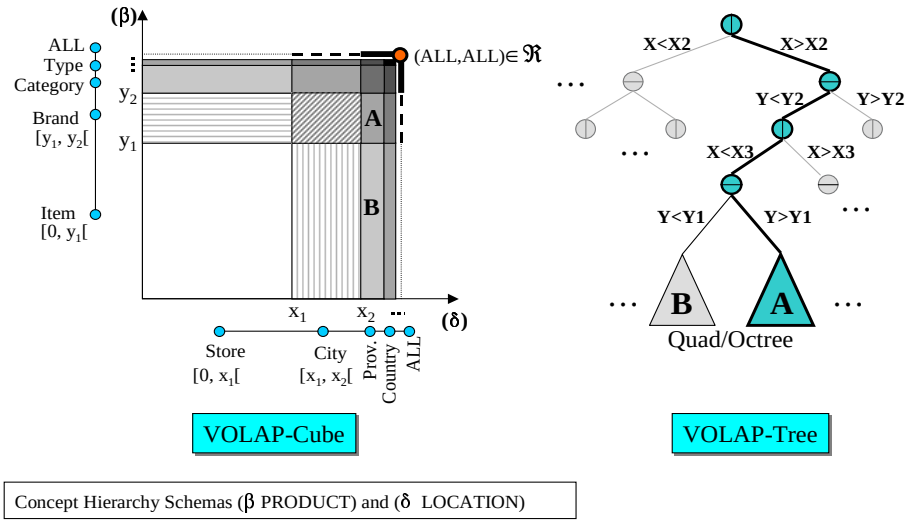


Figure 6.7: The VOLAP-Cube containing a materialized OLAP region for each group-by. The VOLAP-tree's upper portion is a KD-tree that indexes the regions. Each leaf points to an octree that partitions the region.

Using the reference points of OLAP regions (in the concept hierarchy nodes), the VOLAP-tree partitions the dimensions similar to a binary search tree.

The root of the VOLAP-tree is the root for a 3D-tree. At the leaf level, each 3D-tree leaf contains a pointer to the second layer, which is an Octree that recursively partitions that particular OLAP region into octants. Recursion continues until all octants at the leaf level do not contain more than a given number of data points. Within the VCU, the "VR Partition" module (Figure 4.5) uses the the user's location (UIM input) to determine the appropriate set of octree nodes to use for rendering. As the user's position changes through VR, so does the set of rendered octants (Section 6.4.1).

When the user performs an OLAP operation, the address of the new OLAP regions is obtained (Section 6.5.1) and used to traverse the VOLAP-tree in search for the associated leaf pointer. As we have mentioned, every OLAP region constitutes its own virtual world. The pointer obtained from the leaf node after an OLAP operation has been executed contains a reference to the octree that represents the new world. The octree data is then read to synthesize and render the new set of virtual objects. In other words, the upper layer of the tree corresponds to the OLAP-loop while the lower layer corresponds to the visualization-loop (Chapter 5).

The efficiency of this design stems from the fact that performing OLAP operation is simply equivalent to "transporting" the user to a different region within the VOLAP-cube. As an example, consider the two associated concept hierarchies that are shown next the axis that maps them in VR (Figure 6.7). Domain attributes of the dimension LOCATION at the lowest level, "store," are assigned the index values between [0,X_1[along the X axis. Similarly, the attributes defining "categories" in the PRODUCT dimension are assigned the range [Y_2, Y_3[. Assuming that the current view is (Province/Item), region **B**, executing a ROLL-UP operation on the dimension PRODUCT is equivalent to transporting the user into region **A**.

Chapter 7

Connecting the different Components

In the proceeding chapters we have presented the different processes that work together to create a unique environment for the mining and analysis of remote and distributed data sources. The existence of an *intra* and *inter* subsystem communication structure was implicit through the discussion. In this chapter the complete process is presented from end to end as we present the communication layer and how the three main components VCU, UIM and the VDW function with respect to one another.

7.1 The Communication Layer

Currently there are two approaches available that provide an API for communication purposes; **SOAP** (Simple Object Access Protocol)[15] and **CORBA** (Common Object Request Broker Architecture)[9]. The SOAP API has been introduced to the market recently and was intended to be used over TCP/IP (internet) to provide accessibility to various web-based applications. SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an *envelope* that defines a framework for describing what is in a message and how to process it, a set of *encoding rules* for expressing instances of application-defined data types, and a convention for representing *remote procedure calls* and responses. CORBA API consists of a set of methods that provide means, transparent to the client, to invoke (execute) a remote

procedure and await the return of the results in the form of an object. Each the client and server have what is called an ORB (Object Request Broker) that receives and processes the different requests.

CORBA provides a solution for the development of distributed heterogeneous applications. But why would one want to develop a distributed application in the first place? In many applications, the data sources used are distributed across several locations or systems. To be able to maintain this information where it is and access it only as needed saves storage provide a constantly up to date information. Another reason for distributing an application is the need to distribute the computations needed allowing for more focused and independent development of more more than one aspect independently. Both factors are immediately applicable to DIVE-ON. As the system seeks distributed data sources, the VDW performs its own computations to provide such data transparently to the VCU.

7.2 Implementing The CORBA Interfaces

In this section the communication layer implemented in DIVE-ON using CORBA is presented. One of the main design goals of our system is the ability to visually explore remote and distributed data sets; using the well-established CORBA API, neither the VCU nor the VDW have to provide considerations for the type of network, operating system, platform etc. The way that data is viewed in this model is as an object that can be requested by a client. The server in return, who can have several clients, "fulfills" this request by extracting the needed data and assembling it into an object that is then transmitted back. For complete details about implementing a CORBA networking solution, Baker[9] and Henning[45] provide a well presented step-by-step implementation methodology. We begin with the server software at the VDW.

7.2.1 Server-side Communication layer

This section introduces the implementation strategy used in developing the Server-Side of the Communication Layer, which we refer to as the *DCC-Servant*. Because only one data source can be visualized at once, the DCC-Servant only needs to maintain a single CORBA object at any given time. In general only one data source

will be present at a physical location, so exactly one servant will be instantiated for each CORBA object (in this case a DCC module). Currently, all server objects are transient. What this means is that if the system is shut down, state changes are forgotten and object references held become invalid. The DCC operations provided by DCC-Servant will be limited to a read-only service. This is sufficient, as most data mining applications do not perform regular updates to data warehouses.

To extract data from the data warehouse, an API is needed to provide access to the proprietary database encapsulated by the DCC. The system was designed such that changing from one database system to another would not require rebuilding the entire system. A minimal API has been developed to use in this system. The operations provided by the API allow the DCC-Servant to obtain an array of measure data and a set of schema hierarchies. Facilities to set the current level of abstraction have also been added. The DCC operations that are available to the client will be represented in the DCC-Servant interface.

The DCC-Servant is implemented according to the **IDL** (Interface Description Language) specification. Compiling the IDL definition of the interface using an **IDL-C++** compiler does this automatically. Each IDL definition is mapped to C++ skeleton classes. These classes provide all the basic operations of the DCC-Servant in pure virtual form. Therefore, the DCC-Servant must at least provide implementations for these basic operations.

Once implementations for all the basic DCC-Servant operations have been provided, a main server function is needed to start the Servants. This main function is responsible for initializing the ORB at run time, obtaining a root Object Adapter, creating a servant manager and activating the manager. An *Object Adapter* is used to provide the communication facilities to the CORBA objects. They provide objects with a way of interfacing with the ORB. Once this step is complete, the DCC-Servant needs to be coupled with a DCC module to connect it to the database. This gives a complete picture of the server-side communication of the DIVE-ON system.

7.2.2 Client-Side Communication layer

This section outlines the general structure of the Client-Side Communication layer that will be referred to as the *VCU-Client*. The VCU-Client is much simpler than the

DCC-Servant. Part of this is due to the fact that CORBA provides a template class of objects known as "**_var**" types. A _var object is created for each interface defined in the IDL specification. They are similar to pointers to objects but are designed to handle all the memory management issues involved without using pointers. This means that the programmer can create objects on the heap, assign them to their respective _var type and forget about them. This removes much of the complexity of dynamic memory allocation from the programmer.

The client must obtain a *reference* to a remote object before being able to use any of the DCC-Servant operations. This is done by first initializing the local ORB. The ORB's services are then used to obtain a reference to the remote object. Currently this is done by passing a stringified reference to the client. The client can then use ORB services to convert this stringified reference to an actual reference to a remote object. This reference must then be narrowed to a reference to an object of the desired type. Narrowing is similar to down casting but provides more safety. This means that an object reference cannot be narrowed unless the referenced object is an instance of the narrowed class. This is possible in downcast operations and may be a significant source of programming error. Once a narrowed object reference to the DCC-Servant is obtained, the client can request operations in the same way as methods are called on local objects. CORBA hides all the details of the object's location and implementation. This means that requests to objects in different namespaces or the same namespace will be semantically identical (at least on the outside).

Previously it was mentioned that a stringified reference was passed to the client to obtain a reference to a remote object. This may seem to contradict all claims of location transparency made of CORBA. Services are provided by ORBs to be able to obtain references to objects without the use of stringified references. However, at this stage in development, this approach was used for its simplicity. Later versions of the system will provide location independence.

The OMG[1] has provided a specification for a set of standardized services that must be provided with ORB implementations. These three services are the OMG Naming Service, the OMG Event Service and the OMG Trader Service. The nam-

[1]OMG:The Object Management Group page at http://www.omg.org/

ing service allows CORBA programs to obtain object references without the need for them to be explicitly passed to the calling client program. Object references are located through symbolic names. The OMG Trading Services allows clients to locate objects with the help of a program that acts as a trader. This allows the client to perform a dynamic lookup of a service based on the description of the service it needs. The OMG Event Service allows CORBA programs to use distributed callbacks. CORBA programs that do not use the Event Services are generally coupled and communicate synchronously. The OMG Event Service allows for decoupled communication between remote objects. Future versions of the VCU-Client will use the OMG Naming Service to provide location transparency to objects used by the system. This will enable the client to find the DCC-Servant without the need for a stringified reference.

7.3 Inter/Intra Subsystem Communications

All DIVE-ON internal messages communicated between the VCU and the VDW, and between the VDW and the data sources are formed as XML documents. The VDW is designed as a stand-alone server capable of fulfilling incoming inquires from *any* client. As seen in Figure 3.4 the server software (CORBA and SOAP servers) is built using a Java API layer, which is regarded as middleware that interprets incoming requests and encodes outgoing replies between the server modules and the DCC. These server modules are capable of providing their services by constructing a set of objects that a client can remotely call upon (invoke), and awaits the receipt of the XML document containing the response. Adhering to this mechanism, the VCU handles the IVR, OLAP and user interaction by using a client module to invoke the DCC-Shell object responsible for the particular service sought.

Inter-VDW communication is also implemented as client-server using CORBA and SOAP. To allow the DCC-Shell control over the distributed data sources, each source is built within a wrapper working either as a SOAP or a CORBA server. The DCC-Shell in turn uses a client module to access each source. This approach provides a solution to the problem of managing heterogeneous data sources in a manner that is transparent to the DCC. For example, regardless of whether the back-end source is a real data warehouse, relational database or even legacy system, the DCC-Shell

is always provided with a uniformed interface. The DCC-Shell CORBA and SOAP server objects (Figure 3.4) provide the functions needed for warehouse management and query. In a typical session a client first establishes connection to the VDW and then requests information pertaining to its contents. Having learned about the existing entities within the VDW, a client may then initiate a cycle requesting any given available cuboid.

7.4 VCU and VDW Protocol

The visualization experience begins with the user walking into the CAVE, effectively starting a visualization session. This is indicated to the DCC, which creates a new session ID and assign the variables used. Then the user uses the menu to inspect the type of viewable data maintained by the Virtual Data Warehouse (VDW). The VDW is constructed over, possibly, several data marts each of which is subject-specific; as a result, the VDW may contain more than one n-dimensional data cube that are built within the scope of a given analysis task. This information is passed to the VCU in the form of a XML document that contains the "name" of each of these cubes, their size and other descriptive details. Once a particular n-cube is chosen, the VCU needs to know the specifics of this cube. This includes information such as the number of data dimensions, the number of measures and what are their "names." This in formation is gathered by the server and presented to the user.

Chapter 8

Conclusions And a Future Agenda

The reviews obtained from our publications were very encouraging and seem to indicate that this work was well received and heading in the right direction. Deploying OLAP engines to a theme-oriented data warehouse is fairly recent and had shown a tremendous impact in the industry[1]. As the need for IVR applications to include more and more fields of science [12, 59, 54] in collaborative and networked IVR environments [71, 62, 61] grows, so will the need for exploring remote and distributed data sources [25, 84, 16]. Our work with DIVE-ON will continue to monitor these three areas closely and hope to be able to establish a system that will have significant repute. As we have indicated earlier, the number of scientists that are currently working on immersed visual data mining in any one given team around the world is another indication of the magnitude of commitment to this research direction.

In this thesis we have presented a solution that provides an OLAP data mining interface in immersive virtual reality. Our research sought alternate methods for exploring large volumes of heterogeneous data sets in a way that enables a user to effectively identify anomalies and implicit patterns for decision support. To accomplish this, DIVE-ON constructs a task specific navigation and object manipulation interface that is almost invisible to the user. Using tracking devices, the user navigates through data by walking, or flying, and interacts with its objects by simply "reaching out" for them.

[1]For current OLAP market share, vendors and latest technology see the OLAP report at: http://www.olapreport.com/

In Chapters 4 and 5 we have presented our approach for creating an IVR environment with data exploration in mind. According to research [42], the top priority is given to orienteering, object identification and navigation control. Drawing a reference grid and implementing the crosshair directed pointer (skitter) were designed to address the issue of orientation. Providing aggregate-based interaction allows the user to point and query data points, and hence identify the objects. Providing the user with an intuitive way to "fly" within the environment is provided so that the user can navigate through the environment. To produce a transplant interface, we have used the user's motion (walking) as a mean of navigation within a locality. This is equivalent to the way we explore our natural surroundings. To provide a mean for object interaction, the concept of clutched and hand-coupled menus was presented. This solution was the result of evaluating several published alternatives. A major problem with visualizing data is the problem of occlusion. Using of spheres was found to occlude less objects. In addition, the adaptation of 2D clutching to 3D was mainly done due to the occlusion problems noticed early in the project.

DIVE-ON creates a virtual world that is inhabited by one type of geometric objects (spheres or cubes) each one of which uses its colour and size to tell something about what it represents. We would like to examine the possibility of increasing the number of data mining measures presented by introducing more than one type of geometric objects. For example, a pyramid that points upwards could be used to indicate the existence of monotonic increase somewhere at a lower level, which is particularly useful in market analysis studies. The use of such objects could instantly identify several attributes relating to the ever changing consumer trends and utilization forecasting. Shapes can also be used as a cue, like colour and size, to visualize yet another discretized dimension.

8.1 Upcoming Research Agenda

The most immediate goal of our future research involves a user-based formal comparison evaluation between two interaction paradigms. The first, is the already implemented hand-coupled menu system and the new concept, yet to be implemented, is the *rotating torus*. Both systems provide 6-DOF through the use of hand-held trackers, which allow the user to move the menu system into an arbitrary

position within the virtual world. While the first provides 3D equivalent to what most users are accustomed to, the latter could reduce occlusion further and, potentially, provide more information. The objective is to learn why and which of the two paradigms is more accepted by the users while immersed in IVR. Another research issue we plan to investigate is the use of distorted views [18], also called fisheye or detail-in-context, in 3D graphics.

Next, we will seek appropriate means of incorporating a *data sonification* [78, 69] component to DIVE-ON, making it the first application that applies audible cues to data warehouses for the purpose of knowledge extraction. Since hearing is usually a background process, DIVE-ON will use the audible cues mainly to steer the user's foreground process, vision, into a direction that may need in-depth examination. Limiting the use of audible cues in this manner avoids the permeation of the IVR with sensory input that could lead to some undesired perceptual complexities. In addition, we are to examine the possibility of creating a new vision-based input scheme where the user uses hand gestures to execute various commands. A complete system evaluation by different users will be due at that point to compare DIVE-ON to other desktop tools. Results will be published during 2003.

We are also investigating extensions to XMDQL based on DMQL [40] in order to include data mining constraints for association rules and classification. These XMDQL extensions would be used for interactive classification as well as interactive evaluation of association rules from within the immersed virtual environment in the CAVE. The outcome of applying a data mining clustering algorithm is a good candidate for 3D visual manipulation and evaluation. The user can control the parameters and the algorithms that are performing the clustering quickly after a walking through the clusters for evaluation.

8.2 Tele-Immersion Aspects

In the tele-immersion arena, the EVL group have been active since 1995 (shortly after introducing the CAVE). There are many lessons to be learned from their experience to identify our niche in this direction as it pertains to data mining and warehousing. Once it has been shown that users can indeed use DIVE-ON for effective visual data mining in IVR, we will then address the issue of having multiple

users in different geographic locations virtually exploring the same data warehouse. Expanding the communication layer within DIVE-ON to incorporate master/slave synchronization is a possible solution. For this type of experiments we will use the **CAVElet** that was designed and built by Dr. Mark Green here at the University of Alberta. The Cavelet is a small, PC-based, version of the CAVE. Using both the Cavelet and the CAVE we can run our study of methods to synchronize the two IVR systems while providing a means of virtual collaboration between the two groups of users.

8.3 VOLAP Issues

In any IVR system the frame rate must always be kept above 22 frames per second to prevent VR sickness [8]. Since the information presented may be voluminous, increased scene complexity is unavoidable. To prevent frame rate slowdown with the increased flow of data to the IVR, it is important to provide a model that guarantees an acceptable minimum frame rate. What is needed is a method of rendering only visible segments of the virtual world as well as an appropriate multidimensional index structure that allows interactive mining operations. In this regards, we have presented a preliminary solution [5] in the form of a new data structure that indexes a multidimensional data cube as a set of spatially decomposed data mining regions. The new structure, VOLAP, must be examined further to obtain performance evaluation in terms of query and update complexity so that it could be enhanced to support dynamic warehouse updates. It is also important to optimize the size of this structure so that it can address concerns regarding sparse data cubes [13].

8.3.1 Time Travel: Adding Animation

So far the user has been able to maneuver effectively through the three data dimensions being viewed by providing a natural mapping onto the physical X, Y, and Z coordinates. In a future implementation, we would like to incorporate the ability of adding animation as a visual cue. When a person moves from point A to point B, their location with respect to the physical world changes. It is also true that if a person remains still in one point that their location also changes; it changes with respect to *time*. The idea of allowing the user to navigate through time is rather

natural and allows the user to (1) simultaneously travel through four dimension and (2) visually analyze any time varying phenomenon by animating the virtual objects so that they reflect how their visual cues (size and colour) vary as a function of time. The following example will further explain how time travel can indeed easily reveal patterns that are otherwise difficult to detect. It should be clear that this function is offered to the user only when "time" is actually an attribute within the data warehouse.

It has been illustrated how to use size and colour to visually depict certain aggregate attributes; would it also be possible to depict the change in pattern within a range of a given aggregate? For example, consider a view in which each virtual object represents the total *units sold* (size attribute) of a given product and brand by a given employee. The colour attribute represents the *profit margin* for the units sold. The user can view various domain values by simply navigating along the appropriate X, Y or Z axis. It is also possible to display and animate historic data through time.

Using the floating menu, the user would specify the time range that they wish to run. The VCU would then begin to update the colour and size of the virtual object as it progresses through the time record specified. Similar to any other dimension, the user may specify whether the playback is to take place with respect to different granularities such as day-to-day or year-to-year basis. This effectively adds motion as a visual cue that quickly alerts the user to many interesting patterns including market growth or decline, employee productivity, and changing consumer preferences to only name a few. Incorporating animation into the virtual objects is an extremely powerful tool that allows the analyst to explore massive amount of information almost effortlessly. For instance, assume that over time selling product K is becoming less profitable while selling product L is becoming more profitable. This can be easily identified by noticing that the objects on the horizontal plane (y $= K$) continue to include an increasingly higher content of blue while the (y $= L$) objects would show continuously higher red content.

Bibliography

[1] Sameet Agarwal, Rakesh Agrawal, and Prassad Deshpande et al. On the computation of multidimensional aggregates. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 1996.

[2] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawaji. Modeling multidimensional database. In *Proc. 1997 ACM-SIGMOD Conf. Management of Data*, pages 232–243, 1997.

[3] Daniel Aliaga and Anselmo A. Lastra. Automatic image placement to provide a guaranteed frame rate. In *Proc. ACM SIGGRAPH*, pages 307–316, Aug 1999.

[4] Ayman Ammoura, Osmar R. Zaïane, and Randy Goebel. Towards a novel OLAP interface for distributed data warehouses. In *Proc. 3rd Int. Conf. on Data Warehousing and Knowledge Discovery, (DaWaK'01)*, Munich, Germany, September 2001. `http://www.dexa.org`.

[5] Ayman Ammoura, Osmar R. Zaïane, and Yuan Ji. Immersed visual data mining: Walking the walk. In *Proc. 18th British National Conference On Databases (BNCOD'01)*, Oxford, UK, 2001.

[6] Edward Angel. *Interactive Computer Graphics, A Top Down Approach with OpenGL.* Addison-Weseley Longman Inc., Harlow England,, 1997.

[7] Computer Associates. VISION:Journey (A product). Brochure, 2001. `http://ca.com/products/vision/prod_info/vision_journey_pd.htm`.

[8] M. Pauline Baker. Human factors in virtual environments for the visual analysis of scientific data. *NCSA Publications: National Centre for Supercomputer Applications*, 1996.

[9] Seán Baker. *CORBA Distributed Objects Using Orbix.* ACM Press, Addison-Weseley, Longman Inc., Harlow England, 1997.

[10] N. Beckmann, Hans-Peter Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM SIGMOD Int. conf. on Management of Data*, pages 322–331, Atlantic City, NJ, 1990.

[11] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, pages 28–39, Bombay, India, 1996.

[12] Wes Bethel. Chemical flooding in a virtual environment - a survivor's guide to VR development, May 1994. `http://www-vis.lbl.gov/publications/avs94/avs94.html`.

[13] Kevin Beyer and Raghu Ramakrishnan. Bottom-up computation of sparse and Iceberg-CUBES. In *Proc. 1999 ACM-SIGMOD Conf. Management of Data*, pages 359 – 367, 1999.

[14] Eric A. Bier. Skitters and jacks: Interactive 3D positioning tools. In *Proc. Workshop on Interactive 3D Graphics*, pages 183–196. ACM Press, 1986.

[15] Don Boxand, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP). W3C, May 2000. `http://www.w3.org/TR/SOAP/`.

[16] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *ICDE*, pages 389–398, 2000.

[17] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. "distortion viewing techniques for 3D data". In *"INFO-VIS'96: Proceedings of the IEEE Conference on Information Visualization"*, pages 46–53, Oct 1996.

[18] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia. Extending distortion viewing from 2D to 3D. *IEEE Computer Graphics and Applications*, 17(4):42–51, Jul 1997.

[19] Eric Frederick Charlton. *An Octree Solution to Conservation-laws over Arbitrary Regions (OSCAR) with Applications to Aircraft Aerodynamics.* University, University of Michigan, 1997.

[20] Surajit Chauduri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, March 1997.

[21] Paolo Cignoni, Paola Marino, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, April 1997.

[22] Oracle Corporation. Oracle OLAP products. White Paper. `http://www.oracle.com/publications/index.html?content.html`.

[23] C. Cruz-Neira, D. Sandin, and T. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *Proc. ACM SIGGRAPH*, Aug 1993. http://www.evl.uic.edu/EVL/VR/systems.shtml.

[24] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *Proc. Int. Conf. Data Engineering (ICDE'00)*, pages 367–378, San Diego, CA, 2000.

[25] Cristina Dutra de Aguiar Ciferri and Fernando da Fonseca de Souza. Distributing the data warehouses. In *Proc. Conf. Brazilian Symposium on Databases (SBBD'00)*, Brazil, 2000.

[26] Michael F. Deering. The HoloSketch VR sketching system. *Communications of the ACM*, 39(5):54–61, 1996.

[27] T. Ertl, R. Westermann, and R. Grosso. Multiresolution and hierarchical methods for the visualization of volume data. *Visualization of Volume Data*, 15(5/6):31–42, 2000.

[28] Martin Ester, Jorn Kohlhammer, and Hans-Peter Kriegel. The DC-Tree: A fully dynamic index structure for data warehouse. In *Proc. 2000 Int. Conf. Data Engineering (ICDE'00)*, San Diego, CA, 2000.

[29] Usama Fayyad, G. Piatesky-Shapiro, and P. Smyth. Data mining and knowledge discovery in databases. *Communications of the ACM*, 39(11):24–26, 1996.

[30] S. Feiner, B. MacIntyre, M. Haupt, and E. Solomon. Windows on the world: 2D windows for 3D augmented reality. In *Proc. UIST'93*, pages 145–155, 1993.

[31] J.D. Foley, A. VanDam, S.K. Feiner, and J.F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 2 edition, 1995. ISBN: 0-20184-840-6.

[32] NCSA National Center for Super Computing. Projection-based virtual environment sites. This is a web-based publication that is frequently updated. `http://archive.ncsa.uiuc.edu/VR/cavernus/users.html`.

[33] The Natinal Center for Super Computing. NCSA Vis&VE: Intro to visualization and virtual reality. Internet Information. `http://www.ncsa.uiuc.edu/About/NCSA/`.

[34] Duncan Galloway, Eric Wolanski, and Brian King. Coastal oceanography data visualization using data explorer. Technical report, The IBM International Foundation, 1996. White papers.

[35] William A. Giovinazzo. *Object Oriented Data Warehouse Design.* Prentice-Hall Inc., New Jersey, first edition, 2000.

[36] Ian S. Graham and Liam Quin. *XML Specification Guide.* Wiley Computer Publishing, 1999. ISBN: 0-471-32753-0.

[37] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational operator generalizing Group-By, Cross-Tab, and Sub-Totals. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[38] Himanshu Gupta, Venky Harinarayan, and Anand Rajaraman. Index selection for OLAP. In *Proc. 1997 Int. Conf. Data Engineering (ICDE'97)*, 1997.

[39] J. Han, J. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, G. Liu, K. Koperski, Y. Lu, N. Stefanovic, L. Winstone, B. Xia, O. R. Zaïane, S. Zhang, and H. Zhu. DBMiner: A system for data mining in relational databases and data warehouses. In *Proc. CASCON: Meeting of Minds*, pages 249–260, Toronto, Canada, November 1997.

[40] J. Han, Y. Fu, K. Koperski, and O. R. Zaïane. DMQL: A data mining query language for relational databases. In *SIGMOD Workshop on Research Issues on Data mining and Knowledge Discovery*, pages 24–34, Montreal, Canada, June 1996.

[41] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, first edition edition, April 2000.

[42] Chris Hand. A survey of 3D interaction techniques. *Computer Graphics Forum*, 16(5):269–281, Dec. 1997.

[43] Venky Harinarayan, Anand Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. 1996 ACM-SIGMOD Conf. Management of Data*, pages 205–216, 1996.

[44] Donald Hearn and M. Pauline Baker. *Computer Graphics.* Prentice-Hall Inc., 1990. ISBN: 0-13-165382-2.

[45] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Weseley Longman Inc., Reading Massachusetts, 1999.

[46] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Weseley Longman Inc., Reading Massachusetts, 1999.

[47] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in OLAP data cubes. In *Proc. ACM SIGMOD Conference on the Management of Data*, pages 73–88, 1997.

[48] Accrue Software INC. Architecture of Applix iEnterprise mobile processing environment. White Paper, November 2000. `http://www.applix.com/literature/`.

[49] SGI INC. Opengl performer. Web-based information, 2000. `http://www.sgi.com/developers/devtools/apis/performer.html`.

[50] SGI INC. Opengl volumizer. Web-based information, 2000. `http://www.sgi.com/developers/devtools/apis/volumizer.html`.

[51] MicroStrategy Incorporated. The case for relational OLAP. White Papers, 1995. `http://www.microstrategy.com/files/whitepapers/wp_rolap.pdf`.

[52] Informix. Informix MetaCube 4.2. White Papers, 2001. `http://www.informix.com/informix/products/`.

[53] Bill H. Inmon. *Building the Data Warehouse*. John Wiley, 1992.

[54] Vijendra Jaswal. CAVEvis: Distributed real-time visualization of time-varying scalar and vector fields using the CAVE virtual reality theater. In *IEEE Visualization*, Oct 1997.

[55] F. P. Brook Jr. What's real about virtual reality. *IEEE Computer Graphics and Applications*, 19(6):16–27, Nov.-Dec. 1999.

[56] Daniel A. Keim. Visualization techniques for mining large databases: A comparison. *Transactions on Knowledge and Data Engineering*, 8(6):923–938, Dec. 1996.

[57] Daniel A. Keim. Visual data mining (tutorial). In *Int. Conf. on Very Large Databases (VLDB'97)*, Athens, Greece, 1997.

[58] Daniel A. Keim. Visual techniques for exploring databases (invited tutorial). In *Int. Conf. on Knowledge Discovery and Data Mining (KDD97)*, Newport Beach, 1997.

[59] Robert V. Kenyon and Michelle B. Afenya. Training in virtual and real environments. *Annals of Biomedical Engineering*, 23(1):445–455, 1995. `http://www.evl.uic.edu/EVL/RESEARCH/PAPERS/KENYON/ppt.html`.

[60] Kengo Koiso, Takahiro Matsumoto, and Katsumi Tanaka. Spatial presentation and aggregation of georeferenced data. In *Proc. Int. DASFA*, pages 153–160, 1999.

[61] Jason Leigh and Thomas DeFanti et al. A review of tele-immersive applications in the CAVE research network. In *Proceedings of IEEE VR '99*, Houston, TX, March 1999. `http://www.evl.uic.edu/paper/pdf/Review.pdf`.

[62] Jason Leigh, Andrew Johnson, and Thomas DeFanti et al. A methodology for supporting collaborative exploratory analysis of massive data sets in tele-immersive environments. In *Proceedings of the 8th IEEE International Symposium on High Performance and Distributed Computing*, Redundo Beach, CA, June 1999. `http://www.evl.uic.edu/paper/pdf/Review.pdf`.

[63] Robert Lindeman, John L. Sibert, and James K. Hahn. Towards usable VR: An empirical study of user interfaces for immersive virtual environments. In *Proc. CHI'99*, May 1999.

[64] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. *Computer Graphics*, 31(Annual Conference Series):199–208, 1997.

[65] *MR-Toolkit:Virtual Reality, 3D and 2D User Interface Software Tools.* http://www.cs.ualberta.ca/~graphics/MRToolkit.html.

[66] Inderpal Mumick, Dallan Quass, and B. S. Mumick. Maintenance of data cubes and summery tables in a warehouse. In *Proc. 1997 ACM-SIGMOD Conf. Management of Data*, pages 100–111, 1997.

[67] Seigo Muto and Masaru Kitsuregawa. Improving main memory utilization for array-based datacube computation. In *DOLAP '98, Proc. ACM First International Workshop on Data Warehousing and OLAP*, pages 28–33, Bethesda, Maryland, November 1998. ACM.

[68] The NASA ESS project Fy97 annual report on data visualization and sonification. http://sdcd.gsfc.nasa.gov/ESS/annual.reports/ess97/sys/vr.html.

[69] Monique Noirhomme-Fraiture. Multimedia support for complex multidimensional data mining. In *Workshop on Multimedia Data Mining, MDM/KDD2000*, Boston, MA, August 2000. `http://www.cs.ualberta.ca/~zaiane/mdm_kdd2000/proceedings.html`.

[70] Patrick O'Neil. *Database Principles, Programming, Performance.* Morgan Kauffman Publishers, San Fransisco, first edition, 1994.

[71] Kyoung Park, Abhinav Kapoor Chris Scharver, and Jason Leigh. Exploiting multiple perspective in tele-immersion. In *Proceedings of IPT 2000: Immersive Projection Technology Workshop*, June 2000. `http://www.evl.uic.edu/cavern/multiperspective/`.

[72] Emilee Patric, Dennis Cosgrove, Aleksandra Slovkovic, Jennifer Ann Rode, Thom Verratti, and Greg Chiselko. Using a large projection screen as an alternative to head-mounted display for virtual environments. In *Proc. CHI'00*, pages 478–485, Aug 2000.

[73] R. M. Pickett. Visual analysis of texture in the detection and recognition of objects. In B. S. Lipkin, editor, *Picture Processing and Psycho-Pictorics.* Academic Press New York, 1970.

[74] The OLAP Report. Market share analysis. Web-based information, 2001. `http://www.olapreport.com/Market.htm`.

[75] George Robertson, Maarten van Dantzich, Daniel Robbins, Mary Czerwinski, Ken Hinckley, Kirsten Risden, David Thiel, and Vadim Gorokhovsky. The task gallery: A 3d window manager. In *Proc. CHI'00*, pages 494–501, Aug 2000.

[76] Nick Roussopoulos, Yannis Kotidis, and Mema Roussopoulos. Cubetree: organization of and bulk incremental updates on the data cube. In *Proc. ACM SIGMOD*, pages 89–99, Tucso, Arizona, May 1997.

[77] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley Inc., first edition, 1989.

[78] Carla Scaletti and Alan B. Craig. Using sound to extract meaning from complex data. *The International Society for Optical Engineering*, 1459, 1991.

[79] Raj Shekar, Elias Fayyad, Roni Yagel, and J. Frederick Cornhill. Octree-based decimation of marching cubes surfaces. In *Proc. IEEE Visualization*, pages 335–342, 1996.

[80] Hyperion Software. The role of the multidimensional database in a data warehousing solution. White Paper. `http://www.essbase.com/main.asp?webpagekey=14`.

[81] Yin Jenny Tam. Datacube:its implementation and application in olap mining. Masters thesis, University of Simon Fraser, Vancouver, British Columbia, Canada, September 1998.

[82] Kurt Thearling, Barry Becker, and Dennis DeCosta. Visualizing data mining models. In *Proc. Integration of Data Mining and Data Visualization Workshop*, 1998.

[83] Andries van Dam, Andrew S. Forsberg, David H. Laidlaw, Joseph J. LaViola, Jr., and Rosemary M. Simpson. Immersive vr for scientific visualization: A progress report. *IEEE Computer Graphics and Applications*, 20(6):26–52, November 2000.

[84] Panos Vassiliadis and Timos K. Sellis. A survey of logical models for OLAP databases. *SIGMOD Record*, 28(4):64–69, 1999.

[85] Yanqing Wang and Christine MacKenzie. Object manipulation in virtual environments: Relative size matters. In *Proc. CHI'99*, May 1999.

[86] Matthew Ward and Daniel Keim. Screen layout methods for multidimensional visualization. In *Euro-American Workshop on Visualization of Information and Data (CODATA'97)*, page 1. IEEE Press, June 1997.

[87] Colin Ware and Glenn Franck. Evaluating stereo and motion cues for visualizing information nets in three dimensions. *ACM Transactions on Graphics*, 15(2):121–140, 1996. `http://www.ccom.unh.edu/vislab/PDFs/`.

[88] Stephen Wehrend and Clayton Lewis. A problem-oriented classification of visualization techniques. In *Visualization '90*, pages 139–143. IEEE Press, 1990.

[89] Jane Wilhems and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[90] Matthias M. Wloka and Eliot Greenfield. The virtual tricorder: A uniform interface for virtual reality. In *ACM Symposium on User Interface Software and Technology*, pages 39–40. ACM Press, November 1995.

[91] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide.* Addison Wesely, Reading Massachusetts, third edition, 1999. ISBN 0-201-60458-2.

[92] XML-RPC home page: Simple cross-platform distributed computing, based on the standards of the internet. http://www.xmlrpc.com.

[93] Osmar R. Zaïane and Ayman Ammoura. On-line analytic precessing while immersed in a CAVE. In *Proc. Int. IEEE Conference on User Interfaces for Data Intensive Systems (UIDIS'01)*, Zurich, Switzerland, May 2001.

[94] Yihong Zhao, Prassad M. Deshpande, and Jeffrey F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. ACM SIGMOD*, pages 159–170, Tucson, Arizon, May 1997.

Appendix A

Related Arithmetic

This appendix presents some of the arithmetic performed by the UIM to perform rotations and provide the user with the hand-coupled menu system. Such arithmetic is well known and used by most real-time graphical rendering engines such as OpenGL. Most often the VR world that is rendered is based on a mathematical model consisting of nothing more than a set of vertex groupings. In OpenGL, the engine used by DIVE-ON, vertices of polygons are the basic building block for the VR world. Even when the object rendered "looks" like a smooth 3D sphere, its building units are still vertices. First we will look at the quaternion since it is the basic quantity used for providing 6-DOF in most interactive VR applications.

A.1 What is a Quaternion?

Given any rigid object, the set of all possible rotations fit into a coherent algebraic structure called a **quaternion**. Every rotation corresponds to a unique unit quaternion which can be expressed by $a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ where $a, b, c, d \in \mathbf{R}$ such that $a^2 + b^2 + c^2 + d^2 = 1$. It is important to note that only unit quaternion can be used in representing rotation. To convert a quaternion into a unit quaternion, its length must be first calculated. Each component i sthen devided by that this length. The length i ssimply computed by using:

$$|q| = \sqrt{a^2 + b^2 + c^2 + d^2}$$

Quaternions are used often in animation or in IVR due to the simplicity in providing continuous rotating motion. Performing successive rotations on an object is

equivalent to multiplying the corresponding quaternions [31], which is very effective when the rotation is about more than one multiple axis. The quaternions representing the axii of rotation can be simply multiplied and the resultant quaternion can then be used to represent the composite rotation. In OpenGL this is often done by converting the final quaternion into a rotation matrix by which the object's vertices are multiplied.

In the VizRoom, DIVE-ON obtains the information about the user's hand and head orientation through the hardware drivers, which provide a 4-tuple representation of a quaternion $q = (a, b, c, d)$. This information is packaged with the x, y, z coordinates of the trackers in the location structure:

```
struct MR_eye {
    float  x, y, z;
    float quat[4];
}
typedef struct MR_eye *EYE;
```

A.1.1 Converting The Quaternion to a Rotation Matrix

Assume that we have the 4-tuple quaternion $q = (a, b, c, d)$, then q can be transformed into a rotation matrix simply by performing the following computation:

$$\begin{pmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & 1-2b^2-2c^2 \end{pmatrix}$$

A.2 6-DOF and Hand-Coupled Menus

Reading text in VR is very tricky since the text has to be as close to vertical as possible for the user to be able to read it. The user interface manager (UIM) the hand-held tracker provides the information needed for the system to control where the panel should be positioned. To do this, the UIM maps the location and orientation of the hand in the real world to the location and orientation of the menu in VR.

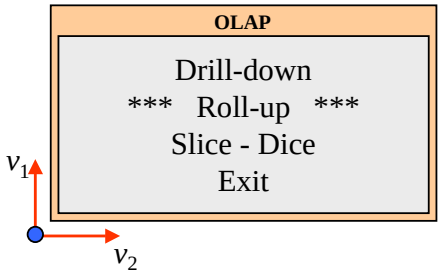


Figure A.1: Each panel must be specified by a lower-left corner point and two orthogonal vectors that are computed from the quaternion and the unit vectors.

To specify the location and orientation of the menu we need to specify two orthogonal vectors and a point on the panel. As in Figure A.1, the two orthogonal vectors $\vec{v_1}$ and $\vec{v_2}$ define the horizontal and vertical axis of the panel. What we want is for the normal of the panel to always point in the direction of the user and that the horizontal vector to be parallel to the X axis. We use the rotations of the standard unit vectors to get at the needed vectors:

```
//Defining the basic x, y, z unit vectors
static vector iVec[3] = {1.0, 0.0, 0.0};  //
static vector jVec[3] = {0.0,1.0, 0.0};   //
static vector kVec[3] = {0.0, 0.0, 1.0};  //
```

The parameter `savedEye` is the value of the current tracker location as it arrives in the structure shown above. Next we show how you can apply some of the MR_Toolit functions to perform the rotations, set the menu parameters and render the result. The following line of code specifies the two vectors $\vec{v_1}$ and $\vec{v_1}$ that are shown in the Figure.

```
MR_panel_set_location( sysPanel, origin, v1, kVec);
```

The V1 parameter is obtained by rotating the Y unit vector $\vec{j}$ by the current quaternion. We then use the Z unit $\vec{k}$, the parameter kVec, to provide the two vectors needed. The example below shows the corresponding method:

```
// Display the panel that has been made and set.

    void
    Interaction::showIt (MR_eye savedEye) {
    int i;

    vector rotatedVec [3];  // The rotated vector.
    if (!menuOn) return;    // Do not display, button has not pressed.

    vrotq (savedEye.quat, jVec, rotatedVec);  // Rotate the j by the quaternion

    // Assign V1 to the rotated unit vic j (Y unit)
    for (i = 0; i < 3; i++) v1[i] = (rotatedVec [i]);

    origin [0] = savedEye.x * factorX;  // The "corner" point of V1, V2 (perp.)
    origin [1] = savedEye.y * factorY;  // or, the translation of the V1V2 plane
    origin [2] = savedEye.z * factorZ;  // Response factor (factor_{x,y,z})

    MR_panel_set_location( sysPanel, origin, v1, kVec); //  Assign the param. of panel
    MR_panel_draw_panel( sysPanel);  // Render
}
```

Appendix B

Colour Coding Scalers

This appendix describes how DIVE-ON colour encodes the scaler values of a measure. There are two approaches to colour encoding on the perceptual level;using *different luminance* values for one colour or using a *colour palette* consisting of highly contrasted colour. Due to the light conditions in the CAVE, we have implemented a the later.

Given a basic colour (red, green, blue), it luminance is a measure of the amount of white light present in the colour. In other words, luminance defines the brightness attribute of a colour. This can be used to identify the value of a scaler; the more luminant the bigger the magnitude. Take the colour red for instance, we could base our model from maximum value represented by a fully saturated colour attribute (R = 255, G = 0, B = 0) in the HLS model this corresponds to (H = 0, L = 128, S = 255). OpenGL uses the RGB colour model and all other models, HLS, HSV, CMYK must be converted to this model [91].

B.1 Normalization

In the mapping phase of the data visualization process, some transformation must take place so that the numeric values can be mapped to their perspective geometric attributes. Normalization is typically a mapping method while projection is used to map multiple dimensions at the same time (such as projecting N dimension onto the plane). The VCU uses the *min-max* normalization function which performs a linear transformation on the original data, and hence preserving inter-value variations in the data set. Assuming that the data set to be normalized is A, then the min-max

functions maps a value v into $v\prime$ according to the following equation:

$$v\prime = \frac{v - min_A}{max_A - minA}(newMax_A - newMin_A) + newMin_A$$

The value min_A and max_A are the minimum and the maximum values present in the set A respectively. The above function maps the A to the new interval $[newMin_A, newMax_A]$ which can be defined by the application depending on the way the normalized values are to be used.

The *z-mean* normalization method can be used when the min and max of the set A are unknown a priori but the standard deviation and the mean are known:

$$v\prime = \frac{v - mean_A}{standardDev_A}$$

B.2 Sample

To represent the aggregates using spheres, the colour coding of the numeric value is reflected by the colour of the sphere. After normalization, the sphere light deflecting, absorbing, and emitting properties have to be set. OpenGL then uses these setting to calculate how the surface of the sphere should react to light and ,inadvertently, display the proper shading of the colour required. the Following is a segment of this is set.

```
// set the material property array for diffusing light
  mat_diffuse [0] = (float) R;  // Red
  mat_diffuse [1] = (float) G;  // Green
  mat_diffuse [2] = (float) B;

  ...

// set the material property array for emitting light
  mat_emission [0] = (float) (R/2.00);
  mat_emission [1] = (float) (G/2.00);
  mat_emission [2] = (float) (B/2.00);
```

```
// set the properties
  glMaterialfv (GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialfv (GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv (GL_FRONT, GL_SHININESS, mat_shininess);
  if (emitt)
    glMaterialfv (GL_FRONT, GL_EMISSION, mat_specular);
  else {
    glMaterialfv (GL_FRONT, GL_EMISSION, mat_nothing);
    glMaterialfv (GL_FRONT, GL_AMBIENT, mat_nothing);
    }
  if (b3on)
      glMaterialfv (GL_FRONT, GL_AMBIENT, mat_emission);
```

Appendix C

State-of-the-art Commercial API Graphics Accelerators

Before designing VOLAP, we had to evaluate some of the available solutions. The purpose of this appendix is to present a brief introduction and links to some of the most recent commercial solutions that implement the state-of-the-art algorithms to address certain types of data, tasks, platforms, and visualization environments.

SGI's **OpenGL Volumizer** is a library of classes that facilitates the visualization of voxel-based data sets common in geographic, medical, scientific, and engineering applications. It was released in 1999 to include various optimizations for rendering complex isosurfaces. The Volumizer is built upon the concept of the voxel as the fundamental volumetric primitive, and the tetrahedron – a four-sided, three-dimensional solid object – as the fundamental large- scale volumetric primitive. In DIVE-ON, the data presented in terms of much simpler voxels, a vertex, which is provided by any graphics API. More information can be found in [50].

The **OpenGL Performer** [49] does indeed seem to be able to solve the problem of guaranteed frame rate, however, this is needed for a situations where the entire data is available on a desk. The OpenGL performer uses multiprocessors to create *scene graphs* (comparable to our octree partition) that are rendered on a need-bases and they provide a LOD (level of detail) for fast terrain rendering.

Appendix D

Glossary of Terms

ALL: Is a reserved world that indicates a total aggregation of the attribute values corresponding to that particular dimension. "ALL" is itself considered as an attribute value of the highest generalization or abstraction.

API: Application Programming Interface is a set of methods of functions that are provided to the programer to include in their code.

Blocking: One of the possible states that a process can exist in is the "blocked" state to indicate that the process is waiting for some event to occur. To say that a multiprocess program runs without blocking, means that all child processes will not enter the blocked state.

Clipping Plane: Using OpenGL one would have to define the viewing volume by specifying the viewing frustum. The plane of the frustum that is lies away from the line of sight is called the clipping plane since no vertices are rendered beyond this plane.

Data Mining Algorithm: An algorithm that inputs a set of data and outputs a set of "interesting" observations. The meaning of interestingness is completely user specified and task dependent.

Data Space: The dimensions of the 3D array containing the data measures for the data being visualized.

Frame Rate: The number of times that the graphics engine is capable of reloading the frame buffer in one second. The frame rate is usually measured by frames

per second or f/s.

Graphics Primitives: Computer graphics programmers rely on a set of "drawing" functions that they can combine to create a scene. This set is called a set of primitives and is usually associated with special hardware circuitry that guarantees fast display. For example, OpenGL primitives are (Point, Line, Polygon, Bitmap, and Image).

Hand-Coupled: We use the term hand-coupled to refer to the direct mapping between the user's hand and a virtual object. Hand-coupling virtual objects creates the illusion that the user is actually grabbing an object with their own hands. This is done through the use of 6 degrees of freedom hand-held tracking devices.

Quadtree: A regular recursive spatial decomposition data structure. The quadtree is usually applied to 2D region or point data for a hierarchical indexing through partitioning. The entire region dimensions are represented by the root node which is divided into 4 quadrants by an integer division of the dimensions. The same procedure is recursively applied to each quadrant until the recursion bottoms out when a certain threshold has been attained.

Real-Time Rendering: The ability to refresh the frame buffer correspondingly with some real life event. That is the ability to synchronizes the graphics update with an external event such as the user movement or the input of a continues data stream.

Recursive Spatial Partitioning: Given a volume, it can be recursively decomposed into a collection of adjoining, disjoint set of sub-volumes. By adjoining we mean that the decomposition preserves the spatial properties of the volume. By recursive we mean that the decomposition algorithm is first applied to the entire set and then to all resulting subvolumes.

State Machine: A finite deterministic automata. That is a device that which can be in one of a given set of states. Given a new input, that state changes according to a state table that dictates the new state transition with respect to this particular input.

Transparent Interface: The ultimate goal of computer-human interaction research is to provide a user interface that makes the computer invisible to the user by stressing the semantics of the interaction not the syntax. An interface that caters to this idea is said to be transparent. In IVR environments this can be achieved by simply adhering to the methodologies by which humans interact with their natural worlds.

OLAP Loop: A term that we use to refer to a session where the user is engaged in executing a sequence of OLAP operations.

Point Tree: A tree index structure that is used to index a set of distinct points. Such trees are utilized when the purpose of use is to to perform point queries rather than range queries. See also, region tree.

Region Tree: This is a type of structure that is used to index multiple ranges of data to specify regions in multidimensional data sets. Such trees are utilized when the purpose of use is to perform region, or range, queries rather than specific point queries. Our VOLAP-tree is a region-based tree structure.

Virtual Space: The dimensions of the minimum bounding box containing all coordinates for each virtual object inhabiting the virtual world.

Visualization Loop: In the virtual environment, a continuous input stream is receive by the visualization module to report the user's current viewpoint. This input is a part of a loop that continuously updates the scene according to the input as the user navigates the environment.

VizRoom: This is the name of the CAVE implementation at the University of Alberta. VizRoom is a 3-wall CAVE with two tracking devices.

VR Sickness: Within an immersive virtual environment, if the image transformation rate corresponding to the user's motion is significantly slower than that expected by the human's sensorimotor system (less than 16 f/s), nausea and/or vomiting may occur. Such symptoms are called VR sickness and may be avoided by guaranteeing a frame rate above 20 f/s.